```
**********************************************************
  267081 Mon May  5 11:11:22 2014
new/usr/src/uts/common/io/fibre-channel/fca/emlxs/emlxs_dhchap.c
4786 emlxs shouldn't abuse ddi_get_time(9f)
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Emulex.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 */
#endif /* ! codereview */


#include <emlxs.h>

#ifdef DHCHAP_SUPPORT

#include <md5.h>
#include <sha1.h>
#ifdef S10
#include <sha1_consts.h>
#else
#include <sys/sha1_consts.h>
#endif /* S10 */
#include <bignum.h>
#include <sys/time.h>

#ifdef  S10
#define BIGNUM_CHUNK_32
#define BIG_CHUNK_TYPE                  uint32_t
#define CHARLEN2BIGNUMLEN(_val)         (_val/4)
#endif  /* S10 */

#define RAND

#ifndef ENABLE
#define ENABLE   1
#endif  /* ENABLE */

#ifndef DISABLE
#define DISABLE   0
#endif  /* DISABLE */
```



```
/* Required for EMLXS_CONTEXT in EMLXS_MSGF calls */
EMLXS_MSG_DEF(EMLXS_DHCHAP_C);

static char *emlxs_dhc_pstate_xlate(uint32_t state);
static char *emlxs_dhc_nstate_xlate(uint32_t state);
static uint32_t emlxs_check_dhgp(emlxs_port_t *port, NODELIST *ndlp,
        uint32_t *dh_id, uint16_t cnt, uint32_t *dhgp_id);
static void emlxs_dhc_set_reauth_time(emlxs_port_t *port,
        emlxs_node_t *ndlp, uint32_t status);

static void emlxs_auth_cfg_init(emlxs_hba_t *hba);
static void emlxs_auth_cfg_fini(emlxs_hba_t *hba);
static void emlxs_auth_cfg_read(emlxs_hba_t *hba);
static uint32_t emlxs_auth_cfg_parse(emlxs_hba_t *hba,
        emlxs_auth_cfg_t *config, char *prop_str);
static emlxs_auth_cfg_t *emlxs_auth_cfg_get(emlxs_hba_t *hba,
        uint8_t *lwwpn, uint8_t *rwwpn);
static emlxs_auth_cfg_t *emlxs_auth_cfg_create(emlxs_hba_t *hba,
        uint8_t *lwwpn, uint8_t *rwwpn);
static void emlxs_auth_cfg_destroy(emlxs_hba_t *hba,
        emlxs_auth_cfg_t *auth_cfg);
static void emlxs_auth_cfg_print(emlxs_hba_t *hba,
        emlxs_auth_cfg_t *auth_cfg);

static void emlxs_auth_key_init(emlxs_hba_t *hba);
static void emlxs_auth_key_fini(emlxs_hba_t *hba);
static void emlxs_auth_key_read(emlxs_hba_t *hba);
static uint32_t emlxs_auth_key_parse(emlxs_hba_t *hba,
        emlxs_auth_key_t *auth_key, char *prop_str);
static emlxs_auth_key_t *emlxs_auth_key_get(emlxs_hba_t *hba,
        uint8_t *lwwpn, uint8_t *rwwpn);
static emlxs_auth_key_t *emlxs_auth_key_create(emlxs_hba_t *hba,
        uint8_t *lwwpn, uint8_t *rwwpn);
static void emlxs_auth_key_destroy(emlxs_hba_t *hba,
        emlxs_auth_key_t *auth_key);
static void emlxs_auth_key_print(emlxs_hba_t *hba,
        emlxs_auth_key_t *auth_key);

static void emlxs_get_random_bytes(NODELIST *ndlp, uint8_t *rdn,
        uint32_t len);
static emlxs_auth_cfg_t *emlxs_auth_cfg_find(emlxs_port_t *port,
        uint8_t *rwwpn);
static emlxs_auth_key_t *emlxs_auth_key_find(emlxs_port_t *port,
        uint8_t *rwwpn);
static void emlxs_dhc_auth_complete(emlxs_port_t *port,
        emlxs_node_t *ndlp, uint32_t status);
static void emlxs_log_auth_event(emlxs_port_t *port, NODELIST *ndlp,
        char *subclass, char *info);
static int emlxs_issue_auth_negotiate(emlxs_port_t *port,
        emlxs_node_t *ndlp, uint8_t retry);
static void emlxs_cmpl_auth_negotiate_issue(fc_packet_t *pkt);
static uint32_t *emlxs_hash_rsp(emlxs_port_t *port,
        emlxs_port_dhc_t *port_dhc, NODELIST *ndlp, uint32_t tran_id,
        union challenge_val un_cval, uint8_t *dhval, uint32_t dhvallen);
static fc_packet_t *emlxs_prep_els_fc_pkt(emlxs_port_t *port,
        uint32_t d_id, uint32_t cmd_size, uint32_t rsp_size,
        uint32_t datalen, int32_t sleepflag);

static uint32_t *emlxs_hash_vrf(emlxs_port_t *port,
        emlxs_port_dhc_t *port_dhc, NODELIST *ndlp, uint32_t tran_id,
        union challenge_val un_cval);


static BIG_ERR_CODE
emlxs_interm_hash(emlxs_port_t *port, emlxs_port_dhc_t *port_dhc,
```

```
128         NODELIST *ndlp, void *hash_val, uint32_t tran_id,
129         union challenge_val un_cval, uint8_t *dhval, uint32_t *);

131 static BIG_ERR_CODE
132 emlxs_BIGNUM_get_pubkey(emlxs_port_t *port, emlxs_port_dhc_t *port_dhc,
133         NODELIST *ndlp, uint8_t *dhval, uint32_t *dhvallen,
134         uint32_t hash_size, uint32_t dhgp_id);
135 static BIG_ERR_CODE
136 emlxs_BIGNUM_get_dhval(emlxs_port_t *port, emlxs_port_dhc_t *port_dhc,
137         NODELIST *ndlp, uint8_t *dhval, uint32_t *dhval_len,
138         uint32_t dhgp_id, uint8_t *priv_key, uint32_t privkey_len);
139 static uint32_t *
140 emlxs_hash_verification(emlxs_port_t *port, emlxs_port_dhc_t *port_dhc,
141         NODELIST *ndlp, uint32_t tran_id, uint8_t *dhval,
142         uint32_t dhval_len, uint32_t flag, uint8_t *bi_cval);

144 static uint32_t *
145 emlxs_hash_get_R2(emlxs_port_t *port, emlxs_port_dhc_t *port_dhc,
146         NODELIST *ndlp, uint32_t tran_id, uint8_t *dhval,
147         uint32_t dhval_len, uint32_t flag, uint8_t *bi_cval);

149 static uint32_t emlxs_issue_auth_reject(emlxs_port_t *port,
150         NODELIST *ndlp, int retry, uint32_t *arg, uint8_t ReasonCode,
151         uint8_t ReasonCodeExplanation);

153 static uint32_t emlxs_disc_neverdev(emlxs_port_t *port, void *arg1,
154         void *arg2, void *arg3, void *arg4, uint32_t evt);
155 static uint32_t emlxs_rcv_auth_msg_unmapped_node(emlxs_port_t *port,
156         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
157 static uint32_t emlxs_rcv_auth_msg_npr_node(emlxs_port_t *port,
158         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
159 static uint32_t emlxs_cmpl_auth_msg_npr_node(emlxs_port_t *port,
160         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
161 static uint32_t emlxs_rcv_auth_msg_auth_negotiate_issue(emlxs_port_t *port,
162         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
163 static uint32_t emlxs_cmpl_auth_msg_auth_negotiate_issue(emlxs_port_t *port,
164         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
165 static uint32_t emlxs_rcv_auth_msg_auth_negotiate_rcv(emlxs_port_t *port,
166         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
167 static uint32_t emlxs_cmpl_auth_msg_auth_negotiate_rcv(emlxs_port_t *port,
168         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
169 static uint32_t
170 emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next(emlxs_port_t *port,
171         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
172 static uint32_t
173 emlxs_cmpl_auth_msg_auth_negotiate_cmpl_wait4next(emlxs_port_t *port,
174         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
175 static uint32_t
176 emlxs_rcv_auth_msg_dhchap_challenge_issue(emlxs_port_t *port, void *arg1,
177         void *arg2, void *arg3, void *arg4, uint32_t evt);
178 static uint32_t
179 emlxs_cmpl_auth_msg_dhchap_challenge_issue(emlxs_port_t *port, void *arg1,
180         void *arg2, void *arg3, void *arg4, uint32_t evt);
181 static uint32_t emlxs_rcv_auth_msg_dhchap_reply_issue(emlxs_port_t *port,
182         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
183 static uint32_t emlxs_cmpl_auth_msg_dhchap_reply_issue(emlxs_port_t *port,
184         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
185 static uint32_t
186 emlxs_rcv_auth_msg_dhchap_challenge_cmpl_wait4next(emlxs_port_t *port,
187         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
188 static uint32_t
189 emlxs_cmpl_auth_msg_dhchap_challenge_cmpl_wait4next(emlxs_port_t *port,
190         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
191 static uint32_t
192 emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next(emlxs_port_t *port,
193         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
```



```
194 static uint32_t
195 emlxs_cmpl_auth_msg_dhchap_reply_cmpl_wait4next(emlxs_port_t *port,
196         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
197 static uint32_t emlxs_rcv_auth_msg_dhchap_success_issue(emlxs_port_t *port,
198         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
199 static uint32_t
200 emlxs_cmpl_auth_msg_dhchap_success_issue(emlxs_port_t *port, void *arg1,
201         void *arg2, void *arg3, void *arg4, uint32_t evt);
202 static uint32_t
203 emlxs_rcv_auth_msg_dhchap_success_issue_wait4next(emlxs_port_t *port,
204         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
205 static uint32_t
206 emlxs_cmpl_auth_msg_dhchap_success_issue_wait4next(emlxs_port_t *port,
207         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
208 static uint32_t
209 emlxs_rcv_auth_msg_dhchap_success_cmpl_wait4next(emlxs_port_t *port,
210         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
211 static uint32_t
212 emlxs_cmpl_auth_msg_dhchap_success_cmpl_wait4next(emlxs_port_t *port,
213         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);


216 static uint32_t emlxs_device_recov_unmapped_node(emlxs_port_t *port,
217         void *arg1, void *arg2, void *arg3, void *arg4, uint32_t evt);
218 static uint32_t emlxs_device_rm_npr_node(emlxs_port_t *port, void *arg1,
219         void *arg2, void *arg3, void *arg4, uint32_t evt);
220 static uint32_t emlxs_device_recov_npr_node(emlxs_port_t *port, void *arg1,
221         void *arg2, void *arg3, void *arg4, uint32_t evt);
222 static uint32_t emlxs_device_rem_auth(emlxs_port_t *port, void *arg1,
223         void *arg2, void *arg3, void *arg4, uint32_t evt);
224 static uint32_t emlxs_device_recov_auth(emlxs_port_t *port, void *arg1,
225         void *arg2, void *arg3, void *arg4, uint32_t evt);

227 static uint8_t emlxs_null_wwn[8] =
228         {0, 0, 0, 0, 0, 0, 0, 0};
229 static uint8_t emlxs_fabric_wwn[8] =
230         {0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff};

232 unsigned char dhgp1_pVal[] =
233 {0xEE, 0xAF, 0x0A, 0xB9, 0xAD, 0xB3, 0x8D, 0xD6, 0x9C, 0x33, 0xF8, 0x0A, 0xFA,
234 0x8F, 0xC5, 0xE8,
235 0x60, 0x72, 0x61, 0x87, 0x75, 0xFF, 0x3C, 0x0B, 0x9E, 0xA2, 0x31, 0x4C, 0x9C,
236 0x25, 0x65, 0x76,
237 0xD6, 0x74, 0xDF, 0x74, 0x96, 0xEA, 0x81, 0xD3, 0x38, 0x3B, 0x48, 0x13, 0xD6,
238 0x92, 0xC6, 0xE0,
239 0xE0, 0xD5, 0xD8, 0xE2, 0x50, 0xB9, 0x8B, 0xE4, 0x8E, 0x49, 0x5C, 0x1D, 0x60,
240 0x89, 0xDA, 0xD1,
241 0x5D, 0xC7, 0xD7, 0xB4, 0x61, 0x54, 0xD6, 0xB6, 0xCE, 0x8E, 0xF4, 0xAD, 0x69,
242 0xB1, 0x5D, 0x49,
243 0x82, 0x55, 0x9B, 0x29, 0x7B, 0xCF, 0x18, 0x85, 0xC5, 0x29, 0xF5, 0x66, 0x66,
244 0x0E, 0x57, 0xEC,
245 0x68, 0xED, 0xBC, 0x3C, 0x05, 0x72, 0x6C, 0xC0, 0x2F, 0xD4, 0xCB, 0xF4, 0x97,
246 0x6E, 0xAA, 0x9A,
247 0xFD, 0x51, 0x38, 0xFE, 0x83, 0x76, 0x43, 0x5B, 0x9F, 0xC6, 0x1D, 0x2F, 0xC0,
248 0xEB, 0x06, 0xE3,
249 };

251 unsigned char dhgp2_pVal[] =
252 {0xD7, 0x79, 0x46, 0x82, 0x6E, 0x81, 0x19, 0x14, 0xB3, 0x94, 0x01, 0xD5, 0x6A,
253 0x0A, 0x78, 0x43,
254 0xA8, 0xE7, 0x57, 0x5D, 0x73, 0x8C, 0x67, 0x2A, 0x09, 0x0A, 0xB1, 0x18, 0x7D,
255 0x69, 0x0D, 0xC4,
256 0x38, 0x72, 0xFC, 0x06, 0xA7, 0xB6, 0xA4, 0x3F, 0x3B, 0x95, 0xBE, 0xAE, 0xC7,
257 0xDF, 0x04, 0xB9,
258 0xD2, 0x42, 0xEB, 0xDC, 0x48, 0x11, 0x11, 0x28, 0x32, 0x16, 0xCE, 0x81, 0x6E,
259 0x00, 0x4B, 0x78,
```

```
0x6C, 0x5F, 0xCE, 0x85, 0x67, 0x80, 0xD4, 0x18, 0x37, 0xD9, 0x5A, 0xD7, 0x87,
0xA5, 0x0B, 0xBE,
0x90, 0xBD, 0x3A, 0x9C, 0x98, 0xAC, 0x0F, 0x5F, 0xC0, 0xDE, 0x74, 0x4B, 0x1C,
0xDE, 0x18, 0x91,
0x69, 0x08, 0x94, 0xBC, 0x1F, 0x65, 0xE0, 0x0D, 0xE1, 0x5B, 0x4B, 0x2A, 0xA6,
0xD8, 0x71, 0x00,
0xC9, 0xEC, 0xC2, 0x52, 0x7E, 0x45, 0xEB, 0x84, 0x9D, 0xEB, 0x14, 0xBB, 0x20,
0x49, 0xB1, 0x63,
0xEA, 0x04, 0x18, 0x7F, 0xD2, 0x7C, 0x1B, 0xD9, 0xC7, 0x95, 0x8C, 0xD4, 0x0C,
0xE7, 0x06, 0x7A,
0x9C, 0x02, 0x4F, 0x9B, 0x7C, 0x5A, 0x0B, 0x4F, 0x50, 0x03, 0x68, 0x61, 0x61,
0xF0, 0x60, 0x5B
};

unsigned char dhgp3_pVal[] =
{0x9D, 0xEF, 0x3C, 0xAF, 0xB9, 0x39, 0x27, 0x7A, 0xB1, 0xF1, 0x2A, 0x86, 0x17,
0xA4, 0x7B, 0xBB,
0xDB, 0xA5, 0x1D, 0xF4, 0x99, 0xAC, 0x4C, 0x80, 0xBE, 0xEE, 0xA9, 0x61, 0x4B,
0x19, 0xCC, 0x4D,
0x5F, 0x4F, 0x5F, 0x55, 0x6E, 0x27, 0xCB, 0xDE, 0x51, 0xC6, 0xA9, 0x4B, 0xE4,
0x60, 0x7A, 0x29,
0x15, 0x58, 0x90, 0x3B, 0xA0, 0xD0, 0xF8, 0x43, 0x80, 0xB6, 0x55, 0xBB, 0x9A,
0x22, 0xE8, 0xDC,
0xDF, 0x02, 0x8A, 0x7C, 0xEC, 0x67, 0xF0, 0xD0, 0x81, 0x34, 0xB1, 0xC8, 0xB9,
0x79, 0x89, 0x14,
0x9B, 0x60, 0x9E, 0x0B, 0xE3, 0xBA, 0xB6, 0x3D, 0x47, 0x54, 0x83, 0x81, 0xDB,
0xC5, 0xB1, 0xFC,
0x76, 0x4E, 0x3F, 0x4B, 0x53, 0xDD, 0x9D, 0xA1, 0x15, 0x8B, 0xFD, 0x3E, 0x2B,
0x9C, 0x8C, 0xF5,
0x6E, 0xDF, 0x01, 0x95, 0x39, 0x34, 0x96, 0x27, 0xDB, 0x2F, 0xD5, 0x3D, 0x24,
0xB7, 0xC4, 0x86,
0x65, 0x77, 0x2E, 0x43, 0x7D, 0x6C, 0x7F, 0x8C, 0xE4, 0x42, 0x73, 0x4A, 0xF7,
0xCC, 0xB7, 0xAE,
0x83, 0x7C, 0x26, 0x4A, 0xE3, 0xA9, 0xBE, 0xB8, 0x7F, 0x8A, 0x2F, 0xE9, 0xB8,
0xB5, 0x29, 0x2E,
0x5A, 0x02, 0x1F, 0xFF, 0x5E, 0x91, 0x47, 0x9E, 0x8C, 0xE7, 0xA2, 0x8C, 0x24,
0x42, 0xC6, 0xF3,
0x15, 0x18, 0x0F, 0x93, 0x49, 0x9A, 0x23, 0x4D, 0xCF, 0x76, 0xE3, 0xFE, 0xD1,
0x35, 0xF9, 0xBB
};

unsigned char dhgp4_pVal[] =
{0xAC, 0x6B, 0xDB, 0x41, 0x32, 0x4A, 0x9A, 0x9B, 0xF1, 0x66, 0xDE, 0x5E, 0x13,
0x89, 0x58, 0x2F,
0xAF, 0x72, 0xB6, 0x65, 0x19, 0x87, 0xEE, 0x07, 0xFC, 0x31, 0x92, 0x94, 0x3D,
0xB5, 0x60, 0x50,
0xA3, 0x73, 0x29, 0xCB, 0xB4, 0xA0, 0x99, 0xED, 0x81, 0x93, 0xE0, 0x75, 0x77,
0x67, 0xA1, 0x3D,
0xD5, 0x23, 0x12, 0xAB, 0x4B, 0x03, 0x31, 0x0D, 0xCD, 0x7F, 0x48, 0xA9, 0xDA,
0x04, 0xFD, 0x50,
0xE8, 0x08, 0x39, 0x69, 0xED, 0xB7, 0x67, 0xB0, 0xCF, 0x60, 0x95, 0x17, 0x9A,
0x16, 0x3A, 0xB3,
0x66, 0x1A, 0x05, 0xFB, 0xD5, 0xFA, 0xAA, 0xE8, 0x29, 0x18, 0xA9, 0x96, 0x2F,
0x0B, 0x93, 0xB8,
0x55, 0xF9, 0x79, 0x93, 0xEC, 0x97, 0x5E, 0xEA, 0xA8, 0x0D, 0x74, 0x0A, 0xDB,
0xF4, 0xFF, 0x74,
0x73, 0x59, 0xD0, 0x41, 0xD5, 0xC3, 0x3E, 0xA7, 0x1D, 0x28, 0x1E, 0x44, 0x6B,
0x14, 0x77, 0x3B,
0xCA, 0x97, 0xB4, 0x3A, 0x23, 0xFB, 0x80, 0x16, 0x76, 0xBD, 0x20, 0x7A, 0x43,
0x6C, 0x64, 0x81,
0xF1, 0xD2, 0xB9, 0x07, 0x87, 0x17, 0x46, 0x1A, 0x5B, 0x9D, 0x32, 0xE6, 0x88,
0xF8, 0x77, 0x48,
0x54, 0x45, 0x23, 0xB5, 0x24, 0xB0, 0xD5, 0x7D, 0x5E, 0xA7, 0x7A, 0x27, 0x75,
0xD2, 0xEC, 0xFA,
0x03, 0x2C, 0xFB, 0xDB, 0xF5, 0x2F, 0xB3, 0x78, 0x61, 0x60, 0x27, 0x90, 0x04,
0xE5, 0x7A, 0xE6,
```



```
0xAF, 0x87, 0x4E, 0x73, 0x03, 0xCE, 0x53, 0x29, 0x9C, 0xCC, 0x04, 0x1C, 0x7B,
0xC3, 0x08, 0xD8,
0x2A, 0x56, 0x98, 0xF3, 0xA8, 0xD0, 0xC3, 0x82, 0x71, 0xAE, 0x35, 0xF8, 0xE9,
0xDB, 0xFB, 0xB6,
0x94, 0xB5, 0xC8, 0x03, 0xD8, 0x9F, 0x7A, 0xE4, 0x35, 0xDE, 0x23, 0x6D, 0x52,
0x5F, 0x54, 0x75,
0x9B, 0x65, 0xE3, 0x72, 0xFC, 0xD6, 0x8E, 0xF2, 0x0F, 0xA7, 0x11, 0x1F, 0x9E,
0x4A, 0xFF, 0x73
};

/*
 * myrand is used for test only, eventually it should be replaced by the random
 * number. AND it is basically the private key.
 */
/* #define      MYRAND */
#ifdef MYRAND
unsigned char myrand[] =
{0x11, 0x11, 0x22, 0x22,
        0x33, 0x33, 0x44, 0x44,
        0x55, 0x55, 0x66, 0x66,
        0x77, 0x77, 0x88, 0x88,
0x99, 0x99, 0x00, 0x00};
#endif  /* MYRAND */




/* Node Events */
#define NODE_EVENT_DEVICE_RM    0x0 /* Auth response timeout & fail */
#define NODE_EVENT_DEVICE_RECOVERY 0x1 /* Auth response timeout & recovery */
#define NODE_EVENT_RCV_AUTH_MSG  0x2 /* Unsolicited Auth received */
#define NODE_EVENT_CMPL_AUTH_MSG 0x3
#define NODE_EVENT_MAX_EVENT     0x4

emlxs_table_t emlxs_event_table[] =
{
        {NODE_EVENT_DEVICE_RM, "DEVICE_REMOVE"},
        {NODE_EVENT_DEVICE_RECOVERY, "DEVICE_RECOVERY"},
        {NODE_EVENT_RCV_AUTH_MSG, "AUTH_MSG_RCVD"},
        {NODE_EVENT_CMPL_AUTH_MSG, "AUTH_MSG_CMPL"},

};      /* emlxs_event_table() */

emlxs_table_t emlxs_pstate_table[] =
{
        {ELX_FABRIC_STATE_UNKNOWN, "FABRIC_STATE_UNKNOWN"},
        {ELX_FABRIC_AUTH_DISABLED, "FABRIC_AUTH_DISABLED"},
        {ELX_FABRIC_AUTH_FAILED, "FABRIC_AUTH_FAILED"},
        {ELX_FABRIC_AUTH_SUCCESS, "FABRIC_AUTH_SUCCESS"},
        {ELX_FABRIC_IN_AUTH, "FABRIC_IN_AUTH"},
        {ELX_FABRIC_IN_REAUTH, "FABRIC_IN_REAUTH"},

};      /* emlxs_pstate_table() */

emlxs_table_t emlxs_nstate_table[] =
{
{NODE_STATE_UNKNOWN, "STATE_UNKNOWN"},
{NODE_STATE_AUTH_DISABLED, "AUTH_DISABLED"},
{NODE_STATE_AUTH_FAILED, "AUTH_FAILED"},
{NODE_STATE_AUTH_SUCCESS, "AUTH_SUCCESS"},
{NODE_STATE_AUTH_NEGOTIATE_ISSUE, "NEGOTIATE_ISSUE"},
{NODE_STATE_AUTH_NEGOTIATE_RCV, "NEGOTIATE_RCV"},
{NODE_STATE_AUTH_NEGOTIATE_CMPL_WAIT4NEXT, "NEGOTIATE_CMPL"},
{NODE_STATE_DHCHAP_CHALLENGE_ISSUE, "DHCHAP_CHALLENGE_ISSUE"},
{NODE_STATE_DHCHAP_REPLY_ISSUE, "DHCHAP_REPLY_ISSUE"},
{NODE_STATE_DHCHAP_CHALLENGE_CMPL_WAIT4NEXT, "DHCHAP_CHALLENGE_CMPL"},
```

```
392 {NODE_STATE_DHCHAP_REPLY_CMPL_WAIT4NEXT, "DHCHAP_REPLY_CMPL"},
393 {NODE_STATE_DHCHAP_SUCCESS_ISSUE, "DHCHAP_SUCCESS_ISSUE"},
394 {NODE_STATE_DHCHAP_SUCCESS_ISSUE_WAIT4NEXT, "DHCHAP_SUCCESS_ISSUE_WAIT"},
395 {NODE_STATE_DHCHAP_SUCCESS_CMPL_WAIT4NEXT, "DHCHAP_SUCCESS_CMPL"},
396 };      /* emlxs_nstate_table() */

398 extern char *
399 emlxs_dhc_event_xlate(uint32_t state)
400 {
401         static char buffer[32];
402         uint32_t i;
403         uint32_t count;

405         count = sizeof (emlxs_event_table) / sizeof (emlxs_table_t);
406         for (i = 0; i < count; i++) {
407                 if (state == emlxs_event_table[i].code) {
408                         return (emlxs_event_table[i].string);
409                 }
410         }

412         (void) sprintf(buffer, "event=0x%x", state);
413         return (buffer);

415 } /* emlxs_dhc_event_xlate() */


418 extern void
419 emlxs_dhc_state(emlxs_port_t *port, emlxs_node_t *ndlp, uint32_t state,
420         uint32_t reason, uint32_t explaination)
421 {
422         emlxs_hba_t *hba = HBA;
423         emlxs_port_dhc_t *port_dhc = &port->port_dhc;
424         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
425         uint32_t pstate;

427         if ((state != NODE_STATE_NOCHANGE) && (node_dhc->state != state)) {
428                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_state_msg,
429                     "Node:0x%x %s --> %s", ndlp->nlp_DID,
430                     emlxs_dhc_nstate_xlate(node_dhc->state),
431                     emlxs_dhc_nstate_xlate(state));

433                 node_dhc->prev_state = node_dhc->state;
434                 node_dhc->state = (uint16_t)state;

436                 /* Perform common functions based on state */
437                 switch (state) {
438                 case NODE_STATE_UNKNOWN:
439                 case NODE_STATE_AUTH_DISABLED:
440                         node_dhc->nlp_authrsp_tmo = 0;
441                         node_dhc->nlp_authrsp_tmocnt = 0;
442                         emlxs_dhc_set_reauth_time(port, ndlp, DISABLE);
443                         break;

445                 case NODE_STATE_AUTH_SUCCESS:
446                         /* Record auth time */
447                         if (ndlp->nlp_DID == FABRIC_DID) {
448                                 port_dhc->auth_time = DRV_TIME;
449                         } else if (node_dhc->parent_auth_cfg) {
450                                 node_dhc->parent_auth_cfg->auth_time = DRV_TIME;
451                         }
452                         hba->rdn_flag = 0;
453                         node_dhc->nlp_authrsp_tmo = 0;

455                         if (node_dhc->flag & NLP_SET_REAUTH_TIME) {
456                                 emlxs_dhc_set_reauth_time(port, ndlp, ENABLE);
457                         }
```



```
458                         break;

460                 default:
461                         break;
462                 }

464                 /* Check for switch port */
465                 if (ndlp->nlp_DID == FABRIC_DID) {
466                         switch (state) {
467                         case NODE_STATE_UNKNOWN:
468                                 pstate = ELX_FABRIC_STATE_UNKNOWN;
469                                 break;

471                         case NODE_STATE_AUTH_DISABLED:
472                                 pstate = ELX_FABRIC_AUTH_DISABLED;
473                                 break;

475                         case NODE_STATE_AUTH_FAILED:
476                                 pstate = ELX_FABRIC_AUTH_FAILED;
477                                 break;

479                         case NODE_STATE_AUTH_SUCCESS:
480                                 pstate = ELX_FABRIC_AUTH_SUCCESS;
481                                 break;

483                                 /* Auth active */
484                         default:
485                                 if (port_dhc->state ==
486                                     ELX_FABRIC_AUTH_SUCCESS) {
487                                         pstate = ELX_FABRIC_IN_REAUTH;
488                                 } else if (port_dhc->state !=
489                                     ELX_FABRIC_IN_REAUTH) {
490                                         pstate = ELX_FABRIC_IN_AUTH;
491                                 }
492                                 break;
493                         }

495                         if (port_dhc->state != pstate) {
496                                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_state_msg,
497                                     "Port: %s --> %s",
498                                     emlxs_dhc_pstate_xlate(port_dhc->state),
499                                     emlxs_dhc_pstate_xlate(pstate));

501                                 port_dhc->state = pstate;
502                         }
503                 }
504         }
505         /* Update auth status */
506         mutex_enter(&hba->auth_lock);
507         emlxs_dhc_status(port, ndlp, reason, explaination);
508         mutex_exit(&hba->auth_lock);

510         return;

512 } /* emlxs_dhc_state() */


515 /* auth_lock must be held when calling this */
516 extern void
517 emlxs_dhc_status(emlxs_port_t *port, emlxs_node_t *ndlp, uint32_t reason,
518         uint32_t explaination)
519 {
520         emlxs_port_dhc_t *port_dhc;
521         emlxs_node_dhc_t *node_dhc;
522         dfc_auth_status_t *auth_status;
523         uint32_t drv_time;
```

```
525         if (!ndlp || !ndlp->nlp_active || ndlp->node_dhc.state ==
526             NODE_STATE_UNKNOWN) {
527                 return;
528         }
529         port_dhc = &port->port_dhc;
530         node_dhc = &ndlp->node_dhc;
531
532         /* Get auth status object */
533         if (ndlp->nlp_DID == FABRIC_DID) {
534                 auth_status = &port_dhc->auth_status;
535         } else if (node_dhc->parent_auth_cfg) {
536                 auth_status = &node_dhc->parent_auth_cfg->auth_status;
537         } else {
538                 /* No auth status to be updated */
539                 return;
540         }
541
542         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_status_msg,
543             "Node:0x%x state=%s rsn=0x%x exp=0x%x (%x,%x)",
544             ndlp->nlp_DID, emlxs_dhc_nstate_xlate(node_dhc->state), reason,
545             explaination, auth_status->auth_state,
546             auth_status->auth_failReason);
547
548         /* Set state and auth_failReason */
549         switch (node_dhc->state) {
550         case NODE_STATE_UNKNOWN:        /* Connection */
551                 if (auth_status->auth_state != DFC_AUTH_STATE_FAILED) {
552                         auth_status->auth_state = DFC_AUTH_STATE_OFF;
553                         auth_status->auth_failReason = 0;
554                 }
555                 break;
556
557         case NODE_STATE_AUTH_DISABLED:
558                 auth_status->auth_state = DFC_AUTH_STATE_OFF;
559                 auth_status->auth_failReason = 0;
560                 break;
561
562         case NODE_STATE_AUTH_FAILED:
563                 /* Check failure reason and update if neccessary */
564                 switch (reason) {
565                 case AUTHRJT_FAILURE:   /* 0x01 */
566                 case AUTHRJT_LOGIC_ERR: /* 0x02 */
567                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
568                         auth_status->auth_failReason = DFC_AUTH_FAIL_REJECTED;
569                         break;
570
571                 case LSRJT_AUTH_REQUIRED:       /* 0x03 */
572                         switch (explaination) {
573                         case LSEXP_AUTH_REQUIRED:
574                                 auth_status->auth_state = DFC_AUTH_STATE_FAILED;
575                                 auth_status->auth_failReason =
576                                     DFC_AUTH_FAIL_LS_RJT;
577                                 break;
578                         default:
579                                 auth_status->auth_state = DFC_AUTH_STATE_FAILED;
580                                 auth_status->auth_failReason =
581                                     DFC_AUTH_FAIL_REJECTED;
582                         }
583                         break;
584
585                 case LSRJT_AUTH_LOGICAL_BSY:    /* 0x05 */
586                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
587                         auth_status->auth_failReason = DFC_AUTH_FAIL_BSY_LS_RJT;
588                         break;
```



```
590                 case LSRJT_AUTH_ELS_NOT_SUPPORTED:      /* 0x0B */
591                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
592                         auth_status->auth_failReason = DFC_AUTH_FAIL_LS_RJT;
593                         break;
594
595                 case LSRJT_AUTH_NOT_LOGGED_IN:  /* 0x09 */
596                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
597                         auth_status->auth_failReason = DFC_AUTH_FAIL_BSY_LS_RJT;
598                         break;
599                 }
600
601                 /* Make sure the state is set to failed at this point */
602                 if (auth_status->auth_state != DFC_AUTH_STATE_FAILED) {
603                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
604                         auth_status->auth_failReason = DFC_AUTH_FAIL_GENERIC;
605                 }
606                 break;
607
608         case NODE_STATE_AUTH_SUCCESS:
609                 auth_status->auth_state = DFC_AUTH_STATE_ON;
610                 auth_status->auth_failReason = 0;
611                 break;
612
613                 /* Authentication currently active */
614         default:
615                 /* Set defaults */
616                 auth_status->auth_state = DFC_AUTH_STATE_INP;
617                 auth_status->auth_failReason = 0;
618
619                 /* Check codes for exceptions */
620                 switch (reason) {
621                 case AUTHRJT_FAILURE:   /* 0x01 */
622                         switch (explaination) {
623                         case AUTHEXP_AUTH_FAILED:       /* 0x05 */
624                         case AUTHEXP_BAD_PAYLOAD:       /* 0x06 */
625                         case AUTHEXP_BAD_PROTOCOL:      /* 0x07 */
626                                 auth_status->auth_state = DFC_AUTH_STATE_FAILED;
627                                 auth_status->auth_failReason =
628                                     DFC_AUTH_FAIL_REJECTED;
629                                 break;
630                         }
631                         break;
632
633                 case AUTHRJT_LOGIC_ERR: /* 0x02 */
634                         switch (explaination) {
635                         case AUTHEXP_MECH_UNUSABLE:     /* 0x01 */
636                         case AUTHEXP_DHGROUP_UNUSABLE:  /* 0x02 */
637                         case AUTHEXP_HASHFUNC_UNUSABLE: /* 0x03 */
638                         case AUTHEXP_CONCAT_UNSUPP:     /* 0x09 */
639                         case AUTHEXP_BAD_PROTOVERS:     /* 0x0A */
640                                 auth_status->auth_state = DFC_AUTH_STATE_FAILED;
641                                 auth_status->auth_failReason =
642                                     DFC_AUTH_FAIL_REJECTED;
643                                 break;
644                         }
645                         break;
646
647                 case LSRJT_AUTH_REQUIRED:       /* 0x03 */
648                         switch (explaination) {
649                         case LSEXP_AUTH_REQUIRED:
650                                 auth_status->auth_state = DFC_AUTH_STATE_FAILED;
651                                 auth_status->auth_failReason =
652                                     DFC_AUTH_FAIL_LS_RJT;
653                                 break;
654                         }
655                         break;
```

```
657                 case LSRJT_AUTH_LOGICAL_BSY:    /* 0x05 */
658                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
659                         auth_status->auth_failReason = DFC_AUTH_FAIL_BSY_LS_RJT;
660                         break;
661
662                 case LSRJT_AUTH_ELS_NOT_SUPPORTED:      /* 0x0B */
663                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
664                         auth_status->auth_failReason = DFC_AUTH_FAIL_LS_RJT;
665                         break;
666
667                 case LSRJT_AUTH_NOT_LOGGED_IN:  /* 0x09 */
668                         auth_status->auth_state = DFC_AUTH_STATE_FAILED;
669                         auth_status->auth_failReason = DFC_AUTH_FAIL_BSY_LS_RJT;
670                         break;
671                 }
672                 break;
673         }
674
675         if (auth_status->auth_state != DFC_AUTH_STATE_ON) {
676                 auth_status->time_until_next_auth = 0;
677                 auth_status->localAuth = 0;
678                 auth_status->remoteAuth = 0;
679                 auth_status->group_priority = 0;
680                 auth_status->hash_priority = 0;
681                 auth_status->type_priority = 0;
682         } else {
683                 switch (node_dhc->nlp_reauth_status) {
684                 case NLP_HOST_REAUTH_ENABLED:
685                 case NLP_HOST_REAUTH_IN_PROGRESS:
686                         drv_time = DRV_TIME;
687
688                         if (node_dhc->nlp_reauth_tmo > drv_time) {
689                                 auth_status->time_until_next_auth =
690                                     node_dhc->nlp_reauth_tmo - drv_time;
691                         } else {
692                                 auth_status->time_until_next_auth = 0;
693                         }
694                         break;
695
696                 case NLP_HOST_REAUTH_DISABLED:
697                 default:
698                         auth_status->time_until_next_auth = 0;
699                         break;
700                 }
701
702                 if (node_dhc->flag & NLP_REMOTE_AUTH) {
703                         auth_status->localAuth = 0;
704                         auth_status->remoteAuth = 1;
705                 } else {
706                         auth_status->localAuth = 1;
707                         auth_status->remoteAuth = 0;
708                 }
709
710                 auth_status->type_priority = DFC_AUTH_TYPE_DHCHAP;
711
712                 switch (node_dhc->nlp_auth_dhgpid) {
713                 case GROUP_NULL:
714                         auth_status->group_priority = ELX_GROUP_NULL;
715                         break;
716
717                 case GROUP_1024:
718                         auth_status->group_priority = ELX_GROUP_1024;
719                         break;
720
721                 case GROUP_1280:
```



```
722                         auth_status->group_priority = ELX_GROUP_1280;
723                         break;
724
725                 case GROUP_1536:
726                         auth_status->group_priority = ELX_GROUP_1536;
727                         break;
728
729                 case GROUP_2048:
730                         auth_status->group_priority = ELX_GROUP_2048;
731                         break;
732                 }
733
734                 switch (node_dhc->nlp_auth_hashid) {
735                 case 0:
736                         auth_status->hash_priority = 0;
737                         break;
738
739                 case AUTH_SHA1:
740                         auth_status->hash_priority = ELX_SHA1;
741                         break;
742
743                 case AUTH_MD5:
744                         auth_status->hash_priority = ELX_MD5;
745                         break;
746                 }
747         }
748
749         return;
750
751 } /* emlxs_dhc_status()  */
752
753 static char *
754 emlxs_dhc_pstate_xlate(uint32_t state)
755 {
756         static char buffer[32];
757         uint32_t i;
758         uint32_t count;
759
760         count = sizeof (emlxs_pstate_table) / sizeof (emlxs_table_t);
761         for (i = 0; i < count; i++) {
762                 if (state == emlxs_pstate_table[i].code) {
763                         return (emlxs_pstate_table[i].string);
764                 }
765         }
766
767         (void) sprintf(buffer, "state=0x%x", state);
768         return (buffer);
769
770 } /* emlxs_dhc_pstate_xlate() */
771
772
773 static char *
774 emlxs_dhc_nstate_xlate(uint32_t state)
775 {
776         static char buffer[32];
777         uint32_t i;
778         uint32_t count;
779
780         count = sizeof (emlxs_nstate_table) / sizeof (emlxs_table_t);
781         for (i = 0; i < count; i++) {
782                 if (state == emlxs_nstate_table[i].code) {
783                         return (emlxs_nstate_table[i].string);
784                 }
785         }
786
787         (void) sprintf(buffer, "state=0x%x", state);
```

```
788         return (buffer);

790 } /* emlxs_dhc_nstate_xlate() */


793 static uint32_t
794 emlxs_check_dhgp(
795         emlxs_port_t *port,
796         NODELIST *ndlp,
797         uint32_t *dh_id,
798         uint16_t cnt,
799         uint32_t *dhgp_id)
800 {
801         uint32_t i, j, rc = 1;
802         uint32_t wnt;
803         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

805         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
806             "dhgp: 0x%x, id[0..4]=0x%x 0x%x 0x%x 0x%x 0x%x pri[1]=0x%x",
807             cnt, dh_id[0], dh_id[1], dh_id[2], dh_id[3], dh_id[4],
808             node_dhc->auth_cfg.dh_group_priority[1]);

810         /*
811          * Here are the rules, as the responder We always try to select ours
812          * highest setup
813          */

815         /* Check to see if there is any repeated dhgp in initiator's list */
816         /* If available, it is a invalid payload */
817         if (cnt >= 2) {
818                 for (i = 0; i <= cnt - 2; i++) {
819                         for (j = i + 1; j <= cnt - 1; j++) {
820                                 if (dh_id[i] == dh_id[j]) {
821                                         rc = 2;
822                                         EMLXS_MSGF(EMLXS_CONTEXT,
823                                             &emlxs_fcsp_detail_msg,
824                                             ":Rpt dhid[%x]=%x dhid[%x]=%x",
825                                             i, dh_id[i], j, dh_id[j]);
826                                         break;
827                                 }
828                         }

830                         if (rc == 2) {
831                                 break;
832                         }
833                 }

835                 if ((i == cnt - 1) && (j == cnt)) {
836                         rc = 1;
837                 }
838                 if (rc == 2) {
839                         /* duplicate invalid payload */
840                         return (rc);
841                 }
842         }
843         /* Check how many dhgps the responder specified */
844         wnt = 0;
845         while (node_dhc->auth_cfg.dh_group_priority[wnt] != 0xF) {
846                 wnt++;
847         }

849         /* Determine the most suitable dhgp the responder should use */
850         for (i = 0; i < wnt; i++) {
851                 for (j = 0; j < cnt; j++) {
852                         if (node_dhc->auth_cfg.dh_group_priority[i] ==
853                             dh_id[j]) {
```



```
854                                 rc = 0;
855                                 *dhgp_id =
856                                     node_dhc->auth_cfg.dh_group_priority[i];
857                                 break;
858                         }
859                 }

861                 if (rc == 0) {
862                         break;
863                 }
864         }

866         if (i == wnt) {
867                 /* no match */
868                 rc = 1;
869                 return (1);
870         }

872         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
873             "emlxs_check_dhgp: dhgp_id=0x%x", *dhgp_id);

875         return (rc);
876 } /* emlxs_check_dhgp */


879 static void
880 emlxs_get_random_bytes(
881         NODELIST *ndlp,
882         uint8_t *rdn,
883         uint32_t len)
884 {
885         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
886         hrtime_t now;
887         uint8_t sha1_digest[20];
888         SHA1_CTX sha1ctx;

890         now = gethrtime();

892         bzero(&sha1ctx, sizeof (SHA1_CTX));
893         SHA1Init(&sha1ctx);
894         SHA1Update(&sha1ctx, (void *) &node_dhc->auth_cfg.local_entity,
895             sizeof (NAME_TYPE));
896         SHA1Update(&sha1ctx, (void *) &now, sizeof (hrtime_t));
897         SHA1Final((void *) sha1_digest, &sha1ctx);
898         bcopy((void *) &sha1_digest[0], (void *) &rdn[0], len);

900         return;

902 } /* emlxs_get_random_bytes */


905 /* ************************** STATE MACHINE ************************ */

907 static void *emlxs_dhchap_action[] =
908 {
909         /* Action routine               Event */

911 /* NODE_STATE_UNKNOWN  0x00 */
912         (void *) emlxs_disc_neverdev,   /* DEVICE_RM */
913         (void *) emlxs_disc_neverdev,   /* DEVICE_RECOVERY */
914         (void *) emlxs_disc_neverdev,   /* RCV_AUTH_MSG */
915         (void *) emlxs_disc_neverdev,   /* CMPL_AUTH_MSG */

917 /* NODE_STATE_AUTH_DISABLED  0x01 */
918         (void *) emlxs_disc_neverdev,   /* DEVICE_RM */
919         (void *) emlxs_disc_neverdev,   /* DEVICE_RECOVERY */
```

```
920         (void *) emlxs_disc_neverdev,   /* RCV_AUTH_MSG */
921         (void *) emlxs_disc_neverdev,   /* CMPL_AUTH_MSG */

923 /* NODE_STATE_AUTH_FAILED  0x02 */
924         (void *) emlxs_device_rm_npr_node,      /* DEVICE_RM */
925         (void *) emlxs_device_recov_npr_node,   /* DEVICE_RECOVERY */
926         (void *) emlxs_rcv_auth_msg_npr_node,   /* RCV_AUTH_MSG */
927         (void *) emlxs_cmpl_auth_msg_npr_node,  /* CMPL_AUTH_MSG */

929 /* NODE_STATE_AUTH_SUCCESS  0x03 */
930         (void *) emlxs_disc_neverdev,                   /* DEVICE_RM */
931         (void *) emlxs_device_recov_unmapped_node,      /* DEVICE_RECOVERY */
932         (void *) emlxs_rcv_auth_msg_unmapped_node,      /* RCV_AUTH_MSG */
933         (void *) emlxs_disc_neverdev,                   /* CMPL_AUTH_MSG */

935 /* NODE_STATE_AUTH_NEGOTIATE_ISSUE  0x04 */
936         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
937         (void *) emlxs_device_recov_auth, /* DEVICE_RECOVERY */
938         (void *) emlxs_rcv_auth_msg_auth_negotiate_issue, /* RCV_AUTH_MSG  */
939         (void *) emlxs_cmpl_auth_msg_auth_negotiate_issue, /* CMPL_AUTH_MSG */

941 /* NODE_STATE_AUTH_NEGOTIATE_RCV  0x05 */
942         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
943         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
944         (void *) emlxs_rcv_auth_msg_auth_negotiate_rcv, /* RCV_AUTH_MSG */
945         (void *) emlxs_cmpl_auth_msg_auth_negotiate_rcv, /* CMPL_AUTH_MSG */

947 /* NODE_STATE_AUTH_NEGOTIATE_CMPL_WAIT4NEXT  0x06 */
948         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
949         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
950         (void *) emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next,
951                                         /* RCV_AUTH_MSG */
952         (void *) emlxs_cmpl_auth_msg_auth_negotiate_cmpl_wait4next,
953                                         /* CMPL_AUTH_MSG */

955 /* NODE_STATE_DHCHAP_CHALLENGE_ISSUE  0x07 */
956         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
957         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
958         (void *) emlxs_rcv_auth_msg_dhchap_challenge_issue, /* RCV_AUTH_MSG */
959         (void *) emlxs_cmpl_auth_msg_dhchap_challenge_issue, /* CMPL_AUTH_MSG */

961 /* NODE_STATE_DHCHAP_REPLY_ISSUE  0x08 */
962         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
963         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
964         (void *) emlxs_rcv_auth_msg_dhchap_reply_issue, /* RCV_AUTH_MSG */
965         (void *) emlxs_cmpl_auth_msg_dhchap_reply_issue, /* CMPL_AUTH_MSG */

967 /* NODE_STATE_DHCHAP_CHALLENGE_CMPL_WAIT4NEXT  0x09 */
968         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
969         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
970         (void *) emlxs_rcv_auth_msg_dhchap_challenge_cmpl_wait4next,
971                                         /* RCV_AUTH_MSG   */
972         (void *) emlxs_cmpl_auth_msg_dhchap_challenge_cmpl_wait4next,
973                                         /* CMPL_AUTH_MSG */

975 /* NODE_STATE_DHCHAP_REPLY_CMPL_WAIT4NEXT  0x0A */
976         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
977         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
978         (void *) emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next,
979                                         /* RCV_AUTH_MSG */
980         (void *) emlxs_cmpl_auth_msg_dhchap_reply_cmpl_wait4next,
981                                         /* CMPL_AUTH_MSG */

983 /* NODE_STATE_DHCHAP_SUCCESS_ISSUE  0x0B */
984         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
985         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
```



```
986         (void *) emlxs_rcv_auth_msg_dhchap_success_issue,
987                                         /* RCV_AUTH_MSG */
988         (void *) emlxs_cmpl_auth_msg_dhchap_success_issue,
989                                         /* CMPL_AUTH_MSG */

991 /* NODE_STATE_DHCHAP_SUCCESS_ISSUE_WAIT4NEXT  0x0C */
992         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
993         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
994         (void *) emlxs_rcv_auth_msg_dhchap_success_issue_wait4next,
995                                         /* RCV_AUTH_MSG */
996         (void *) emlxs_cmpl_auth_msg_dhchap_success_issue_wait4next,
997                                         /* CMPL_AUTH_MSG */

999 /* NODE_STATE_DHCHAP_SUCCESS_CMPL_WAIT4NEXT  0x0D */
1000         (void *) emlxs_device_rem_auth, /* DEVICE_RM */
1001         (void *) emlxs_device_recov_auth,       /* DEVICE_RECOVERY */
1002         (void *) emlxs_rcv_auth_msg_dhchap_success_cmpl_wait4next,
1003                                         /* RCV_AUTH_MSG */
1004         (void *) emlxs_cmpl_auth_msg_dhchap_success_cmpl_wait4next,
1005                                         /* CMPL_AUTH_MSG */

1007 }; /* emlxs_dhchap_action[] */


1010 extern int
1011 emlxs_dhchap_state_machine(emlxs_port_t *port, CHANNEL *cp,
1012                 IOCBQ *iocbq, MATCHMAP *mp,
1013                 NODELIST *ndlp, int evt)
1014 {
1015         emlxs_hba_t *hba = HBA;
1016         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
1017         uint32_t rc;
1018         uint32_t(*func) (emlxs_port_t *, CHANNEL *, IOCBQ *, MATCHMAP *,
1019             NODELIST *, uint32_t);

1021         mutex_enter(&hba->dhc_lock);

1023         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_event_msg,
1024             "%s: did=0x%x",
1025             emlxs_dhc_event_xlate(evt), ndlp->nlp_DID);

1027         node_dhc->disc_refcnt++;

1029         func = (uint32_t(*) (emlxs_port_t *, CHANNEL *, IOCBQ *, MATCHMAP *,
1030             NODELIST *, uint32_t))
1031             emlxs_dhchap_action[(node_dhc->state * NODE_EVENT_MAX_EVENT) + evt];

1033         rc = (func) (port, cp, iocbq, mp, ndlp, evt);

1035         node_dhc->disc_refcnt--;

1037         mutex_exit(&hba->dhc_lock);

1039         return (rc);

1041 } /* emlxs_dhchap_state_machine() */

1043 /* ARGSUSED */
1044 static uint32_t
1045 emlxs_disc_neverdev(
1046 emlxs_port_t *port,
1047 /* CHANNEL * rp, */ void *arg1,
1048 /* IOCBQ * iocbq, */ void *arg2,
1049 /* MATCHMAP * mp, */ void *arg3,
1050 /* NODELIST * ndlp */ void *arg4,
1051 uint32_t evt)
```

```
{
        NODELIST *ndlp = (NODELIST *) arg4;
        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
            "emlxs_disc_neverdev: did=0x%x.",
            ndlp->nlp_DID);

        emlxs_dhc_state(port, ndlp, NODE_STATE_UNKNOWN, 0, 0);

        return (node_dhc->state);

} /* emlxs_disc_neverdev() */


/*
 * ! emlxs_cmpl_dhchap_challenge_issue
 *
 * \pre \post \param   cmdiocb \param   rspiocb \return  void
 *
 * \b Description: iocb_cmpl callback function. when the ELS DHCHAP_Challenge
 * msg sent back got the ACC/RJT from initiator.
 *
 */
static void
emlxs_cmpl_dhchap_challenge_issue(fc_packet_t *pkt)
{
        emlxs_port_t *port = pkt->pkt_ulp_private;
        emlxs_buf_t *sbp;
        NODELIST *ndlp;
        uint32_t did;

        did = pkt->pkt_cmd_fhdr.d_id;
        sbp = (emlxs_buf_t *)pkt->pkt_fca_private;
        ndlp = sbp->node;

        if (!ndlp) {
                ndlp = emlxs_node_find_did(port, did);
        }
        if (pkt->pkt_state != FC_PKT_SUCCESS) {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
                    "emlxs_cmpl_dhchap_challenge_issue: did=0x%x state=%x",
                    did, pkt->pkt_state);
        } else {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
                    "emlxs_cmpl_dhchap_challenge_issue: did=0x%x. Succcess.",
                    did);
        }

        if (ndlp) {
                if (pkt->pkt_state == FC_PKT_SUCCESS) {
                        (void) emlxs_dhchap_state_machine(port, NULL, NULL,
                            NULL, ndlp, NODE_EVENT_CMPL_AUTH_MSG);
                }
        }
        emlxs_pkt_free(pkt);

        return;

} /* emlxs_cmpl_dhchap_challenge_issue */




/*
 * ! emlxs_cmpl_dhchap_success_issue
```



```
 *
 * \pre \post \param   phba \param   cmdiocb \param   rspiocb \return  void
 *
 * \b Description: iocb_cmpl callback function.
 *
 */
static void
emlxs_cmpl_dhchap_success_issue(fc_packet_t *pkt)
{
        emlxs_port_t *port = pkt->pkt_ulp_private;
        NODELIST *ndlp;
        uint32_t did;
        emlxs_buf_t *sbp;

        did = pkt->pkt_cmd_fhdr.d_id;
        sbp = (emlxs_buf_t *)pkt->pkt_fca_private;
        ndlp = sbp->node;

        if (!ndlp) {
                ndlp = emlxs_node_find_did(port, did);
        }
        if (pkt->pkt_state != FC_PKT_SUCCESS) {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
                    "emlxs_cmpl_dhchap_success_issue: 0x%x %x. No retry.",
                    did, pkt->pkt_state);
        } else {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
                    "emlxs_cmpl_dhchap_success_issue: did=0x%x. Succcess.",
                    did);
        }

        if (ndlp) {
                if (pkt->pkt_state == FC_PKT_SUCCESS) {
                        (void) emlxs_dhchap_state_machine(port, NULL, NULL,
                            NULL, ndlp, NODE_EVENT_CMPL_AUTH_MSG);
                }
        }
        emlxs_pkt_free(pkt);

        return;

} /* emlxs_cmpl_dhchap_success_issue */


/*
 * if rsp == NULL, this is only the DHCHAP_Success msg
 *
 * if rsp != NULL, DHCHAP_Success contains rsp to the challenge.
 */
/* ARGSUSED */
uint32_t
emlxs_issue_dhchap_success(
        emlxs_port_t *port,
        NODELIST *ndlp,
        int retry,
        uint8_t *rsp)
{
        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
        fc_packet_t *pkt;
        uint32_t cmd_size;
        uint32_t rsp_size;
        uint8_t *pCmd;
        uint16_t cmdsize;
        DHCHAP_SUCCESS_HDR *ap;
        uint8_t *tmp;
        uint32_t len;
```

```
1184         uint32_t ret;
1185
1186         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1187             "emlxs_issue_dhchap_success: did=0x%x", ndlp->nlp_DID);
1188
1189         if (ndlp->nlp_DID == FABRIC_DID) {
1190                 if (node_dhc->nlp_auth_hashid == AUTH_MD5)
1191                         len = MD5_LEN;
1192                 else
1193                         len = SHA1_LEN;
1194         } else {
1195                 len = (node_dhc->nlp_auth_hashid == AUTH_MD5) ?
1196                     MD5_LEN : SHA1_LEN;
1197         }
1198
1199         if (rsp == NULL) {
1200                 cmdsize = sizeof (DHCHAP_SUCCESS_HDR);
1201         } else {
1202
1203                 cmdsize = sizeof (DHCHAP_SUCCESS_HDR) + len;
1204         }
1205
1206         cmd_size = cmdsize;
1207         rsp_size = 4;
1208
1209         if ((pkt = emlxs_prep_els_fc_pkt(port, ndlp->nlp_DID, cmd_size,
1210             rsp_size, 0, KM_NOSLEEP)) == NULL) {
1211                 return (1);
1212         }
1213         pCmd = (uint8_t *)pkt->pkt_cmd;
1214
1215         ap = (DHCHAP_SUCCESS_HDR *)pCmd;
1216         tmp = (uint8_t *)pCmd;
1217
1218         ap->auth_els_code = ELS_CMD_AUTH_CODE;
1219         ap->auth_els_flags = 0x0;
1220         ap->auth_msg_code = DHCHAP_SUCCESS;
1221         ap->proto_version = 0x01;
1222
1223         /*
1224          * In case of rsp == NULL meaning that this is DHCHAP_Success issued
1225          * when Host is the initiator AND this DHCHAP_Success is issused in
1226          * response to the bi-directional authentication, meaning Host
1227          * authenticate another entity, therefore no more DHCHAP_Success
1228          * expected. OR this DHCHAP_Success is issued by host when host is
1229          * the responder BUT it is uni-directional auth, therefore no more
1230          * DHCHAP_Success expected.
1231          *
1232          * In case of rsp != NULL it indicates this DHCHAP_Success is issued
1233          * when host is the responder AND this DHCHAP_Success has reply
1234          * embedded therefore the host expects DHCHAP_Success from other
1235          * entity in transaction.
1236          */
1237         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1238             "emlxs_issue_dhchap_success: 0x%x 0x%x 0x%x 0x%x 0x%x %p",
1239             ndlp->nlp_DID, node_dhc->nlp_auth_hashid,
1240             node_dhc->nlp_auth_tranid_rsp,
1241             node_dhc->nlp_auth_tranid_ini, cmdsize, rsp);
1242
1243         if (rsp == NULL) {
1244                 ap->msg_len = LE_SWAP32(0x00000004);
1245                 ap->RspVal_len = 0x0;
1246
1247                 node_dhc->fc_dhchap_success_expected = 0;
1248         } else {
1249                 node_dhc->fc_dhchap_success_expected = 1;
```



```
1251                 ap->msg_len = LE_SWAP32(4 + len);
1252
1253                 tmp += sizeof (DHCHAP_SUCCESS_HDR) - sizeof (uint32_t);
1254                 *(uint32_t *)tmp = LE_SWAP32(len);
1255                 tmp += sizeof (uint32_t);
1256                 bcopy((void *)rsp, (void *)tmp, len);
1257         }
1258
1259         if (node_dhc->nlp_reauth_status == NLP_HOST_REAUTH_IN_PROGRESS) {
1260                 ap->tran_id = LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);
1261         } else {
1262                 if (node_dhc->nlp_auth_flag == 2) {
1263                         ap->tran_id =
1264                             LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);
1265                 } else if (node_dhc->nlp_auth_flag == 1) {
1266                         ap->tran_id =
1267                             LE_SWAP32(node_dhc->nlp_auth_tranid_ini);
1268                 } else {
1269                         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_debug_msg,
1270                             "emlxs_is_dhch_success: (1) 0x%x 0x%x 0x%x 0x%x",
1271                             ndlp->nlp_DID, node_dhc->nlp_auth_flag,
1272                             node_dhc->nlp_auth_tranid_rsp,
1273                             node_dhc->nlp_auth_tranid_ini);
1274
1275                         return (1);
1276                 }
1277         }
1278
1279         pkt->pkt_comp = emlxs_cmpl_dhchap_success_issue;
1280
1281         ret = emlxs_pkt_send(pkt, 1);
1282
1283         if (ret != FC_SUCCESS) {
1284                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1285                     "emlxs_issue_dhchap_success: Unable to send packet. 0x%x",
1286                     ret);
1287
1288                 emlxs_pkt_free(pkt);
1289
1290                 return (1);
1291         }
1292         return (0);
1293
1294 } /* emlxs_issue_dhchap_success */
1295
1296
1297 /*
1298  * ! emlxs_cmpl_auth_reject_issue
1299  *
1300  * \pre \post \param   phba \param   cmdiocb \param   rspiocb \return  void
1301  *
1302  * \b Description: iocb_cmpl callback function.
1303  *
1304  */
1305 static void
1306 emlxs_cmpl_auth_reject_issue(fc_packet_t *pkt)
1307 {
1308         emlxs_port_t *port = pkt->pkt_ulp_private;
1309         emlxs_buf_t *sbp;
1310         NODELIST *ndlp;
1311         uint32_t did;
1312
1313         did = pkt->pkt_cmd_fhdr.d_id;
1314         sbp = (emlxs_buf_t *)pkt->pkt_fca_private;
1315         ndlp = sbp->node;
```

```
1317         if (!ndlp) {
1318                 ndlp = emlxs_node_find_did(port, did);
1319         }
1320         if (pkt->pkt_state != FC_PKT_SUCCESS) {
1321                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1322                     "emlxs_cmpl_auth_reject_issue: 0x%x %x. No retry.",
1323                     did, pkt->pkt_state);
1324         } else {
1325                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1326                     "emlxs_cmpl_auth_reject_issue: did=0x%x. Succcess.",
1327                     did);
1328         }

1330         if (ndlp) {
1331                 /* setup the new state */
1332                 emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED, 0, 0);

1334                 if (pkt->pkt_state == FC_PKT_SUCCESS) {
1335                         (void) emlxs_dhchap_state_machine(port, NULL, NULL,
1336                             NULL, ndlp, NODE_EVENT_CMPL_AUTH_MSG);
1337                 }
1338         }
1339         emlxs_pkt_free(pkt);

1341         return;

1343 } /* emlxs_cmpl_auth_reject_issue */


1346 /*
1347  * If Logical Error and Reason Code Explanation is "Restart Authentication
1348  * Protocol" then the Transaction Identifier could be
1349  * any value.
1350  */
1351 /* ARGSUSED */
1352 static uint32_t
1353 emlxs_issue_auth_reject(
1354         emlxs_port_t *port,
1355         NODELIST *ndlp,
1356         int retry,
1357         uint32_t *arg,
1358         uint8_t ReasonCode,
1359         uint8_t ReasonCodeExplanation)
1360 {
1361         fc_packet_t *pkt;
1362         uint32_t cmd_size;
1363         uint32_t rsp_size;
1364         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
1365         uint16_t cmdsize;
1366         AUTH_RJT *ap;
1367         char info[64];

1369         if (node_dhc->nlp_authrsp_tmo) {
1370                 node_dhc->nlp_authrsp_tmo = 0;
1371         }
1372         cmdsize = sizeof (AUTH_RJT);
1373         cmd_size = cmdsize;
1374         rsp_size = 4;

1376         if ((pkt = emlxs_prep_els_fc_pkt(port, ndlp->nlp_DID, cmd_size,
1377             rsp_size, 0, KM_NOSLEEP)) == NULL) {
1378                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
1379                     "Auth reject failed: Unable to allocate pkt. 0x%x %x %x",
1380                     ndlp->nlp_DID, ReasonCode, ReasonCodeExplanation);
```



```
1382                 return (1);
1383         }
1384         ap = (AUTH_RJT *) pkt->pkt_cmd;
1385         ap->auth_els_code = ELS_CMD_AUTH_CODE;
1386         ap->auth_els_flags = 0x0;
1387         ap->auth_msg_code = AUTH_REJECT;
1388         ap->proto_version = 0x01;
1389         ap->msg_len = LE_SWAP32(4);

1391         if (node_dhc->nlp_auth_flag == 2) {
1392                 ap->tran_id = LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);
1393         } else if (node_dhc->nlp_auth_flag == 1) {
1394                 ap->tran_id = LE_SWAP32(node_dhc->nlp_auth_tranid_ini);
1395         } else {
1396                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
1397                     "Auth reject failed.Invalid flag=%d. 0x%x %x expl=%x",
1398                     ndlp->nlp_DID, node_dhc->nlp_auth_flag, ReasonCode,
1399                     ReasonCodeExplanation);

1401                 emlxs_pkt_free(pkt);

1403                 return (1);
1404         }

1406         ap->ReasonCode = ReasonCode;
1407         ap->ReasonCodeExplanation = ReasonCodeExplanation;

1409         pkt->pkt_comp = emlxs_cmpl_auth_reject_issue;

1411         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_debug_msg,
1412             "Auth reject: did=0x%x reason=%x expl=%x",
1413             ndlp->nlp_DID, ReasonCode, ReasonCodeExplanation);

1415         if (emlxs_pkt_send(pkt, 1) != FC_SUCCESS) {
1416                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
1417                     "Auth reject failed. Unable to send pkt. 0x%x %x expl=%x",
1418                     ndlp->nlp_DID, node_dhc->nlp_auth_flag, ReasonCode,
1419                     ReasonCodeExplanation);

1421                 emlxs_pkt_free(pkt);

1423                 return (1);
1424         }
1425         (void) sprintf(info,
1426             "Auth-Reject: ReasonCode=0x%x, ReasonCodeExplanation=0x%x",
1427             ReasonCode, ReasonCodeExplanation);

1429         emlxs_log_auth_event(port, ndlp, ESC_EMLXS_28, info);

1431         return (0);

1433 } /* emlxs_issue_auth_reject */


1436 static fc_packet_t *
1437         emlxs_prep_els_fc_pkt(
1438         emlxs_port_t *port,
1439         uint32_t d_id,
1440         uint32_t cmd_size,
1441         uint32_t rsp_size,
1442         uint32_t datalen,
1443         int32_t sleepflag)
1444 {
1445         fc_packet_t *pkt;

1447         /* simulate the ULP stack's fc_packet send out */
```

```
1448         if (!(pkt = emlxs_pkt_alloc(port, cmd_size, rsp_size,
1449             datalen, sleepflag))) {
1450                 return (NULL);
1451         }
1452         pkt->pkt_tran_type = FC_PKT_EXCHANGE;
1453         pkt->pkt_timeout = 35;

1455         /* Build the fc header */
1456         pkt->pkt_cmd_fhdr.d_id = LE_SWAP24_LO(d_id);
1457         pkt->pkt_cmd_fhdr.r_ctl = R_CTL_ELS_REQ;
1458         pkt->pkt_cmd_fhdr.s_id = LE_SWAP24_LO(port->did);
1459         pkt->pkt_cmd_fhdr.type = FC_TYPE_EXTENDED_LS;
1460         pkt->pkt_cmd_fhdr.f_ctl =
1461             F_CTL_FIRST_SEQ | F_CTL_END_SEQ | F_CTL_SEQ_INITIATIVE;
1462         pkt->pkt_cmd_fhdr.seq_id = 0;
1463         pkt->pkt_cmd_fhdr.df_ctl = 0;
1464         pkt->pkt_cmd_fhdr.seq_cnt = 0;
1465         pkt->pkt_cmd_fhdr.ox_id = 0xFFFF;
1466         pkt->pkt_cmd_fhdr.rx_id = 0xFFFF;
1467         pkt->pkt_cmd_fhdr.ro = 0;

1469         return ((fc_packet_t *)pkt);

1471 } /* emlxs_prep_els_fc_pkt */


1474 /*
1475  * ! emlxs_issue_auth_negotiate
1476  *
1477  * \pre \post \param   port \param   ndlp \param   retry \param   flag \return
1478  * int
1479  *
1480  * \b Description:
1481  *
1482  * The routine is invoked when host as the authentication initiator which
1483  * issue the AUTH_ELS command AUTH_Negotiate to the other
1484  * entity ndlp. When this Auth_Negotiate command is completed, the iocb_cmpl
1485  * will get called as the solicited mbox cmd
1486  * callback. Some switch only support NULL dhchap in which case negotiate
1487  * should be modified to only have NULL DH specificed.
1488  *
1489  */
1490 /* ARGSUSED */
1491 static int
1492         emlxs_issue_auth_negotiate(
1493         emlxs_port_t *port,
1494         emlxs_node_t *ndlp,
1495         uint8_t retry)
1496 {
1497         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
1498         fc_packet_t *pkt;
1499         uint32_t cmd_size;
1500         uint32_t rsp_size;
1501         uint16_t cmdsize;
1502         AUTH_MSG_NEGOT_NULL_1 *null_ap1;
1503         AUTH_MSG_NEGOT_NULL_2 *null_ap2;
1504         uint32_t num_hs = 0;
1505         uint8_t flag;
1506         AUTH_MSG_NEGOT_1 *ap1;
1507         AUTH_MSG_NEGOT_2 *ap2;
1508         uint16_t para_len = 0;
1509         uint16_t hash_wcnt = 0;
1510         uint16_t dhgp_wcnt = 0;


1513         emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_NEGOTIATE_ISSUE, 0, 0);
```



```
1515         /* Full DH group support limit:2, only NULL group support limit:1 */
1516         flag = (node_dhc->nlp_auth_limit == 2) ? 1 : 0;

1518         /* first: determine the cmdsize based on the auth cfg parameters */
1519         if (flag == 1) {
1520                 /* May be Full DH group + 2 hash may not be */
1521                 cmdsize = sizeof (AUTH_MSG_NEGOT_NULL);

1523                 cmdsize += 2 + 2;       /* name tag: 2, name length: 2 */
1524                 cmdsize += 8;   /* WWN: 8 */
1525                 cmdsize += 4;   /* num of protocol: 4 */
1526                 cmdsize += 4;   /* protocol parms length: 4 */
1527                 cmdsize += 4;   /* protocol id: 4 */
1528                 para_len += 4;

1530                 cmdsize += 2 + 2;       /* hashlist: tag: 2, count:2 */
1531                 para_len += 4;

1533                 if (node_dhc->auth_cfg.hash_priority[1] == 0x00) {
1534                         /* only one hash func */
1535                         cmdsize += 4;
1536                         num_hs = 1;
1537                         para_len += 4;
1538                         hash_wcnt = 1;
1539                 } else {
1540                         /* two hash funcs */
1541                         cmdsize += 4 + 4;
1542                         num_hs = 2;
1543                         para_len += 4 + 4;
1544                         hash_wcnt = 2;
1545                 }

1547                 cmdsize += 2 + 2;
1548                 para_len += 4;
1549                 if (node_dhc->auth_cfg.dh_group_priority[1] == 0xf) {
1550                         /* only one dhgp specified: could be NULL or non-NULL */
1551                         cmdsize += 4;
1552                         para_len += 4;
1553                         dhgp_wcnt = 1;

1555                 } else if (node_dhc->auth_cfg.dh_group_priority[2] == 0xf) {
1556                         /* two dhgps specified */
1557                         cmdsize += 4 + 4;
1558                         para_len += 4 + 4;
1559                         dhgp_wcnt = 2;

1561                 } else if (node_dhc->auth_cfg.dh_group_priority[3] == 0xf) {
1562                         /* three dhgps specified */
1563                         cmdsize += 4 + 4 + 4;
1564                         para_len += 4 + 4 + 4;
1565                         dhgp_wcnt = 3;

1567                 } else if (node_dhc->auth_cfg.dh_group_priority[4] == 0xf) {
1568                         /* four dhgps specified */
1569                         cmdsize += 4 + 4 + 4 + 4;
1570                         para_len += 4 + 4 + 4 + 4;
1571                         dhgp_wcnt = 4;

1573                 } else if (node_dhc->auth_cfg.dh_group_priority[5] == 0xf) {
1574                         cmdsize += 4 + 4 + 4 + 4 + 4;
1575                         para_len += 4 + 4 + 4 + 4 + 4;
1576                         dhgp_wcnt = 5;

1578                 }
1579         } else {
```

```
1580                cmdsize = sizeof (AUTH_MSG_NEGOT_NULL);

1582                /*
1583                 * get the right payload size in byte: determined by config
1584                 * parameters
1585                 */
1586                cmdsize += 2 + 2 + 8;   /* name tag:2, name length:2, name */
1587                                        /* value content:8 */
1588                cmdsize += 4;   /* number of usable authentication */
1589                                /* protocols:4 */
1590                cmdsize += 4;   /* auth protocol params length: 4 */
1591                cmdsize += 4;   /* auth protocol identifier: 4 */

1593                /* hash list infor */
1594                cmdsize += 4;   /* hashlist: tag:2, count:2 */

1596                if (node_dhc->auth_cfg.hash_priority[1] == 0x00) {
1597                        cmdsize += 4;   /* only one hash function provided */
1598                        num_hs = 1;
1599                } else {
1600                        num_hs = 2;
1601                        cmdsize += 4 + 4;       /* sha1: 4, md5: 4 */
1602                }

1604                /* dhgp list info */
1605                /* since this is NULL DH group */
1606                cmdsize += 4;   /* dhgroup: tag:2, count:2 */
1607                cmdsize += 4;   /* set it to zero */
1608        }

1610        cmd_size = cmdsize;
1611        rsp_size = 4;

1613        if ((pkt = emlxs_prep_els_fc_pkt(port, ndlp->nlp_DID, cmd_size,
1614            rsp_size, 0, KM_NOSLEEP)) == NULL) {
1615                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
1616                    "issue_auth_negotiate: Unable to allocate pkt. 0x%x %d",
1617                    ndlp->nlp_DID, cmd_size);

1619                return (1);
1620        }
1621        /* Fill in AUTH_MSG_NEGOT payload */
1622        if (flag == 1) {
1623                if (hash_wcnt == 1) {
1624                        ap1 = (AUTH_MSG_NEGOT_1 *)pkt->pkt_cmd;
1625                        ap1->auth_els_code = ELS_CMD_AUTH_CODE;
1626                        ap1->auth_els_flags = 0x00;
1627                        ap1->auth_msg_code = AUTH_NEGOTIATE;
1628                        ap1->proto_version = 0x01;
1629                        ap1->msg_len = LE_SWAP32(cmdsize -
1630                            sizeof (AUTH_MSG_NEGOT_NULL));
1631                } else {
1632                        ap2 = (AUTH_MSG_NEGOT_2 *)pkt->pkt_cmd;
1633                        ap2->auth_els_code = ELS_CMD_AUTH_CODE;
1634                        ap2->auth_els_flags = 0x00;
1635                        ap2->auth_msg_code = AUTH_NEGOTIATE;
1636                        ap2->proto_version = 0x01;
1637                        ap2->msg_len = LE_SWAP32(cmdsize -
1638                            sizeof (AUTH_MSG_NEGOT_NULL));
1639                }
1640        } else {
1641                if (node_dhc->auth_cfg.hash_priority[1] == 0x00) {
1642                        null_ap1 = (AUTH_MSG_NEGOT_NULL_1 *)pkt->pkt_cmd;
1643                        null_ap1->auth_els_code = ELS_CMD_AUTH_CODE;
1644                        null_ap1->auth_els_flags = 0x0;
1645                        null_ap1->auth_msg_code = AUTH_NEGOTIATE;
```



```
1646                        null_ap1->proto_version = 0x01;
1647                        null_ap1->msg_len = LE_SWAP32(cmdsize -
1648                            sizeof (AUTH_MSG_NEGOT_NULL));

1650                } else {
1651                        null_ap2 = (AUTH_MSG_NEGOT_NULL_2 *)pkt->pkt_cmd;
1652                        null_ap2->auth_els_code = ELS_CMD_AUTH_CODE;
1653                        null_ap2->auth_els_flags = 0x0;
1654                        null_ap2->auth_msg_code = AUTH_NEGOTIATE;
1655                        null_ap2->proto_version = 0x01;
1656                        null_ap2->msg_len = LE_SWAP32(cmdsize -
1657                            sizeof (AUTH_MSG_NEGOT_NULL));
1658                }
1659        }

1661        /*
1662         * For host reauthentication heart beat, the tran_id is incremented
1663         * by one for each heart beat being fired and round back to 1 when
1664         * 0xffffffff is reached. tran_id 0 is reserved as the initial linkup
1665         * authentication transaction id.
1666         */

1668        /* responder flag:2, initiator flag:1 */
1669        node_dhc->nlp_auth_flag = 2;    /* ndlp is the always the auth */
1670                                        /* responder */

1672        if (node_dhc->nlp_reauth_status == NLP_HOST_REAUTH_IN_PROGRESS) {
1673                if (node_dhc->nlp_auth_tranid_rsp == 0xffffffff) {
1674                        node_dhc->nlp_auth_tranid_rsp = 1;
1675                } else {
1676                        node_dhc->nlp_auth_tranid_rsp++;
1677                }
1678        } else {        /* !NLP_HOST_REAUTH_IN_PROGRESS */
1679                node_dhc->nlp_auth_tranid_rsp = 0;
1680        }

1682        if (flag == 1) {
1683                if (hash_wcnt == 1) {
1684                        ap1->tran_id =
1685                            LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);

1687                        ap1->params.name_tag = AUTH_NAME_ID;
1688                        ap1->params.name_len = AUTH_NAME_LEN;
1689                        bcopy((void *)&port->wwpn,
1690                            (void *) &ap1->params.nodeName, sizeof (NAME_TYPE));
1691                        ap1->params.proto_num = AUTH_PROTO_NUM;
1692                        ap1->params.para_len = LE_SWAP32(para_len);
1693                        ap1->params.proto_id = AUTH_DHCHAP;
1694                        ap1->params.HashList_tag = HASH_LIST_TAG;
1695                        ap1->params.HashList_wcnt = LE_SWAP16(hash_wcnt);
1696                        ap1->params.HashList_value1 =
1697                            node_dhc->auth_cfg.hash_priority[0];
1698                        ap1->params.DHgIDList_tag = DHGID_LIST_TAG;
1699                        ap1->params.DHgIDList_wnt = LE_SWAP16(dhgp_wcnt);

1701                        switch (dhgp_wcnt) {
1702                        case 5:
1703                                ap1->params.DHgIDList_g4 =
1704                                    (node_dhc->auth_cfg.dh_group_priority[4]);
1705                                ap1->params.DHgIDList_g3 =
1706                                    (node_dhc->auth_cfg.dh_group_priority[3]);
1707                                ap1->params.DHgIDList_g2 =
1708                                    (node_dhc->auth_cfg.dh_group_priority[2]);
1709                                ap1->params.DHgIDList_g1 =
1710                                    (node_dhc->auth_cfg.dh_group_priority[1]);
1711                                ap1->params.DHgIDList_g0 =
```

```
1712                                    (node_dhc->auth_cfg.dh_group_priority[0]);
1713                            break;
1714                    case 4:
1715                            ap1->params.DHgIDList_g3 =
1716                                (node_dhc->auth_cfg.dh_group_priority[3]);
1717                            ap1->params.DHgIDList_g2 =
1718                                (node_dhc->auth_cfg.dh_group_priority[2]);
1719                            ap1->params.DHgIDList_g1 =
1720                                (node_dhc->auth_cfg.dh_group_priority[1]);
1721                            ap1->params.DHgIDList_g0 =
1722                                (node_dhc->auth_cfg.dh_group_priority[0]);
1723                            break;
1724                    case 3:
1725                            ap1->params.DHgIDList_g2 =
1726                                (node_dhc->auth_cfg.dh_group_priority[2]);
1727                            ap1->params.DHgIDList_g1 =
1728                                (node_dhc->auth_cfg.dh_group_priority[1]);
1729                            ap1->params.DHgIDList_g0 =
1730                                (node_dhc->auth_cfg.dh_group_priority[0]);
1731                            break;
1732                    case 2:
1733                            ap1->params.DHgIDList_g1 =
1734                                (node_dhc->auth_cfg.dh_group_priority[1]);
1735                            ap1->params.DHgIDList_g0 =
1736                                (node_dhc->auth_cfg.dh_group_priority[0]);
1737                            break;
1738                    case 1:
1739                            ap1->params.DHgIDList_g0 =
1740                                (node_dhc->auth_cfg.dh_group_priority[0]);
1741                            break;
1742                    }
1743            } else {
1744                    ap2->tran_id =
1745                        LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);

1747                    ap2->params.name_tag = AUTH_NAME_ID;
1748                    ap2->params.name_len = AUTH_NAME_LEN;
1749                    bcopy((void *) &port->wwpn,
1750                        (void *) &ap2->params.nodeName, sizeof (NAME_TYPE));
1751                    ap2->params.proto_num = AUTH_PROTO_NUM;
1752                    ap2->params.para_len = LE_SWAP32(para_len);
1753                    ap2->params.proto_id = AUTH_DHCHAP;
1754                    ap2->params.HashList_tag = HASH_LIST_TAG;
1755                    ap2->params.HashList_wcnt = LE_SWAP16(hash_wcnt);
1756                    ap2->params.HashList_value1 =
1757                        (node_dhc->auth_cfg.hash_priority[0]);
1758                    ap2->params.HashList_value2 =
1759                        (node_dhc->auth_cfg.hash_priority[1]);

1761                    ap2->params.DHgIDList_tag = DHGID_LIST_TAG;
1762                    ap2->params.DHgIDList_wnt = LE_SWAP16(dhgp_wcnt);

1764                    switch (dhgp_wcnt) {
1765                    case 5:
1766                            ap2->params.DHgIDList_g4 =
1767                                (node_dhc->auth_cfg.dh_group_priority[4]);
1768                            ap2->params.DHgIDList_g3 =
1769                                (node_dhc->auth_cfg.dh_group_priority[3]);
1770                            ap2->params.DHgIDList_g2 =
1771                                (node_dhc->auth_cfg.dh_group_priority[2]);
1772                            ap2->params.DHgIDList_g1 =
1773                                (node_dhc->auth_cfg.dh_group_priority[1]);
1774                            ap2->params.DHgIDList_g0 =
1775                                (node_dhc->auth_cfg.dh_group_priority[0]);
1776                            break;
1777                    case 4:
```



```
1778                            ap2->params.DHgIDList_g3 =
1779                                (node_dhc->auth_cfg.dh_group_priority[3]);
1780                            ap2->params.DHgIDList_g2 =
1781                                (node_dhc->auth_cfg.dh_group_priority[2]);
1782                            ap2->params.DHgIDList_g1 =
1783                                (node_dhc->auth_cfg.dh_group_priority[1]);
1784                            ap2->params.DHgIDList_g0 =
1785                                (node_dhc->auth_cfg.dh_group_priority[0]);
1786                            break;
1787                    case 3:
1788                            ap2->params.DHgIDList_g2 =
1789                                (node_dhc->auth_cfg.dh_group_priority[2]);
1790                            ap2->params.DHgIDList_g1 =
1791                                (node_dhc->auth_cfg.dh_group_priority[1]);
1792                            ap2->params.DHgIDList_g0 =
1793                                (node_dhc->auth_cfg.dh_group_priority[0]);
1794                            break;
1795                    case 2:
1796                            ap2->params.DHgIDList_g1 =
1797                                (node_dhc->auth_cfg.dh_group_priority[1]);
1798                            ap2->params.DHgIDList_g0 =
1799                                (node_dhc->auth_cfg.dh_group_priority[0]);
1800                            break;
1801                    case 1:
1802                            ap2->params.DHgIDList_g0 =
1803                                (node_dhc->auth_cfg.dh_group_priority[0]);
1804                            break;
1805                    }
1806            }
1807    } else {
1808            if (num_hs == 1) {
1809                    null_ap1->tran_id =
1810                        LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);

1812                    null_ap1->params.name_tag = AUTH_NAME_ID;
1813                    null_ap1->params.name_len = AUTH_NAME_LEN;
1814                    bcopy((void *) &port->wwpn,
1815                        (void *) &null_ap1->params.nodeName,
1816                        sizeof (NAME_TYPE));
1817                    null_ap1->params.proto_num = AUTH_PROTO_NUM;
1818                    null_ap1->params.para_len = LE_SWAP32(0x00000014);
1819                    null_ap1->params.proto_id = AUTH_DHCHAP;
1820                    null_ap1->params.HashList_tag = HASH_LIST_TAG;
1821                    null_ap1->params.HashList_wcnt = LE_SWAP16(0x0001);
1822                    null_ap1->params.HashList_value1 =
1823                        (node_dhc->auth_cfg.hash_priority[0]);
1824                    null_ap1->params.DHgIDList_tag = DHGID_LIST_TAG;
1825                    null_ap1->params.DHgIDList_wnt = LE_SWAP16(0x0001);
1826                    null_ap1->params.DHgIDList_g0 = 0x0;
1827            } else {
1828                    null_ap2->tran_id =
1829                        LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);

1831                    null_ap2->params.name_tag = AUTH_NAME_ID;
1832                    null_ap2->params.name_len = AUTH_NAME_LEN;
1833                    bcopy((void *) &port->wwpn,
1834                        (void *) &null_ap2->params.nodeName,
1835                        sizeof (NAME_TYPE));
1836                    null_ap2->params.proto_num = AUTH_PROTO_NUM;
1837                    null_ap2->params.para_len = LE_SWAP32(0x00000018);
1838                    null_ap2->params.proto_id = AUTH_DHCHAP;

1840                    null_ap2->params.HashList_tag = HASH_LIST_TAG;
1841                    null_ap2->params.HashList_wcnt = LE_SWAP16(0x0002);
1842                    null_ap2->params.HashList_value1 =
1843                        (node_dhc->auth_cfg.hash_priority[0]);
```

```
1844                        null_ap2->params.HashList_value2 =
1845                            (node_dhc->auth_cfg.hash_priority[1]);
1847                        null_ap2->params.DHgIDList_tag = DHGID_LIST_TAG;
1848                        null_ap2->params.DHgIDList_wnt = LE_SWAP16(0x0001);
1849                        null_ap2->params.DHgIDList_g0 = 0x0;
1850                }
1851        }
1853        pkt->pkt_comp = emlxs_cmpl_auth_negotiate_issue;
1855        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_debug_msg,
1856            "issue_auth_negotiate: %x flag=%d size=%d hash=%x,%x tid=%x,%x",
1857            ndlp->nlp_DID, flag, cmd_size,
1858            node_dhc->auth_cfg.hash_priority[0],
1859            node_dhc->auth_cfg.hash_priority[1],
1860            node_dhc->nlp_auth_tranid_rsp, node_dhc->nlp_auth_tranid_ini);
1862        if (emlxs_pkt_send(pkt, 1) != FC_SUCCESS) {
1863                emlxs_pkt_free(pkt);
1865                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
1866                    "issue_auth_negotiate: Unable to send pkt. did=0x%x",
1867                    ndlp->nlp_DID);
1869                return (1);
1870        }
1871        return (0);
1873 } /* emlxs_issue_auth_negotiate() */
1877 /*
1878  * ! emlxs_cmpl_auth_negotiate_issue
1879  *
1880  * \pre \post \param   phba \param   cmdiocb \param   rspiocb \return  void
1881  *
1882  * \b Description: iocb_cmpl callback function.
1883  *
1884  */
1885 static void
1886 emlxs_cmpl_auth_negotiate_issue(fc_packet_t *pkt)
1887 {
1888        emlxs_port_t *port = pkt->pkt_ulp_private;
1889        emlxs_buf_t *sbp;
1890        NODELIST *ndlp;
1891        emlxs_node_dhc_t *node_dhc;
1892        uint32_t did;
1894        did = pkt->pkt_cmd_fhdr.d_id;
1895        sbp = (emlxs_buf_t *)pkt->pkt_fca_private;
1896        ndlp = sbp->node;
1897        node_dhc = &ndlp->node_dhc;
1899        if (!ndlp) {
1900                ndlp = emlxs_node_find_did(port, did);
1901        }
1902        if (pkt->pkt_state != FC_PKT_SUCCESS) {
1903                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1904                    "emlxs_cmpl_dhchap_negotiate_issue: 0x%x %x. Noretry.",
1905                    did, pkt->pkt_state);
1906        } else {
1907                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1908                    "emlxs_cmpl_dhchap_negotiate_issue: did=0x%x. Succcess.",
1909                    did);
```



```
1910        }
1912        if (ndlp) {
1913                if (pkt->pkt_state == FC_PKT_SUCCESS) {
1914                        (void) emlxs_dhchap_state_machine(port, NULL, NULL,
1915                            NULL, ndlp, NODE_EVENT_CMPL_AUTH_MSG);
1916                } else {
1917                        emlxs_dhc_set_reauth_time(port, ndlp, DISABLE);
1919                        emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED,
1920                            0, 0);
1922                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_debug_msg,
1923                            "Reauth disabled. did=0x%x state=%x",
1924                            ndlp->nlp_DID, node_dhc->state);
1926                        emlxs_dhc_auth_complete(port, ndlp, 1);
1927                }
1928        }
1929        emlxs_pkt_free(pkt);
1931        return;
1933 } /* emlxs_cmpl_auth_negotiate_issue */
1936 /*
1937  * ! emlxs_cmpl_auth_msg_auth_negotiate_issue
1938  *
1939  * \pre \post \param   port \param   CHANNEL * rp \param   arg \param   evt
1940  * \return  uint32_t \b Description:
1941  *
1942  * This routine is invoked when the host receive the solicited ACC/RJT ELS
1943  * cmd from an NxPort or FxPort that has received the ELS
1944  * AUTH Negotiate msg from the host. in case of RJT, Auth_Negotiate should
1945  * be retried in emlxs_cmpl_auth_negotiate_issue
1946  * call. in case of ACC, the host must be the initiator because its current
1947  * state could be "AUTH_NEGOTIATE_RCV" if it is the
1948  * responder. Then the next stat = AUTH_NEGOTIATE_CMPL_WAIT4NEXT
1949  */
1950 /* ARGSUSED */
1951 static uint32_t
1952 emlxs_cmpl_auth_msg_auth_negotiate_issue(
1953 emlxs_port_t *port,
1954 /* CHANNEL * rp, */ void *arg1,
1955 /* IOCBQ * iocbq, */ void *arg2,
1956 /* MATCHMAP * mp, */ void *arg3,
1957 /* NODELIST * ndlp, */ void *arg4,
1958 uint32_t evt)
1959 {
1960        NODELIST *ndlp = (NODELIST *)arg4;
1961        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
1963        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
1964            "emlxs_cmpl_auth_msg_auth_negotiate_issue: did=0x%x",
1965            ndlp->nlp_DID);
1967        /* start the emlxs_dhc_authrsp_timeout timer */
1968        if (node_dhc->nlp_authrsp_tmo == 0) {
1969                node_dhc->nlp_authrsp_tmo = DRV_TIME +
1970                    node_dhc->auth_cfg.authentication_timeout;
1971        }
1972        /*
1973         * The next state should be
1974         * emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next
1975         */
```

```
1976         emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_NEGOTIATE_CMPL_WAIT4NEXT,
1977             0, 0);

1979         return (node_dhc->state);

1981 } /* emlxs_cmpl_auth_msg_auth_negotiate_issue */



1985 /*
1986  * ! emlxs_rcv_auth_msg_auth_negotiate_issue
1987  *
1988  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
1989  * uint32_t \b Description:
1990  *
1991  * This routine is supported for HBA in either auth initiator mode or
1992  * responder mode.
1993  *
1994  * This routine is invoked when the host receive an unsolicited ELS AUTH Msg
1995  * from an NxPort or FxPort to which the host has just
1996  * sent out an ELS AUTH negotiate msg. and the NxPort or FxPort also LS_ACC
1997  * to the host's AUTH_Negotiate msg.
1998  *
1999  * If this unsolicited ELS auth msg is from the FxPort or a NxPort with a
2000  * numerically lower WWPN, the host will be the winner in
2001  * this authentication transaction initiation phase, the host as the
2002  * initiator will send back ACC and then Auth_Reject message
2003  * with the Reason Code 'Logical Error' and Reason Code Explanation'
2004  * Authentication Transaction Already Started' and with the
2005  * current state unchanged and mark itself as auth_initiator.
2006  *
2007  * Otherwise, the host will be the responder that will reply to the received
2008  * AUTH_Negotiate message will ACC (or RJT?) and abort
2009  * its own transaction upon receipt of the AUTH_Reject message. The new state
2010  * will be "AUTH_NEGOTIATE_RCV" and mark the host as
2011  * auth_responder.
2012  */
2013 /* ARGSUSED */
2014 static uint32_t
2015 emlxs_rcv_auth_msg_auth_negotiate_issue(
2016 emlxs_port_t *port,
2017 /* CHANNEL * rp, */ void *arg1,
2018 /* IOCBQ * iocbq, */ void *arg2,
2019 /* MATCHMAP * mp, */ void *arg3,
2020 /* NODELIST * ndlp */ void *arg4,
2021 uint32_t evt)
2022 {
2023         NODELIST *ndlp = (NODELIST *)arg4;
2024         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
2025         IOCBQ *iocbq = (IOCBQ *) arg2;
2026         uint8_t ReasonCode;
2027         uint8_t ReasonCodeExplanation;

2029         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
2030             "emlxs_rcv_auth_msg_auth_negotiate_issue: did=0x%x",
2031             ndlp->nlp_DID);

2033         /* Anyway we accept it first and then send auth_reject */
2034         (void) emlxs_els_reply(port, iocbq, ELS_CMD_ACC, ELS_CMD_AUTH, 0, 0);

2036         /* host is always the initiator and it should win */
2037         ReasonCode = AUTHRJT_LOGIC_ERR;
2038         ReasonCodeExplanation = AUTHEXP_AUTHTRAN_STARTED;

2040         emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_NEGOTIATE_ISSUE,
2041             ReasonCode, ReasonCodeExplanation);
```



```
2042         (void) emlxs_issue_auth_reject(port, ndlp, 0, 0, ReasonCode,
2043             ReasonCodeExplanation);

2045         return (node_dhc->state);

2047 } /* emlxs_rcv_auth_msg_auth_negotiate_issue */



2050 /*
2051  * ! emlxs_cmpl_dhchap_reply_issue
2052  *
2053  * \pre \post \param   phba \param   cmdiocb \param   rspiocb \return  void
2054  *
2055  * \b Description: iocb_cmpl callback function.
2056  *
2057  */
2058 static void
2059 emlxs_cmpl_dhchap_reply_issue(fc_packet_t *pkt)
2060 {
2061         emlxs_port_t *port = pkt->pkt_ulp_private;
2062         emlxs_buf_t *sbp;
2063         NODELIST *ndlp;
2064         uint32_t did;

2066         did = pkt->pkt_cmd_fhdr.d_id;
2067         sbp = (emlxs_buf_t *)pkt->pkt_fca_private;
2068         ndlp = sbp->node;

2070         if (!ndlp) {
2071                 ndlp = emlxs_node_find_did(port, did);
2072         }
2073         if (pkt->pkt_state != FC_PKT_SUCCESS) {
2074                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
2075                     "emlxs_cmpl_dhchap_reply_issue: 0x%x %x. No retry.",
2076                     did, pkt->pkt_state);
2077         } else {
2078                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
2079                     "emlxs_cmpl_dhchap_reply_issue: did=0x%x. Succcess.",
2080                     did);
2081         }

2083         if (ndlp) {
2084                 if (pkt->pkt_state == FC_PKT_SUCCESS) {
2085                         (void) emlxs_dhchap_state_machine(port, NULL, NULL,
2086                             NULL, ndlp, NODE_EVENT_CMPL_AUTH_MSG);
2087                 }
2088         }
2089         emlxs_pkt_free(pkt);

2091         return;

2093 } /* emlxs_cmpl_dhchap_reply_issue */



2096 /*
2097  * arg: the AUTH_Negotiate payload from the initiator. payload_len: the
2098  * payload length
2099  *
2100  * We always send out the challenge parameter based on our preference
2101  * order configured on the host side no matter what perference
2102  * order looks like from auth_negotiate . In other words, if the host issue
2103  * the challenge the host will make the decision as to
2104  * what hash function, what dhgp_id is to be used.
2105  *
2106  * This challenge value should not be confused with the challenge value for
2107  * bi-dir as part of reply when host is the initiator.
```

```
 */
/* ARGSUSED */
uint32_t
emlxs_issue_dhchap_challenge(
        emlxs_port_t *port,
        NODELIST *ndlp,
        int retry,
        void *arg,
        uint32_t payload_len,
        uint32_t hash_id,
        uint32_t dhgp_id)
{
        emlxs_hba_t *hba = HBA;
        fc_packet_t *pkt;
        uint32_t cmd_size;
        uint32_t rsp_size;
        uint16_t cmdsize = 0;
        uint8_t *pCmd;
        emlxs_port_dhc_t *port_dhc = &port->port_dhc;
        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
        DHCHAP_CHALL *chal;
        uint8_t *tmp;
        uint8_t random_number[20];
        uint8_t dhval[256];
        uint32_t dhval_len;
        uint32_t tran_id;
        BIG_ERR_CODE err = BIG_OK;

        /*
         * we assume the HBAnyware should configure the driver the right
         * parameters for challenge. for now, we create our own challenge.
         */
        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
            "emlxs_issue_dhchap_challenge: did=0x%x hashlist=[%x,%x,%x,%x]",
            ndlp->nlp_DID, node_dhc->auth_cfg.hash_priority[0],
            node_dhc->auth_cfg.hash_priority[1],
            node_dhc->auth_cfg.hash_priority[2],
            node_dhc->auth_cfg.hash_priority[3]);

        /*
         * Here is my own challenge structure:
         *
         * 1: AUTH_MSG_HDR (12 bytes + 4 bytes + 8 bytes) 2: hasd_id (4
         * bytes) 3: dhgp_id (4 bytes) 4: cval_len (4 bytes) 5: cval
         * (20 bytes or 16 bytes: cval_len bytes) 6: dhval_len (4 bytes)
         * 7: dhval (dhval_len bytes) all these information should be stored
         * in port_dhc struct
         */
        if (hash_id == AUTH_SHA1) {
                cmdsize = (12 + 4 + 8) + (4 + 4 + 4) + 20 + 4;
        } else if (hash_id == AUTH_MD5) {
                cmdsize = (12 + 4 + 8) + (4 + 4 + 4) + 16 + 4;
        } else {
                return (1);
        }


        switch (dhgp_id) {
        case GROUP_NULL:
                break;

        case GROUP_1024:
                cmdsize += 128;
                break;

        case GROUP_1280:
```



```
                cmdsize += 160;
                break;

        case GROUP_1536:
                cmdsize += 192;
                break;

        case GROUP_2048:
                cmdsize += 256;
                break;

        default:
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                    "emlxs_issue_dhchap_challenge: Invalid dhgp_id=0x%x",
                    dhgp_id);
                return (1);
        }

        cmd_size = cmdsize;
        rsp_size = 4;

        if ((pkt = emlxs_prep_els_fc_pkt(port, ndlp->nlp_DID, cmd_size,
            rsp_size,
            0, KM_NOSLEEP)) == NULL) {
                return (1);
        }
        pCmd = (uint8_t *)pkt->pkt_cmd;

        tmp = (uint8_t *)arg;
        tmp += 8;
        /* collect tran_id: this tran_id is set by the initiator */
        tran_id = *(uint32_t *)tmp;

        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
            "emlxs_issue_dhchap_challenge: 0x%x 0x%x 0x%x 0x%x %d 0x%x 0x%x 0x%x",
            ndlp->nlp_DID, node_dhc->nlp_auth_tranid_ini,
            node_dhc->nlp_auth_tranid_rsp,
            cmdsize, tran_id, hash_id, dhgp_id);

        /* store the tran_id : ndlp is the initiator */
        node_dhc->nlp_auth_tranid_ini = LE_SWAP32(tran_id);

        tmp += sizeof (uint32_t);

        chal = (DHCHAP_CHALL *)pCmd;
        chal->cnul.msg_hdr.auth_els_code = ELS_CMD_AUTH_CODE;
        chal->cnul.msg_hdr.auth_els_flags = 0x0;
        chal->cnul.msg_hdr.auth_msg_code = DHCHAP_CHALLENGE;
        chal->cnul.msg_hdr.proto_version = 0x01;
        chal->cnul.msg_hdr.msg_len = LE_SWAP32(cmdsize - 12);
        chal->cnul.msg_hdr.tran_id = tran_id;
        chal->cnul.msg_hdr.name_tag = (AUTH_NAME_ID);
        chal->cnul.msg_hdr.name_len = (AUTH_NAME_LEN);

        bcopy((void *) &port->wwpn,
            (void *) &chal->cnul.msg_hdr.nodeName, sizeof (NAME_TYPE));

        chal->cnul.hash_id = hash_id;
        chal->cnul.dhgp_id = dhgp_id;

        chal->cnul.cval_len = ((chal->cnul.hash_id == AUTH_SHA1) ?
            LE_SWAP32(SHA1_LEN) : LE_SWAP32(MD5_LEN));

        tmp = (uint8_t *)pCmd;
        tmp += sizeof (DHCHAP_CHALL_NULL);
```

```
#ifdef RAND
        /* generate a random number as the challenge */
        bzero(random_number, LE_SWAP32(chal->cnul.cval_len));

        if (hba->rdn_flag == 1) {
                emlxs_get_random_bytes(ndlp, random_number, 20);
        } else {
                (void) random_get_pseudo_bytes(random_number,
                    LE_SWAP32(chal->cnul.cval_len));
        }

        /*
         * the host should store the challenge for later usage when later on
         * host get the reply msg, host needs to verify it by using its old
         * challenge, its private key as the input to the hash function. the
         * challenge as the random_number should be stored in
         * node_dhc->hrsp_cval[]
         */
        if (ndlp->nlp_DID == FABRIC_DID) {
                bcopy((void *) &random_number[0],
                    (void *) &node_dhc->hrsp_cval[0],
                    LE_SWAP32(chal->cnul.cval_len));
                /* save another copy in partner's ndlp */
                bcopy((void *) &random_number[0],
                    (void *) &node_dhc->nlp_auth_misc.hrsp_cval[0],
                    LE_SWAP32(chal->cnul.cval_len));
        } else {
                bcopy((void *) &random_number[0],
                    (void *) &node_dhc->nlp_auth_misc.hrsp_cval[0],
                    LE_SWAP32(chal->cnul.cval_len));
        }
        bcopy((void *) &random_number[0], (void *) tmp,
            LE_SWAP32(chal->cnul.cval_len));

#endif  /* RAND */

        /* for test only hardcode the challenge value */
#ifdef MYRAND
        if (ndlp->nlp_DID == FABRIC_DID) {
                bcopy((void *) myrand, (void *) &node_dhc->hrsp_cval[0],
                    LE_SWAP32(chal->cnul.cval_len));
                /* save another copy in partner's ndlp */
                bcopy((void *) myrand,
                    (void *) &node_dhc->nlp_auth_misc.hrsp_cval[0],
                    LE_SWAP32(chal->cnul.cval_len));
        } else {
                bcopy((void *) myrand,
                    (void *) &node_dhc->nlp_auth_misc.hrsp_cval[0],
                    LE_SWAP32(chal->cnul.cval_len));
        }
        bcopy((void *) myrand, (void *) tmp,
            LE_SWAP32(chal->cnul.cval_len));

#endif  /* MYRAND */

        if (ndlp->nlp_DID == FABRIC_DID) {
                node_dhc->hrsp_cval_len = LE_SWAP32(chal->cnul.cval_len);
                node_dhc->nlp_auth_misc.hrsp_cval_len =
                    LE_SWAP32(chal->cnul.cval_len);
        } else {
                node_dhc->nlp_auth_misc.hrsp_cval_len =
                    LE_SWAP32(chal->cnul.cval_len);
        }

        tmp += LE_SWAP32(chal->cnul.cval_len);
```



```
        /*
         * we need another random number as the private key x which will be
         * used to compute the public key i.e. g^x mod p we intentionally set
         * the length of private key as the same length of challenge. we have
         * to store the private key in node_dhc->hrsp_priv_key[20].
         */
#ifdef RAND

        if (dhgp_id != GROUP_NULL) {

                bzero(random_number, LE_SWAP32(chal->cnul.cval_len));

                if (hba->rdn_flag == 1) {
                        emlxs_get_random_bytes(ndlp, random_number, 20);
                } else {
                        (void) random_get_pseudo_bytes(random_number,
                            LE_SWAP32(chal->cnul.cval_len));
                }

                if (ndlp->nlp_DID == FABRIC_DID) {
                        bcopy((void *) &random_number[0],
                            (void *) node_dhc->hrsp_priv_key,
                            LE_SWAP32(chal->cnul.cval_len));
                        bcopy((void *) &random_number[0],
                            (void *) node_dhc->nlp_auth_misc.hrsp_priv_key,
                            LE_SWAP32(chal->cnul.cval_len));
                } else {
                        bcopy((void *) &random_number[0],
                            (void *) node_dhc->nlp_auth_misc.hrsp_priv_key,
                            LE_SWAP32(chal->cnul.cval_len));
                }
        }
#endif  /* RAND */

#ifdef MYRAND
        if (dhgp_id != GROUP_NULL) {
                /* For test only we hardcode the priv_key here */
                bcopy((void *) myrand, (void *) node_dhc->hrsp_priv_key,
                    LE_SWAP32(chal->cnul.cval_len));

                if (ndlp->nlp_DID == FABRIC_DID) {
                        bcopy((void *) myrand,
                            (void *) node_dhc->hrsp_priv_key,
                            LE_SWAP32(chal->cnul.cval_len));
                        bcopy((void *) myrand,
                            (void *) node_dhc->nlp_auth_misc.hrsp_priv_key,
                            LE_SWAP32(chal->cnul.cval_len));
                } else {
                        bcopy((void *) myrand,
                            (void *) node_dhc->nlp_auth_misc.hrsp_priv_key,
                            LE_SWAP32(chal->cnul.cval_len));
                }
        }
#endif  /* MYRAND */

        /* also store the hash function and dhgp_id being used in challenge. */
        /* These information could be configurable through HBAnyware */
        node_dhc->nlp_auth_hashid = hash_id;
        node_dhc->nlp_auth_dhgpid = dhgp_id;

        /*
         * generate the DH value DH value is g^x mod p  and it is also called
         * public key in which g is 2, x is the random number ontained above.
         * p is the dhgp3_pVal
         */
```

```
#ifdef MYRAND

        /* to get (g^x mod p) with x private key */
        if (dhgp_id != GROUP_NULL) {

                err = emlxs_BIGNUM_get_dhval(port, port_dhc, ndlp, dhval,
                    &dhval_len, chal->cnul.dhgp_id,
                    myrand, LE_SWAP32(chal->cnul.cval_len));

                if (err != BIG_OK) {
                        emlxs_pkt_free(pkt);

                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                            "emlxs_issue_dhchap_challenge: error. 0x%x",
                            err);

                        return (1);
                }
                /* we are not going to use dhval and dhval_len */

                /* *(uint32_t *)tmp = dhval_len; */
                if (ndlp->nlp_DID == FABRIC_DID) {
                        *(uint32_t *)tmp =
                            LE_SWAP32(node_dhc->hrsp_pubkey_len);
                } else {
                        *(uint32_t *)tmp =
                            LE_SWAP32(
                            node_dhc->nlp_auth_misc.hrsp_pubkey_len);
                }

                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
                    "emlxs_issue_dhchap_challenge: 0x%x: 0x%x 0x%x",
                    ndlp->nlp_DID, *(uint32_t *)tmp, dhval_len);

                tmp += sizeof (uint32_t);

                if (ndlp->nlp_DID == FABRIC_DID) {
                        bcopy((void *) node_dhc->hrsp_pub_key, (void *)tmp,
                            node_dhc->hrsp_pubkey_len);
                } else {
                        bcopy((void *) node_dhc->nlp_auth_misc.hrsp_pub_key,
                            (void *)tmp,
                            node_dhc->nlp_auth_misc.hrsp_pubkey_len);
                }
        } else {
                /* NULL DHCHAP */
                *(uint32_t *)tmp = 0;
        }

#endif  /* MYRAND */

#ifdef RAND

        /* to get (g^x mod p) with x private key */
        if (dhgp_id != GROUP_NULL) {

                err = emlxs_BIGNUM_get_dhval(port, port_dhc, ndlp, dhval,
                    &dhval_len, chal->cnul.dhgp_id,
                    random_number, LE_SWAP32(chal->cnul.cval_len));

                if (err != BIG_OK) {
                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                            "emlxs_issue_dhchap_challenge: error. 0x%x",
                            err);

                        emlxs_pkt_free(pkt);
```



```
                        return (1);
                }
                /* we are not going to use dhval and dhval_len */

                /* *(uint32_t *)tmp = dhval_len; */
                if (ndlp->nlp_DID == FABRIC_DID) {
                        *(uint32_t *)tmp =
                            LE_SWAP32(node_dhc->hrsp_pubkey_len);
                } else {
                        *(uint32_t *)tmp =
                            LE_SWAP32(
                            node_dhc->nlp_auth_misc.hrsp_pubkey_len);
                }

                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
                    "emlxs_issue_dhchap_challenge: did=0x%x: pubkey_len=0x%x",
                    ndlp->nlp_DID, *(uint32_t *)tmp);

                tmp += sizeof (uint32_t);

                if (ndlp->nlp_DID == FABRIC_DID) {
                        bcopy((void *) node_dhc->hrsp_pub_key, (void *)tmp,
                            node_dhc->hrsp_pubkey_len);
                } else {
                        bcopy((void *) node_dhc->nlp_auth_misc.hrsp_pub_key,
                            (void *)tmp,
                            node_dhc->nlp_auth_misc.hrsp_pubkey_len);
                }
        } else {
                /* NULL DHCHAP */
                *(uint32_t *)tmp = 0;
        }

#endif  /* RAND */

        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
            "emlxs_issue_dhchap_challenge: 0x%x 0x%x 0x%x 0x%x 0x%x",
            ndlp->nlp_DID, node_dhc->nlp_auth_tranid_ini,
            node_dhc->nlp_auth_tranid_rsp,
            chal->cnul.hash_id, chal->cnul.dhgp_id);

        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
            "emlxs_issue_dhchap_challenge: 0x%x 0x%x 0x%x 0x%x",
            ndlp->nlp_DID, tran_id, node_dhc->nlp_auth_hashid,
            node_dhc->nlp_auth_dhgpid);

        pkt->pkt_comp = emlxs_cmpl_dhchap_challenge_issue;

        if (emlxs_pkt_send(pkt, 1) != FC_SUCCESS) {
                emlxs_pkt_free(pkt);

                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                    "emlxs_issue_dhchap_challenge: Unable to send fc packet.");

                return (1);
        }
        return (0);

} /* emlxs_issue_dhchap_challenge */


/*
 * DHCHAP_Reply msg
 */
/* ARGSUSED */
uint32_t
```

```
2504 emlxs_issue_dhchap_reply(
2505         emlxs_port_t *port,
2506         NODELIST *ndlp,
2507         int retry,
2508         uint32_t *arg1, /* response */
2509         uint8_t *dhval,
2510         uint32_t dhval_len,
2511         uint8_t *arg2,  /* random number */
2512         uint32_t arg2_len)
2513 {
2514         fc_packet_t *pkt;
2515         uint32_t cmd_size;
2516         uint32_t rsp_size;
2517         uint16_t cmdsize = 0;
2518         DHCHAP_REPLY_HDR *ap;
2519         uint8_t *pCmd;
2520         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

2522         /* Header size */
2523         cmdsize = sizeof (DHCHAP_REPLY_HDR);

2525         /* Rsp value len size (4) + Response value size */
2526         if (ndlp->nlp_DID == FABRIC_DID) {
2527                 if (node_dhc->hash_id == AUTH_MD5) {
2528                         cmdsize += 4 + MD5_LEN;
2529                 }
2530                 if (node_dhc->hash_id == AUTH_SHA1) {
2531                         cmdsize += 4 + SHA1_LEN;
2532                 }
2533         } else {
2534                 if (node_dhc->nlp_auth_hashid == AUTH_MD5) {
2535                         cmdsize += 4 + MD5_LEN;
2536                 }
2537                 if (node_dhc->nlp_auth_hashid == AUTH_SHA1) {
2538                         cmdsize += 4 + SHA1_LEN;
2539                 }
2540         }

2542         /* DH value len size (4) + DH value size */
2543         if (ndlp->nlp_DID == FABRIC_DID) {
2544                 switch (node_dhc->dhgp_id) {
2545                 case GROUP_NULL:

2547                         break;

2549                 case GROUP_1024:
2550                 case GROUP_1280:
2551                 case GROUP_1536:
2552                 case GROUP_2048:
2553                 default:
2554                         break;
2555                 }
2556         }

2558         cmdsize += 4 + dhval_len;

2560         /* Challenge value len size (4) + Challenge value size */
2561         if (node_dhc->auth_cfg.bidirectional == 0) {
2562                 cmdsize += 4;
2563         } else {
2564                 if (ndlp->nlp_DID == FABRIC_DID) {
2565                         cmdsize += 4 + ((node_dhc->hash_id == AUTH_MD5) ?
2566                             MD5_LEN : SHA1_LEN);
2567                 } else {
2568                         cmdsize += 4 +
2569                             ((node_dhc->nlp_auth_hashid == AUTH_MD5) ? MD5_LEN :
```



```
2570                             SHA1_LEN);
2571                 }
2572         }

2574         cmd_size = cmdsize;
2575         rsp_size = 4;

2577         if ((pkt = emlxs_prep_els_fc_pkt(port, ndlp->nlp_DID, cmd_size,
2578             rsp_size, 0, KM_NOSLEEP)) == NULL) {
2579                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2580                     "emlxs_issue_dhchap_reply failed: did=0x%x size=%x,%x",
2581                     ndlp->nlp_DID, cmd_size, rsp_size);

2583                 return (1);
2584         }
2585         pCmd = (uint8_t *)pkt->pkt_cmd;

2587         ap = (DHCHAP_REPLY_HDR *)pCmd;
2588         ap->auth_els_code = ELS_CMD_AUTH_CODE;
2589         ap->auth_els_flags = 0x0;
2590         ap->auth_msg_code = DHCHAP_REPLY;
2591         ap->proto_version = 0x01;
2592         ap->msg_len = LE_SWAP32(cmdsize - sizeof (DHCHAP_REPLY_HDR));
2593         ap->tran_id = LE_SWAP32(node_dhc->nlp_auth_tranid_rsp);

2595         pCmd = (uint8_t *)(pCmd + sizeof (DHCHAP_REPLY_HDR));

2597         if (ndlp->nlp_DID == FABRIC_DID) {
2598                 if (node_dhc->hash_id == AUTH_MD5) {
2599                         *(uint32_t *)pCmd = LE_SWAP32(MD5_LEN);
2600                 } else {
2601                         *(uint32_t *)pCmd = LE_SWAP32(SHA1_LEN);
2602                 }
2603         } else {
2604                 if (node_dhc->nlp_auth_hashid == AUTH_MD5) {
2605                         *(uint32_t *)pCmd = LE_SWAP32(MD5_LEN);
2606                 } else {
2607                         *(uint32_t *)pCmd = LE_SWAP32(SHA1_LEN);
2608                 }
2609         }

2611         pCmd = (uint8_t *)(pCmd + 4);

2613         if (ndlp->nlp_DID == FABRIC_DID) {
2614                 if (node_dhc->hash_id == AUTH_MD5) {
2615                         bcopy((void *)arg1, pCmd, MD5_LEN);
2616                         pCmd = (uint8_t *)(pCmd + MD5_LEN);
2617                 } else {
2618                         bcopy((void *)arg1, (void *)pCmd, SHA1_LEN);

2620                         pCmd = (uint8_t *)(pCmd + SHA1_LEN);
2621                 }
2622         } else {
2623                 if (node_dhc->nlp_auth_hashid == AUTH_MD5) {
2624                         bcopy((void *)arg1, pCmd, MD5_LEN);
2625                         pCmd = (uint8_t *)(pCmd + MD5_LEN);
2626                 } else {
2627                         bcopy((void *)arg1, (void *)pCmd, SHA1_LEN);
2628                         pCmd = (uint8_t *)(pCmd + SHA1_LEN);
2629                 }
2630         }

2632         *(uint32_t *)pCmd = LE_SWAP32(dhval_len);

2634         if (dhval_len != 0) {
2635                 pCmd = (uint8_t *)(pCmd + 4);
```

```
2637                switch (node_dhc->dhgp_id) {
2638                case GROUP_NULL:

2640                        break;

2642                case GROUP_1024:
2643                case GROUP_1280:
2644                case GROUP_1536:
2645                case GROUP_2048:
2646                default:
2647                        break;
2648                }
2649                /* elx_bcopy((void *)dhval, (void *)pCmd, dhval_len); */
2650                /*
2651                 * The new DH parameter (g^y mod p) is stored in
2652                 * node_dhc->pub_key
2653                 */
2654                /* pubkey_len should be equal to dhval_len */

2656                if (ndlp->nlp_DID == FABRIC_DID) {
2657                        bcopy((void *) node_dhc->pub_key, (void *)pCmd,
2658                            node_dhc->pubkey_len);
2659                } else {
2660                        bcopy((void *) node_dhc->nlp_auth_misc.pub_key,
2661                            (void *)pCmd,
2662                            node_dhc->nlp_auth_misc.pubkey_len);
2663                }
2664                pCmd = (uint8_t *)(pCmd + dhval_len);
2665        } else
2666                pCmd = (uint8_t *)(pCmd + 4);

2668        if (node_dhc->auth_cfg.bidirectional == 0) {
2669                *(uint32_t *)pCmd = 0x0;
2670        } else {
2671                if (ndlp->nlp_DID == FABRIC_DID) {
2672                        if (node_dhc->hash_id == AUTH_MD5) {
2673                                *(uint32_t *)pCmd = LE_SWAP32(MD5_LEN);
2674                                pCmd = (uint8_t *)(pCmd + 4);
2675                                bcopy((void *)arg2, (void *)pCmd, arg2_len);
2676                        } else if (node_dhc->hash_id == AUTH_SHA1) {
2677                                *(uint32_t *)pCmd = LE_SWAP32(SHA1_LEN);
2678                                pCmd = (uint8_t *)(pCmd + 4);
2679                                /* store the challenge */
2680                                bcopy((void *)arg2, (void *)pCmd, arg2_len);
2681                        }
2682                } else {
2683                        if (node_dhc->nlp_auth_hashid == AUTH_MD5) {
2684                                *(uint32_t *)pCmd = LE_SWAP32(MD5_LEN);
2685                                pCmd = (uint8_t *)(pCmd + 4);
2686                                bcopy((void *)arg2, (void *)pCmd, arg2_len);
2687                        } else if (node_dhc->nlp_auth_hashid == AUTH_SHA1) {
2688                                *(uint32_t *)pCmd = LE_SWAP32(SHA1_LEN);
2689                                pCmd = (uint8_t *)(pCmd + 4);
2690                                bcopy((void *)arg2, (void *)pCmd, arg2_len);
2691                        }
2692                }
2693        }

2695        pkt->pkt_comp = emlxs_cmpl_dhchap_reply_issue;

2697        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
2698            "emlxs_issue_dhchap_reply: did=0x%x  (%x,%x,%x,%x,%x,%x)",
2699            ndlp->nlp_DID, dhval_len, arg2_len, cmdsize,
2700            node_dhc->hash_id, node_dhc->nlp_auth_hashid,
2701            LE_SWAP32(ap->tran_id));
```



```
2703        if (emlxs_pkt_send(pkt, 1) != FC_SUCCESS) {
2704                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2705                    "emlxs_issue_dhchap_reply failed: Unable to send packet.");

2707                emlxs_pkt_free(pkt);

2709                return (1);
2710        }
2711        return (0);

2713 } /* emlxs_issue_dhchap_reply */



2717 /*
2718  * ! emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next
2719  *
2720  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
2721  * uint32_t \b Description:
2722  *
2723  * This routine is invoked when the host received an unsolicted ELS AUTH MSG
2724  * from an NxPort or FxPort which already replied (ACC)
2725  * the ELS AUTH_Negotiate msg from the host. if msg is DHCHAP_Chellenge,
2726  * based on the msg content (DHCHAP computation etc.,)
2727  * the host send back ACC and 1. send back AUTH_Reject and set next state =
2728  * NPR_NODE or 2. send back DHCHAP_Reply msg and set
2729  * next state = DHCHAP_REPLY_ISSUE for bi-directional, the DHCHAP_Reply
2730  * includes challenge from host. for uni-directional, no
2731  * more challenge. if msg is AUTH_Reject or anything else, host send back
2732  * ACC and set next state = NPR_NODE. And based on the
2733  * reject code, host may need to retry negotiate with NULL DH only
2734  *
2735  * If the msg is AUTH_ELS cmd, cancel the nlp_authrsp_timeout timer immediately.
2736  *
2737  */
2738 /* ARGSUSED */
2739 static uint32_t
2740 emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next(
2741 emlxs_port_t *port,
2742 /* CHANNEL * rp, */ void *arg1,
2743 /* IOCBQ * iocbq, */ void *arg2,
2744 /* MATCHMAP * mp, */ void *arg3,
2745 /* NODELIST * ndlp */ void *arg4,
2746 uint32_t evt)
2747 {
2748        emlxs_hba_t *hba = HBA;
2749        emlxs_port_dhc_t *port_dhc = &port->port_dhc;
2750        IOCBQ *iocbq = (IOCBQ *)arg2;
2751        MATCHMAP *mp = (MATCHMAP *)arg3;
2752        NODELIST *ndlp = (NODELIST *)arg4;
2753        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
2754        uint8_t *bp;
2755        uint32_t *lp;
2756        DHCHAP_CHALL_NULL *ncval;
2757        uint16_t namelen;
2758        uint32_t dhvallen;
2759        uint8_t *tmp;
2760        uint8_t ReasonCode;
2761        uint8_t ReasonCodeExplanation;

2763        union challenge_val un_cval;

2765        uint8_t *dhval = NULL;
2766        uint8_t random_number[20];      /* for both SHA1 and MD5 */
2767        uint32_t *arg5 = NULL;  /* response */
```

```
2768         uint32_t tran_id;       /* Transaction Identifier */
2769         uint32_t arg2len = 0;   /* len of new challenge for bidir auth */

2771         AUTH_RJT *rjt;

2773         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
2774             "emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next: did=0x%x",
2775             ndlp->nlp_DID);

2777         emlxs_dhc_state(port, ndlp, NODE_STATE_DHCHAP_REPLY_ISSUE, 0, 0);

2779         (void) emlxs_els_reply(port, iocbq, ELS_CMD_ACC, ELS_CMD_AUTH, 0, 0);

2781         bp = mp->virt;
2782         lp = (uint32_t *)bp;

2784         /*
2785          * 1. we process the DHCHAP_Challenge 2. ACC it first 3. based on the
2786          * result of 1 we DHCHAP_Reply or AUTH_Reject
2787          */
2788         ncval = (DHCHAP_CHALL_NULL *)((uint8_t *)lp);

2790         if (ncval->msg_hdr.auth_els_code != ELS_CMD_AUTH_CODE) {
2791                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2792                     "rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x %x",
2793                     ndlp->nlp_DID, ncval->msg_hdr.auth_els_code);

2795                 /* need to setup reason code/reason explanation code  */
2796                 ReasonCode = AUTHRJT_FAILURE;
2797                 ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL;
2798                 goto AUTH_Reject;
2799         }
2800         if (ncval->msg_hdr.auth_msg_code == AUTH_REJECT) {
2801                 rjt = (AUTH_RJT *)((uint8_t *)lp);
2802                 ReasonCode = rjt->ReasonCode;
2803                 ReasonCodeExplanation = rjt->ReasonCodeExplanation;

2805                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
2806                     "rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x.%x,%x",
2807                     ndlp->nlp_DID, ReasonCode, ReasonCodeExplanation);

2809                 switch (ReasonCode) {
2810                 case AUTHRJT_LOGIC_ERR:
2811                         switch (ReasonCodeExplanation) {
2812                         case AUTHEXP_MECH_UNUSABLE:
2813                         case AUTHEXP_DHGROUP_UNUSABLE:
2814                         case AUTHEXP_HASHFUNC_UNUSABLE:
2815                                 ReasonCode = AUTHRJT_LOGIC_ERR;
2816                                 ReasonCodeExplanation = AUTHEXP_RESTART_AUTH;
2817                                 break;

2819                         case AUTHEXP_RESTART_AUTH:
2820                                 /*
2821                                  * Cancel the rsp timer if not cancelled yet.
2822                                  * and restart auth tran now.
2823                                  */
2824                                 if (node_dhc->nlp_authrsp_tmo != 0) {
2825                                         node_dhc->nlp_authrsp_tmo = 0;
2826                                         node_dhc->nlp_authrsp_tmocnt = 0;
2827                                 }
2828                                 if (emlxs_dhc_auth_start(port, ndlp, NULL,
2829                                     NULL) != 0) {
2830                                         EMLXS_MSGF(EMLXS_CONTEXT,
2831                                             &emlxs_fcsp_debug_msg,
2832                                             "Reauth timeout. failed. 0x%x %x",
2833                                             ndlp->nlp_DID, node_dhc->state);
```



```
2834                                 }
2835                                 return (node_dhc->state);

2837                         default:
2838                                 ReasonCode = AUTHRJT_FAILURE;
2839                                 ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
2840                                 break;
2841                         }
2842                         break;

2844                 case AUTHRJT_FAILURE:
2845                 default:
2846                         ReasonCode = AUTHRJT_FAILURE;
2847                         ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
2848                         break;
2849                 }

2851                 goto AUTH_Reject;
2852         }
2853         if (ncval->msg_hdr.auth_msg_code != DHCHAP_CHALLENGE) {
2854                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2855                     "emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x.%x",
2856                     ndlp->nlp_DID, ncval->msg_hdr.auth_msg_code);

2858                 ReasonCode = AUTHRJT_FAILURE;
2859                 ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL;
2860                 goto AUTH_Reject;
2861         }
2862         tran_id = ncval->msg_hdr.tran_id;

2864         if (LE_SWAP32(tran_id) != node_dhc->nlp_auth_tranid_rsp) {
2865                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2866                     "rcv_auth_msg_auth_negotiate_cmpl_wait4next:0x%x %x!=%x",
2867                     ndlp->nlp_DID, LE_SWAP32(tran_id),
2868                     node_dhc->nlp_auth_tranid_rsp);

2870                 ReasonCode = AUTHRJT_FAILURE;
2871                 ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;
2872                 goto AUTH_Reject;
2873         }
2874         node_dhc->nlp_authrsp_tmo = 0;

2876         namelen = ncval->msg_hdr.name_len;

2878         if (namelen == AUTH_NAME_LEN) {
2879                 /*
2880                  * store another copy of wwn of fabric/or nport used in
2881                  * AUTH_ELS cmd
2882                  */
2883                 bcopy((void *)&ncval->msg_hdr.nodeName,
2884                     (void *)&node_dhc->nlp_auth_wwn, sizeof (NAME_TYPE));
2885         }
2886         /* Collect the challenge value */
2887         tmp = (uint8_t *)((uint8_t *)lp + sizeof (DHCHAP_CHALL_NULL));

2889         if (ncval->hash_id == AUTH_MD5) {
2890                 if (ncval->cval_len != LE_SWAP32(MD5_LEN)) {
2891                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2892                     "rcv_auth_msg_auth_negotiate_cmpl_wait4next:0x%x.%x!=%x",
2893                     ndlp->nlp_DID, ncval->cval_len, LE_SWAP32(MD5_LEN));

2895                         ReasonCode = AUTHRJT_FAILURE;
2896                         ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;
2897                         goto AUTH_Reject;
2898                 }
2899                 bzero(un_cval.md5.val, sizeof (MD5_CVAL));
```

```
2900                bcopy((void *)tmp, (void *)un_cval.md5.val,
2901                    sizeof (MD5_CVAL));
2902                tmp += sizeof (MD5_CVAL);

2904                arg2len = MD5_LEN;

2906        } else if (ncval->hash_id == AUTH_SHA1) {
2907                if (ncval->cval_len != LE_SWAP32(SHA1_LEN)) {
2908                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2909                    "rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x %x!=%x",
2910                    ndlp->nlp_DID, ncval->cval_len, LE_SWAP32(MD5_LEN));

2912                        ReasonCode = AUTHRJT_FAILURE;
2913                        ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;
2914                        goto AUTH_Reject;
2915                }
2916                bzero(un_cval.sha1.val, sizeof (SHA1_CVAL));
2917                bcopy((void *)tmp, (void *)un_cval.sha1.val,
2918                    sizeof (SHA1_CVAL));
2919                tmp += sizeof (SHA1_CVAL);

2921                arg2len = SHA1_LEN;

2923        } else {
2924        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2925            "emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x %x",
2926            ndlp->nlp_DID, ncval->hash_id);

2928                ReasonCode = AUTHRJT_FAILURE;
2929                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;
2930                goto AUTH_Reject;
2931        }

2933        /*
2934         * store hash_id for later usage : hash_id is set by responder in its
2935         * dhchap_challenge
2936         */
2937        node_dhc->hash_id = ncval->hash_id;

2939        /* always use this */
2940        /* store another copy of the hash_id */
2941        node_dhc->nlp_auth_hashid = ncval->hash_id;

2943        /* store dhgp_id for later usage */
2944        node_dhc->dhgp_id = ncval->dhgp_id;

2946        /* store another copy of dhgp_id */
2947        /* always use this */
2948        node_dhc->nlp_auth_dhgpid = ncval->dhgp_id;

2950        /*
2951         * ndlp->nlp_auth_hashid, nlp_auth_dhgpid store the hashid and dhgpid
2952         * when this very ndlp is the auth transaction responder (in other
2953         * words, responder means that this ndlp is send the host the
2954         * challenge. ndlp could be fffffe or another initiator or target
2955         * nport.
2956         */

2958        dhvallen = *((uint32_t *)(tmp));

2960        switch (ncval->dhgp_id) {
2961        case GROUP_NULL:
2962                /* null DHCHAP only */
2963                if (LE_SWAP32(dhvallen) != 0) {
2964                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2965                    "rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x %x %x",
```



```
2966                    ndlp->nlp_DID, ncval->dhgp_id, LE_SWAP32(dhvallen));

2968                        ReasonCode = AUTHRJT_FAILURE;
2969                        ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;
2970                        goto AUTH_Reject;
2971                }
2972                break;

2974        case GROUP_1024:
2975        case GROUP_1280:
2976        case GROUP_1536:
2977        case GROUP_2048:
2978                /* Collect the DH Value */
2979                tmp += sizeof (uint32_t);

2981                dhval = (uint8_t *)kmem_zalloc(LE_SWAP32(dhvallen),
2982                    KM_NOSLEEP);
2983                if (dhval == NULL) {
2984                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2985                    "rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x %x %x",
2986                    ndlp->nlp_DID, ncval->dhgp_id, dhval);

2988                        ReasonCode = AUTHRJT_LOGIC_ERR;
2989                        ReasonCodeExplanation = AUTHEXP_RESTART_AUTH;
2990                        goto AUTH_Reject;
2991                }
2992                bcopy((void *)tmp, (void *)dhval, LE_SWAP32(dhvallen));
2993                break;

2995        default:
2996                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
2997                    "rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x %x.",
2998                    ndlp->nlp_DID, ncval->dhgp_id);

3000                ReasonCode = AUTHRJT_FAILURE;
3001                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;
3002                goto AUTH_Reject;
3003        }

3005        /*
3006         * Calculate the hash value, hash function, DH group, secret etc.
3007         * could be stored in port_dhc.
3008         */

3010        /* arg5 has the response with NULL or Full DH group support */
3011        arg5 = (uint32_t *)emlxs_hash_rsp(port, port_dhc,
3012            ndlp, tran_id, un_cval, dhval, LE_SWAP32(dhvallen));

3014        /* Or should check ndlp->auth_cfg..... */
3015        if (node_dhc->auth_cfg.bidirectional == 1) {
3016                /* get arg2 here */
3017                /*
3018                 * arg2 is the new challenge C2 from initiator if bi-dir auth
3019                 * is supported
3020                 */
3021                bzero(&random_number, sizeof (random_number));

3023                if (hba->rdn_flag == 1) {
3024                        emlxs_get_random_bytes(ndlp, random_number, 20);
3025                } else {
3026                        (void) random_get_pseudo_bytes(random_number, arg2len);
3027                }

3029                /* cache it for later verification usage */
3030                if (ndlp->nlp_DID == FABRIC_DID) {
3031                        bcopy((void *)&random_number[0],
```

```
3032                            (void *)&node_dhc->bi_cval[0], arg2len);
3033                        node_dhc->bi_cval_len = arg2len;

3035                        /* save another copy in our partner's ndlp */
3036                        bcopy((void *)&random_number[0],
3037                            (void *)&node_dhc->nlp_auth_misc.bi_cval[0],
3038                            arg2len);
3039                        node_dhc->nlp_auth_misc.bi_cval_len = arg2len;
3040                } else {
3041                        bcopy((void *)&random_number[0],
3042                            (void *)&node_dhc->nlp_auth_misc.bi_cval[0],
3043                            arg2len);
3044                        node_dhc->nlp_auth_misc.bi_cval_len = arg2len;
3045                }
3046        }
3047        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3048            "rcv_auth_msg_auth_negotiate_cmpl_wait4next:0x%x(%x,%x,%x,%x,%x)",
3049            ndlp->nlp_DID, node_dhc->nlp_auth_tranid_rsp,
3050            node_dhc->nlp_auth_tranid_ini,
3051            ncval->hash_id, ncval->dhgp_id, dhvallen);

3053        /* Issue ELS DHCHAP_Reply */
3054        /*
3055         * arg1 has the response, arg2 has the new challenge if needed (g^y
3056         * mod p) is the pubkey: all are ready and to go
3057         */

3059        /* return 0 success, otherwise failure */
3060        if (emlxs_issue_dhchap_reply(port, ndlp, 0, arg5, dhval,
3061            LE_SWAP32(dhvallen),
3062            random_number, arg2len)) {
3063        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3064            "rcv_auth_msg_auth_negotiate_cmpl_wait4next: 0x%x.failed.",
3065            ndlp->nlp_DID);

3067                kmem_free(dhval, LE_SWAP32(dhvallen));
3068                ReasonCode = AUTHRJT_LOGIC_ERR;
3069                ReasonCodeExplanation = AUTHEXP_RESTART_AUTH;
3070                goto AUTH_Reject;
3071        }
3072        return (node_dhc->state);

3074 AUTH_Reject:

3076        emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED, ReasonCode,
3077            ReasonCodeExplanation);
3078        (void) emlxs_issue_auth_reject(port, ndlp, 0, 0, ReasonCode,
3079            ReasonCodeExplanation);
3080        emlxs_dhc_auth_complete(port, ndlp, 1);

3082        return (node_dhc->state);

3084 } /* emlxs_rcv_auth_msg_auth_negotiate_cmpl_wait4next */


3087 /*
3088  * This routine should be set to emlxs_disc_neverdev
3089  *
3090  */
3091 /* ARGSUSED */
3092 static uint32_t
3093 emlxs_cmpl_auth_msg_auth_negotiate_cmpl_wait4next(
3094 emlxs_port_t    *port,
3095 /* CHANNEL * rp, */ void *arg1,
3096 /* IOCBQ * iocbq, */ void *arg2,
3097 /* MATCHMAP * mp, */ void *arg3,
```



```
3098 /* NODELIST * ndlp */ void *arg4,
3099 uint32_t evt)
3100 {
3101        NODELIST *ndlp = (NODELIST *)arg4;

3103        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3104            "cmpl_auth_msg_auth_negotiate_cmpl_wait4next.0x%x. Not iplted.",
3105            ndlp->nlp_DID);

3107        return (0);
3108 } /* emlxs_cmpl_auth_msg_auth_negotiate_cmpl_wait4next() */


3111 /*
3112  * ! emlxs_rcv_auth_msg_dhchap_reply_issue
3113  *
3114  * This routine is invoked when the host received an unsolicited ELS AUTH
3115  * msg from an NxPort or FxPort into which the host has
3116  * sent an ELS DHCHAP_Reply msg. since the host is the initiator and the
3117  * AUTH transaction is in progress between host and the
3118  * NxPort or FxPort, as a result, the host will send back ACC and AUTH_Reject
3119  * and set the next state = NPR_NODE.
3120  *
3121  */
3122 /* ARGSUSED */
3123 static uint32_t
3124 emlxs_rcv_auth_msg_dhchap_reply_issue(
3125 emlxs_port_t *port,
3126 /* CHANNEL * rp, */ void *arg1,
3127 /* IOCBQ   * iocbq, */ void *arg2,
3128 /* MATCHMAP * mp, */ void *arg3,
3129 /* NODELIST * ndlp */ void *arg4,
3130 uint32_t evt)
3131 {
3132        NODELIST *ndlp = (NODELIST *)arg4;

3134        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3135            "rcv_auth_msg_dhchap_reply_issue called. 0x%x. Not implemented.",
3136            ndlp->nlp_DID);

3138        return (0);

3140 } /* emlxs_rcv_auth_msg_dhchap_reply_issue */



3144 /*
3145  * ! emlxs_cmpl_auth_msg_dhchap_reply_issue
3146  *
3147  * This routine is invoked when
3148  * the host received a solicited ACC/RJT from ELS command from an NxPort
3149  * or FxPort that already received the ELS DHCHAP_Reply
3150  * msg from the host. in case of ACC, next state = DHCHAP_REPLY_CMPL_WAIT4NEXT
3151  * in case of RJT, next state = NPR_NODE
3152  */
3153 /* ARGSUSED */
3154 static uint32_t
3155 emlxs_cmpl_auth_msg_dhchap_reply_issue(
3156 emlxs_port_t *port,
3157 /* CHANNEL  * rp, */ void *arg1,
3158 /* IOCBQ * iocbq, */ void *arg2,
3159 /* MATCHMAP * mp, */ void *arg3,
3160 /* NODELIST * ndlp */ void *arg4,
3161 uint32_t evt)
3162 {
3163        NODELIST *ndlp = (NODELIST *) arg4;
```

```
3164         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

3166         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3167             "emlxs_cmpl_auth_msg_dhchap_reply_issue: did=0x%x",
3168             ndlp->nlp_DID);

3170         /* start the emlxs_dhc_authrsp_timeout timer now */
3171         if (node_dhc->nlp_authrsp_tmo == 0) {
3172                 node_dhc->nlp_authrsp_tmo = DRV_TIME +
3173                     node_dhc->auth_cfg.authentication_timeout;
3174         }
3175         /*
3176          * The next state should be
3177          * emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next
3178          */
3179         emlxs_dhc_state(port, ndlp,
3180             NODE_STATE_DHCHAP_REPLY_CMPL_WAIT4NEXT, 0, 0);

3182         return (node_dhc->state);

3184 } /* emlxs_cmpl_auth_msg_dhchap_reply_issue */



3188 /*
3189  * ! emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next
3190  *
3191  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
3192  * uint32_t \b Description: This rountine is invoked
3193  * when the host received an unsolicited ELS AUTH Msg from the NxPort or
3194  * FxPort that already sent ACC back to the host after
3195  * receipt of DHCHAP_Reply msg. In normal case, this unsolicited msg could
3196  * be DHCHAP_Success msg.
3197  *
3198  * if msg is ELS DHCHAP_Success, based on the payload, host send back ACC and 1.
3199  * for uni-directional, and set next state =
3200  * REG_LOGIN. 2. for bi-directional,  and host do some computations
3201  * (hash etc) and send back either DHCHAP_Success Msg and set
3202  * next state = DHCHAP_SUCCESS_ISSUE_WAIT4NEXT or AUTH_Reject and set next
3203  * state = NPR_NODE. if msg is ELS AUTH_Reject, then
3204  * send back ACC and set next state = NPR_NODE if msg is anything else, then
3205  * RJT and set next state = NPR_NODE
3206  */
3207 /* ARGSUSED */
3208 static uint32_t
3209 emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next(
3210 emlxs_port_t *port,
3211 /* CHANNEL * rp, */ void *arg1,
3212 /* IOCBQ * iocbq, */ void *arg2,
3213 /* MATCHMAP * mp, */ void *arg3,
3214 /* NODELIST * ndlp */ void *arg4,
3215 uint32_t evt)
3216 {
3217         emlxs_port_dhc_t *port_dhc = &port->port_dhc;
3218         IOCBQ *iocbq = (IOCBQ *)arg2;
3219         MATCHMAP *mp = (MATCHMAP *)arg3;
3220         NODELIST *ndlp = (NODELIST *)arg4;
3221         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
3222         uint8_t *bp;
3223         uint32_t *lp;
3224         DHCHAP_SUCCESS_HDR *dh_success;
3225         uint8_t *tmp;
3226         uint8_t rsp_size;
3227         AUTH_RJT *auth_rjt;
3228         uint32_t tran_id;
3229         uint32_t *hash_val;
```



```
3230         union challenge_val un_cval;
3231         uint8_t ReasonCode;
3232         uint8_t ReasonCodeExplanation;
3233         char info[64];

3235         bp = mp->virt;
3236         lp = (uint32_t *)bp;

3238         /*
3239          * 1. we process the DHCHAP_Success or AUTH_Reject 2. ACC it first 3.
3240          * based on the result of 1 we goto the next stage SCR etc.
3241          */

3243         /* sp = (SERV_PARM *)((uint8_t *)lp + sizeof(uint32_t)); */
3244         dh_success = (DHCHAP_SUCCESS_HDR *)((uint8_t *)lp);

3246         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3247             "rcv_auth_msg_dhchap_reply_cmpl_wait4next: 0x%x 0x%x 0x%x",
3248             ndlp->nlp_DID, dh_success->auth_els_code,
3249             dh_success->auth_msg_code);

3251         node_dhc->nlp_authrsp_tmo = 0;

3253         (void) emlxs_els_reply(port, iocbq, ELS_CMD_ACC, ELS_CMD_AUTH, 0, 0);

3255         if (dh_success->auth_msg_code == AUTH_REJECT) {
3256                 /* ACC it and retry etc.  */
3257                 auth_rjt = (AUTH_RJT *) dh_success;
3258                 ReasonCode = auth_rjt->ReasonCode;
3259                 ReasonCodeExplanation = auth_rjt->ReasonCodeExplanation;

3261         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3262             "emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next: 0x%x.(%x,%x)",
3263             ndlp->nlp_DID, ReasonCode, ReasonCodeExplanation);

3265                 switch (ReasonCode) {
3266                 case AUTHRJT_LOGIC_ERR:
3267                         switch (ReasonCodeExplanation) {
3268                         case AUTHEXP_MECH_UNUSABLE:
3269                         case AUTHEXP_DHGROUP_UNUSABLE:
3270                         case AUTHEXP_HASHFUNC_UNUSABLE:
3271                                 ReasonCode = AUTHRJT_LOGIC_ERR;
3272                                 ReasonCodeExplanation = AUTHEXP_RESTART_AUTH;
3273                                 break;

3275                         case AUTHEXP_RESTART_AUTH:
3276                                 /*
3277                                  * Cancel the rsp timer if not cancelled yet.
3278                                  * and restart auth tran now.
3279                                  */
3280                                 if (node_dhc->nlp_authrsp_tmo != 0) {
3281                                         node_dhc->nlp_authrsp_tmo = 0;
3282                                         node_dhc->nlp_authrsp_tmocnt = 0;
3283                                 }
3284                                 if (emlxs_dhc_auth_start(port, ndlp,
3285                                     NULL, NULL) != 0) {
3286                                         EMLXS_MSGF(EMLXS_CONTEXT,
3287                                             &emlxs_fcsp_debug_msg,
3288                                             "Reauth timeout.failed. 0x%x %x",
3289                                             ndlp->nlp_DID, node_dhc->state);
3290                                 }
3291                                 return (node_dhc->state);

3293                         default:
3294                                 ReasonCode = AUTHRJT_FAILURE;
3295                                 ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
```

```
3296                                    break;
3297                            }
3298                            break;

3300                    case AUTHRJT_FAILURE:
3301                    default:
3302                            ReasonCode = AUTHRJT_FAILURE;
3303                            ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3304                            emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED,
3305                                ReasonCode, ReasonCodeExplanation);
3306                            goto out;
3307                    }

3309                    goto AUTH_Reject;
3310            }
3311            if (dh_success->auth_msg_code == DHCHAP_SUCCESS) {

3313                    /* Verify the tran_id */
3314                    tran_id = dh_success->tran_id;

3316            EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3317                "rcv_auth_msg_dhchap_reply_cmpl_wait4next: 0x%x 0x%x 0x%x 0x%x 0x%x",
3318                ndlp->nlp_DID, LE_SWAP32(tran_id),
3319                node_dhc->nlp_auth_tranid_rsp,
3320                node_dhc->nlp_auth_tranid_ini);

3322                    if (LE_SWAP32(tran_id) != node_dhc->nlp_auth_tranid_rsp) {
3323                    EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3324                        "rcv_auth_msg_dhchap_reply_cmpl_wait4next:0x%x %x!=%x",
3325                        ndlp->nlp_DID, LE_SWAP32(tran_id),
3326                        node_dhc->nlp_auth_tranid_rsp);

3328                            ReasonCode = AUTHRJT_FAILURE;
3329                            ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;
3330                            goto AUTH_Reject;
3331                    }
3332                    if (node_dhc->auth_cfg.bidirectional == 0) {
3333                            node_dhc->flag |=
3334                                (NLP_REMOTE_AUTH | NLP_SET_REAUTH_TIME);

3336                            emlxs_dhc_state(port, ndlp,
3337                                NODE_STATE_AUTH_SUCCESS, 0, 0);
3338                            emlxs_log_auth_event(port, ndlp, ESC_EMLXS_20,
3339                                "Host-initiated-unidir-auth-success");
3340                            emlxs_dhc_auth_complete(port, ndlp, 0);
3341                    } else {
3342                            /* bidir auth needed */
3343                            /* if (LE_SWAP32(dh_success->msg_len) > 4) { */

3345                            tmp = (uint8_t *)((uint8_t *)lp);
3346                            tmp += 8;
3347                            tran_id = *(uint32_t *)tmp;
3348                            tmp += 4;
3349                            rsp_size = *(uint32_t *)tmp;
3350                            tmp += 4;

3352                            /* tmp has the response from responder */

3354                            /*
3355                             * node_dhc->bi_cval has the bidir challenge value
3356                             * from initiator
3357                             */

3359                            if (LE_SWAP32(rsp_size) == 16) {
3360                                    bzero(un_cval.md5.val, LE_SWAP32(rsp_size));
3361                                    if (ndlp->nlp_DID == FABRIC_DID)
```



```
3362                                            bcopy((void *)node_dhc->bi_cval,
3363                                                (void *)un_cval.md5.val,
3364                                                LE_SWAP32(rsp_size));
3365                                    else
3366                                    bcopy(
3367                                        (void *)node_dhc->nlp_auth_misc.bi_cval,
3368                                        (void *)un_cval.md5.val,
3369                                        LE_SWAP32(rsp_size));

3371                            } else if (LE_SWAP32(rsp_size) == 20) {

3373                                    bzero(un_cval.sha1.val, LE_SWAP32(rsp_size));
3374                                    if (ndlp->nlp_DID == FABRIC_DID)
3375                                            bcopy((void *)node_dhc->bi_cval,
3376                                                (void *)un_cval.sha1.val,
3377                                                LE_SWAP32(rsp_size));
3378                                    else
3379                                    bcopy(
3380                                        (void *)node_dhc->nlp_auth_misc.bi_cval,
3381                                        (void *)un_cval.sha1.val,
3382                                        LE_SWAP32(rsp_size));
3383                            }
3384                            /* verify the response */
3385                            /* NULL DHCHAP works for now */
3386                            /* for DH group as well */

3388                            /*
3389                             * Cai2 = H (C2 || ((g^x mod p)^y mod p) ) = H (C2 ||
3390                             * (g^xy mod p) )
3391                             *
3392                             * R = H (Ti || Km || Cai2) R ?= R2
3393                             */
3394                            hash_val = emlxs_hash_vrf(port, port_dhc, ndlp,
3395                                tran_id, un_cval);

3397                            if (bcmp((void *)tmp, (void *)hash_val,
3398                                LE_SWAP32(rsp_size))) {
3399                                    if (hash_val != NULL) {
3400                                            /* not identical */
3401                    EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3402                        "emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next: 0x%x.failed. %x",
3403                        ndlp->nlp_DID, *(uint32_t *)hash_val);
3404                                    }
3405                                    ReasonCode = AUTHRJT_FAILURE;
3406                                    ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3407                                    goto AUTH_Reject;
3408                            }
3409                            emlxs_dhc_state(port, ndlp,
3410                                NODE_STATE_DHCHAP_SUCCESS_ISSUE_WAIT4NEXT, 0, 0);

3412                            /* send out DHCHAP_SUCCESS */
3413                            (void) emlxs_issue_dhchap_success(port, ndlp, 0, 0);
3414                    }
3415            }
3416            return (node_dhc->state);

3418    AUTH_Reject:

3420            emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED,
3421                ReasonCode, ReasonCodeExplanation);
3422            (void) emlxs_issue_auth_reject(port, ndlp, 0, 0, ReasonCode,
3423                ReasonCodeExplanation);
3424            emlxs_dhc_auth_complete(port, ndlp, 1);

3426            return (node_dhc->state);
3427    out:
```

```
3428         (void) sprintf(info,
3429             "Auth Failed: ReasonCode=0x%x, ReasonCodeExplanation=0x%x",
3430             ReasonCode, ReasonCodeExplanation);

3432         emlxs_log_auth_event(port, ndlp, ESC_EMLXS_20, info);
3433         emlxs_dhc_auth_complete(port, ndlp, 1);

3435         return (node_dhc->state);

3437 } /* emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next */



3441 /*
3442  * This routine should be set to emlxs_disc_neverdev as it shouldnot happen.
3443  *
3444  */
3445 /* ARGSUSED */
3446 static uint32_t
3447 emlxs_cmpl_auth_msg_dhchap_reply_cmpl_wait4next(
3448 emlxs_port_t *port,
3449 /* CHANNEL * rp, */ void *arg1,
3450 /* IOCBQ  * iocbq, */ void *arg2,
3451 /* MATCHMAP * mp, */ void *arg3,
3452 /* NODELIST * ndlp */ void *arg4,
3453 uint32_t evt)
3454 {
3455         NODELIST *ndlp = (NODELIST *)arg4;

3457         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3458             "cmpl_auth_msg_dhchap_reply_cmpl_wait4next. 0x%x.Not ipleted.",
3459             ndlp->nlp_DID);

3461         return (0);

3463 } /* emlxs_cmpl_auth_msg_dhchap_reply_cmpl_wait4next */



3466 /*
3467  * emlxs_rcv_auth_msg_dhchap_success_issue_wait4next
3468  *
3469  * This routine is supported
3470  * for HBA in either auth initiator mode or responder mode.
3471  *
3472  * This routine is invoked when the host as the auth responder received
3473  * an unsolicited ELS AUTH msg from the NxPort as the auth
3474  * initiator that already received the ELS DHCHAP_Success.
3475  *
3476  * If the host is the auth initiator and since the AUTH transction is
3477  * already in progress, therefore, any auth els msg should not
3478  * happen and if happened, RJT and move to NPR_NODE.
3479  *
3480  * If the host is the auth reponder, this unsolicited els auth msg should
3481  * be DHCHAP_Success for this bi-directional auth
3482  * transaction. In which case, the host should send ACC back and move state
3483  * to REG_LOGIN. If this unsolicited els auth msg is
3484  * DHCHAP_Reject, which could mean that the auth failed, then host should
3485  * send back ACC and set the next state to NPR_NODE.
3486  *
3487  */
3488 /* ARGSUSED */
3489 static uint32_t
3490 emlxs_rcv_auth_msg_dhchap_success_issue_wait4next(
3491 emlxs_port_t *port,
3492 /* CHANNEL * rp, */ void *arg1,
3493 /* IOCBQ * iocbq, */ void *arg2,
```



```
3494 /* MATCHMAP * mp, */ void *arg3,
3495 /* NODELIST * ndlp */ void *arg4,
3496 uint32_t evt)
3497 {
3498         NODELIST *ndlp = (NODELIST *) arg4;

3500         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3501             "rcv_auth_msg_dhchap_success_issue_wait4next. 0x%x. Not iplted.",
3502             ndlp->nlp_DID);

3504         return (0);

3506 } /* emlxs_rcv_auth_msg_dhchap_success_issue_wait4next */



3510 /*
3511  * ! emlxs_cmpl_auth_msg_dhchap_success_issue_wait4next
3512  *
3513  * This routine is invoked when
3514  * the host as the auth initiator received an solicited ACC/RJT from the
3515  * NxPort or FxPort that already received DHCHAP_Success
3516  * Msg the host sent before. in case of ACC, set next state = REG_LOGIN.
3517  * in case of RJT, set next state = NPR_NODE.
3518  *
3519  */
3520 /* ARGSUSED */
3521 static uint32_t
3522 emlxs_cmpl_auth_msg_dhchap_success_issue_wait4next(
3523 emlxs_port_t *port,
3524 /* CHANNEL * rp, */ void *arg1,
3525 /* IOCBQ * iocbq, */ void *arg2,
3526 /* MATCHMAP * mp, */ void *arg3,
3527 /* NODELIST * ndlp */ void *arg4,
3528 uint32_t evt)
3529 {
3530         NODELIST *ndlp = (NODELIST *)arg4;
3531         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

3533         /*
3534          * Either host is the initiator and auth or (reauth bi-direct) is
3535          * done, so start host reauth heartbeat timer now if host side reauth
3536          * heart beat never get started. Or host is the responder and the
3537          * other entity is done with its reauth heart beat with
3538          * uni-directional auth. Anyway we start host side reauth heart beat
3539          * timer now.
3540          */

3542         node_dhc->flag &= ~NLP_REMOTE_AUTH;
3543         node_dhc->flag |= NLP_SET_REAUTH_TIME;

3545         emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_SUCCESS, 0, 0);
3546         emlxs_log_auth_event(port, ndlp, ESC_EMLXS_25,
3547             "Host-initiated-bidir-auth-success");
3548         emlxs_dhc_auth_complete(port, ndlp, 0);

3550         return (node_dhc->state);

3552 } /* emlxs_cmpl_auth_msg_dhchap_success_issue_wait4next */



3555 /*
3556  * ! emlxs_cmpl_auth_msg_auth_negotiate_rcv
3557  *
3558  * This routine is invoked when
3559  * the host received the solicited ACC/RJT ELS cmd from an FxPort or an
```

```
3560  * NxPort that has received the ELS DHCHAP_Challenge.
3561  * The host is the auth responder and the auth transaction is still in
3562  * progress.
3563  *
3564  */
3565 /* ARGSUSED */
3566 static uint32_t
3567 emlxs_cmpl_auth_msg_auth_negotiate_rcv(
3568 emlxs_port_t *port,
3569 /* CHANNEL * rp, */ void *arg1,
3570 /* IOCBQ * iocbq, */ void *arg2,
3571 /* MATCHMAP * mp, */ void *arg3,
3572 /* NODELIST * ndlp */ void *arg4,
3573 uint32_t evt)
3574 {
3575         NODELIST *ndlp = (NODELIST *)arg4;

3577         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3578             "cmpl_auth_msg_auth_negotiate_rcv called. 0x%x. Not implemented.",
3579             ndlp->nlp_DID);

3581         return (0);

3583 } /* emlxs_cmpl_auth_msg_auth_negotiate_rcv */



3587 /*
3588  * ! emlxs_rcv_auth_msg_dhchap_challenge_issue
3589  *
3590  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
3591  * uint32_t \b Description: This routine should be
3592  * emlxs_disc_neverdev. The host is the auth responder and the auth
3593  * transaction is still in progress, any unsolicited els auth
3594  * msg is unexpected and should not happen in normal case.
3595  *
3596  * If DHCHAP_Reject, ACC and next state = NPR_NODE. anything else, RJT and
3597  * next state = NPR_NODE.
3598  */
3599 /* ARGSUSED */
3600 static uint32_t
3601 emlxs_rcv_auth_msg_dhchap_challenge_issue(
3602 emlxs_port_t *port,
3603 /* CHANNEL * rp, */ void *arg1,
3604 /* IOCBQ * iocbq, */ void *arg2,
3605 /* MATCHMAP * mp, */ void *arg3,
3606 /* NODELIST * ndlp */ void *arg4,
3607 uint32_t evt)
3608 {
3609         NODELIST *ndlp = (NODELIST *)arg4;

3611         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3612             "rcv_auth_msg_dhchap_challenge_issue called. 0x%x. Not iplted.",
3613             ndlp->nlp_DID);

3615         return (0);

3617 } /* emlxs_rcv_auth_msg_dhchap_challenge_issue */



3621 /*
3622  * ! emlxs_cmpl_auth_msg_dhchap_challenge_issue
3623  *
3624  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
3625  * uint32_t \b Description: This routine is invoked when
```



```
3626  * the host as the responder received the solicited response (ACC or RJT)
3627  * from initiator to the DHCHAP_Challenge msg sent from
3628  * host. In case of ACC, the next state = DHCHAP_CHALLENGE_CMPL_WAIT4NEXT
3629  * In case of RJT, the next state = NPR_NODE.
3630  *
3631  */
3632 /* ARGSUSED */
3633 static uint32_t
3634 emlxs_cmpl_auth_msg_dhchap_challenge_issue(
3635 emlxs_port_t *port,
3636 /* CHANNEL * rp, */ void *arg1,
3637 /* IOCBQ * iocbq, */ void *arg2,
3638 /* MATCHMAP * mp, */ void *arg3,
3639 /* NODELIST * ndlp */ void *arg4,
3640 uint32_t evt)
3641 {
3642         NODELIST *ndlp = (NODELIST *)arg4;
3643         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

3645         /*
3646          * The next state should be
3647          * emlxs_rcv_auth_msg_dhchap_challenge_cmpl_wait4next
3648          */
3649         emlxs_dhc_state(port, ndlp,
3650             NODE_STATE_DHCHAP_CHALLENGE_CMPL_WAIT4NEXT, 0, 0);

3652         /* Start the fc_authrsp_timeout timer */
3653         if (node_dhc->nlp_authrsp_tmo == 0) {
3654         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3655             "cmpl_auth_msg_dhchap_challenge_issue: Starting authrsp timer.");

3657                 node_dhc->nlp_authrsp_tmo = DRV_TIME +
3658                     node_dhc->auth_cfg.authentication_timeout;
3659         }
3660         return (node_dhc->state);

3662 } /* emlxs_cmpl_auth_msg_dhchap_challenge_issue */


3665 /*
3666  * ! emlxs_rcv_auth_msg_dhchap_challenge_cmpl_wait4next
3667  *
3668  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
3669  * uint32_t \b Description: This routine is invoked when
3670  * the host as the auth responder received an unsolicited auth msg from the
3671  * FxPort or NxPort that already sent ACC to the DHCH_
3672  * Challenge it received. In normal case this unsolicited auth msg should
3673  * be DHCHAP_Reply msg from the initiator.
3674  *
3675  * For DHCHAP_Reply msg, the host send back ACC and then do verification
3676  * (hash?) and send back DHCHAP_Success and next state as
3677  * DHCHAP_SUCCESS_ISSUE or DHCHAP_Reject and next state as NPR_NODE based on
3678  * the verification result.
3679  *
3680  * For bi-directional auth transaction, Reply msg should have the new
3681  * challenge value from the initiator. thus the Success msg
3682  * sent out should have the corresponding Reply from the responder.
3683  *
3684  * For uni-directional, Reply msg received does not contains the new
3685  * challenge and therefore the Success msg does not include the
3686  * Reply msg.
3687  *
3688  * For DHCHAP_Reject, send ACC and moved to the next state NPR_NODE. For
3689  * anything else, send RJT and moved to NPR_NODE.
3690  *
3691  */
```

```
3692 /* ARGSUSED */
3693 static uint32_t
3694 emlxs_rcv_auth_msg_dhchap_challenge_cmpl_wait4next(
3695 emlxs_port_t *port,
3696 /* CHANNEL * rp, */ void *arg1,
3697 /* IOCBQ * iocbq, */ void *arg2,
3698 /* MATCHMAP * mp, */ void *arg3,
3699 /* NODELIST * ndlp */ void *arg4,
3700 uint32_t evt)
3701 {
3702         emlxs_port_dhc_t *port_dhc = &port->port_dhc;
3703         IOCBQ *iocbq = (IOCBQ *)arg2;
3704         MATCHMAP *mp = (MATCHMAP *)arg3;
3705         NODELIST *ndlp = (NODELIST *)arg4;
3706         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
3707         uint8_t *bp;
3708         uint32_t *lp;
3709         DHCHAP_REPLY_HDR *dh_reply;
3710         uint8_t *tmp;
3711         uint32_t rsp_len;
3712         uint8_t rsp[20];        /* should cover SHA-1 and MD5's rsp */
3713         uint32_t dhval_len;
3714         uint8_t dhval[512];
3715         uint32_t cval_len;
3716         uint8_t cval[20];
3717         uint32_t tran_id;
3718         uint32_t *hash_val = NULL;
3719         uint8_t ReasonCode;
3720         uint8_t ReasonCodeExplanation;
3721         AUTH_RJT *rjt;

3723         /* ACC the ELS DHCHAP_Reply msg first */

3725         (void) emlxs_els_reply(port, iocbq, ELS_CMD_ACC, ELS_CMD_AUTH, 0, 0);

3727         bp = mp->virt;
3728         lp = (uint32_t *)bp;

3730         /*
3731          * send back ELS AUTH_Reject or DHCHAP_Success msg based on the
3732          * verification result. i.e., hash computation etc.
3733          */
3734         dh_reply = (DHCHAP_REPLY_HDR *)((uint8_t *)lp);
3735         tmp = (uint8_t *)((uint8_t *)lp);

3737         tran_id = dh_reply->tran_id;

3739         if (LE_SWAP32(tran_id) != node_dhc->nlp_auth_tranid_ini) {

3741         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3742             "rcv_auth_msg_dhchap_challenge_cmpl_wait4next:0x%x 0x%x 0x%x",
3743             ndlp->nlp_DID, tran_id, node_dhc->nlp_auth_tranid_ini);

3745                 ReasonCode = AUTHRJT_FAILURE;
3746                 ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL;
3747                 goto Reject;
3748         }

3750         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3751             "rcv_a_m_dhch_chll_cmpl_wait4next:0x%x 0x%x 0x%x 0x%x 0x%x",
3752             ndlp->nlp_DID, tran_id, node_dhc->nlp_auth_tranid_ini,
3753             node_dhc->nlp_auth_tranid_rsp, dh_reply->auth_msg_code);

3755         /* cancel the nlp_authrsp_timeout timer and send out Auth_Reject */
3756         if (node_dhc->nlp_authrsp_tmo) {
3757                 node_dhc->nlp_authrsp_tmo = 0;
```



```
3758         }
3759         if (dh_reply->auth_msg_code == AUTH_REJECT) {

3761                 rjt = (AUTH_RJT *)((uint8_t *)lp);
3762                 ReasonCode = rjt->ReasonCode;
3763                 ReasonCodeExplanation = rjt->ReasonCodeExplanation;

3765         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3766             "rcv_a_msg_dhch_chall_cmpl_wait4next:RJT rcved:0x%x 0x%x",
3767             ReasonCode, ReasonCodeExplanation);

3769                 switch (ReasonCode) {
3770                 case AUTHRJT_LOGIC_ERR:
3771                         switch (ReasonCodeExplanation) {
3772                         case AUTHEXP_MECH_UNUSABLE:
3773                         case AUTHEXP_DHGROUP_UNUSABLE:
3774                         case AUTHEXP_HASHFUNC_UNUSABLE:
3775                                 ReasonCode = AUTHRJT_LOGIC_ERR;
3776                                 ReasonCodeExplanation = AUTHEXP_RESTART_AUTH;
3777                                 break;

3779                         case AUTHEXP_RESTART_AUTH:
3780                                 /*
3781                                  * Cancel the rsp timer if not cancelled yet.
3782                                  * and restart auth tran now.
3783                                  */
3784                                 if (node_dhc->nlp_authrsp_tmo != 0) {
3785                                         node_dhc->nlp_authrsp_tmo = 0;
3786                                         node_dhc->nlp_authrsp_tmocnt = 0;
3787                                 }
3788                                 if (emlxs_dhc_auth_start(port, ndlp,
3789                                     NULL, NULL) != 0) {
3790                                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_debug_msg,
3791                                     "Reauth timeout.Auth initfailed. 0x%x %x",
3792                                     ndlp->nlp_DID, node_dhc->state);
3793                                 }
3794                                 return (node_dhc->state);

3796                         default:
3797                                 ReasonCode = AUTHRJT_FAILURE;
3798                                 ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3799                                 break;
3800                         }
3801                         break;

3803                 case AUTHRJT_FAILURE:
3804                 default:
3805                         ReasonCode = AUTHRJT_FAILURE;
3806                         ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3807                         break;
3808                 }

3810                 goto Reject;

3812         }
3813         if (dh_reply->auth_msg_code == DHCHAP_REPLY) {

3815                 /* We must send out DHCHAP_Success msg and wait for ACC */
3816                 /* _AND_ if bi-dir auth, we have to wait for next */

3818                 /*
3819                  * Send back DHCHAP_Success or AUTH_Reject based on the
3820                  * verification result
3821                  */
3822                 tmp += sizeof (DHCHAP_REPLY_HDR);
3823                 rsp_len = LE_SWAP32(*(uint32_t *)tmp);
```

```
3824                tmp += sizeof (uint32_t);

3826                /* collect the response data */
3827                bcopy((void *)tmp, (void *)rsp, rsp_len);

3829                tmp += rsp_len;
3830                dhval_len = LE_SWAP32(*(uint32_t *)tmp);

3832                tmp += sizeof (uint32_t);




3836                if (dhval_len != 0) {
3837                        /* collect the DH value */
3838                        bcopy((void *)tmp, (void *)dhval, dhval_len);
3839                        tmp += dhval_len;
3840                }
3841                /*
3842                 * Check to see if there is any challenge for bi-dir auth in
3843                 * the reply msg
3844                 */
3845                cval_len = LE_SWAP32(*(uint32_t *)tmp);
3846                if (cval_len != 0) {
3847                        /* collect challenge value */
3848                        tmp += sizeof (uint32_t);
3849                        bcopy((void *)tmp, (void *)cval, cval_len);

3851                        if (ndlp->nlp_DID == FABRIC_DID) {
3852                                node_dhc->nlp_auth_bidir = 1;
3853                        } else {
3854                                node_dhc->nlp_auth_bidir = 1;
3855                        }
3856                } else {
3857                        node_dhc->nlp_auth_bidir = 0;
3858                }

3860        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
3861            "rcv_a_m_dhchap_challenge_cmpl_wait4next:Reply:%x %lx %x %x %x\n",
3862            ndlp->nlp_DID, *(uint32_t *)rsp, rsp_len, dhval_len, cval_len);

3864                /* Verify the response based on the hash func, dhgp_id etc. */
3865                /*
3866                 * all the information needed are stored in
3867                 * node_dhc->hrsp_xxx or ndlp->nlp_auth_misc.
3868                 */
3869                /*
3870                 * Basically compare the rsp value with the computed hash
3871                 * value
3872                 */

3874                /* allocate hash_val first as rsp_len bytes */
3875                /*
3876                 * we set bi-cval pointer as NULL because we are using
3877                 * node_dhc->hrsp_cval[]
3878                 */
3879                hash_val = emlxs_hash_verification(port, port_dhc, ndlp,
3880                    (tran_id), dhval, (dhval_len), 1, 0);

3882                if (hash_val == NULL) {
3883                        ReasonCode = AUTHRJT_FAILURE;
3884                        ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3885                        goto Reject;
3886                }
3887                if (bcmp((void *) rsp, (void *)hash_val, rsp_len)) {
3888                        /* not identical */
3889        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
```



```
3890            "rcv_auth_msg_dhchap_challenge_cmpl_wait4next: Not authted(1).");

3892                        ReasonCode = AUTHRJT_FAILURE;
3893                        ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3894                        goto Reject;
3895                }
3896                kmem_free(hash_val, rsp_len);
3897                hash_val = NULL;

3899                /* generate the reply based on the challenge received if any */
3900                if ((cval_len) != 0) {
3901                        /*
3902                         * Cal R2 = H (Ti || Km || Ca2) Ca2 = H (C2 || ((g^y
3903                         * mod p)^x mod p) ) = H (C2 || (g^(x*y) mod p)) = H
3904                         * (C2 || seskey) Km is the password associated with
3905                         * responder. Here cval: C2 dhval: (g^y mod p)
3906                         */
3907                        hash_val = emlxs_hash_get_R2(port, port_dhc,
3908                            ndlp, (tran_id), dhval,
3909                            (dhval_len), 1, cval);

3911                        if (hash_val == NULL) {
3912                                ReasonCode = AUTHRJT_FAILURE;
3913                                ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3914                                goto Reject;
3915                        }
3916                }
3917                emlxs_dhc_state(port, ndlp,
3918                    NODE_STATE_DHCHAP_SUCCESS_ISSUE, 0, 0);

3920                if (emlxs_issue_dhchap_success(port, ndlp, 0,
3921                    (uint8_t *)hash_val)) {
3922                        ReasonCode = AUTHRJT_FAILURE;
3923                        ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
3924                        goto Reject;
3925                }
3926        }
3927        return (node_dhc->state);

3929 Reject:

3931        emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED,
3932            ReasonCode, ReasonCodeExplanation);
3933        (void) emlxs_issue_auth_reject(port, ndlp, 0, 0, ReasonCode,
3934            ReasonCodeExplanation);
3935        emlxs_dhc_auth_complete(port, ndlp, 1);

3937 out:

3939        return (node_dhc->state);

3941 } /* emlxs_rcv_auth_msg_dhchap_challenge_cmpl_wait4next */



3945 /*
3946  * This routine should be emlxs_disc_neverdev.
3947  *
3948  */
3949 /* ARGSUSED */
3950 static uint32_t
3951 emlxs_cmpl_auth_msg_dhchap_challenge_cmpl_wait4next(
3952 emlxs_port_t *port,
3953 /* CHANNEL * rp, */ void *arg1,
3954 /* IOCBQ * iocbq, */ void *arg2,
3955 /* MATCHMAP * mp, */ void *arg3,
```

```
3956 /* NODELIST * ndlp */ void *arg4,
3957 uint32_t evt)
3958 {
3959         NODELIST *ndlp = (NODELIST *)arg4;

3961         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3962             "cmpl_a_m_dhch_chall_cmpl_wait4next.0x%x. Not implemented.",
3963             ndlp->nlp_DID);

3965         return (0);

3967 } /* emlxs_cmpl_auth_msg_dhchap_challenge_cmpl_wait4next */


3970 /*
3971  * ! emlxs_rcv_auth_msg_dhchap_success_issue
3972  *
3973  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
3974  * uint32_t \b Description:
3975  *
3976  * The host is the auth responder and the auth transaction is still in
3977  * progress, any unsolicited els auth msg is unexpected and
3978  * should not happen. If DHCHAP_Reject received, ACC back and move to next
3979  * state NPR_NODE. anything else, RJT and move to
3980  * NPR_NODE.
3981  */
3982 /* ARGSUSED */
3983 static uint32_t
3984 emlxs_rcv_auth_msg_dhchap_success_issue(
3985 emlxs_port_t *port,
3986 /* CHANNEL * rp, */ void *arg1,
3987 /* IOCBQ * iocbq, */ void *arg2,
3988 /* MATCHMAP * mp, */ void *arg3,
3989 /* NODELIST * ndlp */ void *arg4,
3990 uint32_t evt)
3991 {
3992         NODELIST *ndlp = (NODELIST *)arg4;

3994         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
3995             "rcv_a_m_dhch_success_issue called. did=0x%x. Not implemented.",
3996             ndlp->nlp_DID);

3998         return (0);

4000 } /* emlxs_rcv_auth_msg_dhchap_success_issue */



4004 /*
4005  * emlxs_cmpl_auth_msg_dhchap_success_issue
4006  *
4007  * This routine is invoked when
4008  * host as the auth responder received the solicited response (ACC or RJT)
4009  * from the initiator that received DHCHAP_ Success.
4010  *
4011  * For uni-dirctional authentication, we are done so the next state =
4012  * REG_LOGIN for bi-directional authentication, we will expect
4013  * DHCHAP_Success msg. so the next state = DHCHAP_SUCCESS_CMPL_WAIT4NEXT
4014  * and start the emlxs_dhc_authrsp_timeout timer
4015  */
4016 /* ARGSUSED */
4017 static uint32_t
4018 emlxs_cmpl_auth_msg_dhchap_success_issue(
4019 emlxs_port_t *port,
4020 /* CHANNEL * rp, */ void *arg1,
4021 /* IOCBQ * iocbq, */ void *arg2,
```



```
4022 /* MATCHMAP * mp, */ void *arg3,
4023 /* NODELIST * ndlp */ void *arg4,
4024 uint32_t evt)
4025 {
4026         NODELIST *ndlp = (NODELIST *)arg4;
4027         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

4029         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4030             "cmpl_a_m_dhch_success_issue: did=0x%x auth_bidir=0x%x",
4031             ndlp->nlp_DID, node_dhc->nlp_auth_bidir);

4033         if (node_dhc->nlp_auth_bidir == 1) {
4034                 /* we would expect the bi-dir authentication result */

4036                 /*
4037                  * the next state should be
4038                  * emlxs_rcv_auth_msg_dhchap_success_cmpl_wait4next
4039                  */
4040                 emlxs_dhc_state(port, ndlp,
4041                     NODE_STATE_DHCHAP_SUCCESS_CMPL_WAIT4NEXT, 0, 0);

4043                 /* start the emlxs_dhc_authrsp_timeout timer */
4044                 node_dhc->nlp_authrsp_tmo = DRV_TIME +
4045                     node_dhc->auth_cfg.authentication_timeout;
4046         } else {
4047                 node_dhc->flag &= ~NLP_REMOTE_AUTH;

4049                 emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_SUCCESS, 0, 0);
4050                 emlxs_log_auth_event(port, ndlp, ESC_EMLXS_22,
4051                     "Node-initiated-unidir-reauth-success");
4052                 emlxs_dhc_auth_complete(port, ndlp, 0);
4053         }

4055         return (node_dhc->state);

4057 } /* emlxs_cmpl_auth_msg_dhchap_success_issue */


4060 /* ARGSUSED */
4061 static uint32_t
4062 emlxs_device_recov_unmapped_node(
4063         emlxs_port_t *port,
4064         void *arg1,
4065         void *arg2,
4066         void *arg3,
4067         void *arg4,
4068         uint32_t evt)
4069 {
4070         NODELIST *ndlp = (NODELIST *)arg4;

4072         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4073             "emlxs_device_recov_unmapped_node called. 0x%x. Not implemented.",
4074             ndlp->nlp_DID);

4076         return (0);

4078 } /* emlxs_device_recov_unmapped_node */



4082 /* ARGSUSED */
4083 static uint32_t
4084 emlxs_device_rm_npr_node(
4085         emlxs_port_t *port,
4086         void *arg1,
4087         void *arg2,
```

```
4088         void *arg3,
4089         void *arg4,
4090         uint32_t evt)
4091 {
4092         NODELIST *ndlp = (NODELIST *)arg4;

4094         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4095             "emlxs_device_rm_npr_node called. 0x%x. Not implemented.",
4096             ndlp->nlp_DID);

4098         return (0);

4100 } /* emlxs_device_rm_npr_node */


4103 /* ARGSUSED */
4104 static uint32_t
4105 emlxs_device_recov_npr_node(
4106         emlxs_port_t *port,
4107         void *arg1,
4108         void *arg2,
4109         void *arg3,
4110         void *arg4,
4111         uint32_t evt)
4112 {
4113         NODELIST *ndlp = (NODELIST *)arg4;

4115         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4116             "emlxs_device_recov_npr_node called. 0x%x. Not implemented.",
4117             ndlp->nlp_DID);

4119         return (0);

4121 } /* emlxs_device_recov_npr_node */


4124 /* ARGSUSED */
4125 static uint32_t
4126 emlxs_device_rem_auth(
4127 emlxs_port_t *port,
4128 /* CHANNEL * rp, */ void *arg1,
4129 /* IOCBQ * iocbq, */ void *arg2,
4130 /* MATCHMAP * mp, */ void *arg3,
4131 /* NODELIST * ndlp */ void *arg4,
4132 uint32_t evt)
4133 {
4134         NODELIST *ndlp = (NODELIST *)arg4;
4135         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

4137         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4138             "emlxs_device_rem_auth: 0x%x.",
4139             ndlp->nlp_DID);

4141         emlxs_dhc_state(port, ndlp, NODE_STATE_UNKNOWN, 0, 0);

4143         return (node_dhc->state);

4145 } /* emlxs_device_rem_auth */


4148 /*
4149  * This routine is invoked when linkdown event happens during authentication
4150  */
4151 /* ARGSUSED */
4152 static uint32_t
4153 emlxs_device_recov_auth(
```



```
4154 emlxs_port_t *port,
4155 /* CHANNEL * rp, */ void *arg1,
4156 /* IOCBQ * iocbq, */ void *arg2,
4157 /* MATCHMAP * mp, */ void *arg3,
4158 /* NODELIST * ndlp */ void *arg4,
4159 uint32_t evt)
4160 {
4161         NODELIST *ndlp = (NODELIST *)arg4;
4162         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

4164         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4165             "emlxs_device_recov_auth: 0x%x.",
4166             ndlp->nlp_DID);

4168         node_dhc->nlp_authrsp_tmo = 0;

4170         emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED, 0, 0);

4172         return (node_dhc->state);

4174 } /* emlxs_device_recov_auth */



4178 /*
4179  * This routine is invoked when the host as the responder sent out the
4180  * ELS DHCHAP_Success to the initiator, the initiator ACC
4181  * it. AND then the host received an unsolicited auth msg from the initiator,
4182  * this msg is supposed to be the ELS DHCHAP_Success
4183  * msg for the bi-directional authentication.
4184  *
4185  * next state should be REG_LOGIN
4186  */
4187 /* ARGSUSED */
4188 static uint32_t
4189 emlxs_rcv_auth_msg_dhchap_success_cmpl_wait4next(
4190 emlxs_port_t *port,
4191 /* CHANNEL * rp, */ void *arg1,
4192 /* IOCBQ * iocbq, */ void *arg2,
4193 /* MATCHMAP * mp, */ void *arg3,
4194 /* NODELIST * ndlp */ void *arg4,
4195 uint32_t evt)
4196 {
4197         IOCBQ *iocbq = (IOCBQ *)arg2;
4198         MATCHMAP *mp = (MATCHMAP *)arg3;
4199         NODELIST *ndlp = (NODELIST *)arg4;
4200         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
4201         uint8_t *bp;
4202         uint32_t *lp;
4203         DHCHAP_SUCCESS_HDR *dh_success;
4204         AUTH_RJT *auth_rjt;
4205         uint8_t ReasonCode;
4206         uint8_t ReasonCodeExplanation;

4208         bp = mp->virt;
4209         lp = (uint32_t *)bp;

4211         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4212             "emlxs_rcv_auth_msg_dhchap_success_cmpl_wait4next: did=0x%x",
4213             ndlp->nlp_DID);

4215         dh_success = (DHCHAP_SUCCESS_HDR *)((uint8_t *)lp);

4217         (void) emlxs_els_reply(port, iocbq, ELS_CMD_ACC, ELS_CMD_AUTH, 0, 0);

4219         if (dh_success->auth_msg_code == AUTH_REJECT) {
```

```
4220                /* ACC it and retry etc.  */
4221                auth_rjt = (AUTH_RJT *)dh_success;
4222                ReasonCode = auth_rjt->ReasonCode;
4223                ReasonCodeExplanation = auth_rjt->ReasonCodeExplanation;

4225        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4226            "rcv_a_m_dhch_success_cmpl_wait4next:REJECT rvd. 0x%x 0x%x 0x%x",
4227            ndlp->nlp_DID, ReasonCode, ReasonCodeExplanation);

4229                switch (ReasonCode) {
4230                case AUTHRJT_LOGIC_ERR:
4231                        switch (ReasonCodeExplanation) {
4232                        case AUTHEXP_MECH_UNUSABLE:
4233                        case AUTHEXP_DHGROUP_UNUSABLE:
4234                        case AUTHEXP_HASHFUNC_UNUSABLE:
4235                                ReasonCode = AUTHRJT_LOGIC_ERR;
4236                                ReasonCodeExplanation = AUTHEXP_RESTART_AUTH;
4237                                break;

4239                        case AUTHEXP_RESTART_AUTH:
4240                                /*
4241                                 * Cancel the rsp timer if not cancelled yet.
4242                                 * and restart auth tran now.
4243                                 */
4244                                if (node_dhc->nlp_authrsp_tmo != 0) {
4245                                        node_dhc->nlp_authrsp_tmo = 0;
4246                                        node_dhc->nlp_authrsp_tmocnt = 0;
4247                                }
4248                                if (emlxs_dhc_auth_start(port, ndlp,
4249                                    NULL, NULL) != 0) {
4250                                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_debug_msg,
4251                                    "Reauth timeout. Auth initfailed. 0x%x %x",
4252                                    ndlp->nlp_DID, node_dhc->state);
4253                                }
4254                                return (node_dhc->state);

4256                        default:
4257                                ReasonCode = AUTHRJT_FAILURE;
4258                                ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
4259                                break;

4261                        }
4262                        break;

4264                case AUTHRJT_FAILURE:
4265                default:
4266                        ReasonCode = AUTHRJT_FAILURE;
4267                        ReasonCodeExplanation = AUTHEXP_AUTH_FAILED;
4268                        break;

4270                }

4272                goto Reject;

4274        } else if (dh_success->auth_msg_code == DHCHAP_SUCCESS) {
4275                if (LE_SWAP32(dh_success->tran_id) !=
4276                    node_dhc->nlp_auth_tranid_ini) {
4277        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4278            "rcv_a_m_dhch_success_cmpl_wait4next: 0x%x 0x%lx, 0x%lx",
4279            ndlp->nlp_DID, dh_success->tran_id, node_dhc->nlp_auth_tranid_ini);

4281                        ReasonCode = AUTHRJT_FAILURE;
4282                        ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL;
4283                        goto Reject;
4284                }
4285                node_dhc->flag |= NLP_REMOTE_AUTH;
```



```
4287                emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_SUCCESS, 0, 0);
4288                emlxs_log_auth_event(port, ndlp, ESC_EMLXS_26,
4289                    "Node-initiated-bidir-reauth-success");
4290                emlxs_dhc_auth_complete(port, ndlp, 0);
4291        } else {
4292                ReasonCode = AUTHRJT_FAILURE;
4293                ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL;
4294                goto Reject;
4295        }

4297        return (node_dhc->state);

4299 Reject:

4301        emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED,
4302            ReasonCode, ReasonCodeExplanation);
4303        (void) emlxs_issue_auth_reject(port, ndlp, 0, 0, ReasonCode,
4304            ReasonCodeExplanation);
4305        emlxs_dhc_auth_complete(port, ndlp, 1);

4307 out:

4309        return (node_dhc->state);

4311 } /* emlxs_rcv_auth_msg_dhchap_success_cmpl_wait4next */


4314 /* ARGSUSED */
4315 static uint32_t
4316 emlxs_cmpl_auth_msg_dhchap_success_cmpl_wait4next(
4317 emlxs_port_t *port,
4318 /* CHANNEL * rp, */ void *arg1,
4319 /* IOCBQ * iocbq, */ void *arg2,
4320 /* MATCHMAP * mp, */ void *arg3,
4321 /* NODELIST * ndlp */ void *arg4,
4322 uint32_t evt)
4323 {

4325        return (0);

4327 } /* emlxs_cmpl_auth_msg_dhchap_success_cmpl_wait4next */


4330 /* ARGSUSED */
4331 static uint32_t
4332 emlxs_rcv_auth_msg_auth_negotiate_rcv(
4333 emlxs_port_t *port,
4334 /* CHANNEL * rp, */ void *arg1,
4335 /* IOCBQ * iocbq, */ void *arg2,
4336 /* MATCHMAP * mp, */ void *arg3,
4337 /* NODELIST * ndlp */ void *arg4,
4338 uint32_t evt)
4339 {
4340        NODELIST *ndlp = (NODELIST *)arg4;

4342        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4343            "rcv_a_m_auth_negotiate_rcv called. did=0x%x. Not implemented.",
4344            ndlp->nlp_DID);

4346        return (0);

4348 } /* emlxs_rcv_auth_msg_auth_negotiate_rcv */


4351 /* ARGSUSED */
```

```
4352 static uint32_t
4353 emlxs_rcv_auth_msg_npr_node(
4354 emlxs_port_t *port,
4355 /* CHANNEL  * rp, */ void *arg1,
4356 /* IOCBQ * iocbq, */ void *arg2,
4357 /* MATCHMAP * mp, */ void *arg3,
4358 /* NODELIST * ndlp */ void *arg4,
4359         uint32_t evt)
4360 {
4361         IOCBQ *iocbq = (IOCBQ *)arg2;
4362         MATCHMAP *mp = (MATCHMAP *)arg3;
4363         NODELIST *ndlp = (NODELIST *)arg4;
4364         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
4365         uint8_t *bp;

4367         uint32_t *lp;
4368         uint32_t msglen;
4369         uint8_t *tmp;

4371         AUTH_MSG_HDR *msg;

4373         uint8_t *temp;
4374         uint32_t rc, i, hs_id[2], dh_id[5];
4375                                         /* from initiator */
4376         uint32_t hash_id, dhgp_id;      /* to be used by responder */
4377         uint16_t num_hs = 0;
4378         uint16_t num_dh = 0;

4380         bp = mp->virt;
4381         lp = (uint32_t *)bp;

4383         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4384             "emlxs_rcv_auth_msg_npr_node:");

4386         /*
4387          * 1. process the auth msg, should acc first no matter what. 2.
4388          * return DHCHAP_Challenge for AUTH_Negotiate auth msg, AUTH_Reject
4389          * for anything else.
4390          */
4391         (void) emlxs_els_reply(port, iocbq, ELS_CMD_ACC, ELS_CMD_AUTH, 0, 0);

4393         msg = (AUTH_MSG_HDR *)((uint8_t *)lp);
4394         msglen = msg->msg_len;
4395         tmp = ((uint8_t *)lp);

4397         /* temp is used for error checking */
4398         temp = (uint8_t *)((uint8_t *)lp);
4399         /* Check the auth_els_code */
4400         if (((*(uint32_t *)temp) & 0xFFFFFFFF) != LE_SWAP32(0x90000B01)) {
4401                 /* ReasonCode = AUTHRJT_FAILURE; */
4402                 /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */

4404                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4405                     "emlxs_rcv_auth_msg_npr_node: payload(1)=0x%x",
4406                     (*(uint32_t *)temp));

4408                 goto AUTH_Reject;
4409         }
4410         temp += 3 * sizeof (uint32_t);
4411         /* Check name tag and name length */
4412         if (((*(uint32_t *)temp) & 0xFFFFFFFF) != LE_SWAP32(0x00010008)) {
4413                 /* ReasonCode = AUTHRJT_FAILURE; */
4414                 /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */

4416                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4417                     "emlxs_rcv_auth_msg_npr_node: payload(2)=0x%x",
```



```
4418                     (*(uint32_t *)temp));

4420                 goto AUTH_Reject;
4421         }
4422         temp += sizeof (uint32_t) + 8;
4423         /* Check proto_num */
4424         if (((*(uint32_t *)temp) & 0xFFFFFFFF) != LE_SWAP32(0x00000001)) {
4425                 /* ReasonCode = AUTHRJT_FAILURE; */
4426                 /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */

4428                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4429                     "emlxs_rcv_auth_msg_npr_node: payload(3)=0x%x",
4430                     (*(uint32_t *)temp));

4432                 goto AUTH_Reject;
4433         }
4434         temp += sizeof (uint32_t);
4435         /* Get para_len */
4436         /* para_len = LE_SWAP32(*(uint32_t *)temp); */

4438         temp += sizeof (uint32_t);
4439         /* Check proto_id */
4440         if (((*(uint32_t *)temp) & 0xFFFFFFFF) != AUTH_DHCHAP) {
4441                 /* ReasonCode = AUTHRJT_FAILURE; */
4442                 /* ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL; */

4444                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4445                     "emlxs_rcv_auth_msg_npr_node: payload(4)=0x%x",
4446                     (*(uint32_t *)temp));

4448                 goto AUTH_Reject;
4449         }
4450         temp += sizeof (uint32_t);
4451         /* Check hashlist tag */
4452         if ((LE_SWAP32(*(uint32_t *)temp) & 0xFFFF0000) >> 16 !=
4453             LE_SWAP16(HASH_LIST_TAG)) {
4454                 /* ReasonCode = AUTHRJT_FAILURE; */
4455                 /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */

4457                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4458                     "emlxs_rcv_auth_msg_npr_node: payload(5)=0x%x",
4459                     (LE_SWAP32(*(uint32_t *)temp) & 0xFFFF0000) >> 16);

4461                 goto AUTH_Reject;
4462         }
4463         /* Get num_hs  */
4464         num_hs = LE_SWAP32(*(uint32_t *)temp) & 0x0000FFFF;

4466         temp += sizeof (uint32_t);
4467         /* Check HashList_value1 */
4468         hs_id[0] = *(uint32_t *)temp;

4470         if ((hs_id[0] != AUTH_MD5) && (hs_id[0] != AUTH_SHA1)) {
4471                 /* ReasonCode = AUTHRJT_LOGIC_ERR; */
4472                 /* ReasonCodeExplanation = AUTHEXP_HASHFUNC_UNUSABLE; */

4474                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4475                     "emlxs_rcv_auth_msg_npr_node: payload(6)=0x%x",
4476                     (*(uint32_t *)temp));

4478                 goto AUTH_Reject;
4479         }
4480         if (num_hs == 1) {
4481                 hs_id[1] = 0;
4482         } else if (num_hs == 2) {
4483                 temp += sizeof (uint32_t);
```

```
4484                hs_id[1] = *(uint32_t *)temp;
4485
4486                if ((hs_id[1] != AUTH_MD5) && (hs_id[1] != AUTH_SHA1)) {
4487                        /* ReasonCode = AUTHRJT_LOGIC_ERR; */
4488                        /* ReasonCodeExplanation = AUTHEXP_HASHFUNC_UNUSABLE; */
4489
4490                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4491                            "emlxs_rcv_auth_msg_npr_node: payload(7)=0x%x",
4492                            (*(uint32_t *)temp));
4493
4494                        goto AUTH_Reject;
4495                }
4496                if (hs_id[0] == hs_id[1]) {
4497                        /* ReasonCode = AUTHRJT_FAILURE; */
4498                        /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */
4499
4500                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4501                            "emlxs_rcv_auth_msg_npr_node: payload(8)=0x%x",
4502                            (*(uint32_t *)temp));
4503
4504                        goto AUTH_Reject;
4505                }
4506        } else {
4507                /* ReasonCode = AUTHRJT_FAILURE; */
4508                /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */
4509
4510                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4511                    "emlxs_rcv_auth_msg_npr_node: payload(9)=0x%x",
4512                    (*(uint32_t *)(temp - sizeof (uint32_t))));
4513
4514                goto AUTH_Reject;
4515        }
4516
4517        /* Which hash_id should we use */
4518        if (num_hs == 1) {
4519                /*
4520                 * We always use the highest priority specified by us if we
4521                 * match initiator's , Otherwise, we use the next higher we
4522                 * both have. CR 26238
4523                 */
4524                if (node_dhc->auth_cfg.hash_priority[0] == hs_id[0]) {
4525                        hash_id = node_dhc->auth_cfg.hash_priority[0];
4526                } else if (node_dhc->auth_cfg.hash_priority[1] == hs_id[0]) {
4527                        hash_id = node_dhc->auth_cfg.hash_priority[1];
4528                } else {
4529                        /* ReasonCode = AUTHRJT_LOGIC_ERR; */
4530                        /* ReasonCodeExplanation = AUTHEXP_HASHFUNC_UNUSABLE; */
4531
4532                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4533                            "emlxs_rcv_auth_msg_npr_node: payload(10)=0x%lx",
4534                            (*(uint32_t *)temp));
4535
4536                        goto AUTH_Reject;
4537                }
4538        } else {
4539                /*
4540                 * Since the initiator specified two hashs, we always select
4541                 * our first one.
4542                 */
4543                hash_id = node_dhc->auth_cfg.hash_priority[0];
4544        }
4545
4546        temp += sizeof (uint32_t);
4547        /* Check DHgIDList_tag */
4548        if ((LE_SWAP32(*(uint32_t *)temp) & 0xFFFF0000) >> 16 !=
4549            LE_SWAP16(DHGID_LIST_TAG)) {
```



```
4550                /* ReasonCode = AUTHRJT_FAILURE; */
4551                /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */
4552
4553                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4554                    "emlxs_rcv_auth_msg_npr_node: payload(11)=0x%lx",
4555                    (*(uint32_t *)temp));
4556
4557                goto AUTH_Reject;
4558        }
4559        /* Get num_dh */
4560        num_dh = LE_SWAP32(*(uint32_t *)temp) & 0x0000FFFF;
4561
4562        if (num_dh == 0) {
4563                /* ReasonCode = AUTHRJT_FAILURE; */
4564                /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */
4565
4566                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4567                    "emlxs_rcv_auth_msg_npr_node: payload(12)=0x%lx",
4568                    (*(uint32_t *)temp));
4569
4570                goto AUTH_Reject;
4571        }
4572        for (i = 0; i < num_dh; i++) {
4573                temp += sizeof (uint32_t);
4574                /* Check DHgIDList_g0 */
4575                dh_id[i] = (*(uint32_t *)temp);
4576        }
4577
4578        rc = emlxs_check_dhgp(port, ndlp, dh_id, num_dh, &dhgp_id);
4579
4580        if (rc == 1) {
4581                /* ReasonCode = AUTHRJT_LOGIC_ERR; */
4582                /* ReasonCodeExplanation = AUTHEXP_DHGROUP_UNUSABLE; */
4583
4584                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4585                    "emlxs_rcv_auth_msg_npr_node: payload(13)=0x%lx",
4586                    (*(uint32_t *)temp));
4587
4588                goto AUTH_Reject;
4589        } else if (rc == 2) {
4590                /* ReasonCode = AUTHRJT_FAILURE; */
4591                /* ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD; */
4592
4593                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
4594                    "emlxs_rcv_auth_msg_npr_node: payload(14)=0x%lx",
4595                    (*(uint32_t *)temp));
4596
4597                goto AUTH_Reject;
4598        }
4599        /* We should update the tran_id */
4600        node_dhc->nlp_auth_tranid_ini = msg->tran_id;
4601
4602        if (msg->auth_msg_code == AUTH_NEGOTIATE) {
4603                node_dhc->nlp_auth_flag = 1;    /* ndlp is the initiator */
4604
4605                /* Send back the DHCHAP_Challenge with the proper paramaters */
4606                if (emlxs_issue_dhchap_challenge(port, ndlp, 0, tmp,
4607                    LE_SWAP32(msglen),
4608                    hash_id, dhgp_id)) {
4609                        goto AUTH_Reject;
4610                }
4611                emlxs_dhc_state(port, ndlp,
4612                    NODE_STATE_DHCHAP_CHALLENGE_ISSUE, 0, 0);
4613
4614        } else {
4615                goto AUTH_Reject;
```

```
4616         }

4618         return (node_dhc->state);

4620 AUTH_Reject:

4622         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4623             "emlxs_rcv_auth_msg_npr_node: AUTH_Reject it.");

4625         return (node_dhc->state);

4627 } /* emlxs_rcv_auth_msg_npr_node */


4630 /* ARGSUSED */
4631 static uint32_t
4632 emlxs_cmpl_auth_msg_npr_node(
4633 emlxs_port_t *port,
4634 /* CHANNEL * rp, */ void *arg1,
4635 /* IOCBQ * iocbq, */ void *arg2,
4636 /* MATCHMAP * mp, */ void *arg3,
4637 /* NODELIST * ndlp */ void *arg4,
4638 uint32_t evt)
4639 {
4640         NODELIST *ndlp = (NODELIST *)arg4;
4641         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

4643         /*
4644          * we donot cancel the nodev timeout here because we donot know if we
4645          * can get the authentication restarted from other side once we got
4646          * the new auth transaction kicked off we cancel nodev tmo
4647          * immediately.
4648          */
4649         /* we goto change the hba state back to where it used to be */
4650         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4651             "emlxs_cmpl_auth_msg_npr_node: 0x%x 0x%x prev_state=0x%x\n",
4652             ndlp->nlp_DID, node_dhc->state, node_dhc->prev_state);

4654         return (node_dhc->state);

4656 } /* emlxs_cmpl_auth_msg_npr_node */


4659 /*
4660  * ! emlxs_rcv_auth_msg_unmapped_node
4661  *
4662  * \pre \post \param   phba \param   ndlp \param   arg \param   evt \return
4663  * uint32_t
4664  *
4665  * \b Description: This routine is invoked when the host received an
4666  * unsolicited els authentication msg from the Fx_Port which is
4667  * wellknown port 0xFFFFFE in unmapped state, or from Nx_Port which is
4668  * in the unmapped state meaning that it is either a target
4669  * which there is no scsi id associated with it or it could be another
4670  * initiator. (end-to-end)
4671  *
4672  * For the Fabric F_Port (FFFFFE) we mark the port to the state in re_auth
4673  * state without disruppting the traffic. Then the fabric
4674  * will go through the authentication processes until it is done.
4675  *
4676  * most of the cases, the fabric should send us AUTH_Negotiate ELS msg. Once
4677  * host received this auth_negotiate els msg, host
4678  * should sent back ACC first and then send random challenge, plus DH value
4679  * (i.e., host's publick key)
4680  *
4681  * Host side needs to store the challenge value and public key for later
```



```
4682  * verification usage. (i.e., to verify the response from
4683  * initiator)
4684  *
4685  * If two FC_Ports start the reauthentication transaction at the same time,
4686  * one of the two authentication transactions shall be
4687  * aborted. In case of Host and Fabric the Nx_Port shall remain the
4688  * authentication initiator, while the Fx_Port shall become
4689  * the authentication responder.
4690  *
4691  */
4692 /* ARGSUSED */
4693 static uint32_t
4694 emlxs_rcv_auth_msg_unmapped_node(
4695 emlxs_port_t *port,
4696 /* CHANNEL * rp, */ void *arg1,
4697 /* IOCBQ * iocbq, */ void *arg2,
4698 /* MATCHMAP * mp, */ void *arg3,
4699 /* NODELIST * ndlp */ void *arg4,
4700         uint32_t evt)
4701 {
4702         IOCBQ *iocbq = (IOCBQ *)arg2;
4703         MATCHMAP *mp = (MATCHMAP *)arg3;
4704         NODELIST *ndlp = (NODELIST *)arg4;
4705         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
4706         uint8_t *bp;
4707         uint32_t *lp;
4708         uint32_t msglen;
4709         uint8_t *tmp;

4711         uint8_t ReasonCode;
4712         uint8_t ReasonCodeExplanation;
4713         AUTH_MSG_HDR *msg;
4714         uint8_t *temp;
4715         uint32_t rc, i, hs_id[2], dh_id[5];
4716                                         /* from initiator */
4717         uint32_t hash_id, dhgp_id;      /* to be used by responder */
4718         uint16_t num_hs = 0;
4719         uint16_t num_dh = 0;

4721         /*
4722          * 1. process the auth msg, should acc first no matter what. 2.
4723          * return DHCHAP_Challenge for AUTH_Negotiate auth msg, AUTH_Reject
4724          * for anything else.
4725          */
4726         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4727             "emlxs_rcv_auth_msg_unmapped_node: Sending ACC: did=0x%x",
4728             ndlp->nlp_DID);

4730         (void) emlxs_els_reply(port, iocbq, ELS_CMD_ACC, ELS_CMD_AUTH, 0, 0);

4732         bp = mp->virt;
4733         lp = (uint32_t *)bp;

4735         msg = (AUTH_MSG_HDR *)((uint8_t *)lp);
4736         msglen = msg->msg_len;

4738         tmp = ((uint8_t *)lp);

4740         /* temp is used for error checking */
4741         temp = (uint8_t *)((uint8_t *)lp);
4742         /* Check the auth_els_code */
4743         if (((*(uint32_t *)temp) & 0xFFFFFFFF) != LE_SWAP32(0x90000B01)) {
4744                 ReasonCode = AUTHRJT_FAILURE;
4745                 ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4747                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
```

```
4748                    "emlxs_rcv_auth_msg_unmapped_node: payload(1)=0x%x",
4749                    (*(uint32_t *)temp));

4751                goto AUTH_Reject;
4752        }
4753        temp += 3 * sizeof (uint32_t);
4754        /* Check name tag and name length */
4755        if (((*(uint32_t *)temp) & 0xFFFFFFFF) != LE_SWAP32(0x00010008)) {
4756                ReasonCode = AUTHRJT_FAILURE;
4757                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4759                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4760                    "emlxs_rcv_auth_msg_unmapped_node: payload(2)=0x%x",
4761                    (*(uint32_t *)temp));

4763                goto AUTH_Reject;
4764        }
4765        temp += sizeof (uint32_t) + 8;
4766        /* Check proto_num */
4767        if (((*(uint32_t *)temp) & 0xFFFFFFFF) != LE_SWAP32(0x00000001)) {
4768                ReasonCode = AUTHRJT_FAILURE;
4769                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4771                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4772                    "emlxs_rcv_auth_msg_unmapped_node: payload(3)=0x%x",
4773                    (*(uint32_t *)temp));

4775                goto AUTH_Reject;
4776        }
4777        temp += sizeof (uint32_t);

4779        /* Get para_len */
4780        /* para_len = *(uint32_t *)temp; */
4781        temp += sizeof (uint32_t);

4783        /* Check proto_id */
4784        if (((*(uint32_t *)temp) & 0xFFFFFFFF) != AUTH_DHCHAP) {
4785                ReasonCode = AUTHRJT_FAILURE;
4786                ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL;

4788                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4789                    "emlxs_rcv_auth_msg_unmapped_node: payload(4)=0x%x",
4790                    (*(uint32_t *)temp));

4792                goto AUTH_Reject;
4793        }
4794        temp += sizeof (uint32_t);
4795        /* Check hashlist tag */
4796        if ((LE_SWAP32(*(uint32_t *)temp) & 0xFFFF0000) >> 16 !=
4797            LE_SWAP16(HASH_LIST_TAG)) {
4798                ReasonCode = AUTHRJT_FAILURE;
4799                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4801                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4802                    "emlxs_rcv_auth_msg_unmapped_node: payload(5)=0x%x",
4803                    (LE_SWAP32(*(uint32_t *)temp) & 0xFFFF0000) >> 16);

4805                goto AUTH_Reject;
4806        }
4807        /* Get num_hs  */
4808        num_hs = LE_SWAP32(*(uint32_t *)temp) & 0x0000FFFF;

4810        temp += sizeof (uint32_t);
4811        /* Check HashList_value1 */
4812        hs_id[0] = *(uint32_t *)temp;
```



```
4814        if ((hs_id[0] != AUTH_MD5) && (hs_id[0] != AUTH_SHA1)) {
4815                ReasonCode = AUTHRJT_LOGIC_ERR;
4816                ReasonCodeExplanation = AUTHEXP_HASHFUNC_UNUSABLE;

4818                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4819                    "emlxs_rcv_auth_msg_unmapped_node: payload(6)=0x%x",
4820                    (*(uint32_t *)temp));

4822                goto AUTH_Reject;
4823        }
4824        if (num_hs == 1) {
4825                hs_id[1] = 0;
4826        } else if (num_hs == 2) {
4827                temp += sizeof (uint32_t);
4828                hs_id[1] = *(uint32_t *)temp;

4830                if ((hs_id[1] != AUTH_MD5) && (hs_id[1] != AUTH_SHA1)) {
4831                        ReasonCode = AUTHRJT_LOGIC_ERR;
4832                        ReasonCodeExplanation = AUTHEXP_HASHFUNC_UNUSABLE;

4834                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4835                            "emlxs_rcv_auth_msg_unmapped_node: payload(7)=0x%x",
4836                            (*(uint32_t *)temp));

4838                        goto AUTH_Reject;
4839                }
4840                if (hs_id[0] == hs_id[1]) {
4841                        ReasonCode = AUTHRJT_FAILURE;
4842                        ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4844                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4845                            "emlxs_rcv_auth_msg_unmapped_node: payload(8)=0x%x",
4846                            (*(uint32_t *)temp));

4848                        goto AUTH_Reject;
4849                }
4850        } else {
4851                ReasonCode = AUTHRJT_FAILURE;
4852                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4854                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4855                    "emlxs_rcv_auth_msg_unmapped_node: payload(9)=0x%x",
4856                    (*(uint32_t *)(temp - sizeof (uint32_t))));

4858                goto AUTH_Reject;
4859        }

4861        /* Which hash_id should we use */
4862        if (num_hs == 1) {
4863                /*
4864                 * We always use the highest priority specified by us if we
4865                 * match initiator's , Otherwise, we use the next higher we
4866                 * both have. CR 26238
4867                 */
4868                if (node_dhc->auth_cfg.hash_priority[0] == hs_id[0]) {
4869                        hash_id = node_dhc->auth_cfg.hash_priority[0];
4870                } else if (node_dhc->auth_cfg.hash_priority[1] == hs_id[0]) {
4871                        hash_id = node_dhc->auth_cfg.hash_priority[1];
4872                } else {
4873                        ReasonCode = AUTHRJT_LOGIC_ERR;
4874                        ReasonCodeExplanation = AUTHEXP_HASHFUNC_UNUSABLE;

4876                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4877                            "emlxs_rcv_auth_msg_unmapped_node: pload(10)=0x%x",
4878                            (*(uint32_t *)temp));
```

```
4880                        goto AUTH_Reject;
4881                }
4882        } else {
4883                /*
4884                 * Since the initiator specified two hashs, we always select
4885                 * our first one.
4886                 */
4887                hash_id = node_dhc->auth_cfg.hash_priority[0];
4888        }

4890        temp += sizeof (uint32_t);
4891        /* Check DHgIDList_tag */
4892        if ((LE_SWAP32(*(uint32_t *)temp) & 0xFFFF0000) >> 16 !=
4893            LE_SWAP16(DHGID_LIST_TAG)) {
4894                ReasonCode = AUTHRJT_FAILURE;
4895                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4897                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4898                    "emlxs_rcv_auth_msg_unmapped_node: payload(11)=0x%x",
4899                    (*(uint32_t *)temp));

4901                goto AUTH_Reject;
4902        }
4903        /* Get num_dh */
4904        num_dh = LE_SWAP32(*(uint32_t *)temp) & 0x0000FFFF;

4906        if (num_dh == 0) {
4907                ReasonCode = AUTHRJT_FAILURE;
4908                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4910                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4911                    "emlxs_rcv_auth_msg_unmapped_node: payload(12)=0x%x",
4912                    (*(uint32_t *)temp));

4914                goto AUTH_Reject;
4915        }
4916        for (i = 0; i < num_dh; i++) {
4917                temp += sizeof (uint32_t);
4918                /* Check DHgIDList_g0 */
4919                dh_id[i] = (*(uint32_t *)temp);
4920        }

4922        rc = emlxs_check_dhgp(port, ndlp, dh_id, num_dh, &dhgp_id);

4924        if (rc == 1) {
4925                ReasonCode = AUTHRJT_LOGIC_ERR;
4926                ReasonCodeExplanation = AUTHEXP_DHGROUP_UNUSABLE;

4928                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4929                    "emlxs_rcv_auth_msg_unmapped_node: payload(13)=0x%x",
4930                    (*(uint32_t *)temp));

4932                goto AUTH_Reject;
4933        } else if (rc == 2) {
4934                ReasonCode = AUTHRJT_FAILURE;
4935                ReasonCodeExplanation = AUTHEXP_BAD_PAYLOAD;

4937                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4938                    "emlxs_rcv_auth_msg_unmapped_node: payload(14)=0x%x",
4939                    (*(uint32_t *)temp));

4941                goto AUTH_Reject;
4942        }
4943        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4944            "emlxs_rcv_auth_msg_unmapped_node: 0x%x 0x%x 0x%x 0x%x 0x%x 0x%x",
4945            hash_id, dhgp_id, msg->auth_msg_code, msglen, msg->tran_id);
```



```
4947        /*
4948         * since ndlp is the initiator, tran_id is store in
4949         * nlp_auth_tranid_ini
4950         */
4951        node_dhc->nlp_auth_tranid_ini = LE_SWAP32(msg->tran_id);

4953        if (msg->auth_msg_code == AUTH_NEGOTIATE) {

4955                /*
4956                 * at this point, we know for sure we received the
4957                 * auth-negotiate msg from another entity, so cancel the
4958                 * auth-rsp timeout timer if we are expecting it. should
4959                 * never happen?
4960                 */
4961                node_dhc->nlp_auth_flag = 1;

4963                if (node_dhc->nlp_authrsp_tmo) {
4964                        node_dhc->nlp_authrsp_tmo = 0;
4965                }
4966                /*
4967                 * If at this point, the host is doing reauthentication
4968                 * (reauth heart beat) to this ndlp, then Host should remain
4969                 * as the auth initiator, host should reply to the received
4970                 * AUTH_Negotiate message with an AUTH_Reject message with
4971                 * Reason Code 'Logical Error' and Reason Code Explanation
4972                 * 'Authentication Transaction Already Started'.
4973                 */
4974                if (node_dhc->nlp_reauth_status ==
4975                    NLP_HOST_REAUTH_IN_PROGRESS) {
4976                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
4977                    "emlxs_rcv_auth_msg_unmapped_node: Ht reauth inprgress.");

4979                        ReasonCode = AUTHRJT_LOGIC_ERR;
4980                        ReasonCodeExplanation = AUTHEXP_AUTHTRAN_STARTED;

4982                        goto AUTH_Reject;
4983                }
4984                /* Send back the DHCHAP_Challenge with the proper paramaters */
4985                if (emlxs_issue_dhchap_challenge(port, ndlp, 0, tmp,
4986                    LE_SWAP32(msglen),
4987                    hash_id, dhgp_id)) {

4989                        goto AUTH_Reject;
4990                }
4991                /* setup the proper state */
4992                emlxs_dhc_state(port, ndlp,
4993                    NODE_STATE_DHCHAP_CHALLENGE_ISSUE, 0, 0);

4995        } else {
4996                ReasonCode = AUTHRJT_FAILURE;
4997                ReasonCodeExplanation = AUTHEXP_BAD_PROTOCOL;

4999                goto AUTH_Reject;
5000        }

5002        return (node_dhc->state);

5004 AUTH_Reject:

5006        emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED,
5007            ReasonCode, ReasonCodeExplanation);
5008        (void) emlxs_issue_auth_reject(port, ndlp, 0, 0, ReasonCode,
5009            ReasonCodeExplanation);
5010        emlxs_dhc_auth_complete(port, ndlp, 1);
```

```
5012         return (node_dhc->state);

5014 } /* emlxs_rcv_auth_msg_unmapped_node */




5019 /*
5020  * emlxs_hash_vrf for verification only the host is the initiator in
5021  * the routine.
5022  */
5023 /* ARGSUSED */
5024 static uint32_t *
5025 emlxs_hash_vrf(
5026         emlxs_port_t *port,
5027         emlxs_port_dhc_t *port_dhc,
5028         NODELIST *ndlp,
5029         uint32_t tran_id,
5030         union challenge_val un_cval)
5031 {
5032         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
5033         uint32_t dhgp_id;
5034         uint32_t hash_id;
5035         uint32_t *hash_val;
5036         uint32_t hash_size;
5037         MD5_CTX mdctx;
5038         SHA1_CTX sha1ctx;
5039         uint8_t sha1_digest[20];
5040         uint8_t md5_digest[16];
5041         uint8_t mytran_id = 0x00;

5043         char *remote_key;

5045         tran_id = (AUTH_TRAN_ID_MASK & tran_id);
5046         mytran_id = (uint8_t)(LE_SWAP32(tran_id));


5049         if (ndlp->nlp_DID == FABRIC_DID) {
5050                 remote_key = (char *)node_dhc->auth_key.remote_password;
5051                 hash_id = node_dhc->hash_id;
5052                 dhgp_id = node_dhc->dhgp_id;
5053         } else {
5054                 remote_key = (char *)node_dhc->auth_key.remote_password;
5055                 hash_id = node_dhc->nlp_auth_hashid;
5056                 dhgp_id = node_dhc->nlp_auth_dhgpid;
5057         }

5059         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
5060             "emlxs_hash_vrf: 0x%x 0x%x 0x%x tran_id=0x%x",
5061             ndlp->nlp_DID, hash_id, dhgp_id, mytran_id);

5063         if (dhgp_id == 0) {
5064                 /* NULL DHCHAP */
5065                 if (hash_id == AUTH_MD5) {
5066                         bzero(&mdctx, sizeof (MD5_CTX));

5068                         hash_size = MD5_LEN;

5070                         MD5Init(&mdctx);

5072                         /* Transaction Identifier T */
5073                         MD5Update(&mdctx, (unsigned char *) &mytran_id, 1);

5075                         MD5Update(&mdctx, (unsigned char *) remote_key,
5076                             node_dhc->auth_key.remote_password_length);
```



```
5078                         /* Augmented challenge: NULL DHCHAP i.e., Challenge */
5079                         MD5Update(&mdctx,
5080                             (unsigned char *)&(un_cval.md5.val[0]), MD5_LEN);

5082                         MD5Final((uint8_t *)md5_digest, &mdctx);

5084                         hash_val = (uint32_t *)kmem_alloc(hash_size,
5085                             KM_NOSLEEP);
5086                         if (hash_val == NULL) {
5087                                 return (NULL);
5088                         } else {
5089                                 bcopy((void *)&md5_digest,
5090                                     (void *)hash_val, MD5_LEN);
5091                         }
5092                         /*
5093                          * emlxs_md5_digest_to_hex((uint8_t *)hash_val,
5094                          * output);
5095                          */
5096                 }
5097                 if (hash_id == AUTH_SHA1) {
5098                         bzero(&sha1ctx, sizeof (SHA1_CTX));
5099                         hash_size = SHA1_LEN;
5100                         SHA1Init(&sha1ctx);

5102                         SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

5104                         SHA1Update(&sha1ctx, (void *)remote_key,
5105                             node_dhc->auth_key.remote_password_length);

5107                         SHA1Update(&sha1ctx,
5108                             (void *)&(un_cval.sha1.val[0]), SHA1_LEN);

5110                         SHA1Final((void *)sha1_digest, &sha1ctx);

5112                         /*
5113                          * emlxs_sha1_digest_to_hex((uint8_t *)hash_val,
5114                          * output);
5115                          */

5117                         hash_val = (uint32_t *)kmem_alloc(hash_size,
5118                             KM_NOSLEEP);
5119                         if (hash_val == NULL) {
5120                                 return (NULL);
5121                         } else {
5122                                 bcopy((void *)&sha1_digest,
5123                                     (void *)hash_val, SHA1_LEN);
5124                         }
5125                 }
5126                 return ((uint32_t *)hash_val);
5127         } else {
5128                 /* Verification of bi-dir auth for DH-CHAP group */
5129                 /* original challenge is node_dhc->bi_cval[] */
5130                 /* session key is node_dhc->ses_key[] */
5131                 /* That's IT */
5132                 /*
5133                  * H(bi_cval || ses_key) = C H(Ti || Km || C)  = hash_val
5134                  */
5135                 if (hash_id == AUTH_MD5) {
5136                         bzero(&mdctx, sizeof (MD5_CTX));
5137                         hash_size = MD5_LEN;

5139                         MD5Init(&mdctx);

5141                         MD5Update(&mdctx,
5142                             (void *)&(un_cval.md5.val[0]), MD5_LEN);
```

```
5144                        if (ndlp->nlp_DID == FABRIC_DID) {
5145                                MD5Update(&mdctx,
5146                                    (void *)&node_dhc->ses_key[0],
5147                                    node_dhc->seskey_len);
5148                        } else {
5149                                /* ses_key is obtained in emlxs_hash_rsp */
5150                                MD5Update(&mdctx,
5151                                    (void *)&node_dhc->nlp_auth_misc.ses_key[0],
5152                                    node_dhc->nlp_auth_misc.seskey_len);
5153                        }

5155                        MD5Final((void *)md5_digest, &mdctx);

5157                        MD5Init(&mdctx);

5159                        MD5Update(&mdctx, (void *)&mytran_id, 1);

5161                        MD5Update(&mdctx, (void *)remote_key,
5162                            node_dhc->auth_key.remote_password_length);

5164                        MD5Update(&mdctx, (void *)md5_digest, MD5_LEN);

5166                        MD5Final((void *)md5_digest, &mdctx);

5168                        hash_val = (uint32_t *)kmem_alloc(hash_size,
5169                            KM_NOSLEEP);
5170                        if (hash_val == NULL) {
5171                                return (NULL);
5172                        } else {
5173                                bcopy((void *)&md5_digest,
5174                                    (void *)hash_val, MD5_LEN);
5175                        }
5176                }
5177                if (hash_id == AUTH_SHA1) {
5178                        bzero(&sha1ctx, sizeof (SHA1_CTX));
5179                        hash_size = SHA1_LEN;

5181                        SHA1Init(&sha1ctx);

5183                        SHA1Update(&sha1ctx,
5184                            (void *)&(un_cval.sha1.val[0]), SHA1_LEN);

5186                        if (ndlp->nlp_DID == FABRIC_DID) {
5187                                SHA1Update(&sha1ctx,
5188                                    (void *)&node_dhc->ses_key[0],
5189                                    node_dhc->seskey_len);
5190                        } else {
5191                                /* ses_key was obtained in emlxs_hash_rsp */
5192                                SHA1Update(&sha1ctx,
5193                                    (void *)&node_dhc->nlp_auth_misc.ses_key[0],
5194                                    node_dhc->nlp_auth_misc.seskey_len);
5195                        }

5197                        SHA1Final((void *)sha1_digest, &sha1ctx);

5199                        SHA1Init(&sha1ctx);

5201                        SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

5203                        SHA1Update(&sha1ctx, (void *)remote_key,
5204                            node_dhc->auth_key.remote_password_length);

5206                        SHA1Update(&sha1ctx, (void *)sha1_digest, SHA1_LEN);

5208                        SHA1Final((void *)sha1_digest, &sha1ctx);
```



```
5210                        hash_val = (uint32_t *)kmem_alloc(hash_size,
5211                            KM_NOSLEEP);
5212                        if (hash_val == NULL) {
5213                                return (NULL);
5214                        } else {
5215                                bcopy((void *)&sha1_digest,
5216                                    (void *)hash_val, SHA1_LEN);
5217                        }
5218                }
5219                return ((uint32_t *)hash_val);
5220        }

5222 } /* emlxs_hash_vrf */


5225 /*
5226  * If dhval == NULL, NULL DHCHAP else, DHCHAP group.
5227  *
5228  * This routine is used by the auth transaction initiator (Who does the
5229  * auth-negotiate) to calculate the R1 (response) based on
5230  * the dh value it received, its own random private key, the challenge it
5231  * received, and Transaction id, as well as the password
5232  * associated with this very initiator in the auth pair.
5233  */
5234 uint32_t *
5235 emlxs_hash_rsp(
5236 emlxs_port_t *port,
5237 emlxs_port_dhc_t *port_dhc,
5238 NODELIST *ndlp,
5239 uint32_t tran_id,
5240 union challenge_val un_cval,
5241 uint8_t *dhval,
5242 uint32_t dhvallen)
5243 {
5244        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
5245        uint32_t dhgp_id;
5246        uint32_t hash_id;
5247        uint32_t *hash_val;
5248        uint32_t hash_size;
5249        MD5_CTX mdctx;
5250        SHA1_CTX sha1ctx;
5251        uint8_t sha1_digest[20];
5252        uint8_t md5_digest[16];
5253        uint8_t Cai[20];
5254        uint8_t mytran_id = 0x00;
5255        char *mykey;
5256        BIG_ERR_CODE err = BIG_OK;

5258        if (ndlp->nlp_DID == FABRIC_DID) {
5259                hash_id = node_dhc->hash_id;
5260                dhgp_id = node_dhc->dhgp_id;
5261        } else {
5262                hash_id = node_dhc->nlp_auth_hashid;
5263                dhgp_id = node_dhc->nlp_auth_dhgpid;
5264        }

5266        tran_id = (AUTH_TRAN_ID_MASK & tran_id);
5267        mytran_id = (uint8_t)(LE_SWAP32(tran_id));

5269        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
5270            "emlxs_hash_rsp: 0x%x 0x%x 0x%x 0x%x dhvallen=0x%x",
5271            ndlp->nlp_DID, hash_id, dhgp_id, mytran_id, dhvallen);

5273        if (ndlp->nlp_DID == FABRIC_DID) {
5274                mykey = (char *)node_dhc->auth_key.local_password;
```

```
5276         } else {
5277                 mykey = (char *)node_dhc->auth_key.local_password;
5278         }

5280         if (dhval == NULL) {
5281                 /* NULL DHCHAP */
5282                 if (hash_id == AUTH_MD5) {
5283                         bzero(&mdctx, sizeof (MD5_CTX));
5284                         hash_size = MD5_LEN;

5286                         MD5Init(&mdctx);

5288                         MD5Update(&mdctx, (unsigned char *)&mytran_id, 1);

5290                         MD5Update(&mdctx, (unsigned char *)mykey,
5291                             node_dhc->auth_key.local_password_length);

5293                         MD5Update(&mdctx,
5294                             (unsigned char *)&(un_cval.md5.val[0]),
5295                             MD5_LEN);

5297                         MD5Final((uint8_t *)md5_digest, &mdctx);

5299                         hash_val = (uint32_t *)kmem_alloc(hash_size,
5300                             KM_NOSLEEP);
5301                         if (hash_val == NULL) {
5302                                 return (NULL);
5303                         } else {
5304                                 bcopy((void *)&md5_digest,
5305                                     (void *)hash_val, MD5_LEN);
5306                         }

5308                         /*
5309                          * emlxs_md5_digest_to_hex((uint8_t *)hash_val,
5310                          * output);
5311                          */

5313                 }
5314                 if (hash_id == AUTH_SHA1) {
5315                         bzero(&sha1ctx, sizeof (SHA1_CTX));
5316                         hash_size = SHA1_LEN;
5317                         SHA1Init(&sha1ctx);

5319                         SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

5321                         SHA1Update(&sha1ctx, (void *)mykey,
5322                             node_dhc->auth_key.local_password_length);

5324                         SHA1Update(&sha1ctx,
5325                             (void *)&(un_cval.sha1.val[0]), SHA1_LEN);

5327                         SHA1Final((void *)sha1_digest, &sha1ctx);

5329                         /*
5330                          * emlxs_sha1_digest_to_hex((uint8_t *)hash_val,
5331                          * output);
5332                          */

5334                         hash_val = (uint32_t *)kmem_alloc(hash_size,
5335                             KM_NOSLEEP);
5336                         if (hash_val == NULL) {
5337                                 return (NULL);
5338                         } else {
5339                                 bcopy((void *)&sha1_digest,
5340                                     (void *)hash_val, SHA1_LEN);
5341                         }
```



```
5342                 }
5343                 return ((uint32_t *)hash_val);
5344         } else {

5346                 /* process DH grops */
5347                 /*
5348                  * calculate interm hash value Ca1 Ca1 = H(C1 || (g^x mod
5349                  * p)^y mod p) in which C1 is the challenge received. g^x mod
5350                  * p is the dhval received y is the random number in 16 bytes
5351                  * for MD5, 20 bytes for SHA1 p is hardcoded value based on
5352                  * different DH groups.
5353                  *
5354                  * To calculate hash value R1 R1 = H (Ti || Kn || Cai) in which
5355                  * Ti is the transaction identifier Kn is the shared secret.
5356                  * Cai is the result from interm hash.
5357                  *
5358                  * g^y mod p is reserved in port_dhc as pubkey (public key).for
5359                  * bi-dir challenge is another random number. y is prikey
5360                  * (private key). ((g^x mod p)^y mod p) is sekey (session
5361                  * key)
5362                  */
5363                 err = emlxs_interm_hash(port, port_dhc, ndlp,
5364                     (void *)&Cai, tran_id,
5365                     un_cval, dhval, &dhvallen);

5367                 if (err != BIG_OK) {
5368                         return (NULL);
5369                 }
5370                 if (hash_id == AUTH_MD5) {
5371                         bzero(&mdctx, sizeof (MD5_CTX));
5372                         hash_size = MD5_LEN;

5374                         MD5Init(&mdctx);

5376                         MD5Update(&mdctx, (unsigned char *)&mytran_id, 1);

5378                         MD5Update(&mdctx, (unsigned char *)mykey,
5379                             node_dhc->auth_key.local_password_length);

5381                         MD5Update(&mdctx, (unsigned char *)Cai, MD5_LEN);

5383                         MD5Final((uint8_t *)md5_digest, &mdctx);

5385                         hash_val = (uint32_t *)kmem_alloc(hash_size,
5386                             KM_NOSLEEP);
5387                         if (hash_val == NULL) {
5388                                 return (NULL);
5389                         } else {
5390                                 bcopy((void *)&md5_digest,
5391                                     (void *)hash_val, MD5_LEN);
5392                         }
5393                 }
5394                 if (hash_id == AUTH_SHA1) {
5395                         bzero(&sha1ctx, sizeof (SHA1_CTX));
5396                         hash_size = SHA1_LEN;

5398                         SHA1Init(&sha1ctx);

5400                         SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

5402                         SHA1Update(&sha1ctx, (void *)mykey,
5403                             node_dhc->auth_key.local_password_length);

5405                         SHA1Update(&sha1ctx, (void *)&Cai[0], SHA1_LEN);

5407                         SHA1Final((void *)sha1_digest, &sha1ctx);
```

```
5409                        hash_val = (uint32_t *)kmem_alloc(hash_size,
5410                            KM_NOSLEEP);
5411                        if (hash_val == NULL) {
5412                                return (NULL);
5413                        } else {
5414                                bcopy((void *)&sha1_digest,
5415                                    (void *)hash_val, SHA1_LEN);
5416                        }
5417                }
5418                return ((uint32_t *)hash_val);
5419        }

5421 } /* emlxs_hash_rsp */


5424 /*
5425  * To get the augmented challenge Cai Stored in hash_val
5426  *
5427  * Cai = Hash (C1 || ((g^x mod p)^y mod p)) = Hash (C1 || (g^(x*y) mod p)
5428  *
5429  * C1:challenge received from the remote entity (g^x mod p): dh val
5430  * received from the remote entity (remote entity's pubkey) y:
5431  * random private key from the local entity Hash: hash function used in
5432  * agreement. (g^(x*y) mod p): shared session key (aka
5433  * shared secret) (g^y mod p): local entity's pubkey
5434  */
5435 /* ARGSUSED */
5436 BIG_ERR_CODE
5437 emlxs_interm_hash(
5438 emlxs_port_t *port,
5439 emlxs_port_dhc_t *port_dhc,
5440 NODELIST *ndlp,
5441 void *hash_val,
5442 uint32_t tran_id,
5443 union challenge_val un_cval,
5444 uint8_t *dhval,
5445 uint32_t *dhvallen)
5446 {
5447         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
5448         uint32_t dhgp_id;
5449         uint32_t hash_id;
5450         MD5_CTX mdctx;
5451         SHA1_CTX sha1ctx;
5452         uint8_t sha1_digest[20];
5453         uint8_t md5_digest[16];
5454         uint32_t hash_size;
5455         BIG_ERR_CODE err = BIG_OK;

5457         if (ndlp->nlp_DID == FABRIC_DID) {
5458                 hash_id = node_dhc->hash_id;
5459                 dhgp_id = node_dhc->dhgp_id;
5460         } else {
5461                 hash_id = node_dhc->nlp_auth_hashid;
5462                 dhgp_id = node_dhc->nlp_auth_dhgpid;
5463         }

5465         if (hash_id == AUTH_MD5) {
5466                 bzero(&mdctx, sizeof (MD5_CTX));
5467                 hash_size = MD5_LEN;
5468                 MD5Init(&mdctx);
5469                 MD5Update(&mdctx,
5470                     (unsigned char *)&(un_cval.md5.val[0]), MD5_LEN);

5472                 /*
5473                  * get the pub key (g^y mod p) and session key (g^(x*y) mod
```



```
5474                  * p) and stored them in the partner's ndlp structure
5475                  */
5476                 err = emlxs_BIGNUM_get_pubkey(port, port_dhc, ndlp,
5477                     dhval, dhvallen, hash_size, dhgp_id);

5479                 if (err != BIG_OK) {
5480                         return (err);
5481                 }
5482                 if (ndlp->nlp_DID == FABRIC_DID) {
5483                         MD5Update(&mdctx,
5484                             (unsigned char *)&node_dhc->ses_key[0],
5485                             node_dhc->seskey_len);
5486                 } else {
5487                 MD5Update(&mdctx,
5488                     (unsigned char *)&node_dhc->nlp_auth_misc.ses_key[0],
5489                     node_dhc->nlp_auth_misc.seskey_len);
5490                 }

5492                 MD5Final((uint8_t *)md5_digest, &mdctx);

5494                 bcopy((void *)&md5_digest, (void *)hash_val, MD5_LEN);
5495         }
5496         if (hash_id == AUTH_SHA1) {
5497                 bzero(&sha1ctx, sizeof (SHA1_CTX));

5499                 hash_size = SHA1_LEN;

5501                 SHA1Init(&sha1ctx);

5503                 SHA1Update(&sha1ctx, (void *)&(un_cval.sha1.val[0]), SHA1_LEN);

5505                 /* get the pub key and session key */
5506                 err = emlxs_BIGNUM_get_pubkey(port, port_dhc, ndlp,
5507                     dhval, dhvallen, hash_size, dhgp_id);

5509                 if (err != BIG_OK) {
5510                         return (err);
5511                 }
5512                 if (ndlp->nlp_DID == FABRIC_DID) {
5513                         SHA1Update(&sha1ctx, (void *)&node_dhc->ses_key[0],
5514                             node_dhc->seskey_len);
5515                 } else {
5516                         SHA1Update(&sha1ctx,
5517                             (void *)&node_dhc->nlp_auth_misc.ses_key[0],
5518                             node_dhc->nlp_auth_misc.seskey_len);
5519                 }

5521                 SHA1Final((void *)sha1_digest, &sha1ctx);

5523                 bcopy((void *)&sha1_digest, (void *)hash_val, SHA1_LEN);
5524         }
5525         return (err);

5527 } /* emlxs_interm_hash */

5529 /*
5530  * This routine get the pubkey and session key. these pubkey and session
5531  * key are stored in the partner's ndlp structure.
5532  */
5533 /* ARGSUSED */
5534 BIG_ERR_CODE
5535 emlxs_BIGNUM_get_pubkey(
5536                         emlxs_port_t *port,
5537                         emlxs_port_dhc_t *port_dhc,
5538                         NODELIST *ndlp,
5539                         uint8_t *dhval,
```

```
5540                        uint32_t *dhvallen,
5541                        uint32_t hash_size,
5542                        uint32_t dhgp_id)
5543 {
5544         emlxs_hba_t *hba = HBA;

5546         BIGNUM a, e, n, result;
5547         uint32_t plen;
5548         uint8_t random_number[20];
5549         unsigned char *tmp = NULL;
5550         BIGNUM g, result1;

5552 #ifdef BIGNUM_CHUNK_32
5553         uint8_t gen[] = {0x00, 0x00, 0x00, 0x02};
5554 #else
5555         uint8_t gen[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02};
5556 #endif /* BIGNUM_CHUNK_32 */

5558         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
5559         BIG_ERR_CODE err = BIG_OK;

5561         /*
5562          * compute a^e mod n assume a < n, n odd, result->value at least as
5563          * long as n->value.
5564          *
5565          * a is the public key received from responder. e is the private key
5566          * generated by me. n is the wellknown modulus.
5567          */

5569         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
5570             "emlxs_BIGNUM_get_pubkey: 0x%x 0x%x 0x%x 0x%x",
5571             ndlp->nlp_DID, *dhvallen, hash_size, dhgp_id);

5573         /* size should be in the unit of (BIG_CHUNK_TYPE) words */
5574         if (big_init(&a, CHARLEN2BIGNUMLEN(*dhvallen))  != BIG_OK) {
5575                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5576                     "emlxs_BIGNUM_get_pubkey: big_init failed. a size=%d",
5577                     CHARLEN2BIGNUMLEN(*dhvallen));

5579                 err = BIG_NO_MEM;
5580                 return (err);
5581         }
5582         /* a: (g^x mod p) */
5583         /*
5584          * dhval is in big-endian format. This call converts from
5585          * byte-big-endian format to big number format (words in little
5586          * endian order, but bytes within the words big endian)
5587          */
5588         bytestring2bignum(&a, (unsigned char *)dhval, *dhvallen);

5590         if (big_init(&e, CHARLEN2BIGNUMLEN(hash_size)) != BIG_OK) {
5591                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5592                     "emlxs_BIGNUM_get_pubkey: big_init failed. e size=%d",
5593                     CHARLEN2BIGNUMLEN(hash_size));

5595                 err = BIG_NO_MEM;
5596                 goto ret1;
5597         }
5598 #ifdef RAND

5600         bzero(&random_number, hash_size);

5602         /* to get random private key: y */
5603         /* remember y is short lived private key */
5604         if (hba->rdn_flag == 1) {
5605                 emlxs_get_random_bytes(ndlp, random_number, 20);
```



```
5606         } else {
5607                 (void) random_get_pseudo_bytes(random_number, hash_size);
5608         }

5610         /* e: y */
5611         bytestring2bignum(&e, (unsigned char *)random_number, hash_size);

5613 #endif  /* RAND */

5615 #ifdef MYRAND
5616         bytestring2bignum(&e, (unsigned char *)myrand, hash_size);

5618         printf("myrand random_number as Y ================\n");
5619         for (i = 0; i < 5; i++) {
5620                 for (j = 0; j < 4; j++) {
5621                         printf("%x", myrand[(i * 4) + j]);
5622                 }
5623                 printf("\n");
5624         }
5625 #endif  /* MYRAND */

5627         switch (dhgp_id) {
5628         case GROUP_1024:
5629                 plen = 128;
5630                 tmp = dhgp1_pVal;
5631                 break;

5633         case GROUP_1280:
5634                 plen = 160;
5635                 tmp = dhgp2_pVal;
5636                 break;

5638         case GROUP_1536:
5639                 plen = 192;
5640                 tmp = dhgp3_pVal;
5641                 break;

5643         case GROUP_2048:
5644                 plen = 256;
5645                 tmp = dhgp4_pVal;
5646                 break;
5647         }

5649         if (big_init(&n, CHARLEN2BIGNUMLEN(plen)) != BIG_OK) {
5650                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5651                     "emlxs_BIGNUM_get_pubkey: big_init failed. n size=%d",
5652                     CHARLEN2BIGNUMLEN(plen));
5653                 err = BIG_NO_MEM;
5654                 goto ret2;
5655         }
5656         bytestring2bignum(&n, (unsigned char *)tmp, plen);

5658         if (big_init(&result, CHARLEN2BIGNUMLEN(512)) != BIG_OK) {
5659                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5660                     "emlxs_BIGNUM_get_pubkey: big_init failed. result size=%d",
5661                     CHARLEN2BIGNUMLEN(512));

5663                 err = BIG_NO_MEM;
5664                 goto ret3;
5665         }
5666         if (big_cmp_abs(&a, &n) > 0) {
5667                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5668                     "emlxs_BIGNUM_get_pubkey: big_cmp_abs error.");
5669                 err = BIG_GENERAL_ERR;
5670                 goto ret4;
5671         }
```

```
5672         /* perform computation on big numbers to get seskey  */
5673         /* a^e mod n */
5674         /* i.e., (g^x mod p)^y mod p  */

5676         if (big_modexp(&result, &a, &e, &n, NULL) != BIG_OK) {
5677                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5678                     "emlxs_BIGNUM_get_pubkey: big_modexp result error");
5679                 err = BIG_NO_MEM;
5680                 goto ret4;
5681         }
5682         /* convert big number ses_key to bytestring */
5683         if (ndlp->nlp_DID == FABRIC_DID) {
5684                 /*
5685                  * This call converts from big number format to
5686                  * byte-big-endian format. big number format is words in
5687                  * little endian order, but bytes within words in native byte
5688                  * order
5689                  */
5690                 bignum2bytestring(node_dhc->ses_key, &result,
5691                     sizeof (BIG_CHUNK_TYPE) * (result.len));
5692                 node_dhc->seskey_len = sizeof (BIG_CHUNK_TYPE) * (result.len);

5694                 /* we can store another copy in ndlp */
5695                 bignum2bytestring(node_dhc->nlp_auth_misc.ses_key, &result,
5696                     sizeof (BIG_CHUNK_TYPE) * (result.len));
5697                 node_dhc->nlp_auth_misc.seskey_len =
5698                     sizeof (BIG_CHUNK_TYPE) * (result.len);
5699         } else {
5700                 /* for end-to-end auth */
5701                 bignum2bytestring(node_dhc->nlp_auth_misc.ses_key, &result,
5702                     sizeof (BIG_CHUNK_TYPE) * (result.len));
5703                 node_dhc->nlp_auth_misc.seskey_len =
5704                     sizeof (BIG_CHUNK_TYPE) * (result.len);
5705         }

5707         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
5708             "emlxs_BIGNUM_get_pubkey: after seskey cal: 0x%x 0x%x 0x%x",
5709             node_dhc->nlp_auth_misc.seskey_len, result.size, result.len);


5712         /* to get pub_key: g^y mod p, g is 2 */

5714         if (big_init(&g, 1) != BIG_OK) {
5715                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5716                     "emlxs_BIGNUM_get_pubkey: big_init failed. g size=1");

5718                 err = BIG_NO_MEM;
5719                 goto ret4;
5720         }
5721         if (big_init(&result1, CHARLEN2BIGNUMLEN(512)) != BIG_OK) {
5722                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5723                     "emlxs_BIGNUM_get_pubkey: big_init failed. result1 size=%d",
5724                     CHARLEN2BIGNUMLEN(512));
5725                 err = BIG_NO_MEM;
5726                 goto ret5;
5727         }

5729         bytestring2bignum(&g,
5730             (unsigned char *)&gen, sizeof (BIG_CHUNK_TYPE));

5732         if (big_modexp(&result1, &g, &e, &n, NULL) != BIG_OK) {
5733                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5734                     "emlxs_BIGNUM_get_pubkey: big_modexp result1 error");
5735                 err = BIG_NO_MEM;
5736                 goto ret6;
5737         }
```



```
5738         /* convert big number pub_key to bytestring */
5739         if (ndlp->nlp_DID == FABRIC_DID) {

5741                 bignum2bytestring(node_dhc->pub_key, &result1,
5742                     sizeof (BIG_CHUNK_TYPE) * (result1.len));
5743                 node_dhc->pubkey_len = (result1.len) * sizeof (BIG_CHUNK_TYPE);

5745                 /* save another copy in ndlp */
5746                 bignum2bytestring(node_dhc->nlp_auth_misc.pub_key, &result1,
5747                     sizeof (BIG_CHUNK_TYPE) * (result1.len));
5748                 node_dhc->nlp_auth_misc.pubkey_len =
5749                     (result1.len) * sizeof (BIG_CHUNK_TYPE);

5751         } else {
5752                 /* for end-to-end auth */
5753                 bignum2bytestring(node_dhc->nlp_auth_misc.pub_key, &result1,
5754                     sizeof (BIG_CHUNK_TYPE) * (result1.len));
5755                 node_dhc->nlp_auth_misc.pubkey_len =
5756                     (result1.len) * sizeof (BIG_CHUNK_TYPE);
5757         }

5759         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
5760             "emlxs_BIGNUM_get_pubkey: after pubkey cal: 0x%x 0x%x 0x%x",
5761             node_dhc->nlp_auth_misc.pubkey_len, result1.size, result1.len);


5764 ret6:
5765         big_finish(&result1);
5766 ret5:
5767         big_finish(&g);
5768 ret4:
5769         big_finish(&result);
5770 ret3:
5771         big_finish(&n);
5772 ret2:
5773         big_finish(&e);
5774 ret1:
5775         big_finish(&a);

5777         return (err);

5779 } /* emlxs_BIGNUM_get_pubkey */


5782 /*
5783  * g^x mod p x is the priv_key g and p are wellknow based on dhgp_id
5784  */
5785 /* ARGSUSED */
5786 static BIG_ERR_CODE
5787 emlxs_BIGNUM_get_dhval(
5788 emlxs_port_t *port,
5789 emlxs_port_dhc_t *port_dhc,
5790 NODELIST *ndlp,
5791 uint8_t *dhval,
5792 uint32_t *dhval_len,
5793 uint32_t dhgp_id,
5794 uint8_t *priv_key,
5795 uint32_t privkey_len)
5796 {
5797         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
5798         BIGNUM g, e, n, result1;
5799         uint32_t plen;
5800         unsigned char *tmp = NULL;

5802 #ifdef BIGNUM_CHUNK_32
5803         uint8_t gen[] = {0x00, 0x00, 0x00, 0x02};
```

```
#else
        uint8_t gen[] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x02};
#endif /* BIGNUM_CHUNK_32 */

        BIG_ERR_CODE err = BIG_OK;

        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
            "emlxs_BIGNUM_get_dhval: did=0x%x privkey_len=0x%x dhgp_id=0x%x",
            ndlp->nlp_DID, privkey_len, dhgp_id);

        if (big_init(&result1, CHARLEN2BIGNUMLEN(512)) != BIG_OK) {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                    "emlxs_BIGNUM_get_dhval: big_init failed. result1 size=%d",
                    CHARLEN2BIGNUMLEN(512));

                err = BIG_NO_MEM;
                return (err);
        }
        if (big_init(&g, 1) != BIG_OK) {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                    "emlxs_BIGNUM_get_dhval: big_init failed. g size=1");

                err = BIG_NO_MEM;
                goto ret1;
        }
        /* get g */
        bytestring2bignum(&g, (unsigned char *)gen, sizeof (BIG_CHUNK_TYPE));

        if (big_init(&e, CHARLEN2BIGNUMLEN(privkey_len)) != BIG_OK) {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                    "emlxs_BIGNUM_get_dhval: big_init failed. e size=%d",
                    CHARLEN2BIGNUMLEN(privkey_len));

                err = BIG_NO_MEM;
                goto ret2;
        }
        /* get x */
        bytestring2bignum(&e, (unsigned char *)priv_key, privkey_len);

        switch (dhgp_id) {
        case GROUP_1024:
                plen = 128;
                tmp = dhgp1_pVal;
                break;

        case GROUP_1280:
                plen = 160;
                tmp = dhgp2_pVal;
                break;

        case GROUP_1536:
                plen = 192;
                tmp = dhgp3_pVal;
                break;

        case GROUP_2048:
                plen = 256;
                tmp = dhgp4_pVal;
                break;
        }

        if (big_init(&n, CHARLEN2BIGNUMLEN(plen)) != BIG_OK) {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                    "emlxs_BIGNUM_get_dhval: big_init failed. n size=%d",
                    CHARLEN2BIGNUMLEN(plen));
```



```
                err = BIG_NO_MEM;
                goto ret3;
        }
        /* get p */
        bytestring2bignum(&n, (unsigned char *)tmp, plen);

        /* to cal: (g^x mod p) */
        if (big_modexp(&result1, &g, &e, &n, NULL) != BIG_OK) {
                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
                    "emlxs_BIGNUM_get_dhval: big_modexp result1 error");

                err = BIG_GENERAL_ERR;
                goto ret4;
        }
        /* convert big number pub_key to bytestring */
        if (ndlp->nlp_DID == FABRIC_DID) {
                bignum2bytestring(node_dhc->hrsp_pub_key, &result1,
                    sizeof (BIG_CHUNK_TYPE) * (result1.len));
                node_dhc->hrsp_pubkey_len =
                    (result1.len) * sizeof (BIG_CHUNK_TYPE);

                /* save another copy in partner's ndlp */
                bignum2bytestring(node_dhc->nlp_auth_misc.hrsp_pub_key,
                    &result1,
                    sizeof (BIG_CHUNK_TYPE) * (result1.len));

                node_dhc->nlp_auth_misc.hrsp_pubkey_len =
                    (result1.len) * sizeof (BIG_CHUNK_TYPE);
        } else {
                bignum2bytestring(node_dhc->nlp_auth_misc.hrsp_pub_key,
                    &result1,
                    sizeof (BIG_CHUNK_TYPE) * (result1.len));
                node_dhc->nlp_auth_misc.hrsp_pubkey_len =
                    (result1.len) * sizeof (BIG_CHUNK_TYPE);
        }


        if (ndlp->nlp_DID == FABRIC_DID) {
                bcopy((void *)node_dhc->hrsp_pub_key, (void *)dhval,
                    node_dhc->hrsp_pubkey_len);
        } else {
                bcopy((void *)node_dhc->nlp_auth_misc.hrsp_pub_key,
                    (void *)dhval,
                    node_dhc->nlp_auth_misc.hrsp_pubkey_len);
        }

        *(uint32_t *)dhval_len = (result1.len) * sizeof (BIG_CHUNK_TYPE);


ret4:
        big_finish(&result1);
ret3:
        big_finish(&e);
ret2:
        big_finish(&n);
ret1:
        big_finish(&g);

        return (err);

} /* emlxs_BIGNUM_get_dhval */


/*
 * to get ((g^y mod p)^x mod p) a^e mod n
 */
```

```
5936 BIG_ERR_CODE
5937 emlxs_BIGNUM_pubkey(
5938                     emlxs_port_t *port,
5939                     void *pubkey,
5940                     uint8_t *dhval,     /* g^y mod p */
5941                     uint32_t dhvallen,
5942                     uint8_t *key,       /* x */
5943                     uint32_t key_size,
5944                     uint32_t dhgp_id,
5945                     uint32_t *pubkeylen)
5946 {
5947         BIGNUM a, e, n, result;
5948         uint32_t plen;
5949         unsigned char *tmp = NULL;
5950         BIG_ERR_CODE err = BIG_OK;

5952         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
5953             "emlxs_BIGNUM_pubkey: dhvallen=0x%x dhgp_id=0x%x",
5954             dhvallen, dhgp_id);

5956         if (big_init(&a, CHARLEN2BIGNUMLEN(dhvallen)) != BIG_OK) {
5957                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5958                     "emlxs_BIGNUM_pubkey: big_init failed. a size=%d",
5959                     CHARLEN2BIGNUMLEN(dhvallen));

5961                 err = BIG_NO_MEM;
5962                 return (err);
5963         }
5964         /* get g^y mod p */
5965         bytestring2bignum(&a, (unsigned char *)dhval, dhvallen);

5967         if (big_init(&e, CHARLEN2BIGNUMLEN(key_size)) != BIG_OK) {
5968                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
5969                     "emlxs_BIGNUM_pubkey: big_init failed. e size=%d",
5970                     CHARLEN2BIGNUMLEN(key_size));

5972                 err = BIG_NO_MEM;
5973                 goto ret1;
5974         }
5975         /* get x */
5976         bytestring2bignum(&e, (unsigned char *)key, key_size);

5978         switch (dhgp_id) {
5979         case GROUP_1024:
5980                 plen = 128;
5981                 tmp = dhgp1_pVal;
5982                 break;

5984         case GROUP_1280:
5985                 plen = 160;
5986                 tmp = dhgp2_pVal;
5987                 break;

5989         case GROUP_1536:
5990                 plen = 192;
5991                 tmp = dhgp3_pVal;
5992                 break;

5994         case GROUP_2048:
5995                 plen = 256;
5996                 tmp = dhgp4_pVal;
5997                 break;
5998         }

6000         if (big_init(&n, CHARLEN2BIGNUMLEN(plen)) != BIG_OK) {
6001                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
```



```
6002                     "emlxs_BIGNUM_pubkey: big_init failed. n size=%d",
6003                     CHARLEN2BIGNUMLEN(plen));

6005                 err = BIG_NO_MEM;
6006                 goto ret2;
6007         }
6008         bytestring2bignum(&n, (unsigned char *)tmp, plen);

6010         if (big_init(&result, CHARLEN2BIGNUMLEN(512)) != BIG_OK) {
6011                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6012                     "emlxs_BIGNUM_pubkey: big_init failed. result size=%d",
6013                     CHARLEN2BIGNUMLEN(512));

6015                 err = BIG_NO_MEM;
6016                 goto ret3;
6017         }
6018         if (big_cmp_abs(&a, &n) > 0) {
6019                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6020                     "emlxs_BIGNUM_pubkey: big_cmp_abs error");

6022                 err = BIG_GENERAL_ERR;
6023                 goto ret4;
6024         }
6025         if (big_modexp(&result, &a, &e, &n, NULL) != BIG_OK) {
6026                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6027                     "emlxs_BIGNUM_pubkey: big_modexp result error");

6029                 err = BIG_NO_MEM;
6030                 goto ret4;
6031         }
6032         bignum2bytestring(pubkey, &result,
6033             sizeof (BIG_CHUNK_TYPE) * (result.len));
6034         *pubkeylen = sizeof (BIG_CHUNK_TYPE) * (result.len);

6036         /* This pubkey is actually session key */

6038 ret4:
6039         big_finish(&result);
6040 ret3:
6041         big_finish(&n);
6042 ret2:
6043         big_finish(&e);
6044 ret1:
6045         big_finish(&a);

6047         return (err);

6049 } /* emlxs_BIGNUM_pubkey */


6052 /*
6053  * key: x dhval: (g^y mod p) tran_id: Ti bi_cval: C2 hash_id: H dhgp_id: p/g
6054  *
6055  * Cai = H (C2 || ((g^y mod p)^x mod p) )
6056  *
6057  */
6058 /* ARGSUSED */
6059 BIG_ERR_CODE
6060 emlxs_hash_Cai(
6061         emlxs_port_t *port,
6062         emlxs_port_dhc_t *port_dhc,
6063         NODELIST *ndlp,
6064         void *Cai,
6065         uint32_t hash_id,
6066         uint32_t dhgp_id,
6067         uint32_t tran_id,
```

```
6068         uint8_t *cval,
6069         uint32_t cval_len,
6070         uint8_t *key,
6071         uint8_t *dhval,
6072         uint32_t dhvallen)
6073 {
6074         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
6075         MD5_CTX mdctx;
6076         SHA1_CTX sha1ctx;
6077         uint8_t sha1_digest[20];
6078         uint8_t md5_digest[16];
6079         uint8_t pubkey[512];
6080         uint32_t pubkey_len = 0;
6081         uint32_t key_size;
6082         BIG_ERR_CODE err = BIG_OK;

6084         key_size = cval_len;
6085         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
6086             "emlxs_hash_Cai: 0x%x 0x%x 0x%x 0x%x 0x%x",
6087             ndlp->nlp_DID, hash_id, dhgp_id, tran_id, dhvallen);

6089         if (hash_id == AUTH_MD5) {
6090                 bzero(&mdctx, sizeof (MD5_CTX));
6091                 MD5Init(&mdctx);
6092                 MD5Update(&mdctx, (unsigned char *)cval, cval_len);

6094                 /* this pubkey obtained is actually the session key */
6095                 /*
6096                  * pubkey: ((g^y mod p)^x mod p)
6097                  */
6098                 err = emlxs_BIGNUM_pubkey(port, pubkey, dhval, dhvallen,
6099                     key, key_size, dhgp_id, &pubkey_len);

6101                 if (err != BIG_OK) {
6102                         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6103                             "emlxs_hash_Cai: MD5 BIGNUM_pubkey error: 0x%x",
6104                             err);

6106                         err = BIG_GENERAL_ERR;
6107                         return (err);
6108                 }
6109                 if (pubkey_len == 0) {
6110                         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6111                             "emlxs_hash_Cai: MD5 BIGNUM_pubkey error: len=0");

6113                         err = BIG_GENERAL_ERR;
6114                         return (err);
6115                 }
6116                 if (ndlp->nlp_DID == FABRIC_DID) {
6117                         bcopy((void *)pubkey,
6118                             (void *)node_dhc->hrsp_ses_key, pubkey_len);
6119                         node_dhc->hrsp_seskey_len = pubkey_len;

6121                         /* store extra copy */
6122                         bcopy((void *)pubkey,
6123                             (void *)node_dhc->nlp_auth_misc.hrsp_ses_key,
6124                             pubkey_len);
6125                         node_dhc->nlp_auth_misc.hrsp_seskey_len = pubkey_len;

6127                 } else {
6128                         bcopy((void *)pubkey,
6129                             (void *)node_dhc->nlp_auth_misc.hrsp_ses_key,
6130                             pubkey_len);
6131                         node_dhc->nlp_auth_misc.hrsp_seskey_len = pubkey_len;
6132                 }
```



```
6134                 MD5Update(&mdctx, (unsigned char *)pubkey, pubkey_len);
6135                 MD5Final((uint8_t *)md5_digest, &mdctx);
6136                 bcopy((void *)&md5_digest, (void *)Cai, MD5_LEN);
6137         }
6138         if (hash_id == AUTH_SHA1) {
6139                 bzero(&sha1ctx, sizeof (SHA1_CTX));
6140                 SHA1Init(&sha1ctx);

6142                 SHA1Update(&sha1ctx, (void *)cval, cval_len);

6144                 err = emlxs_BIGNUM_pubkey(port, pubkey, dhval, dhvallen,
6145                     key, key_size, dhgp_id, &pubkey_len);

6147                 if (err != BIG_OK) {
6148                         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6149                             "emlxs_hash_Cai: SHA1 BIGNUM_pubkey error: 0x%x",
6150                             err);

6152                         err = BIG_GENERAL_ERR;
6153                         return (err);
6154                 }
6155                 if (pubkey_len == 0) {
6156                         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6157                             "emlxs_hash_Cai: SA1 BUM_pubkey error: key_len=0");

6159                         err = BIG_GENERAL_ERR;
6160                         return (err);
6161                 }
6162                 if (ndlp->nlp_DID == FABRIC_DID) {
6163                         bcopy((void *)pubkey,
6164                             (void *)node_dhc->hrsp_ses_key,
6165                             pubkey_len);
6166                         node_dhc->hrsp_seskey_len = pubkey_len;

6168                         /* store extra copy */
6169                         bcopy((void *)pubkey,
6170                             (void *)node_dhc->nlp_auth_misc.hrsp_ses_key,
6171                             pubkey_len);
6172                         node_dhc->nlp_auth_misc.hrsp_seskey_len = pubkey_len;

6174                 } else {
6175                         bcopy((void *)pubkey,
6176                             (void *)node_dhc->nlp_auth_misc.hrsp_ses_key,
6177                             pubkey_len);
6178                         node_dhc->nlp_auth_misc.hrsp_seskey_len = pubkey_len;
6179                 }

6181                 SHA1Update(&sha1ctx, (void *)pubkey, pubkey_len);
6182                 SHA1Final((void *)sha1_digest, &sha1ctx);
6183                 bcopy((void *)&sha1_digest, (void *)Cai, SHA1_LEN);
6184         }
6185         return (err);

6187 } /* emlxs_hash_Cai */


6190 /*
6191  * This routine is to verify the DHCHAP_Reply from initiator by the host
6192  * as the responder.
6193  *
6194  * flag: 1: if host is the responder 0: if host is the initiator
6195  *
6196  * if bi_cval != NULL, this routine is used to calculate the response based
6197  * on the challenge from initiator as part of
6198  * DHCHAP_Reply for bi-dirctional authentication.
6199  *
```

```
6200  */
6201 /* ARGSUSED */
6202 static uint32_t *
6203 emlxs_hash_verification(
6204         emlxs_port_t *port,
6205         emlxs_port_dhc_t *port_dhc,
6206         NODELIST *ndlp,
6207         uint32_t tran_id,
6208         uint8_t *dhval,
6209         uint32_t dhval_len,
6210         uint32_t flag,  /* always 1 for now */
6211         uint8_t *bi_cval)
6212 {                       /* always 0 for now */
6213         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
6214         uint32_t dhgp_id;
6215         uint32_t hash_id;
6216         uint32_t *hash_val = NULL;
6217         uint32_t hash_size;
6218         MD5_CTX mdctx;
6219         SHA1_CTX sha1ctx;
6220         uint8_t sha1_digest[20];
6221         uint8_t md5_digest[16];
6222         uint8_t Cai[20];
6223         /* union challenge_val un_cval; */
6224         uint8_t key[20];
6225         uint8_t cval[20];
6226         uint32_t cval_len;
6227         uint8_t mytran_id = 0x00;
6228         char *remote_key;
6229         BIG_ERR_CODE err = BIG_OK;

6231         tran_id = (AUTH_TRAN_ID_MASK & tran_id);
6232         mytran_id = (uint8_t)(LE_SWAP32(tran_id));

6234         if (ndlp->nlp_DID == FABRIC_DID) {
6235                 remote_key = (char *)node_dhc->auth_key.remote_password;
6236         } else {
6237                 /*
6238                  * in case of end-to-end auth, this remote password should be
6239                  * the password associated with the remote entity. (i.e.,)
6240                  * for now it is actually local_password.
6241                  */
6242                 remote_key = (char *)node_dhc->auth_key.remote_password;
6243         }

6245         if (flag == 0) {
6246                 dhgp_id = node_dhc->dhgp_id;
6247                 hash_id = node_dhc->hash_id;
6248         } else {
6249                 dhgp_id = node_dhc->nlp_auth_dhgpid;
6250                 hash_id = node_dhc->nlp_auth_hashid;
6251         }

6253         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
6254             "emlxs_hash_verification: 0x%x 0x%x hash_id=0x%x dhgp_id=0x%x",
6255             ndlp->nlp_DID, mytran_id, hash_id, dhgp_id);

6257         if (dhval_len == 0) {
6258                 /* NULL DHCHAP group */
6259                 if (hash_id == AUTH_MD5) {
6260                         bzero(&mdctx, sizeof (MD5_CTX));
6261                         hash_size = MD5_LEN;
6262                         MD5Init(&mdctx);

6264                         MD5Update(&mdctx, (unsigned char *)&mytran_id, 1);
```



```
6266                         if (ndlp->nlp_DID == FABRIC_DID) {
6267                                 MD5Update(&mdctx,
6268                                     (unsigned char *)remote_key,
6269                                     node_dhc->auth_key.remote_password_length);
6270                         } else {
6271                                 MD5Update(&mdctx,
6272                                     (unsigned char *)remote_key,
6273                                     node_dhc->auth_key.remote_password_length);
6274                         }

6276                         if (ndlp->nlp_DID == FABRIC_DID) {
6277                                 MD5Update(&mdctx,
6278                                     (unsigned char *)&node_dhc->hrsp_cval[0],
6279                                     MD5_LEN);
6280                         } else {
6281                 MD5Update(&mdctx,
6282                     (unsigned char *)&node_dhc->nlp_auth_misc.hrsp_cval[0],
6283                     MD5_LEN);
6284                         }

6286                         MD5Final((uint8_t *)md5_digest, &mdctx);

6288                         hash_val = (uint32_t *)kmem_alloc(hash_size,
6289                             KM_NOSLEEP);
6290                         if (hash_val == NULL) {
6291                                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6292                                     "emlxs_hash_verification: alloc failed");

6294                                 return (NULL);
6295                         } else {
6296                                 bcopy((void *)md5_digest,
6297                                     (void *)hash_val, MD5_LEN);
6298                         }
6299                 }
6300                 if (hash_id == AUTH_SHA1) {
6301                         bzero(&sha1ctx, sizeof (SHA1_CTX));
6302                         hash_size = SHA1_LEN;
6303                         SHA1Init(&sha1ctx);
6304                         SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

6306                         if (ndlp->nlp_DID == FABRIC_DID) {
6307                                 SHA1Update(&sha1ctx, (void *)remote_key,
6308                                     node_dhc->auth_key.remote_password_length);
6309                         } else {
6310                                 SHA1Update(&sha1ctx, (void *)remote_key,
6311                                     node_dhc->auth_key.remote_password_length);
6312                         }

6314                         if (ndlp->nlp_DID == FABRIC_DID) {
6315                                 SHA1Update(&sha1ctx,
6316                                     (void *)&node_dhc->hrsp_cval[0],
6317                                     SHA1_LEN);
6318                         } else {
6319                         SHA1Update(&sha1ctx,
6320                             (void *)&node_dhc->nlp_auth_misc.hrsp_cval[0],
6321                             SHA1_LEN);
6322                         }

6324                         SHA1Final((void *)sha1_digest, &sha1ctx);
6325                         hash_val = (uint32_t *)kmem_zalloc(hash_size,
6326                             KM_NOSLEEP);
6327                         if (hash_val == NULL) {
6328                                 EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6329                                     "emlxs_hash_verification: alloc failed");

6331                                 return (NULL);
```

```
6332                        } else {
6333                                bcopy((void *)sha1_digest,
6334                                    (void *)hash_val, SHA1_LEN);
6335                        }
6336                }
6337                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
6338                    "emlxs_hash_verification: hash_val=0x%x",
6339                    *(uint32_t *)hash_val);

6341                return ((uint32_t *)hash_val);
6342        } else {

6344                /* DHCHAP group 1,2,3,4 */
6345                /*
6346                 * host received (g^x mod p) as dhval host has its own
6347                 * private key y as node_dhc->hrsp_priv_key[] host has its
6348                 * original challenge c as node_dhc->hrsp_cval[]
6349                 *
6350                 * H(c || (g^x mod p)^y mod p) = Cai H(Ti || Km || Cai) =
6351                 * hash_val returned. Ti : tran_id, Km : shared secret, Cai:
6352                 * obtained above.
6353                 */
6354                if (hash_id == AUTH_MD5) {
6355                        if (ndlp->nlp_DID == FABRIC_DID) {
6356                                bcopy((void *)node_dhc->hrsp_priv_key,
6357                                    (void *)key, MD5_LEN);
6358                        } else {
6359                        bcopy(
6360                            (void *)node_dhc->nlp_auth_misc.hrsp_priv_key,
6361                            (void *)key, MD5_LEN);
6362                        }
6363                }
6364                if (hash_id == AUTH_SHA1) {
6365                        if (ndlp->nlp_DID == FABRIC_DID) {
6366                                bcopy((void *)node_dhc->hrsp_priv_key,
6367                                    (void *)key, SHA1_LEN);
6368                        } else {
6369                        bcopy(
6370                            (void *)node_dhc->nlp_auth_misc.hrsp_priv_key,
6371                            (void *)key, SHA1_LEN);
6372                        }
6373                }
6374                if (ndlp->nlp_DID == FABRIC_DID) {
6375                        bcopy((void *)node_dhc->hrsp_cval,
6376                            (void *)cval, node_dhc->hrsp_cval_len);
6377                        cval_len = node_dhc->hrsp_cval_len;
6378                } else {
6379                        bcopy((void *)node_dhc->nlp_auth_misc.hrsp_cval,
6380                            (void *)cval,
6381                            node_dhc->nlp_auth_misc.hrsp_cval_len);
6382                        cval_len = node_dhc->nlp_auth_misc.hrsp_cval_len;
6383                }

6385                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
6386                    "emlxs_hash_verification: N-Null gp. 0x%x 0x%x",
6387                    ndlp->nlp_DID, cval_len);

6389                err = emlxs_hash_Cai(port, port_dhc, ndlp, (void *)Cai,
6390                    hash_id, dhgp_id,
6391                    tran_id, cval, cval_len,
6392                    key, dhval, dhval_len);

6394                if (err != BIG_OK) {
6395                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6396                            "emlxs_hash_verification: Cai error. ret=0x%x",
6397                            err);
```



```
6399                        return (NULL);
6400                }
6401                if (hash_id == AUTH_MD5) {
6402                        bzero(&mdctx, sizeof (MD5_CTX));
6403                        hash_size = MD5_LEN;

6405                        MD5Init(&mdctx);
6406                        MD5Update(&mdctx, (unsigned char *)&mytran_id, 1);

6408                        if (ndlp->nlp_DID == FABRIC_DID) {
6409                                MD5Update(&mdctx,
6410                                    (unsigned char *)remote_key,
6411                                    node_dhc->auth_key.remote_password_length);
6412                        } else {
6413                                MD5Update(&mdctx,
6414                                    (unsigned char *)remote_key,
6415                                    node_dhc->auth_key.remote_password_length);
6416                        }

6418                        MD5Update(&mdctx, (unsigned char *)Cai, MD5_LEN);
6419                        MD5Final((uint8_t *)md5_digest, &mdctx);

6421                        hash_val = (uint32_t *)kmem_zalloc(hash_size,
6422                            KM_NOSLEEP);
6423                        if (hash_val == NULL) {
6424                                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6425                                    "emlxs_hash_vf: alloc failed(Non-NULL dh)");

6427                                return (NULL);
6428                        } else {
6429                                bcopy((void *)&md5_digest,
6430                                    (void *)hash_val, MD5_LEN);
6431                        }
6432                }
6433                if (hash_id == AUTH_SHA1) {
6434                        bzero(&sha1ctx, sizeof (SHA1_CTX));
6435                        hash_size = SHA1_LEN;

6437                        SHA1Init(&sha1ctx);
6438                        SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

6440                        if (ndlp->nlp_DID == FABRIC_DID) {
6441                                SHA1Update(&sha1ctx, (void *)remote_key,
6442                                    node_dhc->auth_key.remote_password_length);
6443                        } else {
6444                                SHA1Update(&sha1ctx, (void *)remote_key,
6445                                    node_dhc->auth_key.remote_password_length);
6446                        }

6448                        SHA1Update(&sha1ctx, (void *)Cai, SHA1_LEN);
6449                        SHA1Final((void *)sha1_digest, &sha1ctx);

6451                        hash_val = (uint32_t *)kmem_zalloc(hash_size,
6452                            KM_NOSLEEP);
6453                        if (hash_val == NULL) {
6454                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6455                            "emlxs_hash_vf: val alloc failed (Non-NULL dh)");

6457                                return (NULL);
6458                        } else {
6459                                bcopy((void *)&sha1_digest,
6460                                    (void *)hash_val, SHA1_LEN);
6461                        }
6462                }
6463                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
```

```
6464                    "emlxs_hash_verification: hash_val=0x%x",
6465                    *(uint32_t *)hash_val);

6467                return ((uint32_t *)hash_val);
6468        }

6470 } /* emlxs_hash_verification */



6474 /*
6475  * When DHCHAP_Success msg was sent from responder to the initiator,
6476  * with bi-directional authentication requested, the
6477  * DHCHAP_Success contains the response R2 to the challenge C2 received.
6478  *
6479  * DHCHAP response R2: The value of R2 is computed using the hash function
6480  * H() selected by the HashID parameter of the
6481  * DHCHAP_Challenge msg, and the augmented challenge Ca2.
6482  *
6483  * NULL DH group: Ca2 = C2 Non NULL DH group: Ca2 = H(C2 ||
6484  * (g^y mod p)^x mod p)) x is selected by the authentication responder
6485  * which is the node_dhc->hrsp_priv_key[] (g^y mod p) is dhval received
6486  * from authentication initiator.
6487  *
6488  * R2 = H(Ti || Km || Ca2) Ti is the least significant byte of the
6489  * transaction id. Km is the secret associated with the
6490  * authentication responder.
6491  *
6492  * emlxs_hash_get_R2 and emlxs_hash_verification could be mergerd into one
6493  * function later.
6494  *
6495  */
6496 static uint32_t *
6497 emlxs_hash_get_R2(
6498         emlxs_port_t *port,
6499         emlxs_port_dhc_t *port_dhc,
6500         NODELIST *ndlp,
6501         uint32_t tran_id,
6502         uint8_t *dhval,
6503         uint32_t dhval_len,
6504         uint32_t flag,  /* flag 1 rsponder or 0 initiator */
6505         uint8_t *bi_cval)
6506 {
6507         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

6509         uint32_t dhgp_id;
6510         uint32_t hash_id;
6511         uint32_t *hash_val = NULL;
6512         uint32_t hash_size;
6513         MD5_CTX mdctx;
6514         SHA1_CTX sha1ctx;
6515         uint8_t sha1_digest[20];
6516         uint8_t md5_digest[16];
6517         uint8_t Cai[20];
6518         /* union challenge_val un_cval; */
6519         uint8_t key[20];
6520         uint32_t cval_len;
6521         uint8_t mytran_id = 0x00;

6523         char *mykey;
6524         BIG_ERR_CODE err = BIG_OK;

6526         if (ndlp->nlp_DID == FABRIC_DID) {
6527                 dhgp_id = node_dhc->nlp_auth_dhgpid;
6528                 hash_id = node_dhc->nlp_auth_hashid;
6529         } else {
```



```
6530                 if (flag == 0) {
6531                         dhgp_id = node_dhc->dhgp_id;
6532                         hash_id = node_dhc->hash_id;
6533                 } else {
6534                         dhgp_id = node_dhc->nlp_auth_dhgpid;
6535                         hash_id = node_dhc->nlp_auth_hashid;
6536                 }
6537         }

6539         tran_id = (AUTH_TRAN_ID_MASK & tran_id);
6540         mytran_id = (uint8_t)(LE_SWAP32(tran_id));

6542         EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_detail_msg,
6543             "emlxs_hash_get_R2:0x%x 0x%x dhgp_id=0x%x mytran_id=0x%x",
6544             ndlp->nlp_DID, hash_id, dhgp_id, mytran_id);

6546         if (ndlp->nlp_DID == FABRIC_DID) {
6547                 mykey = (char *)node_dhc->auth_key.local_password;

6549         } else {
6550                 /* in case of end-to-end mykey should be remote_password */
6551                 mykey = (char *)node_dhc->auth_key.remote_password;
6552         }

6554         if (dhval_len == 0) {
6555                 /* NULL DHCHAP group */
6556                 if (hash_id == AUTH_MD5) {
6557                         bzero(&mdctx, sizeof (MD5_CTX));
6558                         hash_size = MD5_LEN;
6559                         MD5Init(&mdctx);

6561                         MD5Update(&mdctx, (unsigned char *)&mytran_id, 1);

6563                         if (ndlp->nlp_DID == FABRIC_DID) {
6564                                 MD5Update(&mdctx, (unsigned char *)mykey,
6565                                     node_dhc->auth_key.local_password_length);
6566                         } else {
6567                                 MD5Update(&mdctx, (unsigned char *)mykey,
6568                                     node_dhc->auth_key.remote_password_length);
6569                         }

6571                         MD5Update(&mdctx, (unsigned char *)bi_cval, MD5_LEN);

6573                         MD5Final((uint8_t *)md5_digest, &mdctx);

6575                         hash_val = (uint32_t *)kmem_alloc(hash_size,
6576                             KM_NOSLEEP);
6577                         if (hash_val == NULL) {
6578                                 return (NULL);
6579                         } else {
6580                                 bcopy((void *)md5_digest,
6581                                     (void *)hash_val, MD5_LEN);
6582                         }
6583                 }
6584                 if (hash_id == AUTH_SHA1) {
6585                         bzero(&sha1ctx, sizeof (SHA1_CTX));
6586                         hash_size = SHA1_LEN;
6587                         SHA1Init(&sha1ctx);
6588                         SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

6590                         if (ndlp->nlp_DID == FABRIC_DID) {
6591                                 SHA1Update(&sha1ctx, (void *)mykey,
6592                                     node_dhc->auth_key.local_password_length);
6593                         } else {
6594                                 SHA1Update(&sha1ctx, (void *)mykey,
6595                                     node_dhc->auth_key.remote_password_length);
```

```
6596                        }

6598                        SHA1Update(&sha1ctx, (void *)bi_cval, SHA1_LEN);
6599                        SHA1Final((void *)sha1_digest, &sha1ctx);
6600                        hash_val = (uint32_t *)kmem_alloc(hash_size,
6601                            KM_NOSLEEP);
6602                        if (hash_val == NULL) {
6603                                return (NULL);
6604                        } else {
6605                                bcopy((void *)sha1_digest,
6606                                    (void *)hash_val, SHA1_LEN);
6607                        }
6608                }
6609        } else {
6610                /* NON-NULL DHCHAP */
6611                if (ndlp->nlp_DID == FABRIC_DID) {
6612                        if (hash_id == AUTH_MD5) {
6613                                bcopy((void *)node_dhc->hrsp_priv_key,
6614                                    (void *)key, MD5_LEN);
6615                        }
6616                        if (hash_id == AUTH_SHA1) {
6617                                bcopy((void *)node_dhc->hrsp_priv_key,
6618                                    (void *)key, SHA1_LEN);
6619                        }
6620                        cval_len = node_dhc->hrsp_cval_len;
6621                } else {
6622                        if (hash_id == AUTH_MD5) {
6623                        bcopy(
6624                            (void *)node_dhc->nlp_auth_misc.hrsp_priv_key,
6625                            (void *)key, MD5_LEN);
6626                        }
6627                        if (hash_id == AUTH_SHA1) {
6628                        bcopy(
6629                            (void *)node_dhc->nlp_auth_misc.hrsp_priv_key,
6630                            (void *)key, SHA1_LEN);
6631                        }
6632                        cval_len = node_dhc->nlp_auth_misc.hrsp_cval_len;
6633                }

6635                /* use bi_cval here */
6636                /*
6637                 * key: x dhval: (g^y mod p) tran_id: Ti bi_cval: C2 hash_id:
6638                 * H dhgp_id: p/g
6639                 *
6640                 * Cai = H (C2 || ((g^y mod p)^x mod p) )
6641                 *
6642                 * R2 = H (Ti || Km || Cai)
6643                 */
6644                err = emlxs_hash_Cai(port, port_dhc, ndlp, (void *)Cai,
6645                    hash_id, dhgp_id, tran_id, bi_cval, cval_len,
6646                    key, dhval, dhval_len);

6648                if (err != BIG_OK) {
6649                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6650                            "emlxs_hash_get_R2: emlxs_hash_Cai error. ret=0x%x",
6651                            err);

6653                        return (NULL);
6654                }
6655                if (hash_id == AUTH_MD5) {
6656                        bzero(&mdctx, sizeof (MD5_CTX));
6657                        hash_size = MD5_LEN;

6659                        MD5Init(&mdctx);
6660                        MD5Update(&mdctx, (unsigned char *) &mytran_id, 1);
```



```
6662                        /*
6663                         * Here we use the same key: mykey, note: this mykey
6664                         * should be the key associated with the
6665                         * authentication responder i.e. the remote key.
6666                         */
6667                        if (ndlp->nlp_DID == FABRIC_DID)
6668                                MD5Update(&mdctx, (unsigned char *)mykey,
6669                                    node_dhc->auth_key.local_password_length);
6670                        else
6671                                MD5Update(&mdctx, (unsigned char *)mykey,
6672                                    node_dhc->auth_key.remote_password_length);

6674                        MD5Update(&mdctx, (unsigned char *)Cai, MD5_LEN);
6675                        MD5Final((uint8_t *)md5_digest, &mdctx);

6677                        hash_val = (uint32_t *)kmem_alloc(hash_size,
6678                            KM_NOSLEEP);
6679                        if (hash_val == NULL) {
6680                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6681                            "emlxs_hash_get_R2: hash_val MD5 alloc failed.");

6683                                return (NULL);
6684                        } else {
6685                                bcopy((void *)md5_digest,
6686                                    (void *)hash_val, MD5_LEN);
6687                        }
6688                }
6689                if (hash_id == AUTH_SHA1) {
6690                        bzero(&sha1ctx, sizeof (SHA1_CTX));
6691                        hash_size = SHA1_LEN;

6693                        SHA1Init(&sha1ctx);
6694                        SHA1Update(&sha1ctx, (void *)&mytran_id, 1);

6696                        if (ndlp->nlp_DID == FABRIC_DID) {
6697                                SHA1Update(&sha1ctx, (void *)mykey,
6698                                    node_dhc->auth_key.local_password_length);
6699                        } else {
6700                                SHA1Update(&sha1ctx, (void *)mykey,
6701                                    node_dhc->auth_key.remote_password_length);
6702                        }

6704                        SHA1Update(&sha1ctx, (void *)Cai, SHA1_LEN);
6705                        SHA1Final((void *)sha1_digest, &sha1ctx);

6707                        hash_val = (uint32_t *)kmem_alloc(hash_size,
6708                            KM_NOSLEEP);
6709                        if (hash_val == NULL) {
6710                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_error_msg,
6711                            "emlxs_hash_get_R2: hash_val SHA1 alloc failed.");

6713                                return (NULL);
6714                        } else {
6715                                bcopy((void *)sha1_digest,
6716                                    (void *)hash_val, SHA1_LEN);
6717                        }
6718                }
6719        }

6721        return ((uint32_t *)hash_val);

6723 } /* emlxs_hash_get_R2 */



6727 /*
```

```
 */
static void
emlxs_log_auth_event(
        emlxs_port_t *port,
        NODELIST *ndlp,
        char *subclass,
        char *info)
{
        emlxs_hba_t *hba = HBA;
        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
        nvlist_t *attr_list = NULL;
        dev_info_t *dip = hba->dip;
        emlxs_auth_cfg_t *auth_cfg;
        char *tmp = "No_more_logging_information_available";

        uint8_t lwwn[8];
        uint8_t rwwn[8];
        char *lwwn_tmp = NULL;
        char *rwwn_tmp = NULL;
        char *mytmp_lwwn, *mytmp_rwwn;
        int i;

        auth_cfg = &(node_dhc->auth_cfg);

        if (info == NULL) {
                info = tmp;
        }
        bcopy((void *) &auth_cfg->local_entity, (void *)lwwn, 8);
        lwwn_tmp = (char *)kmem_zalloc(32, KM_NOSLEEP);
        if (lwwn_tmp == NULL) {
                return;
        }
        mytmp_lwwn = lwwn_tmp;

        for (i = 0; i < 8; i++) {
                lwwn_tmp = (char *)sprintf((char *)lwwn_tmp, "%02X", lwwn[i]);
                lwwn_tmp += 2;
        }
        mytmp_lwwn[16] = '\0';

        bcopy((void *)&auth_cfg->remote_entity, (void *)rwwn, 8);
        rwwn_tmp = (char *)kmem_zalloc(32, KM_NOSLEEP);

        mytmp_rwwn = rwwn_tmp;

        if (rwwn_tmp == NULL) {
                kmem_free(mytmp_lwwn, 32);
                return;
        }
        for (i = 0; i < 8; i++) {
                rwwn_tmp = (char *)sprintf((char *)rwwn_tmp, "%02X", rwwn[i]);
                rwwn_tmp += 2;
        }
        mytmp_rwwn[16] = '\0';

        if (nvlist_alloc(&attr_list, NV_UNIQUE_NAME_TYPE, KM_NOSLEEP)
            == DDI_SUCCESS) {
                if ((nvlist_add_uint32(attr_list, "instance",
                    ddi_get_instance(dip)) == DDI_SUCCESS) &&
                    (nvlist_add_string(attr_list, "lwwn",
                    (char *)mytmp_lwwn) == DDI_SUCCESS) &&
                    (nvlist_add_string(attr_list, "rwwn",
                    (char *)mytmp_rwwn) == DDI_SUCCESS) &&
                    (nvlist_add_string(attr_list, "Info",
                    info) == DDI_SUCCESS) &&
                    (nvlist_add_string(attr_list, "Class",
```



```
                    "EC_emlx") == DDI_SUCCESS) &&
                    (nvlist_add_string(attr_list, "SubClass",
                    subclass) == DDI_SUCCESS)) {

                        (void) ddi_log_sysevent(dip,
                            DDI_VENDOR_EMLX,
                            EC_EMLXS,
                            subclass,
                            attr_list,
                            NULL,
                            DDI_NOSLEEP);
                }
                nvlist_free(attr_list);
                attr_list = NULL;
        }
        kmem_free(mytmp_lwwn, 32);
        kmem_free(mytmp_rwwn, 32);

        return;

} /* emlxs_log_auth_event() */


/* ************************** AUTH DHC INTERFACE ********************** */

extern int
emlxs_dhc_auth_start(
        emlxs_port_t *port,
        emlxs_node_t *ndlp,
        uint8_t *deferred_sbp,
        uint8_t *deferred_ubp)
{
        emlxs_hba_t *hba = HBA;
        emlxs_config_t *cfg = &CFG;
        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
        emlxs_auth_cfg_t *auth_cfg;
        emlxs_auth_key_t *auth_key;
        uint32_t i;
        uint32_t fabric;
        uint32_t fabric_switch;

        /* The ubp represents an unsolicted PLOGI */
        /* The sbp represents a solicted PLOGI    */

        fabric = ((ndlp->nlp_DID & FABRIC_DID_MASK) == FABRIC_DID_MASK) ? 1 : 0;
        fabric_switch = ((ndlp->nlp_DID == FABRIC_DID) ? 1 : 0);

        /* Return is authentication is not enabled */
        if (cfg[CFG_AUTH_ENABLE].current == 0) {
                EMLXS_MSGF(EMLXS_CONTEXT,
                    &emlxs_fcsp_start_msg,
                    "Not started. Auth disabled. did=0x%x", ndlp->nlp_DID);

                emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_DISABLED, 0, 0);

                return (1);
        }
        if (port->vpi != 0 && cfg[CFG_AUTH_NPIV].current == 0) {
                EMLXS_MSGF(EMLXS_CONTEXT,
                    &emlxs_fcsp_start_msg,
                    "Not started. NPIV auth disabled. did=0x%x", ndlp->nlp_DID);

                emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_DISABLED, 0, 0);

                return (1);
        }
```

```
6860         if (!fabric_switch && fabric) {
6861                 EMLXS_MSGF(EMLXS_CONTEXT,
6862                     &emlxs_fcsp_start_msg,
6863                     "Not started. FS auth disabled. did=0x%x", ndlp->nlp_DID);

6865                 emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_DISABLED, 0, 0);

6867                 return (1);
6868         }
6869         /* Return if fcsp support to this node is not enabled */
6870         if (!fabric_switch && cfg[CFG_AUTH_E2E].current == 0) {
6871                 EMLXS_MSGF(EMLXS_CONTEXT,
6872                     &emlxs_fcsp_start_msg,
6873                     "Not started. E2E auth disabled. did=0x%x", ndlp->nlp_DID);

6875                 emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_DISABLED, 0, 0);

6877                 return (1);
6878         }
6879         if ((deferred_sbp && node_dhc->deferred_sbp) ||
6880             (deferred_ubp && node_dhc->deferred_ubp)) {
6881                 /* Clear previous authentication */
6882                 emlxs_dhc_auth_stop(port, ndlp);
6883         }
6884         mutex_enter(&hba->auth_lock);

6886         /* Intialize node */
6887         node_dhc->parent_auth_cfg = NULL;
6888         node_dhc->parent_auth_key = NULL;

6890         /* Acquire auth configuration */
6891         if (fabric_switch) {
6892                 auth_cfg = emlxs_auth_cfg_find(port,
6893                     (uint8_t *)emlxs_fabric_wwn);
6894                 auth_key = emlxs_auth_key_find(port,
6895                     (uint8_t *)emlxs_fabric_wwn);
6896         } else {
6897                 auth_cfg = emlxs_auth_cfg_find(port,
6898                     (uint8_t *)&ndlp->nlp_portname);
6899                 auth_key = emlxs_auth_key_find(port,
6900                     (uint8_t *)&ndlp->nlp_portname);
6901         }

6903         if (!auth_cfg) {
6904                 mutex_exit(&hba->auth_lock);

6906                 EMLXS_MSGF(EMLXS_CONTEXT,
6907                     &emlxs_fcsp_start_msg,
6908                     "Not started. No auth cfg entry found. did=0x%x",
6909                     ndlp->nlp_DID);

6911                 emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_DISABLED, 0, 0);

6913                 return (1);
6914         }
6915         if (fabric_switch) {
6916                 auth_cfg->node = NULL;
6917         } else {
6918                 node_dhc->parent_auth_cfg = auth_cfg;
6919                 auth_cfg->node = ndlp;
6920         }

6922         if (!auth_key) {
6923                 mutex_exit(&hba->auth_lock);

6925                 EMLXS_MSGF(EMLXS_CONTEXT,
```



```
6926                     &emlxs_fcsp_start_msg,
6927                     "Not started. No auth key entry found. did=0x%x",
6928                     ndlp->nlp_DID);

6930                 emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_DISABLED, 0, 0);

6932                 return (1);
6933         }
6934         if (fabric_switch) {
6935                 auth_key->node = NULL;
6936         } else {
6937                 node_dhc->parent_auth_key = auth_key;
6938                 auth_key->node = ndlp;
6939         }

6941         /* Remote port does not support fcsp */
6942         if (ndlp->sparm.cmn.fcsp_support == 0) {
6943                 switch (auth_cfg->authentication_mode) {
6944                 case AUTH_MODE_PASSIVE:
6945                         mutex_exit(&hba->auth_lock);

6947                         EMLXS_MSGF(EMLXS_CONTEXT,
6948                             &emlxs_fcsp_start_msg,
6949                             "Not started. Auth unsupported. did=0x%x",
6950                             ndlp->nlp_DID);

6952                         emlxs_dhc_state(port, ndlp,
6953                             NODE_STATE_AUTH_DISABLED, 0, 0);
6954                         return (1);

6956                 case AUTH_MODE_ACTIVE:
6957                         mutex_exit(&hba->auth_lock);

6959                         EMLXS_MSGF(EMLXS_CONTEXT,
6960                             &emlxs_fcsp_start_msg,
6961                             "Failed. Auth unsupported. did=0x%x",
6962                             ndlp->nlp_DID);

6964                         /*
6965                          * Save packet for deferred completion until
6966                          * authentication is complete
6967                          */
6968                         ndlp->node_dhc.deferred_sbp = deferred_sbp;
6969                         ndlp->node_dhc.deferred_ubp = deferred_ubp;

6971                         goto failed;

6973                 case AUTH_MODE_DISABLED:
6974                 default:
6975                         mutex_exit(&hba->auth_lock);

6977                         EMLXS_MSGF(EMLXS_CONTEXT,
6978                             &emlxs_fcsp_start_msg,
6979                             "Not started. Auth mode=disabled. did=0x%x",
6980                             ndlp->nlp_DID);

6982                         emlxs_dhc_state(port, ndlp,
6983                             NODE_STATE_AUTH_DISABLED, 0, 0);
6984                         return (1);
6985                 }
6986         } else {        /* Remote port supports fcsp */
6987                 switch (auth_cfg->authentication_mode) {
6988                 case AUTH_MODE_PASSIVE:
6989                 case AUTH_MODE_ACTIVE:
6990                         /* start auth */
6991                         break;
```

```
6993                case AUTH_MODE_DISABLED:
6994                default:
6995                        mutex_exit(&hba->auth_lock);

6997                        EMLXS_MSGF(EMLXS_CONTEXT,
6998                            &emlxs_fcsp_start_msg,
6999                            "Failed. Auth mode=disabled. did=0x%x",
7000                            ndlp->nlp_DID);

7002                        /*
7003                         * Save packet for deferred completion until
7004                         * authentication is complete
7005                         */
7006                        ndlp->node_dhc.deferred_sbp = deferred_sbp;
7007                        ndlp->node_dhc.deferred_ubp = deferred_ubp;

7009                        goto failed;
7010                }
7011        }

7013        /* We have a GO for authentication */

7015        /*
7016         * Save pointers for deferred completion until authentication is
7017         * complete
7018         */
7019        node_dhc->deferred_sbp = deferred_sbp;
7020        node_dhc->deferred_ubp = deferred_ubp;

7022        bzero(&node_dhc->auth_cfg, sizeof (node_dhc->auth_cfg));
7023        bzero(&node_dhc->auth_key, sizeof (node_dhc->auth_key));

7025        /* Program node's auth cfg */
7026        bcopy((uint8_t *)&port->wwpn,
7027            (uint8_t *)&node_dhc->auth_cfg.local_entity, 8);
7028        bcopy((uint8_t *)&ndlp->nlp_portname,
7029            (uint8_t *)&node_dhc->auth_cfg.remote_entity, 8);

7031        node_dhc->auth_cfg.authentication_timeout =
7032            auth_cfg->authentication_timeout;
7033        node_dhc->auth_cfg.authentication_mode =
7034            auth_cfg->authentication_mode;

7036        /*
7037         * If remote password type is "ignore", then only unidirectional auth
7038         * is allowed
7039         */
7040        if (auth_key->remote_password_type == 3) {
7041                node_dhc->auth_cfg.bidirectional = 0;
7042        } else {
7043                node_dhc->auth_cfg.bidirectional = auth_cfg->bidirectional;
7044        }

7046        node_dhc->auth_cfg.reauthenticate_time_interval =
7047            auth_cfg->reauthenticate_time_interval;

7049        for (i = 0; i < 4; i++) {
7050                switch (auth_cfg->authentication_type_priority[i]) {
7051                case ELX_DHCHAP:
7052                        node_dhc->auth_cfg.authentication_type_priority[i] =
7053                            AUTH_DHCHAP;
7054                        break;

7056                case ELX_FCAP:
7057                        node_dhc->auth_cfg.authentication_type_priority[i] =
```



```
7058                            AUTH_FCAP;
7059                        break;

7061                case ELX_FCPAP:
7062                        node_dhc->auth_cfg.authentication_type_priority[i] =
7063                            AUTH_FCPAP;
7064                        break;

7066                case ELX_KERBEROS:
7067                        node_dhc->auth_cfg.authentication_type_priority[i] =
7068                            AUTH_KERBEROS;
7069                        break;

7071                default:
7072                        node_dhc->auth_cfg.authentication_type_priority[i] =
7073                            0;
7074                        break;
7075                }

7077                switch (auth_cfg->hash_priority[i]) {
7078                case ELX_SHA1:
7079                        node_dhc->auth_cfg.hash_priority[i] = AUTH_SHA1;
7080                        break;

7082                case ELX_MD5:
7083                        node_dhc->auth_cfg.hash_priority[i] = AUTH_MD5;
7084                        break;

7086                default:
7087                        node_dhc->auth_cfg.hash_priority[i] = 0;
7088                        break;
7089                }
7090        }

7092        for (i = 0; i < 8; i++) {
7093                switch (auth_cfg->dh_group_priority[i]) {
7094                case ELX_GROUP_NULL:
7095                        node_dhc->auth_cfg.dh_group_priority[i] = GROUP_NULL;
7096                        break;

7098                case ELX_GROUP_1024:
7099                        node_dhc->auth_cfg.dh_group_priority[i] = GROUP_1024;
7100                        break;

7102                case ELX_GROUP_1280:
7103                        node_dhc->auth_cfg.dh_group_priority[i] = GROUP_1280;
7104                        break;

7106                case ELX_GROUP_1536:
7107                        node_dhc->auth_cfg.dh_group_priority[i] = GROUP_1536;
7108                        break;

7110                case ELX_GROUP_2048:
7111                        node_dhc->auth_cfg.dh_group_priority[i] = GROUP_2048;
7112                        break;

7114                default:
7115                        node_dhc->auth_cfg.dh_group_priority[i] = 0xF;
7116                        break;
7117                }
7118        }

7120        /* Program the node's key */
7121        if (auth_key) {
7122                bcopy((uint8_t *)auth_key,
7123                    (uint8_t *)&node_dhc->auth_key,
```

```
7124                    sizeof (emlxs_auth_key_t));
7125                node_dhc->auth_key.next = NULL;
7126                node_dhc->auth_key.prev = NULL;

7128                bcopy((uint8_t *)&port->wwpn,
7129                    (uint8_t *)&node_dhc->auth_key.local_entity, 8);
7130                bcopy((uint8_t *)&ndlp->nlp_portname,
7131                    (uint8_t *)&node_dhc->auth_key.remote_entity,
7132                    8);
7133        }
7134        mutex_exit(&hba->auth_lock);

7136        node_dhc->nlp_auth_limit = 2;
7137        node_dhc->nlp_fb_vendor = 1;

7139        node_dhc->nlp_authrsp_tmocnt = 0;
7140        node_dhc->nlp_authrsp_tmo = 0;

7142        if (deferred_ubp) {
7143                /* Acknowledge the unsolicited PLOGI */
7144                /* This should trigger the other port to start authentication */
7145                if (emlxs_ub_send_login_acc(port,
7146                    (fc_unsol_buf_t *)deferred_ubp) != FC_SUCCESS) {
7147                        EMLXS_MSGF(EMLXS_CONTEXT,
7148                            &emlxs_fcsp_start_msg,
7149                            "Not started. Unable to send PLOGI ACC. did=0x%x",
7150                            ndlp->nlp_DID);

7152                        goto failed;
7153                }
7154                /* Start the auth rsp timer */
7155                node_dhc->nlp_authrsp_tmo = DRV_TIME +
7156                    node_dhc->auth_cfg.authentication_timeout;

7158                EMLXS_MSGF(EMLXS_CONTEXT,
7159                    &emlxs_fcsp_start_msg,
7160                    "Authrsp timer activated. did=0x%x",
7161                    ndlp->nlp_DID);

7163                /* The next state should be emlxs_rcv_auth_msg_unmapped_node */
7164                emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_SUCCESS, 0, 0);
7165        } else {
7166                node_dhc->nlp_auth_flag = 1;    /* host is the initiator */

7168                EMLXS_MSGF(EMLXS_CONTEXT,
7169                    &emlxs_fcsp_start_msg,
7170                    "Auth initiated. did=0x%x limit=%d sbp=%p",
7171                    ndlp->nlp_DID, node_dhc->nlp_auth_limit, deferred_sbp);

7173                if (emlxs_issue_auth_negotiate(port, ndlp, 0)) {
7174                        EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_start_msg,
7175                            "Failed. Auth initiation failed. did=0x%x",
7176                            ndlp->nlp_DID);

7178                        goto failed;
7179                }
7180        }

7182        return (0);

7184 failed:

7186        emlxs_dhc_state(port, ndlp, NODE_STATE_AUTH_FAILED, 0, 0);

7188        /* Complete authentication with failed status */
7189        emlxs_dhc_auth_complete(port, ndlp, 1);
```



```
7191        return (0);

7193 } /* emlxs_dhc_auth_start() */



7197 /* This is called to indicate the driver has lost connection with this node */
7198 extern void
7199 emlxs_dhc_auth_stop(
7200        emlxs_port_t *port,
7201        emlxs_node_t *ndlp)
7202 {
7203        emlxs_port_dhc_t *port_dhc = &port->port_dhc;
7204        emlxs_node_dhc_t *node_dhc;
7205        uint32_t i;

7207        if (port_dhc->state == ELX_FABRIC_STATE_UNKNOWN) {
7208                /* Nothing to stop */
7209                return;
7210        }
7211        if (ndlp) {
7212                node_dhc = &ndlp->node_dhc;

7214                if (node_dhc->state == NODE_STATE_UNKNOWN) {
7215                        /* Nothing to stop */
7216                        return;
7217                }
7218                if (ndlp->nlp_DID != FABRIC_DID) {
7219                        emlxs_dhc_state(port, ndlp, NODE_STATE_UNKNOWN, 0, 0);
7220                }
7221                emlxs_dhc_auth_complete(port, ndlp, 2);
7222        } else {        /* Lost connection to all nodes for this port */
7223                rw_enter(&port->node_rwlock, RW_READER);
7224                for (i = 0; i < EMLXS_NUM_HASH_QUES; i++) {
7225                        ndlp = port->node_table[i];

7227                        if (!ndlp) {
7228                                continue;
7229                        }
7230                        node_dhc = &ndlp->node_dhc;

7232                        if (node_dhc->state == NODE_STATE_UNKNOWN) {
7233                                continue;
7234                        }
7235                        if (ndlp->nlp_DID != FABRIC_DID) {
7236                                emlxs_dhc_state(port, ndlp,
7237                                    NODE_STATE_UNKNOWN, 0, 0);
7238                        }
7239                        emlxs_dhc_auth_complete(port, ndlp, 2);
7240                }
7241                rw_exit(&port->node_rwlock);
7242        }

7244        return;

7246 } /* emlxs_dhc_auth_stop */


7249 /* state = 0   - Successful completion. Continue connection to node */
7250 /* state = 1   - Failed completion. Do not continue with connection to node */
7251 /* state = 2   - Stopped completion. Do not continue with connection to node */

7253 static void
7254 emlxs_dhc_auth_complete(
7255                        emlxs_port_t *port,
```

```
7256                        emlxs_node_t *ndlp,
7257                        uint32_t status)
7258 {
7259         emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
7260         uint32_t fabric;
7261         uint32_t fabric_switch;

7263         fabric = ((ndlp->nlp_DID & FABRIC_DID_MASK) == FABRIC_DID_MASK) ? 1 : 0;
7264         fabric_switch = ((ndlp->nlp_DID == FABRIC_DID) ? 1 : 0);

7266         EMLXS_MSGF(EMLXS_CONTEXT,
7267             &emlxs_fcsp_complete_msg,
7268             "did=0x%x status=%d sbp=%p ubp=%p",
7269             ndlp->nlp_DID, status, node_dhc->deferred_sbp,
7270             node_dhc->deferred_ubp);

7272         if (status == 1) {
7273                 if (fabric_switch) {
7274                         /* Virtual link down */
7275                         (void) emlxs_port_offline(port, 0xffffffff);
7276                 } else if (!fabric) {
7277                         /* Port offline */
7278                         (void) emlxs_port_offline(port, ndlp->nlp_DID);
7279                 }
7280         }
7281         /* Send a LOGO if authentication was not successful */
7282         if (status == 1) {
7283                 EMLXS_MSGF(EMLXS_CONTEXT,
7284                     &emlxs_fcsp_complete_msg,
7285                     "Sending LOGO to did=0x%x...",
7286                     ndlp->nlp_DID);
7287                 emlxs_send_logo(port, ndlp->nlp_DID);
7288         }

7290         /* Process deferred cmpl now */
7291         emlxs_mb_deferred_cmpl(port, status,
7292             (emlxs_buf_t *)node_dhc->deferred_sbp,
7293             (fc_unsol_buf_t *)node_dhc->deferred_ubp, 0);

7295         node_dhc->deferred_sbp = 0;
7296         node_dhc->deferred_ubp = 0;

7298         return;

7300 } /* emlxs_dhc_auth_complete */


7303 extern void
7304 emlxs_dhc_attach(emlxs_hba_t *hba)
7305 {
7306         char buf[32];

7308         (void) sprintf(buf, "%s_auth_lock mutex", DRIVER_NAME);
7309         mutex_init(&hba->auth_lock, buf, MUTEX_DRIVER, NULL);

7311         (void) sprintf(buf, "%s_dhc_lock mutex", DRIVER_NAME);
7312         mutex_init(&hba->dhc_lock, buf, MUTEX_DRIVER, NULL);

7314         emlxs_auth_cfg_init(hba);

7316         emlxs_auth_key_init(hba);

7318         hba->rdn_flag = 1;

7320         return;
```



```
7322 } /* emlxs_dhc_attach() */


7325 extern void
7326 emlxs_dhc_detach(emlxs_hba_t *hba)
7327 {
7328         emlxs_auth_cfg_fini(hba);

7330         emlxs_auth_key_fini(hba);

7332         mutex_destroy(&hba->dhc_lock);
7333         mutex_destroy(&hba->auth_lock);

7335         return;

7337 } /* emlxs_dhc_detach() */


7340 extern void
7341 emlxs_dhc_init_sp(emlxs_port_t *port, uint32_t did, SERV_PARM *sp, char **msg)
7342 {
7343         emlxs_hba_t *hba = HBA;
7344         emlxs_config_t *cfg = &CFG;
7345         uint32_t fabric;
7346         uint32_t fabric_switch;
7347         emlxs_auth_cfg_t *auth_cfg = NULL;
7348         emlxs_auth_key_t *auth_key = NULL;

7350         fabric = ((did & FABRIC_DID_MASK) == FABRIC_DID_MASK) ? 1 : 0;
7351         fabric_switch = ((did == FABRIC_DID) ? 1 : 0);

7353         /* Return is authentication is not enabled */
7354         if (cfg[CFG_AUTH_ENABLE].current == 0) {
7355                 sp->cmn.fcsp_support = 0;
7356                 bcopy("fcsp:Disabled (0)", (void *) &msg[0],
7357                     sizeof ("fcsp:Disabled (0)"));
7358                 return;
7359         }

7361         if (port->vpi != 0 && cfg[CFG_AUTH_NPIV].current == 0) {
7362                 sp->cmn.fcsp_support = 0;
7363                 bcopy("fcsp:Disabled (npiv)", (void *) &msg[0],
7364                     sizeof ("fcsp:Disabled (npiv)"));
7365                 return;
7366         }
7367         if (!fabric_switch && fabric) {
7368                 sp->cmn.fcsp_support = 0;
7369                 bcopy("fcsp:Disabled (fs)", (void *) &msg[0],
7370                     sizeof ("fcsp:Disabled (fs)"));
7371                 return;
7372         }
7373         /* Return if fcsp support to this node is not enabled */
7374         if (!fabric_switch && cfg[CFG_AUTH_E2E].current == 0) {
7375                 sp->cmn.fcsp_support = 0;
7376                 bcopy("fcsp:Disabled (e2e)", (void *) &msg[0],
7377                     sizeof ("fcsp:Disabled (e2e)"));
7378                 return;
7379         }

7381         mutex_enter(&hba->auth_lock);
7382         if (fabric_switch) {
7383                 auth_cfg = emlxs_auth_cfg_find(port,
7384                     (uint8_t *)emlxs_fabric_wwn);
7385                 auth_key = emlxs_auth_key_find(port,
7386                     (uint8_t *)emlxs_fabric_wwn);
7387                 if ((!auth_cfg) || (!auth_key)) {
```

```
7388                        sp->cmn.fcsp_support = 0;
7389                        bcopy("fcsp:Disabled (1)", (void *) &msg[0],
7390                            sizeof ("fcsp:Disabled (1)"));
7391                        mutex_exit(&hba->auth_lock);
7392                        return;
7393                }
7394        }
7395        mutex_exit(&hba->auth_lock);

7397        sp->cmn.fcsp_support = 1;

7399        return;

7401 } /* emlxs_dhc_init_sp() */


7404 extern uint32_t
7405 emlxs_dhc_verify_login(emlxs_port_t *port, uint32_t sid, SERV_PARM *sp)
7406 {
7407        emlxs_hba_t *hba = HBA;
7408        emlxs_config_t *cfg = &CFG;
7409        emlxs_auth_cfg_t *auth_cfg;
7410        emlxs_auth_key_t *auth_key;
7411        uint32_t fabric;
7412        uint32_t fabric_switch;

7414        fabric = ((sid & FABRIC_DID_MASK) == FABRIC_DID_MASK) ? 1 : 0;
7415        fabric_switch = ((sid == FABRIC_DID) ? 1 : 0);

7417        if (port->port_dhc.state == ELX_FABRIC_AUTH_FAILED) {
7418                /* Reject login */
7419                return (1);
7420        }
7421        /* Remote host supports FCSP */
7422        if (sp->cmn.fcsp_support) {
7423                /* Continue login */
7424                return (0);
7425        }
7426        /* Auth disabled in host */
7427        if (cfg[CFG_AUTH_ENABLE].current == 0) {
7428                /* Continue login */
7429                return (0);
7430        }
7431        /* Auth disabled for npiv */
7432        if (port->vpi != 0 && cfg[CFG_AUTH_NPIV].current == 0) {
7433                /* Continue login */
7434                return (0);
7435        }
7436        if (!fabric_switch && fabric) {
7437                /* Continue login */
7438                return (0);
7439        }
7440        /* Auth disabled for p2p */
7441        if (!fabric_switch && cfg[CFG_AUTH_E2E].current == 0) {
7442                /* Continue login */
7443                return (0);
7444        }

7446        /* Remote port does NOT support FCSP */
7447        /* Host has FCSP enabled */
7448        /* Now check to make sure auth mode for this port is also enabled */

7450        mutex_enter(&hba->auth_lock);

7452        /* Acquire auth configuration */
7453        if (fabric_switch) {
```



```
7454                auth_cfg = emlxs_auth_cfg_find(port,
7455                    (uint8_t *)emlxs_fabric_wwn);
7456                auth_key = emlxs_auth_key_find(port,
7457                    (uint8_t *)emlxs_fabric_wwn);
7458        } else {
7459                auth_cfg = emlxs_auth_cfg_find(port,
7460                    (uint8_t *)&sp->portName);
7461                auth_key = emlxs_auth_key_find(port,
7462                    (uint8_t *)&sp->portName);
7463        }

7465        if (auth_key && auth_cfg &&
7466            (auth_cfg->authentication_mode == AUTH_MODE_ACTIVE)) {
7467                mutex_exit(&hba->auth_lock);

7469                /* Reject login */
7470                return (1);
7471        }
7472        mutex_exit(&hba->auth_lock);

7474        return (0);

7476 } /* emlxs_dhc_verify_login() */


7479 /*
7480  * ! emlxs_dhc_reauth_timeout
7481  *
7482  * \pre \post \param phba \param arg1: \param arg2: ndlp to which the host
7483  * is to be authenticated. \return void
7484  *
7485  * \b Description:
7486  *
7487  * Timeout handler for reauthentication heartbeat.
7488  *
7489  * The reauthentication heart beat will be triggered 1 min by default after
7490  * the first authentication success. reauth_intval is
7491  * configurable. if reauth_intval is set to zero, it means no reauth heart
7492  * beat anymore.
7493  *
7494  * reauth heart beat will be triggered by IOCTL call from user space. Reauth
7495  * heart beat will go through the authentication process
7496  * all over again without causing IO traffic disruption. Initially it should
7497  * be triggered after authentication success.
7498  * Subsequently disable/enable reauth heart beat will be performed by
7499  * HBAnyware or other utility.
7500  *
7501  */
7502 /* ARGSUSED */
7503 extern void
7504 emlxs_dhc_reauth_timeout(
7505        emlxs_port_t *port,
7506        void *arg1,
7507        void *arg2)
7508 {
7509        emlxs_port_dhc_t *port_dhc = &port->port_dhc;
7510        NODELIST *ndlp = (NODELIST *) arg2;
7511        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;

7513        if (node_dhc->auth_cfg.reauthenticate_time_interval == 0) {
7514                EMLXS_MSGF(EMLXS_CONTEXT,
7515                    &emlxs_fcsp_debug_msg,
7516                    "Reauth timeout. Reauth no longer enabled. 0x%x %x",
7517                    ndlp->nlp_DID, node_dhc->state);

7519                emlxs_dhc_set_reauth_time(port, ndlp, DISABLE);
```

```
7521                return;
7522        }
7523        /* This should not happen!! */
7524        if (port_dhc->state == ELX_FABRIC_IN_AUTH) {
7525                EMLXS_MSGF(EMLXS_CONTEXT,
7526                    &emlxs_fcsp_error_msg,
7527                    "Reauth timeout. Fabric in auth. Quiting. 0x%x %x",
7528                    ndlp->nlp_DID, node_dhc->state);

7530                emlxs_dhc_set_reauth_time(port, ndlp, DISABLE);

7532                return;
7533        }
7534        if (node_dhc->state != NODE_STATE_AUTH_SUCCESS) {
7535                EMLXS_MSGF(EMLXS_CONTEXT,
7536                    &emlxs_fcsp_debug_msg,
7537                    "Reauth timeout. Auth not done. Restarting. 0x%x %x",
7538                    ndlp->nlp_DID, node_dhc->state);

7540                goto restart;
7541        }
7542        /*
7543         * This might happen, the ndlp is doing reauthencation. meaning ndlp
7544         * is being re-authenticated to the host. Thus not necessary to have
7545         * host re-authenticated to the ndlp at this point because ndlp might
7546         * support bi-directional auth. we can just simply donothing and
7547         * restart the timer.
7548         */
7549        if (port_dhc->state == ELX_FABRIC_IN_REAUTH) {
7550                EMLXS_MSGF(EMLXS_CONTEXT,
7551                    &emlxs_fcsp_debug_msg,
7552                    "Reauth timeout. Fabric in reauth. Restarting. 0x%x %x",
7553                    ndlp->nlp_DID, node_dhc->state);

7555                goto restart;
7556        }
7557        /*
7558         * node's reauth heart beat is running already, cancel it first and
7559         * then restart
7560         */
7561        if (node_dhc->nlp_reauth_status == NLP_HOST_REAUTH_IN_PROGRESS) {
7562                EMLXS_MSGF(EMLXS_CONTEXT,
7563                    &emlxs_fcsp_debug_msg,
7564                    "Reauth timeout. Fabric in reauth. Restarting. 0x%x %x",
7565                    ndlp->nlp_DID, node_dhc->state);

7567                goto restart;
7568        }
7569        EMLXS_MSGF(EMLXS_CONTEXT,
7570            &emlxs_fcsp_debug_msg,
7571            "Reauth timeout. Auth initiated. did=0x%x",
7572            ndlp->nlp_DID);

7574        emlxs_dhc_set_reauth_time(port, ndlp, ENABLE);
7575        node_dhc->nlp_reauth_status = NLP_HOST_REAUTH_IN_PROGRESS;

7577        /* Attempt to restart authentication */
7578        if (emlxs_dhc_auth_start(port, ndlp, NULL, NULL) != 0) {
7579                EMLXS_MSGF(EMLXS_CONTEXT,
7580                    &emlxs_fcsp_debug_msg,
7581                    "Reauth timeout. Auth initiation failed. 0x%x %x",
7582                    ndlp->nlp_DID, node_dhc->state);

7584                return;
7585        }
```



```
7586        return;

7588 restart:

7590        emlxs_dhc_set_reauth_time(port, ndlp, ENABLE);

7592        return;

7594 } /* emlxs_dhc_reauth_timeout */


7597 static void
7598 emlxs_dhc_set_reauth_time(
7599        emlxs_port_t *port,
7600        emlxs_node_t *ndlp,
7601        uint32_t status)
7602 {
7603        emlxs_port_dhc_t *port_dhc = &port->port_dhc;
7604        emlxs_node_dhc_t *node_dhc = &ndlp->node_dhc;
7605        uint32_t drv_time;
7606        uint32_t timeout;
7607        uint32_t reauth_tmo;
7608        uint32_t last_auth_time;
  26        time_t last_auth_time;

7610        node_dhc->flag &= ~NLP_SET_REAUTH_TIME;

7612        if ((status == ENABLE) &&
7613            node_dhc->auth_cfg.reauthenticate_time_interval) {

7615                timeout =
7616                    (60 * node_dhc->auth_cfg.reauthenticate_time_interval);
7617                drv_time = DRV_TIME;

7619                /* Get last successful auth time */
7620                if (ndlp->nlp_DID == FABRIC_DID) {
7621                        last_auth_time = port_dhc->auth_time;
7622                } else if (node_dhc->parent_auth_cfg) {
7623                        last_auth_time = node_dhc->parent_auth_cfg->auth_time;
7624                } else {
7625                        last_auth_time = 0;
7626                }

7628                if (last_auth_time) {
7629                        reauth_tmo = last_auth_time + timeout;

7631                        /* Validate reauth_tmo */
7632                        if ((reauth_tmo < drv_time) ||
7633                            (reauth_tmo > drv_time + timeout)) {
7634                                reauth_tmo = drv_time + timeout;
7635                        }
7636                } else {
7637                        reauth_tmo = drv_time + timeout;
7638                }

7640                node_dhc->nlp_reauth_tmo = reauth_tmo;
7641                node_dhc->nlp_reauth_status = NLP_HOST_REAUTH_ENABLED;

7643                EMLXS_MSGF(EMLXS_CONTEXT,
7644                    &emlxs_fcsp_debug_msg,
7645                    "Reauth enabled. did=0x%x state=%x tmo=%d,%d",
7646                    ndlp->nlp_DID, node_dhc->state,
7647                    node_dhc->auth_cfg.reauthenticate_time_interval,
7648                    (reauth_tmo - drv_time));

7650        } else {
```

```
7651                node_dhc->nlp_reauth_tmo = 0;
7652                node_dhc->nlp_reauth_status = NLP_HOST_REAUTH_DISABLED;

7654                EMLXS_MSGF(EMLXS_CONTEXT,
7655                    &emlxs_fcsp_debug_msg,
7656                    "Reauth disabled. did=0x%x state=%x",
7657                    ndlp->nlp_DID, node_dhc->state);
7658        }

7660        return;

7662 } /* emlxs_dhc_set_reauth_time */
______unchanged_portion_omitted_


9651 /* Provides DFC support for emlxs_dfc_get_auth_status() */
9652 extern uint32_t
9653 emlxs_dhc_get_auth_status(emlxs_hba_t *hba, dfc_auth_status_t *fcsp_status)
9654 {
9655        emlxs_port_t *port = &PPORT;
9656        emlxs_config_t *cfg = &CFG;
9657        char s_lwwpn[64];
9658        char s_rwwpn[64];
9659        emlxs_auth_cfg_t *auth_cfg;
9660        dfc_auth_status_t *auth_status;
9661        NODELIST *ndlp;
9662        uint32_t rc;
9663        uint32_t auth_time;
2081        time_t auth_time;
9664        uint32_t update;

9666        /* Return is authentication is not enabled */
9667        if (cfg[CFG_AUTH_ENABLE].current == 0) {
9668                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_fcsp_debug_msg,
9669                    "emlxs_dhc_get_auth_status. Auth disabled.");

9671                return (DFC_AUTH_AUTHENTICATION_DISABLED);
9672        }
9673        mutex_enter(&hba->auth_lock);

9675        auth_cfg = emlxs_auth_cfg_get(hba, (uint8_t *)&fcsp_status->lwwpn,
9676            (uint8_t *)&fcsp_status->rwwpn);

9678        if (!auth_cfg) {
9679                EMLXS_MSGF(EMLXS_CONTEXT, &emlxs_dfc_error_msg,
9680                    "emlxs_dhc_get_auth_status: entry not found. %s:%s",
9681                    emlxs_wwn_xlate(s_lwwpn, (uint8_t *)&fcsp_status->lwwpn),
9682                    emlxs_wwn_xlate(s_rwwpn, (uint8_t *)&fcsp_status->rwwpn));

9684                mutex_exit(&hba->auth_lock);

9686                return (DFC_AUTH_NOT_CONFIGURED);
9687        }
9688        if (bcmp((uint8_t *)&fcsp_status->rwwpn,
9689            (uint8_t *)emlxs_fabric_wwn, 8) == 0) {
9690                auth_status = &port->port_dhc.auth_status;
9691                auth_time = port->port_dhc.auth_time;
9692                ndlp = emlxs_node_find_did(port, FABRIC_DID);
9693        } else {
9694                auth_status = &auth_cfg->auth_status;
9695                auth_time = auth_cfg->auth_time;
9696                ndlp = auth_cfg->node;
9697        }

9699        update = 0;
```



```
9701        /* Check if node is still available */
9702        if (ndlp && ndlp->nlp_active) {
9703                emlxs_dhc_status(port, ndlp, 0, 0);
9704                update = 1;
9705        } else {
9706                rc = DFC_AUTH_WWN_NOT_FOUND;
9707        }


9710        if (update) {
9711                fcsp_status->auth_state = auth_status->auth_state;
9712                fcsp_status->auth_failReason = auth_status->auth_failReason;
9713                fcsp_status->type_priority = auth_status->type_priority;
9714                fcsp_status->group_priority = auth_status->group_priority;
9715                fcsp_status->hash_priority = auth_status->hash_priority;
9716                fcsp_status->localAuth = auth_status->localAuth;
9717                fcsp_status->remoteAuth = auth_status->remoteAuth;
9718                fcsp_status->time_from_last_auth = DRV_TIME - auth_time;
9719                fcsp_status->time_until_next_auth =
9720                    auth_status->time_until_next_auth;

9722                rc = 0;
9723        } else {
9724                rc = DFC_AUTH_WWN_NOT_FOUND;
9725        }

9727        mutex_exit(&hba->auth_lock);

9729        return (rc);

9731 } /* emlxs_dhc_get_auth_status() */
______unchanged_portion_omitted_
```

```
**********************************************************
  275456 Mon May  5 11:11:23 2014
new/usr/src/uts/common/io/fibre-channel/fca/emlxs/emlxs_solaris.c
4786 emlxs shouldn't abuse ddi_get_time(9f)
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Emulex.  All rights reserved.
 * Use is subject to license terms.
 * Copyright (c) 2011 Bayard G. Bell. All rights reserved.
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
#endif /* ! codereview */
 */


#define DEF_ICFG        1

#include <emlxs.h>
#include <emlxs_version.h>


char emlxs_revision[] = EMLXS_REVISION;
char emlxs_version[] = EMLXS_VERSION;
char emlxs_name[] = EMLXS_NAME;
char emlxs_label[] = EMLXS_LABEL;

/* Required for EMLXS_CONTEXT in EMLXS_MSGF calls */
EMLXS_MSG_DEF(EMLXS_SOLARIS_C);

#ifdef MENLO_SUPPORT
static int32_t  emlxs_send_menlo(emlxs_port_t *port, emlxs_buf_t *sbp);
#endif /* MENLO_SUPPORT */

static void     emlxs_fca_attach(emlxs_hba_t *hba);
static void     emlxs_fca_detach(emlxs_hba_t *hba);
static void     emlxs_drv_banner(emlxs_hba_t *hba);

static int32_t  emlxs_get_props(emlxs_hba_t *hba);
static int32_t  emlxs_send_fcp_cmd(emlxs_port_t *port, emlxs_buf_t *sbp,
                    uint32_t *pkt_flags);
static int32_t  emlxs_send_fct_status(emlxs_port_t *port, emlxs_buf_t *sbp);
static int32_t  emlxs_send_fct_abort(emlxs_port_t *port, emlxs_buf_t *sbp);
static int32_t  emlxs_send_ip(emlxs_port_t *port, emlxs_buf_t *sbp);
static int32_t  emlxs_send_els(emlxs_port_t *port, emlxs_buf_t *sbp);
static int32_t  emlxs_send_els_rsp(emlxs_port_t *port, emlxs_buf_t *sbp);
static int32_t  emlxs_send_ct(emlxs_port_t *port, emlxs_buf_t *sbp);
```



```
static int32_t  emlxs_send_ct_rsp(emlxs_port_t *port, emlxs_buf_t *sbp);
static uint32_t emlxs_add_instance(int32_t ddiinst);
static void     emlxs_iodone(emlxs_buf_t *sbp);
static int      emlxs_pm_lower_power(dev_info_t *dip);
static int      emlxs_pm_raise_power(dev_info_t *dip);
static void     emlxs_driver_remove(dev_info_t *dip, uint32_t init_flag,
                    uint32_t failed);
static void     emlxs_iodone_server(void *arg1, void *arg2, void *arg3);
static uint32_t emlxs_integrity_check(emlxs_hba_t *hba);
static uint32_t emlxs_test(emlxs_hba_t *hba, uint32_t test_code,
                    uint32_t args, uint32_t *arg);

#if (EMLXS_MODREV >= EMLXS_MODREV3) && (EMLXS_MODREV <= EMLXS_MODREV4)
static void     emlxs_read_vport_prop(emlxs_hba_t *hba);
#endif  /* EMLXS_MODREV3 || EMLXS_MODREV4 */



extern int
emlxs_msiid_to_chan(emlxs_hba_t *hba, int msi_id);
extern int
emlxs_select_msiid(emlxs_hba_t *hba);

/*
 * Driver Entry Routines.
 */
static int32_t  emlxs_detach(dev_info_t *, ddi_detach_cmd_t);
static int32_t  emlxs_attach(dev_info_t *, ddi_attach_cmd_t);
static int32_t  emlxs_open(dev_t *, int32_t, int32_t, cred_t *);
static int32_t  emlxs_close(dev_t, int32_t, int32_t, cred_t *);
static int32_t  emlxs_ioctl(dev_t, int32_t, intptr_t, int32_t,
                    cred_t *, int32_t *);
static int32_t  emlxs_info(dev_info_t *, ddi_info_cmd_t, void *, void **);


/*
 * FC_AL Transport Functions.
 */
static opaque_t emlxs_fca_bind_port(dev_info_t *, fc_fca_port_info_t *,
                    fc_fca_bind_info_t *);
static void     emlxs_fca_unbind_port(opaque_t);
static void     emlxs_initialize_pkt(emlxs_port_t *, emlxs_buf_t *);
static int32_t  emlxs_fca_get_cap(opaque_t, char *, void *);
static int32_t  emlxs_fca_set_cap(opaque_t, char *, void *);
static int32_t  emlxs_fca_get_map(opaque_t, fc_lilpmap_t *);
static int32_t  emlxs_fca_ub_alloc(opaque_t, uint64_t *, uint32_t,
                    uint32_t *, uint32_t);
static int32_t  emlxs_fca_ub_free(opaque_t, uint32_t, uint64_t *);

static opaque_t emlxs_fca_get_device(opaque_t, fc_portid_t);
static int32_t  emlxs_fca_notify(opaque_t, uint32_t);
static void     emlxs_ub_els_reject(emlxs_port_t *, fc_unsol_buf_t *);

/*
 * Driver Internal Functions.
 */

static void     emlxs_poll(emlxs_port_t *, emlxs_buf_t *);
static int32_t  emlxs_power(dev_info_t *, int32_t, int32_t);
#ifdef EMLXS_I386
#ifdef S11
static int32_t  emlxs_quiesce(dev_info_t *);
#endif
#endif
static int32_t  emlxs_hba_resume(dev_info_t *);
static int32_t  emlxs_hba_suspend(dev_info_t *);
```

```
128 static int32_t  emlxs_hba_detach(dev_info_t *);
129 static int32_t  emlxs_hba_attach(dev_info_t *);
130 static void     emlxs_lock_destroy(emlxs_hba_t *);
131 static void     emlxs_lock_init(emlxs_hba_t *);

133 char *emlxs_pm_components[] = {
134         "NAME=emlxx000",
135         "0=Device D3 State",
136         "1=Device D0 State"
137 };


140 /*
141  * Default emlx dma limits
142  */
143 ddi_dma_lim_t emlxs_dma_lim = {
144         (uint32_t)0,                            /* dlim_addr_lo */
145         (uint32_t)0xffffffff,                   /* dlim_addr_hi */
146         (uint_t)0x00ffffff,                     /* dlim_cntr_max */
147         DEFAULT_BURSTSIZE | BURST32 | BURST64,  /* dlim_burstsizes */
148         1,                                      /* dlim_minxfer */
149         0x00ffffff                              /* dlim_dmaspeed */
150 };

152 /*
153  * Be careful when using these attributes; the defaults listed below are
154  * (almost) the most general case, permitting allocation in almost any
155  * way supported by the LightPulse family.  The sole exception is the
156  * alignment specified as requiring memory allocation on a 4-byte boundary;
157  * the Lightpulse can DMA memory on any byte boundary.
158  *
159  * The LightPulse family currently is limited to 16M transfers;
160  * this restriction affects the dma_attr_count_max and dma_attr_maxxfer fields.
161  */
162 ddi_dma_attr_t emlxs_dma_attr = {
163         DMA_ATTR_V0,                            /* dma_attr_version */
164         (uint64_t)0,                            /* dma_attr_addr_lo */
165         (uint64_t)0xffffffffffffffff,           /* dma_attr_addr_hi */
166         (uint64_t)0x00ffffff,                   /* dma_attr_count_max */
167         1,                                      /* dma_attr_align */
168         DEFAULT_BURSTSIZE | BURST32 | BURST64,  /* dma_attr_burstsizes */
169         1,                                      /* dma_attr_minxfer */
170         (uint64_t)0x00ffffff,                   /* dma_attr_maxxfer */
171         (uint64_t)0xffffffff,                   /* dma_attr_seg */
172         EMLXS_SGLLEN,                           /* dma_attr_sgllen */
173         1,                                      /* dma_attr_granular */
174         0                                       /* dma_attr_flags */
175 };

177 ddi_dma_attr_t emlxs_dma_attr_ro = {
178         DMA_ATTR_V0,                            /* dma_attr_version */
179         (uint64_t)0,                            /* dma_attr_addr_lo */
180         (uint64_t)0xffffffffffffffff,           /* dma_attr_addr_hi */
181         (uint64_t)0x00ffffff,                   /* dma_attr_count_max */
182         1,                                      /* dma_attr_align */
183         DEFAULT_BURSTSIZE | BURST32 | BURST64,  /* dma_attr_burstsizes */
184         1,                                      /* dma_attr_minxfer */
185         (uint64_t)0x00ffffff,                   /* dma_attr_maxxfer */
186         (uint64_t)0xffffffff,                   /* dma_attr_seg */
187         EMLXS_SGLLEN,                           /* dma_attr_sgllen */
188         1,                                      /* dma_attr_granular */
189         DDI_DMA_RELAXED_ORDERING                /* dma_attr_flags */
190 };

192 ddi_dma_attr_t emlxs_dma_attr_1sg = {
193         DMA_ATTR_V0,                            /* dma_attr_version */
```



```
194         (uint64_t)0,                            /* dma_attr_addr_lo */
195         (uint64_t)0xffffffffffffffff,           /* dma_attr_addr_hi */
196         (uint64_t)0x00ffffff,                   /* dma_attr_count_max */
197         1,                                      /* dma_attr_align */
198         DEFAULT_BURSTSIZE | BURST32 | BURST64,  /* dma_attr_burstsizes */
199         1,                                      /* dma_attr_minxfer */
200         (uint64_t)0x00ffffff,                   /* dma_attr_maxxfer */
201         (uint64_t)0xffffffff,                   /* dma_attr_seg */
202         1,                                      /* dma_attr_sgllen */
203         1,                                      /* dma_attr_granular */
204         0                                       /* dma_attr_flags */
205 };

207 #if (EMLXS_MODREV >= EMLXS_MODREV3)
208 ddi_dma_attr_t emlxs_dma_attr_fcip_rsp = {
209         DMA_ATTR_V0,                            /* dma_attr_version */
210         (uint64_t)0,                            /* dma_attr_addr_lo */
211         (uint64_t)0xffffffffffffffff,           /* dma_attr_addr_hi */
212         (uint64_t)0x00ffffff,                   /* dma_attr_count_max */
213         1,                                      /* dma_attr_align */
214         DEFAULT_BURSTSIZE | BURST32 | BURST64,  /* dma_attr_burstsizes */
215         1,                                      /* dma_attr_minxfer */
216         (uint64_t)0x00ffffff,                   /* dma_attr_maxxfer */
217         (uint64_t)0xffffffff,                   /* dma_attr_seg */
218         EMLXS_SGLLEN,                           /* dma_attr_sgllen */
219         1,                                      /* dma_attr_granular */
220         0                                       /* dma_attr_flags */
221 };
222 #endif  /* >= EMLXS_MODREV3 */

224 /*
225  * DDI access attributes for device
226  */
227 ddi_device_acc_attr_t emlxs_dev_acc_attr = {
228         DDI_DEVICE_ATTR_V1,     /* devacc_attr_version          */
229         DDI_STRUCTURE_LE_ACC,   /* PCI is Little Endian         */
230         DDI_STRICTORDER_ACC,    /* devacc_attr_dataorder        */
231         DDI_DEFAULT_ACC         /* devacc_attr_access           */
232 };

234 /*
235  * DDI access attributes for data
236  */
237 ddi_device_acc_attr_t emlxs_data_acc_attr = {
238         DDI_DEVICE_ATTR_V1,     /* devacc_attr_version          */
239         DDI_NEVERSWAP_ACC,      /* don't swap for Data          */
240         DDI_STRICTORDER_ACC,    /* devacc_attr_dataorder        */
241         DDI_DEFAULT_ACC         /* devacc_attr_access           */
242 };

244 /*
245  * Fill in the FC Transport structure,
246  * as defined in the Fibre Channel Transport Programmming Guide.
247  */
248 #if (EMLXS_MODREV == EMLXS_MODREV5)
249         static fc_fca_tran_t emlxs_fca_tran = {
250         FCTL_FCA_MODREV_5,              /* fca_version, with SUN NPIV support */
251         MAX_VPORTS,                     /* fca numerb of ports */
252         sizeof (emlxs_buf_t),           /* fca pkt size */
253         2048,                           /* fca cmd max */
254         &emlxs_dma_lim,                 /* fca dma limits */
255         0,                              /* fca iblock, to be filled in later */
256         &emlxs_dma_attr,                /* fca dma attributes */
257         &emlxs_dma_attr_1sg,            /* fca dma fcp cmd attributes */
258         &emlxs_dma_attr_1sg,            /* fca dma fcp rsp attributes */
259         &emlxs_dma_attr_ro,             /* fca dma fcp data attributes */
```

```
260         &emlxs_dma_attr_1sg,            /* fca dma fcip cmd attributes */
261         &emlxs_dma_attr_fcip_rsp,       /* fca dma fcip rsp attributes */
262         &emlxs_dma_attr_1sg,            /* fca dma fcsm cmd attributes */
263         &emlxs_dma_attr,                /* fca dma fcsm rsp attributes */
264         &emlxs_data_acc_attr,           /* fca access atributes */
265         0,                              /* fca_num_npivports */
266         {0, 0, 0, 0, 0, 0, 0, 0},       /* Physical port WWPN */
267         emlxs_fca_bind_port,
268         emlxs_fca_unbind_port,
269         emlxs_fca_pkt_init,
270         emlxs_fca_pkt_uninit,
271         emlxs_fca_transport,
272         emlxs_fca_get_cap,
273         emlxs_fca_set_cap,
274         emlxs_fca_get_map,
275         emlxs_fca_transport,
276         emlxs_fca_ub_alloc,
277         emlxs_fca_ub_free,
278         emlxs_fca_ub_release,
279         emlxs_fca_pkt_abort,
280         emlxs_fca_reset,
281         emlxs_fca_port_manage,
282         emlxs_fca_get_device,
283         emlxs_fca_notify
284 };
285 #endif  /* EMLXS_MODREV5 */


288 #if (EMLXS_MODREV == EMLXS_MODREV4)
289 static fc_fca_tran_t emlxs_fca_tran = {
290         FCTL_FCA_MODREV_4,              /* fca_version */
291         MAX_VPORTS,                     /* fca numerb of ports */
292         sizeof (emlxs_buf_t),           /* fca pkt size */
293         2048,                           /* fca cmd max */
294         &emlxs_dma_lim,                 /* fca dma limits */
295         0,                              /* fca iblock, to be filled in later */
296         &emlxs_dma_attr,                /* fca dma attributes */
297         &emlxs_dma_attr_1sg,            /* fca dma fcp cmd attributes */
298         &emlxs_dma_attr_1sg,            /* fca dma fcp rsp attributes */
299         &emlxs_dma_attr_ro,             /* fca dma fcp data attributes */
300         &emlxs_dma_attr_1sg,            /* fca dma fcip cmd attributes */
301         &emlxs_dma_attr_fcip_rsp,       /* fca dma fcip rsp attributes */
302         &emlxs_dma_attr_1sg,            /* fca dma fcsm cmd attributes */
303         &emlxs_dma_attr,                /* fca dma fcsm rsp attributes */
304         &emlxs_data_acc_attr,           /* fca access atributes */
305         emlxs_fca_bind_port,
306         emlxs_fca_unbind_port,
307         emlxs_fca_pkt_init,
308         emlxs_fca_pkt_uninit,
309         emlxs_fca_transport,
310         emlxs_fca_get_cap,
311         emlxs_fca_set_cap,
312         emlxs_fca_get_map,
313         emlxs_fca_transport,
314         emlxs_fca_ub_alloc,
315         emlxs_fca_ub_free,
316         emlxs_fca_ub_release,
317         emlxs_fca_pkt_abort,
318         emlxs_fca_reset,
319         emlxs_fca_port_manage,
320         emlxs_fca_get_device,
321         emlxs_fca_notify
322 };
323 #endif  /* EMLXS_MODEREV4 */
```



```
326 #if (EMLXS_MODREV == EMLXS_MODREV3)
327 static fc_fca_tran_t emlxs_fca_tran = {
328         FCTL_FCA_MODREV_3,              /* fca_version */
329         MAX_VPORTS,                     /* fca numerb of ports */
330         sizeof (emlxs_buf_t),           /* fca pkt size */
331         2048,                           /* fca cmd max */
332         &emlxs_dma_lim,                 /* fca dma limits */
333         0,                              /* fca iblock, to be filled in later */
334         &emlxs_dma_attr,                /* fca dma attributes */
335         &emlxs_dma_attr_1sg,            /* fca dma fcp cmd attributes */
336         &emlxs_dma_attr_1sg,            /* fca dma fcp rsp attributes */
337         &emlxs_dma_attr_ro,             /* fca dma fcp data attributes */
338         &emlxs_dma_attr_1sg,            /* fca dma fcip cmd attributes */
339         &emlxs_dma_attr_fcip_rsp,       /* fca dma fcip rsp attributes */
340         &emlxs_dma_attr_1sg,            /* fca dma fcsm cmd attributes */
341         &emlxs_dma_attr,                /* fca dma fcsm rsp attributes */
342         &emlxs_data_acc_attr,           /* fca access atributes */
343         emlxs_fca_bind_port,
344         emlxs_fca_unbind_port,
345         emlxs_fca_pkt_init,
346         emlxs_fca_pkt_uninit,
347         emlxs_fca_transport,
348         emlxs_fca_get_cap,
349         emlxs_fca_set_cap,
350         emlxs_fca_get_map,
351         emlxs_fca_transport,
352         emlxs_fca_ub_alloc,
353         emlxs_fca_ub_free,
354         emlxs_fca_ub_release,
355         emlxs_fca_pkt_abort,
356         emlxs_fca_reset,
357         emlxs_fca_port_manage,
358         emlxs_fca_get_device,
359         emlxs_fca_notify
360 };
361 #endif  /* EMLXS_MODREV3 */


364 #if (EMLXS_MODREV == EMLXS_MODREV2)
365 static fc_fca_tran_t emlxs_fca_tran = {
366         FCTL_FCA_MODREV_2,              /* fca_version */
367         MAX_VPORTS,                     /* number of ports */
368         sizeof (emlxs_buf_t),           /* pkt size */
369         2048,                           /* max cmds */
370         &emlxs_dma_lim,                 /* DMA limits */
371         0,                              /* iblock, to be filled in later */
372         &emlxs_dma_attr,                /* dma attributes */
373         &emlxs_data_acc_attr,           /* access atributes */
374         emlxs_fca_bind_port,
375         emlxs_fca_unbind_port,
376         emlxs_fca_pkt_init,
377         emlxs_fca_pkt_uninit,
378         emlxs_fca_transport,
379         emlxs_fca_get_cap,
380         emlxs_fca_set_cap,
381         emlxs_fca_get_map,
382         emlxs_fca_transport,
383         emlxs_fca_ub_alloc,
384         emlxs_fca_ub_free,
385         emlxs_fca_ub_release,
386         emlxs_fca_pkt_abort,
387         emlxs_fca_reset,
388         emlxs_fca_port_manage,
389         emlxs_fca_get_device,
390         emlxs_fca_notify
391 };
```

```
392 #endif  /* EMLXS_MODREV2 */

394 /*
395  * state pointer which the implementation uses as a place to
396  * hang a set of per-driver structures;
397  *
398  */
399 void            *emlxs_soft_state = NULL;

401 /*
402  * Driver Global variables.
403  */
404 int32_t         emlxs_scsi_reset_delay = 3000;  /* milliseconds */

406 emlxs_device_t  emlxs_device;

408 uint32_t        emlxs_instance[MAX_FC_BRDS];    /* uses emlxs_device.lock */
409 uint32_t        emlxs_instance_count = 0;       /* uses emlxs_device.lock */
410 uint32_t        emlxs_instance_flag = 0;        /* uses emlxs_device.lock */
411 #define EMLXS_FW_SHOW           0x00000001


414 /*
415  * Single private "global" lock used to gain access to
416  * the hba_list and/or any other case where we want need to be
417  * single-threaded.
418  */
419 uint32_t        emlxs_diag_state;

421 /*
422  * CB ops vector.  Used for administration only.
423  */
424 static struct cb_ops emlxs_cb_ops = {
425         emlxs_open,     /* cb_open      */
426         emlxs_close,    /* cb_close     */
427         nodev,          /* cb_strategy  */
428         nodev,          /* cb_print     */
429         nodev,          /* cb_dump      */
430         nodev,          /* cb_read      */
431         nodev,          /* cb_write     */
432         emlxs_ioctl,    /* cb_ioctl     */
433         nodev,          /* cb_devmap    */
434         nodev,          /* cb_mmap      */
435         nodev,          /* cb_segmap    */
436         nochpoll,       /* cb_chpoll    */
437         ddi_prop_op,    /* cb_prop_op   */
438         0,              /* cb_stream    */
439 #ifdef _LP64
440         D_64BIT | D_HOTPLUG | D_MP | D_NEW,     /* cb_flag */
441 #else
442         D_HOTPLUG | D_MP | D_NEW,               /* cb_flag */
443 #endif
444         CB_REV,         /* rev          */
445         nodev,          /* cb_aread     */
446         nodev           /* cb_awrite    */
447 };

449 static struct dev_ops emlxs_ops = {
450         DEVO_REV,       /* rev */
451         0,      /* refcnt */
452         emlxs_info,     /* getinfo      */
453         nulldev,        /* identify     */
454         nulldev,        /* probe        */
455         emlxs_attach,   /* attach       */
456         emlxs_detach,   /* detach       */
457         nodev,          /* reset        */
```



```
458         &emlxs_cb_ops,  /* devo_cb_ops  */
459         NULL,           /* devo_bus_ops */
460         emlxs_power,    /* power ops    */
461 #ifdef EMLXS_I386
462 #ifdef S11
463         emlxs_quiesce,  /* quiesce      */
464 #endif
465 #endif
466 };

468 #include <sys/modctl.h>
469 extern struct mod_ops mod_driverops;

471 #ifdef SAN_DIAG_SUPPORT
472 extern kmutex_t         sd_bucket_mutex;
473 extern sd_bucket_info_t sd_bucket;
474 #endif /* SAN_DIAG_SUPPORT */

476 /*
477  * Module linkage information for the kernel.
478  */
479 static struct modldrv emlxs_modldrv = {
480         &mod_driverops, /* module type - driver */
481         emlxs_name,     /* module name */
482         &emlxs_ops,     /* driver ops */
483 };


486 /*
487  * Driver module linkage structure
488  */
489 static struct modlinkage emlxs_modlinkage = {
490         MODREV_1,       /* ml_rev - must be MODREV_1 */
491         &emlxs_modldrv, /* ml_linkage */
492         NULL    /* end of driver linkage */
493 };


496 /* We only need to add entries for non-default return codes. */
497 /* Entries do not need to be in order. */
498 /* Default:     FC_PKT_TRAN_ERROR,      FC_REASON_ABORTED, */
499 /*              FC_EXPLN_NONE,          FC_ACTION_RETRYABLE */

501 emlxs_xlat_err_t emlxs_iostat_tbl[] = {
502 /*      {f/w code, pkt_state, pkt_reason,       */
503 /*              pkt_expln, pkt_action}          */

505         /* 0x00 - Do not remove */
506         {IOSTAT_SUCCESS, FC_PKT_SUCCESS, FC_REASON_NONE,
507                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

509         /* 0x01 - Do not remove */
510         {IOSTAT_FCP_RSP_ERROR, FC_PKT_SUCCESS, FC_REASON_NONE,
511                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

513         /* 0x02 */
514         {IOSTAT_REMOTE_STOP, FC_PKT_REMOTE_STOP, FC_REASON_ABTS,
515                 FC_EXPLN_NONE, FC_ACTION_NON_RETRYABLE},

517         /*
518          * This is a default entry.
519          * The real codes are written dynamically in emlxs_els.c
520          */
521         /* 0x09 */
522         {IOSTAT_LS_RJT, FC_PKT_LS_RJT, FC_REASON_CMD_UNABLE,
523                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},
```

```
525         /* Special error code */
526         /* 0x10 */
527         {IOSTAT_DATA_OVERRUN, FC_PKT_TRAN_ERROR, FC_REASON_OVERRUN,
528                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

530         /* Special error code */
531         /* 0x11 */
532         {IOSTAT_DATA_UNDERRUN, FC_PKT_TRAN_ERROR, FC_REASON_ABORTED,
533                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

535         /* CLASS 2 only */
536         /* 0x04 */
537         {IOSTAT_NPORT_RJT, FC_PKT_NPORT_RJT, FC_REASON_PROTOCOL_ERROR,
538                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

540         /* CLASS 2 only */
541         /* 0x05 */
542         {IOSTAT_FABRIC_RJT, FC_PKT_FABRIC_RJT, FC_REASON_PROTOCOL_ERROR,
543                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

545         /* CLASS 2 only */
546         /* 0x06 */
547         {IOSTAT_NPORT_BSY, FC_PKT_NPORT_BSY, FC_REASON_PHYSICAL_BUSY,
548                 FC_EXPLN_NONE, FC_ACTION_SEQ_TERM_RETRY},

550         /* CLASS 2 only */
551         /* 0x07 */
552         {IOSTAT_FABRIC_BSY, FC_PKT_FABRIC_BSY, FC_REASON_FABRIC_BSY,
553                 FC_EXPLN_NONE, FC_ACTION_SEQ_TERM_RETRY},
554 };

556 #define IOSTAT_MAX (sizeof (emlxs_iostat_tbl)/sizeof (emlxs_xlat_err_t))


559 /* We only need to add entries for non-default return codes. */
560 /* Entries do not need to be in order. */
561 /* Default:     FC_PKT_TRAN_ERROR,      FC_REASON_ABORTED, */
562 /*              FC_EXPLN_NONE,          FC_ACTION_RETRYABLE} */

564 emlxs_xlat_err_t emlxs_ioerr_tbl[] = {
565 /*      {f/w code, pkt_state, pkt_reason,       */
566 /*              pkt_expln, pkt_action}          */

568         /* 0x01 */
569         {IOERR_MISSING_CONTINUE, FC_PKT_TRAN_ERROR, FC_REASON_OVERRUN,
570                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

572         /* 0x02 */
573         {IOERR_SEQUENCE_TIMEOUT, FC_PKT_TIMEOUT, FC_REASON_SEQ_TIMEOUT,
574                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

576         /* 0x04 */
577         {IOERR_INVALID_RPI, FC_PKT_PORT_OFFLINE, FC_REASON_OFFLINE,
578                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

580         /* 0x05 */
581         {IOERR_NO_XRI, FC_PKT_LOCAL_RJT, FC_REASON_XCHG_DROPPED,
582                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

584         /* 0x06 */
585         {IOERR_ILLEGAL_COMMAND, FC_PKT_LOCAL_RJT, FC_REASON_ILLEGAL_REQ,
586                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

588         /* 0x07 */
589         {IOERR_XCHG_DROPPED, FC_PKT_LOCAL_RJT,  FC_REASON_XCHG_DROPPED,
```



```
590                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

592         /* 0x08 */
593         {IOERR_ILLEGAL_FIELD, FC_PKT_LOCAL_RJT, FC_REASON_ILLEGAL_REQ,
594                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

596         /* 0x0B */
597         {IOERR_RCV_BUFFER_WAITING, FC_PKT_LOCAL_RJT, FC_REASON_NOMEM,
598                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

600         /* 0x0D */
601         {IOERR_TX_DMA_FAILED, FC_PKT_LOCAL_RJT, FC_REASON_DMA_ERROR,
602                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

604         /* 0x0E */
605         {IOERR_RX_DMA_FAILED, FC_PKT_LOCAL_RJT, FC_REASON_DMA_ERROR,
606                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

608         /* 0x0F */
609         {IOERR_ILLEGAL_FRAME, FC_PKT_LOCAL_RJT, FC_REASON_ILLEGAL_FRAME,
610                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

612         /* 0x11 */
613         {IOERR_NO_RESOURCES, FC_PKT_LOCAL_RJT,  FC_REASON_NOMEM,
614                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

616         /* 0x13 */
617         {IOERR_ILLEGAL_LENGTH, FC_PKT_LOCAL_RJT, FC_REASON_ILLEGAL_LENGTH,
618                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

620         /* 0x14 */
621         {IOERR_UNSUPPORTED_FEATURE, FC_PKT_LOCAL_RJT, FC_REASON_UNSUPPORTED,
622                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

624         /* 0x15 */
625         {IOERR_ABORT_IN_PROGRESS, FC_PKT_LOCAL_RJT, FC_REASON_ABORTED,
626                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

628         /* 0x16 */
629         {IOERR_ABORT_REQUESTED, FC_PKT_LOCAL_RJT, FC_REASON_ABORTED,
630                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

632         /* 0x17 */
633         {IOERR_RCV_BUFFER_TIMEOUT, FC_PKT_LOCAL_RJT, FC_REASON_RX_BUF_TIMEOUT,
634                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

636         /* 0x18 */
637         {IOERR_LOOP_OPEN_FAILURE, FC_PKT_LOCAL_RJT, FC_REASON_FCAL_OPN_FAIL,
638                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

640         /* 0x1A */
641         {IOERR_LINK_DOWN, FC_PKT_PORT_OFFLINE, FC_REASON_OFFLINE,
642                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

644         /* 0x21 */
645         {IOERR_BAD_HOST_ADDRESS, FC_PKT_LOCAL_RJT, FC_REASON_BAD_SID,
646                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

648         /* Occurs at link down */
649         /* 0x28 */
650         {IOERR_BUFFER_SHORTAGE, FC_PKT_PORT_OFFLINE, FC_REASON_OFFLINE,
651                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},

653         /* 0xF0 */
654         {IOERR_ABORT_TIMEOUT, FC_PKT_TIMEOUT, FC_REASON_SEQ_TIMEOUT,
655                 FC_EXPLN_NONE, FC_ACTION_RETRYABLE},
```

```
};

#define IOERR_MAX    (sizeof (emlxs_ioerr_tbl)/sizeof (emlxs_xlat_err_t))



emlxs_table_t emlxs_error_table[] = {
        {IOERR_SUCCESS, "No error."},
        {IOERR_MISSING_CONTINUE, "Missing continue."},
        {IOERR_SEQUENCE_TIMEOUT, "Sequence timeout."},
        {IOERR_INTERNAL_ERROR, "Internal error."},
        {IOERR_INVALID_RPI, "Invalid RPI."},
        {IOERR_NO_XRI, "No XRI."},
        {IOERR_ILLEGAL_COMMAND, "Illegal command."},
        {IOERR_XCHG_DROPPED, "Exchange dropped."},
        {IOERR_ILLEGAL_FIELD, "Illegal field."},
        {IOERR_RCV_BUFFER_WAITING, "RX buffer waiting."},
        {IOERR_TX_DMA_FAILED, "TX DMA failed."},
        {IOERR_RX_DMA_FAILED, "RX DMA failed."},
        {IOERR_ILLEGAL_FRAME, "Illegal frame."},
        {IOERR_NO_RESOURCES, "No resources."},
        {IOERR_ILLEGAL_LENGTH, "Illegal length."},
        {IOERR_UNSUPPORTED_FEATURE, "Unsupported feature."},
        {IOERR_ABORT_IN_PROGRESS, "Abort in progess."},
        {IOERR_ABORT_REQUESTED, "Abort requested."},
        {IOERR_RCV_BUFFER_TIMEOUT, "RX buffer timeout."},
        {IOERR_LOOP_OPEN_FAILURE, "Loop open failed."},
        {IOERR_RING_RESET, "Ring reset."},
        {IOERR_LINK_DOWN, "Link down."},
        {IOERR_CORRUPTED_DATA, "Corrupted data."},
        {IOERR_CORRUPTED_RPI, "Corrupted RPI."},
        {IOERR_OUT_OF_ORDER_DATA, "Out-of-order data."},
        {IOERR_OUT_OF_ORDER_ACK, "Out-of-order ack."},
        {IOERR_DUP_FRAME, "Duplicate frame."},
        {IOERR_LINK_CONTROL_FRAME, "Link control frame."},
        {IOERR_BAD_HOST_ADDRESS, "Bad host address."},
        {IOERR_RCV_HDRBUF_WAITING, "RX header buffer waiting."},
        {IOERR_MISSING_HDR_BUFFER, "Missing header buffer."},
        {IOERR_MSEQ_CHAIN_CORRUPTED, "MSEQ chain corrupted."},
        {IOERR_ABORTMULT_REQUESTED, "Abort multiple requested."},
        {IOERR_BUFFER_SHORTAGE, "Buffer shortage."},
        {IOERR_XRIBUF_WAITING, "XRI buffer shortage"},
        {IOERR_XRIBUF_MISSING, "XRI buffer missing"},
        {IOERR_ROFFSET_INVAL, "Relative offset invalid."},
        {IOERR_ROFFSET_MISSING, "Relative offset missing."},
        {IOERR_INSUF_BUFFER, "Buffer too small."},
        {IOERR_MISSING_SI, "ELS frame missing SI"},
        {IOERR_MISSING_ES, "Exhausted burst without ES"},
        {IOERR_INCOMP_XFER, "Transfer incomplete."},
        {IOERR_ABORT_TIMEOUT, "Abort timeout."}

};      /* emlxs_error_table */


emlxs_table_t emlxs_state_table[] = {
        {IOSTAT_SUCCESS, "Success."},
        {IOSTAT_FCP_RSP_ERROR, "FCP response error."},
        {IOSTAT_REMOTE_STOP, "Remote stop."},
        {IOSTAT_LOCAL_REJECT, "Local reject."},
        {IOSTAT_NPORT_RJT, "NPort reject."},
        {IOSTAT_FABRIC_RJT, "Fabric reject."},
        {IOSTAT_NPORT_BSY, "Nport busy."},
        {IOSTAT_FABRIC_BSY, "Fabric busy."},
        {IOSTAT_INTERMED_RSP, "Intermediate response."},
        {IOSTAT_LS_RJT, "LS reject."},
        {IOSTAT_CMD_REJECT,             "Cmd reject."},
```



```
        {IOSTAT_FCP_TGT_LENCHK, "TGT length check."},
        {IOSTAT_NEED_BUFF_ENTRY, "Need buffer entry."},
        {IOSTAT_DATA_UNDERRUN, "Data underrun."},
        {IOSTAT_DATA_OVERRUN,  "Data overrun."},

};      /* emlxs_state_table */


#ifdef MENLO_SUPPORT
emlxs_table_t emlxs_menlo_cmd_table[] = {
        {MENLO_CMD_INITIALIZE,          "MENLO_INIT"},
        {MENLO_CMD_FW_DOWNLOAD,         "MENLO_FW_DOWNLOAD"},
        {MENLO_CMD_READ_MEMORY,         "MENLO_READ_MEM"},
        {MENLO_CMD_WRITE_MEMORY,        "MENLO_WRITE_MEM"},
        {MENLO_CMD_FTE_INSERT,          "MENLO_FTE_INSERT"},
        {MENLO_CMD_FTE_DELETE,          "MENLO_FTE_DELETE"},

        {MENLO_CMD_GET_INIT,            "MENLO_GET_INIT"},
        {MENLO_CMD_GET_CONFIG,          "MENLO_GET_CONFIG"},
        {MENLO_CMD_GET_PORT_STATS,      "MENLO_GET_PORT_STATS"},
        {MENLO_CMD_GET_LIF_STATS,       "MENLO_GET_LIF_STATS"},
        {MENLO_CMD_GET_ASIC_STATS,      "MENLO_GET_ASIC_STATS"},
        {MENLO_CMD_GET_LOG_CONFIG,      "MENLO_GET_LOG_CFG"},
        {MENLO_CMD_GET_LOG_DATA,        "MENLO_GET_LOG_DATA"},
        {MENLO_CMD_GET_PANIC_LOG,       "MENLO_GET_PANIC_LOG"},
        {MENLO_CMD_GET_LB_MODE,         "MENLO_GET_LB_MODE"},

        {MENLO_CMD_SET_PAUSE,           "MENLO_SET_PAUSE"},
        {MENLO_CMD_SET_FCOE_COS,        "MENLO_SET_FCOE_COS"},
        {MENLO_CMD_SET_UIF_PORT_TYPE,   "MENLO_SET_UIF_TYPE"},

        {MENLO_CMD_DIAGNOSTICS,         "MENLO_DIAGNOSTICS"},
        {MENLO_CMD_LOOPBACK,            "MENLO_LOOPBACK"},

        {MENLO_CMD_RESET,               "MENLO_RESET"},
        {MENLO_CMD_SET_MODE,            "MENLO_SET_MODE"}

};      /* emlxs_menlo_cmd_table */

emlxs_table_t emlxs_menlo_rsp_table[] = {
        {MENLO_RSP_SUCCESS,             "SUCCESS"},
        {MENLO_ERR_FAILED,              "FAILED"},
        {MENLO_ERR_INVALID_CMD,         "INVALID_CMD"},
        {MENLO_ERR_INVALID_CREDIT,      "INVALID_CREDIT"},
        {MENLO_ERR_INVALID_SIZE,        "INVALID_SIZE"},
        {MENLO_ERR_INVALID_ADDRESS,     "INVALID_ADDRESS"},
        {MENLO_ERR_INVALID_CONTEXT,     "INVALID_CONTEXT"},
        {MENLO_ERR_INVALID_LENGTH,      "INVALID_LENGTH"},
        {MENLO_ERR_INVALID_TYPE,        "INVALID_TYPE"},
        {MENLO_ERR_INVALID_DATA,        "INVALID_DATA"},
        {MENLO_ERR_INVALID_VALUE1,      "INVALID_VALUE1"},
        {MENLO_ERR_INVALID_VALUE2,      "INVALID_VALUE2"},
        {MENLO_ERR_INVALID_MASK,        "INVALID_MASK"},
        {MENLO_ERR_CHECKSUM,            "CHECKSUM_ERROR"},
        {MENLO_ERR_UNKNOWN_FCID,        "UNKNOWN_FCID"},
        {MENLO_ERR_UNKNOWN_WWN,         "UNKNOWN_WWN"},
        {MENLO_ERR_BUSY,                "BUSY"},

};      /* emlxs_menlo_rsp_table */

#endif /* MENLO_SUPPORT */


emlxs_table_t emlxs_mscmd_table[] = {
        {SLI_CT_RESPONSE_FS_ACC, "CT_ACC"},
        {SLI_CT_RESPONSE_FS_RJT, "CT_RJT"},
```

```
788         {MS_GTIN, "MS_GTIN"},
789         {MS_GIEL, "MS_GIEL"},
790         {MS_GIET, "MS_GIET"},
791         {MS_GDID, "MS_GDID"},
792         {MS_GMID, "MS_GMID"},
793         {MS_GFN, "MS_GFN"},
794         {MS_GIELN, "MS_GIELN"},
795         {MS_GMAL, "MS_GMAL"},
796         {MS_GIEIL, "MS_GIEIL"},
797         {MS_GPL, "MS_GPL"},
798         {MS_GPT, "MS_GPT"},
799         {MS_GPPN, "MS_GPPN"},
800         {MS_GAPNL, "MS_GAPNL"},
801         {MS_GPS, "MS_GPS"},
802         {MS_GPSC, "MS_GPSC"},
803         {MS_GATIN, "MS_GATIN"},
804         {MS_GSES, "MS_GSES"},
805         {MS_GPLNL, "MS_GPLNL"},
806         {MS_GPLT, "MS_GPLT"},
807         {MS_GPLML, "MS_GPLML"},
808         {MS_GPAB, "MS_GPAB"},
809         {MS_GNPL, "MS_GNPL"},
810         {MS_GPNL, "MS_GPNL"},
811         {MS_GPFCP, "MS_GPFCP"},
812         {MS_GPLI, "MS_GPLI"},
813         {MS_GNID, "MS_GNID"},
814         {MS_RIELN, "MS_RIELN"},
815         {MS_RPL, "MS_RPL"},
816         {MS_RPLN, "MS_RPLN"},
817         {MS_RPLT, "MS_RPLT"},
818         {MS_RPLM, "MS_RPLM"},
819         {MS_RPAB, "MS_RPAB"},
820         {MS_RPFCP, "MS_RPFCP"},
821         {MS_RPLI, "MS_RPLI"},
822         {MS_DPL, "MS_DPL"},
823         {MS_DPLN, "MS_DPLN"},
824         {MS_DPLM, "MS_DPLM"},
825         {MS_DPLML, "MS_DPLML"},
826         {MS_DPLI, "MS_DPLI"},
827         {MS_DPAB, "MS_DPAB"},
828         {MS_DPALL, "MS_DPALL"}

830 };      /* emlxs_mscmd_table */


833 emlxs_table_t emlxs_ctcmd_table[] = {
834         {SLI_CT_RESPONSE_FS_ACC, "CT_ACC"},
835         {SLI_CT_RESPONSE_FS_RJT, "CT_RJT"},
836         {SLI_CTNS_GA_NXT, "GA_NXT"},
837         {SLI_CTNS_GPN_ID, "GPN_ID"},
838         {SLI_CTNS_GNN_ID, "GNN_ID"},
839         {SLI_CTNS_GCS_ID, "GCS_ID"},
840         {SLI_CTNS_GFT_ID, "GFT_ID"},
841         {SLI_CTNS_GSPN_ID, "GSPN_ID"},
842         {SLI_CTNS_GPT_ID, "GPT_ID"},
843         {SLI_CTNS_GID_PN, "GID_PN"},
844         {SLI_CTNS_GID_NN, "GID_NN"},
845         {SLI_CTNS_GIP_NN, "GIP_NN"},
846         {SLI_CTNS_GIPA_NN, "GIPA_NN"},
847         {SLI_CTNS_GSNN_NN, "GSNN_NN"},
848         {SLI_CTNS_GNN_IP, "GNN_IP"},
849         {SLI_CTNS_GIPA_IP, "GIPA_IP"},
850         {SLI_CTNS_GID_FT, "GID_FT"},
851         {SLI_CTNS_GID_PT, "GID_PT"},
852         {SLI_CTNS_RPN_ID, "RPN_ID"},
853         {SLI_CTNS_RNN_ID, "RNN_ID"},
```



```
854         {SLI_CTNS_RCS_ID, "RCS_ID"},
855         {SLI_CTNS_RFT_ID, "RFT_ID"},
856         {SLI_CTNS_RSPN_ID, "RSPN_ID"},
857         {SLI_CTNS_RPT_ID, "RPT_ID"},
858         {SLI_CTNS_RIP_NN, "RIP_NN"},
859         {SLI_CTNS_RIPA_NN, "RIPA_NN"},
860         {SLI_CTNS_RSNN_NN, "RSNN_NN"},
861         {SLI_CTNS_DA_ID, "DA_ID"},
862         {SLI_CT_LOOPBACK, "LOOPBACK"} /* Driver special */

864 };      /* emlxs_ctcmd_table */



868 emlxs_table_t emlxs_rmcmd_table[] = {
869         {SLI_CT_RESPONSE_FS_ACC, "CT_ACC"},
870         {SLI_CT_RESPONSE_FS_RJT, "CT_RJT"},
871         {CT_OP_GSAT, "RM_GSAT"},
872         {CT_OP_GHAT, "RM_GHAT"},
873         {CT_OP_GPAT, "RM_GPAT"},
874         {CT_OP_GDAT, "RM_GDAT"},
875         {CT_OP_GPST, "RM_GPST"},
876         {CT_OP_GDP, "RM_GDP"},
877         {CT_OP_GDPG, "RM_GDPG"},
878         {CT_OP_GEPS, "RM_GEPS"},
879         {CT_OP_GLAT, "RM_GLAT"},
880         {CT_OP_SSAT, "RM_SSAT"},
881         {CT_OP_SHAT, "RM_SHAT"},
882         {CT_OP_SPAT, "RM_SPAT"},
883         {CT_OP_SDAT, "RM_SDAT"},
884         {CT_OP_SDP, "RM_SDP"},
885         {CT_OP_SBBS, "RM_SBBS"},
886         {CT_OP_RPST, "RM_RPST"},
887         {CT_OP_VFW, "RM_VFW"},
888         {CT_OP_DFW, "RM_DFW"},
889         {CT_OP_RES, "RM_RES"},
890         {CT_OP_RHD, "RM_RHD"},
891         {CT_OP_UFW, "RM_UFW"},
892         {CT_OP_RDP, "RM_RDP"},
893         {CT_OP_GHDR, "RM_GHDR"},
894         {CT_OP_CHD, "RM_CHD"},
895         {CT_OP_SSR, "RM_SSR"},
896         {CT_OP_RSAT, "RM_RSAT"},
897         {CT_OP_WSAT, "RM_WSAT"},
898         {CT_OP_RSAH, "RM_RSAH"},
899         {CT_OP_WSAH, "RM_WSAH"},
900         {CT_OP_RACT, "RM_RACT"},
901         {CT_OP_WACT, "RM_WACT"},
902         {CT_OP_RKT, "RM_RKT"},
903         {CT_OP_WKT, "RM_WKT"},
904         {CT_OP_SSC, "RM_SSC"},
905         {CT_OP_QHBA, "RM_QHBA"},
906         {CT_OP_GST, "RM_GST"},
907         {CT_OP_GFTM, "RM_GFTM"},
908         {CT_OP_SRL, "RM_SRL"},
909         {CT_OP_SI, "RM_SI"},
910         {CT_OP_SRC, "RM_SRC"},
911         {CT_OP_GPB, "RM_GPB"},
912         {CT_OP_SPB, "RM_SPB"},
913         {CT_OP_RPB, "RM_RPB"},
914         {CT_OP_RAPB, "RM_RAPB"},
915         {CT_OP_GBC, "RM_GBC"},
916         {CT_OP_GBS, "RM_GBS"},
917         {CT_OP_SBS, "RM_SBS"},
918         {CT_OP_GANI, "RM_GANI"},
919         {CT_OP_GRV, "RM_GRV"},
```

```
920         {CT_OP_GAPBS, "RM_GAPBS"},
921         {CT_OP_APBC, "RM_APBC"},
922         {CT_OP_GDT, "RM_GDT"},
923         {CT_OP_GDLMI, "RM_GDLMI"},
924         {CT_OP_GANA, "RM_GANA"},
925         {CT_OP_GDLV, "RM_GDLV"},
926         {CT_OP_GWUP, "RM_GWUP"},
927         {CT_OP_GLM, "RM_GLM"},
928         {CT_OP_GABS, "RM_GABS"},
929         {CT_OP_SABS, "RM_SABS"},
930         {CT_OP_RPR, "RM_RPR"},
931         {SLI_CT_LOOPBACK, "LOOPBACK"} /* Driver special */

933 };      /* emlxs_rmcmd_table */


936 emlxs_table_t emlxs_elscmd_table[] = {
937         {ELS_CMD_ACC, "ACC"},
938         {ELS_CMD_LS_RJT, "LS_RJT"},
939         {ELS_CMD_PLOGI, "PLOGI"},
940         {ELS_CMD_FLOGI, "FLOGI"},
941         {ELS_CMD_LOGO, "LOGO"},
942         {ELS_CMD_ABTX, "ABTX"},
943         {ELS_CMD_RCS, "RCS"},
944         {ELS_CMD_RES, "RES"},
945         {ELS_CMD_RSS, "RSS"},
946         {ELS_CMD_RSI, "RSI"},
947         {ELS_CMD_ESTS, "ESTS"},
948         {ELS_CMD_ESTC, "ESTC"},
949         {ELS_CMD_ADVC, "ADVC"},
950         {ELS_CMD_RTV, "RTV"},
951         {ELS_CMD_RLS, "RLS"},
952         {ELS_CMD_ECHO, "ECHO"},
953         {ELS_CMD_TEST, "TEST"},
954         {ELS_CMD_RRQ, "RRQ"},
955         {ELS_CMD_REC, "REC"},
956         {ELS_CMD_PRLI, "PRLI"},
957         {ELS_CMD_PRLO, "PRLO"},
958         {ELS_CMD_SCN, "SCN"},
959         {ELS_CMD_TPLS, "TPLS"},
960         {ELS_CMD_GPRLO, "GPRLO"},
961         {ELS_CMD_GAID, "GAID"},
962         {ELS_CMD_FACT, "FACT"},
963         {ELS_CMD_FDACT, "FDACT"},
964         {ELS_CMD_NACT, "NACT"},
965         {ELS_CMD_NDACT, "NDACT"},
966         {ELS_CMD_QoSR, "QoSR"},
967         {ELS_CMD_RVCS, "RVCS"},
968         {ELS_CMD_PDISC, "PDISC"},
969         {ELS_CMD_FDISC, "FDISC"},
970         {ELS_CMD_ADISC, "ADISC"},
971         {ELS_CMD_FARP, "FARP"},
972         {ELS_CMD_FARPR, "FARPR"},
973         {ELS_CMD_FAN, "FAN"},
974         {ELS_CMD_RSCN, "RSCN"},
975         {ELS_CMD_SCR, "SCR"},
976         {ELS_CMD_LINIT, "LINIT"},
977         {ELS_CMD_RNID, "RNID"},
978         {ELS_CMD_AUTH, "AUTH"}

980 };      /* emlxs_elscmd_table */


983 /*
984  *
985  *      Device Driver Entry Routines
```



```
986  *
987  */

989 #ifdef MODSYM_SUPPORT
990 static void emlxs_fca_modclose();
991 static int  emlxs_fca_modopen();
992 emlxs_modsym_t emlxs_modsym;    /* uses emlxs_device.lock */

994 static int
995 emlxs_fca_modopen()
996 {
997         int err;

999         if (emlxs_modsym.mod_fctl) {
1000                return (0);
1001        }

1003        /* Leadville (fctl) */
1004        err = 0;
1005        emlxs_modsym.mod_fctl =
1006            ddi_modopen("misc/fctl", KRTLD_MODE_FIRST, &err);
1007        if (!emlxs_modsym.mod_fctl) {
1008                cmn_err(CE_WARN,
1009                    "?%s: misc/fctl: ddi_modopen misc/fctl failed: error=%d",
1010                    DRIVER_NAME, err);

1012                goto failed;
1013        }

1015        err = 0;
1016        /* Check if the fctl fc_fca_attach is present */
1017        emlxs_modsym.fc_fca_attach =
1018            (int (*)())ddi_modsym(emlxs_modsym.mod_fctl, "fc_fca_attach",
1019            &err);
1020        if ((void *)emlxs_modsym.fc_fca_attach == NULL) {
1021                cmn_err(CE_WARN,
1022                    "?%s: misc/fctl: fc_fca_attach not present", DRIVER_NAME);
1023                goto failed;
1024        }

1026        err = 0;
1027        /* Check if the fctl fc_fca_detach is present */
1028        emlxs_modsym.fc_fca_detach =
1029            (int (*)())ddi_modsym(emlxs_modsym.mod_fctl, "fc_fca_detach",
1030            &err);
1031        if ((void *)emlxs_modsym.fc_fca_detach == NULL) {
1032                cmn_err(CE_WARN,
1033                    "?%s: misc/fctl: fc_fca_detach not present", DRIVER_NAME);
1034                goto failed;
1035        }

1037        err = 0;
1038        /* Check if the fctl fc_fca_init is present */
1039        emlxs_modsym.fc_fca_init =
1040            (int (*)())ddi_modsym(emlxs_modsym.mod_fctl, "fc_fca_init", &err);
1041        if ((void *)emlxs_modsym.fc_fca_init == NULL) {
1042                cmn_err(CE_WARN,
1043                    "?%s: misc/fctl: fc_fca_init not present", DRIVER_NAME);
1044                goto failed;
1045        }

1047        return (0);

1049 failed:

1051        emlxs_fca_modclose();
```

```
1053         return (1);


1056 } /* emlxs_fca_modopen() */


1059 static void
1060 emlxs_fca_modclose()
1061 {
1062         if (emlxs_modsym.mod_fctl) {
1063                 (void) ddi_modclose(emlxs_modsym.mod_fctl);
1064                 emlxs_modsym.mod_fctl = 0;
1065         }

1067         emlxs_modsym.fc_fca_attach = NULL;
1068         emlxs_modsym.fc_fca_detach = NULL;
1069         emlxs_modsym.fc_fca_init   = NULL;

1071         return;

1073 } /* emlxs_fca_modclose() */

1075 #endif /* MODSYM_SUPPORT */



1079 /*
1080  * Global driver initialization, called once when driver is loaded
1081  */
1082 int
1083 _init(void)
1084 {
1085         int ret;
1086         char buf[64];

1088         /*
1089          * First init call for this driver,
1090          * so initialize the emlxs_dev_ctl structure.
1091          */
1092         bzero(&emlxs_device, sizeof (emlxs_device));

1094 #ifdef MODSYM_SUPPORT
1095         bzero(&emlxs_modsym, sizeof (emlxs_modsym_t));
1096 #endif /* MODSYM_SUPPORT */

1098         (void) sprintf(buf, "%s_device mutex", DRIVER_NAME);
1099         mutex_init(&emlxs_device.lock, buf, MUTEX_DRIVER, NULL);

1101         (void) drv_getparm(LBOLT, &emlxs_device.log_timestamp);
1102         emlxs_device.drv_timestamp = gethrtime();
  26         emlxs_device.drv_timestamp = ddi_get_time();

1104         for (ret = 0; ret < MAX_FC_BRDS; ret++) {
1105                 emlxs_instance[ret] = (uint32_t)-1;
1106         }

1108         /*
1109          * Provide for one ddiinst of the emlxs_dev_ctl structure
1110          * for each possible board in the system.
1111          */
1112         if ((ret = ddi_soft_state_init(&emlxs_soft_state,
1113             sizeof (emlxs_hba_t), MAX_FC_BRDS)) != 0) {
1114                 cmn_err(CE_WARN,
1115                     "?%s: _init: ddi_soft_state_init failed. rval=%x",
1116                     DRIVER_NAME, ret);
```



```
1118                 return (ret);
1119         }

1121 #ifdef MODSYM_SUPPORT
1122         /* Open SFS */
1123         (void) emlxs_fca_modopen();
1124 #endif /* MODSYM_SUPPORT */

1126         /* Setup devops for SFS */
1127         MODSYM(fc_fca_init)(&emlxs_ops);

1129         if ((ret = mod_install(&emlxs_modlinkage)) != 0) {
1130                 (void) ddi_soft_state_fini(&emlxs_soft_state);
1131 #ifdef MODSYM_SUPPORT
1132                 /* Close SFS */
1133                 emlxs_fca_modclose();
1134 #endif /* MODSYM_SUPPORT */

1136                 return (ret);
1137         }

1139 #ifdef SAN_DIAG_SUPPORT
1140         (void) sprintf(buf, "%s_sd_bucket mutex", DRIVER_NAME);
1141         mutex_init(&sd_bucket_mutex, buf, MUTEX_DRIVER, NULL);
1142 #endif /* SAN_DIAG_SUPPORT */

1144         return (ret);

1146 } /* _init() */
```

*unchanged_portion_omitted_*

****
1694 Mon May 5 11:11:23 2014
new/usr/src/uts/common/sys/fibre-channel/fca/emlxs/emlxs_device.h
4786 emlxs shouldn't abuse ddi_get_time(9f)
****

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2009 Emulex.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 */
#endif /* ! codereview */

#ifndef _EMLXS_DEVICE_H
#define _EMLXS_DEVICE_H

#ifdef  __cplusplus
extern "C" {
#endif

/*
 * This is the global device driver control structure
 */

#ifndef EMLXS_HBA_T
typedef struct emlxs_hba emlxs_hba_t;
#endif

/* This structure must match the one in ./mdb/msgblib.c */
typedef struct emlxs_device
{
        uint32_t hba_count;
        emlxs_hba_t *hba[MAX_FC_BRDS];
        kmutex_t lock;

        hrtime_t drv_timestamp;
        time_t drv_timestamp;
        clock_t log_timestamp;
        emlxs_msg_log_t *log[MAX_FC_BRDS];

#ifdef DUMP_SUPPORT
        emlxs_file_t *dump_txtfile[MAX_FC_BRDS];
        emlxs_file_t *dump_dmpfile[MAX_FC_BRDS];
        emlxs_file_t *dump_ceefile[MAX_FC_BRDS];
```



```
#endif /* DUMP_SUPPORT */

} emlxs_device_t;
```

*_____unchanged_portion_omitted_*

```
**********************************************************
   24349 Mon May  5 11:11:23 2014
new/usr/src/uts/common/sys/fibre-channel/fca/emlxs/emlxs_dhchap.h
4786 emlxs shouldn't abuse ddi_get_time(9f)
**********************************************************
```

```
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Emulex.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 */
#endif /* ! codereview */

#ifndef _EMLXS_DHCHAP_H
#define _EMLXS_DHCHAP_H

#ifdef  __cplusplus
extern "C" {
#endif

#ifdef DHCHAP_SUPPORT
#include <sys/random.h>


/* emlxs_auth_cfg_t */
#define PASSWORD_TYPE_ASCII     1
#define PASSWORD_TYPE_BINARY    2
#define PASSWORD_TYPE_IGNORE    3

#define AUTH_MODE_DISABLED      1
#define AUTH_MODE_ACTIVE        2
#define AUTH_MODE_PASSIVE       3

#define ELX_DHCHAP              0x01    /* Only one currently supported */
#define ELX_FCAP                0x02
#define ELX_FCPAP               0x03
#define ELX_KERBEROS            0x04

#define ELX_MD5                 0x01
#define ELX_SHA1                0x02

#define ELX_GROUP_NULL          0x01
#define ELX_GROUP_1024          0x02
#define ELX_GROUP_1280          0x03
```



```
#define ELX_GROUP_1536          0x04
#define ELX_GROUP_2048          0x05


/* AUTH_ELS Code */
#define ELS_CMD_AUTH_CODE       0x90

/* AUTH_ELS Flags */

/* state ? */
#define AUTH_FINISH             0xFF
#define AUTH_ABORT              0xFE

/* auth_msg code for DHCHAP */
#define AUTH_REJECT             0x0A
#define AUTH_NEGOTIATE          0x0B
#define AUTH_DONE               0x0C
#define DHCHAP_CHALLENGE        0x10
#define DHCHAP_REPLY            0x11
#define DHCHAP_SUCCESS          0x12

/* BIG ENDIAN and LITTLE ENDIAN */

/* authentication protocol identifiers */
#ifdef EMLXS_BIG_ENDIAN

#define AUTH_DHCHAP             0x00000001
#define AUTH_FCAP               0x00000002
#define AUTH_FCPAP              0x00000003
#define AUTH_KERBEROS           0x00000004

#define HASH_LIST_TAG           0x0001
#define DHGID_LIST_TAG          0x0002

/* hash function identifiers */
#define AUTH_SHA1               0x00000006
#define AUTH_MD5                0x00000005

/* DHCHAP group ids */
#define GROUP_NULL              0x00000000
#define GROUP_1024              0x00000001
#define GROUP_1280              0x00000002
#define GROUP_1536              0x00000003
#define GROUP_2048              0x00000004

/* Tran_id Mask */
#define AUTH_TRAN_ID_MASK       0x000000FF

#endif  /* EMLXS_BIG_ENDIAN */

#ifdef EMLXS_LITTLE_ENDIAN

#define AUTH_DHCHAP             0x01000000
#define AUTH_FCAP               0x02000000
#define AUTH_FCPAP              0x03000000
#define AUTH_KERBEROS           0x04000000

#define HASH_LIST_TAG           0x0100
#define DHGID_LIST_TAG          0x0200

/* hash function identifiers */
#define AUTH_SHA1               0x06000000
#define AUTH_MD5                0x05000000

/* DHCHAP group ids */
#define GROUP_NULL              0x00000000
```

```
128 #define GROUP_1024              0x01000000
129 #define GROUP_1280              0x02000000
130 #define GROUP_1536              0x03000000
131 #define GROUP_2048              0x04000000

133 /* Tran_id Mask */
134 #define AUTH_TRAN_ID_MASK       0xFF000000

136 #endif  /* EMLXS_LITTLE_ENDIAN */

138 /* hash funcs hash length in byte */
139 #define SHA1_LEN                0x00000014      /* 20 bytes */
140 #define MD5_LEN                 0x00000010      /* 16 bytes */

142 #define HBA_SECURITY                    0x20

144 /* AUTH_Reject Reason Codes */
145 #define AUTHRJT_FAILURE                 0x01
146 #define AUTHRJT_LOGIC_ERR               0x02

148 /* LS_RJT Reason Codes for AUTH_ELS */
149 #define LSRJT_AUTH_REQUIRED             0x03
150 #define LSRJT_AUTH_LOGICAL_BSY          0x05
151 #define LSRJT_AUTH_ELS_NOT_SUPPORTED    0x0B
152 #define LSRJT_AUTH_NOT_LOGGED_IN        0x09

154 /* AUTH_Reject Reason Code Explanations */
155 #define AUTHEXP_MECH_UNUSABLE           0x01 /* AUTHRJT_LOGIC_ERR */
156 #define AUTHEXP_DHGROUP_UNUSABLE        0x02 /* AUTHRJT_LOGIC_ERR */
157 #define AUTHEXP_HASHFUNC_UNUSABLE       0x03 /* AUTHRJT_LOGIC_ERR */
158 #define AUTHEXP_AUTHTRAN_STARTED        0x04 /* AUTHRJT_LOGIC_ERR */
159 #define AUTHEXP_AUTH_FAILED             0x05 /* AUTHRJT_FAILURE */
160 #define AUTHEXP_BAD_PAYLOAD             0x06 /* AUTHRJT_FAILURE */
161 #define AUTHEXP_BAD_PROTOCOL            0x07 /* AUTHRJT_FAILURE */
162 #define AUTHEXP_RESTART_AUTH            0x08 /* AUTHRJT_LOGIC_ERR */
163 #define AUTHEXP_CONCAT_UNSUPP           0x09 /* AUTHRJT_LOGIC_ERR */
164 #define AUTHEXP_BAD_PROTOVERS           0x0A /* AUTHRJT_LOGIC_ERR */

166 /* LS_RJT Reason Code Explanations for AUTH_ELS */
167 #define LSEXP_AUTH_REQUIRED             0x48
168 #define LSEXP_AUTH_ELS_NOT_SUPPORTED    0x2C
169 #define LSEXP_AUTH_ELS_NOT_LOGGED_IN    0x1E
170 #define LSEXP_AUTH_LOGICAL_BUSY         0x00


173 #define MAX_AUTH_MSA_SIZE 1024

175 #define MAX_AUTH_PID    0x4     /* Max auth proto identifier list */

177 /* parameter tag */
178 #define HASH_LIST       0x0001
179 #define DHG_ID_LIST     0x0002

181 /* name tag from Table 13 v1.8 pp 30 */
182 #ifdef EMLXS_BIG_ENDIAN
183 #define AUTH_NAME_ID            0x0001
184 #define AUTH_NAME_LEN           0x0008
185 #define AUTH_PROTO_NUM          0x00000001
186 #define AUTH_NULL_PARA_LEN      0x00000028
187 #endif  /* EMLXS_BIG_ENDIAN */

189 #ifdef EMLXS_LITTLE_ENDIAN
190 #define AUTH_NAME_ID            0x0100
191 #define AUTH_NAME_LEN           0x0800
192 #define AUTH_PROTO_NUM          0x01000000
193 #define AUTH_NULL_PARA_LEN      0x28000000
```



```
194 #endif  /* EMLXS_LITTLE_ENDIAN */

196 /* name tag from Table 103 v 1.8 pp 123 */
197 #define AUTH_NODE_NAME          0x0002
198 #define AUTH_PORT_NAME          0x0003

200 /*
201  * Sysevent support
202  */
203 /* ddi_log_sysevent() vendors */
204 #define DDI_VENDOR_EMLX         "EMLXS"

206 /* Class */
207 #define EC_EMLXS        "EC_emlxs"

209 /* Subclass */
210 #define ESC_EMLXS_01    "ESC_emlxs_issue_auth_negotiate"
211 #define ESC_EMLXS_02    "ESC_emlxs_cmpl_auth_negotiate_issue"

213 #define ESC_EMLXS_03    "ESC_emlxs_rcv_auth_msg_auth_negotiate_issue"
214 #define ESC_EMLXS_04    "ESC_emlxs_cmpl_auth_msg_auth_negotiate_issue"

216 #define ESC_EMLXS_05    "ESC_emlxs_rcv_auth_msg_unmapped_node"
217 #define ESC_EMLXS_06    "ESC_emlxs_issue_dhchap_challenge"
218 #define ESC_EMLXS_07    "ESC_emlxs_cmpl_dhchap_challenge_issue"

220 #define ESC_EMLXS_08    "ESC_emlxs_rcv_auth_msg_dhchap_challenge_cmpl_wait4next"

222 #define ESC_EMLXS_09    "ESC_emlxs_rcv_auth_msg_auth_negotiate_rcv"
223 #define ESC_EMLXS_10    "ESC_emlxs_cmpl_auth_msg_auth_negotiate_rcv"

225 #define ESC_EMLXS_11    "ESC_emlxs_cmpl_cmpl_dhchap_reply_issue"
226 #define ESC_EMLXS_12    "ESC_emlxs_cmpl_dhchap_reply_issue"
227 #define ESC_EMLXS_13    "ESC_emlxs_cmpl_auth_msg_dhchap_reply_issue"

229 #define ESC_EMLXS_14    "ESC_emlxs_cmpl_auth_msg_auth_negotiate_cmpl_wait4next"

231 #define ESC_EMLXS_15    "ESC_emlxs_issue_dhchap_success"

233 #define ESC_EMLXS_16    "ESC_emlxs_rcv_auth_msg_dhchap_challenge_issue"
234 #define ESC_EMLXS_17    "ESC_emlxs_cmpl_auth_msg_dhchap_challenge_issue"

236 #define ESC_EMLXS_18    "ESC_emlxs_rcv_auth_msg_dhchap_reply_issue"

238 #define ESC_EMLXS_19 \
239         "ESC_emlxs_cmpl_auth_msg_dhchap_challenge_cmpl_wait4next"

241 #define ESC_EMLXS_20    "ESC_emlxs_rcv_auth_msg_dhchap_reply_cmpl_wait4next"
242 #define ESC_EMLXS_21    "ESC_emlxs_cmpl_dhchap_success_issue"
243 #define ESC_EMLXS_22    "ESC_emlxs_cmpl_auth_msg_dhchap_success_issue"

245 #define ESC_EMLXS_23    "ESC_emlxs_cmpl_auth_msg_dhchap_reply_cmpl_wait4next"

247 #define ESC_EMLXS_24    "ESC_emlxs_rcv_auth_msg_dhchap_success_issue_wait4next"
248 #define ESC_EMLXS_25    "ESC_emlxs_cmpl_auth_msg_dhchap_success_issue_wait4next"

250 #define ESC_EMLXS_26    "ESC_emlxs_rcv_auth_msg_dhchap_success_cmpl_wait4next"
251 #define ESC_EMLXS_27    "ESC_emlxs_cmpl_auth_msg_dhchap_success_cmpl_wait4next"

253 #define ESC_EMLXS_28    "ESC_emlxs_issue_auth_reject"
254 #define ESC_EMLXS_29    "ESC_emlxs_cmpl_auth_reject_issue"

256 #define ESC_EMLXS_30    "ESC_emlxs_rcv_auth_msg_npr_node"

258 #define ESC_EMLXS_31    "ESC_emlxs_dhc_reauth_timeout"
```

```
#define ESC_EMLXS_32    "ESC_emlxs_dhc_authrsp_timeout"

#define ESC_EMLXS_33    "ESC_emlxs_ioctl_auth_setcfg"
#define ESC_EMLXS_34    "ESC_emlxs_ioctl_auth_setpwd"
#define ESC_EMLXS_35    "ESC_emlxs_ioctl_auth_delcfg"
#define ESC_EMLXS_36    "ESC_emlxs_ioctl_auth_delpwd"


/* From HBAnyware dfc lib FC-SP */
typedef struct emlxs_auth_cfg
{
        NAME_TYPE               local_entity;   /* host wwpn (NPIV support) */
        NAME_TYPE               remote_entity;  /* switch or target wwpn */
        uint32_t                authentication_timeout;
        uint32_t                authentication_mode;
        uint32_t                bidirectional:1;
        uint32_t                reserved:31;
        uint32_t                authentication_type_priority[4];
        uint32_t                hash_priority[4];
        uint32_t                dh_group_priority[8];
        uint32_t                reauthenticate_time_interval;

        dfc_auth_status_t       auth_status;
        uint32_t                auth_time;
        time_t                  auth_time;
        struct emlxs_node       *node;

        struct emlxs_auth_cfg   *prev;
        struct emlxs_auth_cfg   *next;
} emlxs_auth_cfg_t;
______unchanged_portion_omitted_


/*
 * emlxs_port_dhc struct to be used by emlxs_port_t in emlxs_fc.h
 *
 * This structure contains all the data used by DHCHAP.
 * They are from EMLXSHBA_t in emlxs driver.
 *
 */
typedef struct emlxs_port_dhc
{

        int32_t                 state;
#define ELX_FABRIC_STATE_UNKNOWN        0x00
#define ELX_FABRIC_AUTH_DISABLED        0x01
#define ELX_FABRIC_AUTH_FAILED          0x02
#define ELX_FABRIC_AUTH_SUCCESS         0x03
#define ELX_FABRIC_IN_AUTH              0x04
#define ELX_FABRIC_IN_REAUTH            0x05

        dfc_auth_status_t       auth_status;  /* Fabric auth status */
        uint32_t                auth_time;
        time_t                  auth_time;

} emlxs_port_dhc_t;
______unchanged_portion_omitted_
```

```
**********************************************************
   57703 Mon May  5 11:11:24 2014
new/usr/src/uts/common/sys/fibre-channel/fca/emlxs/emlxs_fc.h
4786 emlxs shouldn't abuse ddi_get_time(9f)
**********************************************************
/*
 * CDDL HEADER START
 *
 * The contents of this file are subject to the terms of the
 * Common Development and Distribution License (the "License").
 * You may not use this file except in compliance with the License.
 *
 * You can obtain a copy of the license at usr/src/OPENSOLARIS.LICENSE
 * or http://www.opensolaris.org/os/licensing.
 * See the License for the specific language governing permissions
 * and limitations under the License.
 *
 * When distributing Covered Code, include this CDDL HEADER in each
 * file and include the License file at usr/src/OPENSOLARIS.LICENSE.
 * If applicable, add the following below this CDDL HEADER, with the
 * fields enclosed by brackets "[]" replaced with your own identifying
 * information: Portions Copyright [yyyy] [name of copyright owner]
 *
 * CDDL HEADER END
 */

/*
 * Copyright 2010 Emulex.  All rights reserved.
 * Use is subject to license terms.
 */
/*
 * Copyright 2014 Nexenta Systems, Inc.  All rights reserved.
 */
#endif /* ! codereview */

#ifndef _EMLXS_FC_H
#define _EMLXS_FC_H

#ifdef  __cplusplus
extern "C" {
#endif

typedef struct emlxs_buf
{
        fc_packet_t             *pkt;           /* scsi_pkt reference */
        struct emlxs_port       *port;          /* pointer to port */
        void                    *bmp;           /* Save the buffer pointer */
                                                /* list for later use. */
        struct emlxs_buf        *fc_fwd;        /* Use it by chip_Q */
        struct emlxs_buf        *fc_bkwd;       /* Use it by chip_Q */
        struct emlxs_buf        *next;          /* Use it when the iodone */
        struct emlxs_node       *node;
        void                    *channel;       /* Save channel and used by */
                                                /* abort */
        struct emlxs_buf        *fpkt;          /* Flush pkt pointer */
        struct XRIobj           *xrip;          /* Exchange resource */
        IOCBQ                   iocbq;
        kmutex_t                mtx;
        uint32_t                pkt_flags;
        uint32_t                iotag;          /* iotag for this cmd */
        uint32_t                ticks;          /* save the timeout ticks */
                                                /* for the fc_packet_t */
        uint32_t                abort_attempts;
        uint32_t                lun;
#define EMLXS_LUN_NONE          0xFFFFFFFF
```



```
        uint32_t                class;          /* Save class and used by */
                                                /* abort */
        uint32_t                ucmd;           /* Unsolicted command that */
                                                /* this packet is responding */
                                                /* to, if any */
        int32_t                 flush_count;    /* Valid only in flush pkts */
        uint32_t                did;

#ifdef SFCT_SUPPORT
        kmutex_t                fct_mtx;
        fc_packet_t             *fct_pkt;
        fct_cmd_t               *fct_cmd;

        uint8_t                 fct_type;

#define EMLXS_FCT_ELS_CMD               0x01    /* Unsolicted */
#define EMLXS_FCT_ELS_REQ               0x02    /* Solicited */
#define EMLXS_FCT_ELS_RSP               0x04
#define EMLXS_FCT_CT_REQ                0x08    /* Solicited */
#define EMLXS_FCT_FCP_CMD               0x10    /* Unsolicted */
#define EMLXS_FCT_FCP_DATA              0x20
#define EMLXS_FCT_FCP_STATUS            0x40


        uint8_t                 fct_flags;

#define EMLXS_FCT_SEND_STATUS           0x01
#define EMLXS_FCT_ABORT_INP             0x02
#define EMLXS_FCT_IO_INP                0x04
#define EMLXS_FCT_PLOGI_RECEIVED        0x10
#define EMLXS_FCT_REGISTERED            0x20

        uint16_t                fct_state;

#define EMLXS_FCT_FCP_CMD_RECEIVED      1
#define EMLXS_FCT_ELS_CMD_RECEIVED      2
#define EMLXS_FCT_CMD_POSTED            3
#define EMLXS_FCT_CMD_WAITQ             4
#define EMLXS_FCT_SEND_CMD_RSP          5
#define EMLXS_FCT_SEND_ELS_RSP          6
#define EMLXS_FCT_SEND_ELS_REQ          7
#define EMLXS_FCT_SEND_CT_REQ           8
#define EMLXS_FCT_RSP_PENDING           9
#define EMLXS_FCT_REQ_PENDING           10
#define EMLXS_FCT_REG_PENDING           11
#define EMLXS_FCT_REG_COMPLETE          12
#define EMLXS_FCT_OWNED                 13
#define EMLXS_FCT_SEND_FCP_DATA         14
#define EMLXS_FCT_SEND_FCP_STATUS       15
#define EMLXS_FCT_DATA_PENDING          16
#define EMLXS_FCT_STATUS_PENDING        17
#define EMLXS_FCT_PKT_COMPLETE          18
#define EMLXS_FCT_PKT_FCPRSP_COMPLETE   19
#define EMLXS_FCT_PKT_ELSRSP_COMPLETE   20
#define EMLXS_FCT_PKT_ELSCMD_COMPLETE   21
#define EMLXS_FCT_PKT_CTCMD_COMPLETE    22
#define EMLXS_FCT_REQ_COMPLETE          23
#define EMLXS_FCT_CLOSE_PENDING         24
#define EMLXS_FCT_ABORT_PENDING         25
#define EMLXS_FCT_ABORT_DONE            26
#define EMLXS_FCT_IO_DONE               27

#define EMLXS_FCT_IOCB_ISSUED           256 /* For tracing only */
#define EMLXS_FCT_IOCB_COMPLETE         257 /* For tracing only */

        stmf_data_buf_t         *fct_buf;
```

```
#endif /* SFCT_SUPPORT */

#ifdef SAN_DIAG_SUPPORT
        hrtime_t                sd_start_time;
#endif
} emlxs_buf_t;



#ifdef FCT_IO_TRACE
#define EMLXS_FCT_STATE_CHG(_fct_cmd, _cmd_sbp, _state) \
        (_cmd_sbp)->fct_state = _state;                 \
        emlxs_fct_io_trace((_cmd_sbp)->port, _fct_cmd, _state)
#else
/* define to set fct_state */
#define EMLXS_FCT_STATE_CHG(_fct_cmd, _cmd_sbp, _state) \
        (_cmd_sbp)->fct_state = _state
#endif /* FCT_IO_TRACE */


/* pkt_flags */
#define PACKET_IN_COMPLETION    0x00000001
#define PACKET_IN_TXQ           0x00000002
#define PACKET_IN_CHIPQ         0x00000004
#define PACKET_IN_DONEQ         0x00000008

#define PACKET_FCP_RESET        0x00000030
#define PACKET_FCP_TGT_RESET    0x00000010
#define PACKET_FCP_LUN_RESET    0x00000020
#define PACKET_POLLED           0x00000040

#ifdef EMLXS_I386
#define PACKET_FCP_SWAPPED      0x00000100
#define PACKET_ELS_SWAPPED      0x00000200
#define PACKET_CT_SWAPPED       0x00000400
#define PACKET_CSP_SWAPPED      0x00000800
#endif  /* EMLXS_I386 */

#define PACKET_STALE            0x00001000

#define PACKET_IN_TIMEOUT       0x00010000
#define PACKET_IN_FLUSH         0x00020000
#define PACKET_IN_ABORT         0x00040000
#define PACKET_XRI_CLOSED       0x00080000 /* An XRI abort/close was issued */

#define PACKET_CHIP_COMP        0x00100000
#define PACKET_COMPLETED        0x00200000
#define PACKET_ULP_OWNED        0x00400000

#define PACKET_STATE_VALID      0x01000000
#define PACKET_FCP_RSP_VALID    0x02000000
#define PACKET_ELS_RSP_VALID    0x04000000
#define PACKET_CT_RSP_VALID     0x08000000

#define PACKET_DELAY_REQUIRED   0x10000000
#define PACKET_ALLOCATED        0x40000000
#define PACKET_VALID            0x80000000


#define STALE_PACKET            ((emlxs_buf_t *)0xFFFFFFFF)


/*
 * From fc_error.h pkt_reason (except for state = NPORT_RJT, FABRIC_RJT,
 * NPORT_BSY, FABRIC_BSY, LS_RJT, BA_RJT, FS_RJT)
```



```
 *
 * FCA unique error codes can begin after FC_REASON_FCA_UNIQUE.
 * Each FCA defines its own set with values greater >= 0x7F
 */
#define FC_REASON_FCA_DEFINED   0x100


/*
 * Device VPD save area
 */

typedef struct emlxs_vpd
{
        uint32_t        biuRev;
        uint32_t        smRev;
        uint32_t        smFwRev;
        uint32_t        endecRev;
        uint16_t        rBit;
        uint8_t         fcphHigh;
        uint8_t         fcphLow;
        uint8_t         feaLevelHigh;
        uint8_t         feaLevelLow;

        uint32_t        postKernRev;
        char            postKernName[32];

        uint32_t        opFwRev;
        char            opFwName[32];
        char            opFwLabel[32];

        uint32_t        sli1FwRev;
        char            sli1FwName[32];
        char            sli1FwLabel[32];

        uint32_t        sli2FwRev;
        char            sli2FwName[32];
        char            sli2FwLabel[32];

        uint32_t        sli3FwRev;
        char            sli3FwName[32];
        char            sli3FwLabel[32];

        uint32_t        sli4FwRev;
        char            sli4FwName[32];
        char            sli4FwLabel[32];

        char            fw_version[32];
        char            fw_label[32];

        char            fcode_version[32];
        char            boot_version[32];

        char            serial_num[32];
        char            part_num[32];
        char            port_num[20];
        char            eng_change[32];
        char            manufacturer[80];
        char            model[80];
        char            model_desc[256];
        char            prog_types[256];
        char            id[80];

        uint32_t        port_index;
        uint16_t        link_speed;
} emlxs_vpd_t;
```

```
typedef struct emlxs_queue
{
        void            *q_first;       /* queue first element */
        void            *q_last;        /* queue last element */
        uint16_t        q_cnt;  /* current length of queue */
        uint16_t        q_max;  /* max length queue can get */
} emlxs_queue_t;
typedef emlxs_queue_t Q;



/*
 * This structure is used when allocating a buffer pool.
 * Note: this should be identical to gasket buf_info (fldl.h).
 */
typedef struct emlxs_buf_info
{
        int32_t         size;   /* Specifies the number of bytes to allocate. */
        int32_t         align;  /* The desired address boundary. */

        int32_t         flags;

#define FC_MBUF_DMA             0x01    /* blocks are for DMA */
#define FC_MBUF_PHYSONLY        0x02    /* For malloc - map a given virtual */
                                        /* address to physical address (skip */
                                        /* the malloc). */
                                        /* For free - just unmap the given */
                                        /* physical address (skip the free). */
#define FC_MBUF_IOCTL           0x04    /* called from dfc_ioctl */
#define FC_MBUF_UNLOCK          0x08    /* called with driver unlocked */
#define FC_MBUF_SNGLSG          0x10    /* allocate a single contiguous */
                                        /* physical memory */
#define FC_MBUF_DMA32           0x20

        uint64_t        phys;           /* specifies physical buffer pointer */
        void            *virt;          /* specifies virtual buffer pointer */
        void            *data_handle;
        void            *dma_handle;
} emlxs_buf_info_t;
typedef emlxs_buf_info_t MBUF_INFO;


#define EMLXS_MAX_HBQ           16      /* Max HBQs handled by firmware */
#define EMLXS_ELS_HBQ_ID        0
#define EMLXS_IP_HBQ_ID         1
#define EMLXS_CT_HBQ_ID         2
#define EMLXS_FCT_HBQ_ID        3

#ifdef SFCT_SUPPORT
#define EMLXS_NUM_HBQ           4       /* Number of HBQs supported by driver */
#else
#define EMLXS_NUM_HBQ           3       /* Number of HBQs supported by driver */
#endif /* SFCT_SUPPORT */


/*
 * An IO Channel is a object that comprises a xmit/cmpl
 * path for IOs.
 * For SLI3, an IO path maps to a ring (cmd/rsp)
 * For SLI4, an IO path map to a queue pair (WQ/CQ)
 */
typedef struct emlxs_channel
{
        struct emlxs_hba *hba;                  /* ptr to hba for channel */
        void            *iopath;                /* ptr to SLI3/4 io path */
```



```
        kmutex_t        rsp_lock;
        IOCBQ           *rsp_head;      /* deferred completion head */
        IOCBQ           *rsp_tail;      /* deferred completion tail */
        emlxs_thread_t  intr_thread;


        uint16_t        channelno;
        uint16_t        chan_flag;

#define EMLXS_NEEDS_TRIGGER 1

        /* Protected by EMLXS_TX_CHANNEL_LOCK */
        emlxs_queue_t   nodeq;                  /* Node service queue */

        kmutex_t        channel_cmd_lock;
        uint32_t        timeout;

        /* Channel command counters */
        uint32_t        ulpSendCmd;
        uint32_t        ulpCmplCmd;
        uint32_t        hbaSendCmd;
        uint32_t        hbaCmplCmd;
        uint32_t        hbaSendCmd_sbp;
        uint32_t        hbaCmplCmd_sbp;

} emlxs_channel_t;
typedef emlxs_channel_t CHANNEL;

/*
 * Should be able to handle max number of io paths for a
 * SLI4 HBA (EMLXS_MAX_WQS) or for a SLI3 HBA (MAX_RINGS)
 */
#define MAX_CHANNEL EMLXS_MSI_MAX_INTRS


/* Structure used to access adapter rings */
typedef struct emlxs_ring
{
        void            *fc_cmdringaddr;        /* virtual offset for cmd */
                                                /* rings */
        void            *fc_rspringaddr;        /* virtual offset for rsp */
                                                /* rings */

        void            *fc_mpon;               /* index ptr for match */
                                                /* structure */
        void            *fc_mpoff;              /* index ptr for match */
                                                /* structure */
        struct emlxs_hba *hba;                  /* ptr to hba for ring */

        uint8_t         fc_numCiocb;            /* number of command iocb's */
                                                /* per ring */
        uint8_t         fc_numRiocb;            /* number of response iocb's */
                                                /* per ring */
        uint8_t         fc_rspidx;              /* current index in response */
                                                /* ring */
        uint8_t         fc_cmdidx;              /* current index in command */
                                                /* ring */
        uint8_t         fc_port_rspidx;
        uint8_t         fc_port_cmdidx;
        uint8_t         ringno;

        uint16_t        fc_missbufcnt;          /* buf cnt we need to repost */
        CHANNEL         *channelp;
```

```
392 } emlxs_ring_t;
393 typedef emlxs_ring_t RING;


396 #ifdef SAN_DIAG_SUPPORT
397 /*
398  * Although right now it's just 1 field, SAN Diag anticipates that this
399  * structure will grow in the future.
400  */
401 typedef struct sd_timestat_level0 {
402         int             count;
403 } sd_timestat_level0_t;
404 #endif

406 typedef struct emlxs_node
407 {
408         struct emlxs_node       *nlp_list_next;
409         struct emlxs_node       *nlp_list_prev;

411         NAME_TYPE               nlp_portname;   /* port name */
412         NAME_TYPE               nlp_nodename;   /* node name */

414         uint32_t                nlp_DID;        /* fibre channel D_ID */
415         uint32_t                nlp_oldDID;

417         uint16_t                nlp_Rpi;        /* login id returned by */
418                                                 /* REG_LOGIN */
419         uint16_t                nlp_Xri;        /* login id returned by */
420                                                 /* REG_LOGIN */

422         uint8_t                 nlp_fcp_info;   /* Remote class info */

424         /* nlp_fcp_info */
425 #define NLP_FCP_TGT_DEVICE      0x10    /* FCP TGT device */
426 #define NLP_FCP_INI_DEVICE      0x20    /* FCP Initiator device */
427 #define NLP_FCP_2_DEVICE        0x40    /* FCP-2 TGT device */
428 #define NLP_EMLX_VPORT          0x80    /* Virtual port */

430         uint32_t                nlp_force_rscn;
431         uint32_t                nlp_tag;        /* Tag used by port_offline */
432         uint32_t                flag;

434 #define NODE_POOL_ALLOCATED     0x00000001

436         SERV_PARM               sparm;

438         /* Protected by EMLXS_TX_CHANNEL_LOCK */
439         uint32_t                nlp_active;     /* Node active flag */
440         uint32_t                nlp_base;
441         uint32_t                nlp_flag[MAX_CHANNEL];  /* Node level channel */
442                                                         /* flags */

444         /* nlp_flag */
445 #define NLP_CLOSED              0x1
446 #define NLP_OFFLINE             0x2
447 #define NLP_RPI_XRI             0x4

449         uint32_t                nlp_tics[MAX_CHANNEL];  /* gate timeout */
450         emlxs_queue_t           nlp_tx[MAX_CHANNEL];    /* Transmit Q head */
451         emlxs_queue_t           nlp_ptx[MAX_CHANNEL];   /* Priority transmit */
452                                                         /* Queue head */
453         void                    *nlp_next[MAX_CHANNEL]; /* Service Request */
454                                                         /* Queue pointer used */
455                                                         /* when node needs */
456                                                         /* servicing */
457 #ifdef DHCHAP_SUPPORT
```



```
458         emlxs_node_dhc_t        node_dhc;
459 #endif  /* DHCHAP_SUPPORT */

461 #ifdef SAN_DIAG_SUPPORT
462         sd_timestat_level0_t    sd_dev_bucket[SD_IO_LATENCY_MAX_BUCKETS];
463 #endif

465         struct RPIobj           *rpip;  /* SLI4 only */
466 #define EMLXS_NODE_TO_RPI(_p, _n)       \
467         ((_n)?((_n->rpip)?_n->rpip:emlxs_rpi_find(_p, _n->nlp_Rpi)):NULL)

469 } emlxs_node_t;
470 typedef emlxs_node_t NODELIST;



474 #define NADDR_LEN       6       /* MAC network address length */
475 typedef struct emlxs_fcip_nethdr
476 {
477         NAME_TYPE       fc_destname;    /* destination port name */
478         NAME_TYPE       fc_srcname;     /* source port name */
479 } emlxs_fcip_nethdr_t;
480 typedef emlxs_fcip_nethdr_t NETHDR;


483 #define MEM_NLP         0       /* memory segment to hold node list entries */
484 #define MEM_IOCB        1       /* memory segment to hold iocb commands */
485 #define MEM_MBOX        2       /* memory segment to hold mailbox cmds  */
486 #define MEM_BPL         3       /* and to hold buffer ptr lists - SLI2   */
487 #define MEM_BUF         4       /* memory segment to hold buffer data   */
488 #define MEM_ELSBUF      4       /* memory segment to hold buffer data   */
489 #define MEM_IPBUF       5       /* memory segment to hold IP buffer data */
490 #define MEM_CTBUF       6       /* memory segment to hold CT buffer data */
491 #define MEM_FCTBUF      7       /* memory segment to hold FCT buffer data */

493 #ifdef SFCT_SUPPORT
494 #define FC_MAX_SEG      8
495 #else
496 #define FC_MAX_SEG      7
497 #endif /* SFCT_SUPPORT */


500 /* A BPL entry is 12 bytes. Subtract 2 for command and response buffers */
501 #define BPL_TO_SGLLEN(_bpl)   ((_bpl/12)-2)
502 #define MEM_BPL_SIZE            1024  /* Default size */

504 /* A SGL entry is 16 bytes. Subtract 2 for command and response buffers */
505 #define SGL_TO_SGLLEN(_sgl)   ((_sgl/16)-2)
506 #define MEM_SGL_SIZE            4096  /* Default size */

508 #ifdef EMLXS_I386
509 #define EMLXS_SGLLEN            BPL_TO_SGLLEN(MEM_BPL_SIZE)
510 #else   /* EMLXS_SPARC */
511 #define EMLXS_SGLLEN            1
512 #endif  /* EMLXS_I386 */

514 #define MEM_BUF_SIZE            1024
515 #define MEM_BUF_COUNT           64

517 #define MEM_ELSBUF_SIZE         MEM_BUF_SIZE
518 #define MEM_ELSBUF_COUNT        hba->max_nodes
519 #define MEM_IPBUF_SIZE          65535
520 #define MEM_IPBUF_COUNT         60
521 #define MEM_CTBUF_SIZE          MAX_CT_PAYLOAD  /* (1024*320) */
522 #define MEM_CTBUF_COUNT         8
523 #define MEM_FCTBUF_SIZE         65535
```

```
524 #define MEM_FCTBUF_COUNT        128

526 typedef struct emlxs_memseg
527 {
528         void                    *fc_memget_ptr;
529         void                    *fc_memget_end;
530         void                    *fc_memput_ptr;
531         void                    *fc_memput_end;

533         void                    *fc_memstart_virt;      /* beginning address */
534                                                         /* of memory block */
535         uint64_t                fc_memstart_phys;       /* beginning address */
536                                                         /* of memory block */
537         ddi_dma_handle_t        fc_mem_dma_handle;
538         ddi_acc_handle_t        fc_mem_dat_handle;
539         uint32_t                fc_total_memsize;
540         uint32_t                fc_memsize;             /* size of mem blks */
541         uint32_t                fc_numblks;             /* no of mem blks */
542         uint32_t                fc_memget_cnt;          /* no of mem get blks */
543         uint32_t                fc_memput_cnt;          /* no of mem put blks */
544         uint32_t                fc_memflag;  /* emlxs_buf_info_t FLAGS */
545         uint32_t                fc_reserved; /* used with priority flag */
546         uint32_t                fc_memalign;
547         uint32_t                fc_memtag;
548         char                    fc_label[32];

550 } emlxs_memseg_t;
551 typedef emlxs_memseg_t MEMSEG;


554 /* Board stat counters */
555 typedef struct emlxs_stats
556 {
557         uint32_t        LinkUp;
558         uint32_t        LinkDown;
559         uint32_t        LinkEvent;
560         uint32_t        LinkMultiEvent;

562         uint32_t        MboxIssued;
563         uint32_t        MboxCompleted;  /* MboxError + MbxGood */
564         uint32_t        MboxGood;
565         uint32_t        MboxError;
566         uint32_t        MboxBusy;
567         uint32_t        MboxInvalid;

569         uint32_t        IocbIssued[MAX_CHANNEL];
570         uint32_t        IocbReceived[MAX_CHANNEL];
571         uint32_t        IocbTxPut[MAX_CHANNEL];
572         uint32_t        IocbTxGet[MAX_CHANNEL];
573         uint32_t        IocbRingFull[MAX_CHANNEL];
574         uint32_t        IocbThrottled;

576         uint32_t        IntrEvent[8];

578         uint32_t        FcpIssued;
579         uint32_t        FcpCompleted;   /* FcpGood + FcpError */
580         uint32_t        FcpGood;
581         uint32_t        FcpError;

583         uint32_t        FcpEvent;       /* FcpStray + FcpCompleted */
584         uint32_t        FcpStray;
585 #ifdef SFCT_SUPPORT
586         uint32_t        FctRingEvent;
587         uint32_t        FctRingError;
588         uint32_t        FctRingDropped;
589 #endif /* SFCT_SUPPORT */
```



```
591         uint32_t        ElsEvent;       /* ElsStray + ElsCmplt (cmd + rsp) */
592         uint32_t        ElsStray;

594         uint32_t        ElsCmdIssued;
595         uint32_t        ElsCmdCompleted;        /* ElsCmdGood + ElsCmdError */
596         uint32_t        ElsCmdGood;
597         uint32_t        ElsCmdError;

599         uint32_t        ElsRspIssued;
600         uint32_t        ElsRspCompleted;

602         uint32_t        ElsRcvEvent;    /* ElsRcvErr + ElsRcvDrop + ElsCmdRcv */
603         uint32_t        ElsRcvError;
604         uint32_t        ElsRcvDropped;
605         uint32_t        ElsCmdReceived; /* ElsRscnRcv + ElsPlogiRcv + ... */
606         uint32_t        ElsRscnReceived;
607         uint32_t        ElsFlogiReceived;
608         uint32_t        ElsPlogiReceived;
609         uint32_t        ElsPrliReceived;
610         uint32_t        ElsPrloReceived;
611         uint32_t        ElsLogoReceived;
612         uint32_t        ElsAdiscReceived;
613         uint32_t        ElsAuthReceived;
614         uint32_t        ElsGenReceived;

616         uint32_t        CtEvent;        /* CtStray + CtCompleted (cmd + rsp) */
617         uint32_t        CtStray;

619         uint32_t        CtCmdIssued;
620         uint32_t        CtCmdCompleted; /* CtCmdGood + CtCmdError */
621         uint32_t        CtCmdGood;
622         uint32_t        CtCmdError;

624         uint32_t        CtRspIssued;
625         uint32_t        CtRspCompleted;

627         uint32_t        CtRcvEvent;     /* CtRcvError + CtRcvDrop + CtCmdRcvd */
628         uint32_t        CtRcvError;
629         uint32_t        CtRcvDropped;
630         uint32_t        CtCmdReceived;

632         uint32_t        IpEvent;        /* IpStray + IpSeqCmpl + IpBcastCmpl */
633         uint32_t        IpStray;

635         uint32_t        IpSeqIssued;
636         uint32_t        IpSeqCompleted; /* IpSeqGood + IpSeqError */
637         uint32_t        IpSeqGood;
638         uint32_t        IpSeqError;

640         uint32_t        IpBcastIssued;
641         uint32_t        IpBcastCompleted;       /* IpBcastGood + IpBcastError */
642         uint32_t        IpBcastGood;
643         uint32_t        IpBcastError;

645         uint32_t        IpRcvEvent;     /* IpDrop + IpSeqRcv + IpBcastRcv */
646         uint32_t        IpDropped;
647         uint32_t        IpSeqReceived;
648         uint32_t        IpBcastReceived;

650         uint32_t        IpUbPosted;
651         uint32_t        ElsUbPosted;
652         uint32_t        CtUbPosted;
653 #ifdef SFCT_SUPPORT
654         uint32_t        FctUbPosted;
655 #endif /* SFCT_SUPPORT */
```

```
657         uint32_t        ResetTime;      /* Time of last reset */

659         uint32_t        ElsTestReceived;
660         uint32_t        ElsEstcReceived;
661         uint32_t        ElsFarprReceived;
662         uint32_t        ElsEchoReceived;
663         uint32_t        ElsRlsReceived;
664         uint32_t        ElsRtvReceived;

666 } emlxs_stats_t;


669 #define FC_MAX_ADPTMSG   (8*28) /* max size of a msg from adapter */

671 #define EMLXS_NUM_THREADS       8
672 #define EMLXS_MIN_TASKS         8
673 #define EMLXS_MAX_TASKS         8

675 #define EMLXS_NUM_HASH_QUES     32
676 #define EMLXS_DID_HASH(x)       ((x) & (EMLXS_NUM_HASH_QUES - 1))


679 /* pkt_tran_flag */
680 #define FC_TRAN_COMPLETED       0x8000


683 typedef struct emlxs_dfc_event
684 {
685         uint32_t        pid;
686         uint32_t        event;
687         uint32_t        last_id;

689         void            *dataout;
690         uint32_t        size;
691         uint32_t        mode;
692 } emlxs_dfc_event_t;


695 typedef struct emlxs_hba_event
696 {
697         uint32_t        last_id;
698         uint32_t        new;
699         uint32_t        missed;
700 } emlxs_hba_event_t;


703 #ifdef SFCT_SUPPORT

705 #define TGTPORTSTAT                     port->fct_stat

707 /*
708  * FctP2IOXcnt will count IOs by their fcpDL. Counters
709  * are for buckets of various power of 2 sizes.
710  * Bucket 0  <  512  > 0
711  * Bucket 1  >= 512  < 1024
712  * Bucket 2  >= 1024 < 2048
713  * Bucket 3  >= 2048 < 4096
714  * Bucket 4  >= 4096 < 8192
715  * Bucket 5  >= 8192 < 16K
716  * Bucket 6  >= 16K  < 32K
717  * Bucket 7  >= 32K  < 64K
718  * Bucket 8  >= 64K  < 128K
719  * Bucket 9  >= 128K < 256K
720  * Bucket 10 >= 256K < 512K
721  * Bucket 11 >= 512K < 1MB
```



```
722  * Bucket 12 >= 1MB  < 2MB
723  * Bucket 13 >= 2MB  < 4MB
724  * Bucket 14 >= 4MB  < 8MB
725  * Bucket 15 >= 8MB
726  */
727 #define MAX_TGTPORT_IOCNT  16


730 /*
731  * These routines will bump the right counter, based on
732  * the size of the IO inputed, with the least number of
733  * comparisions.  A max of 5 comparisions is only needed
734  * to classify the IO in one of 16 ranges. A binary search
735  * to locate the high bit in the size is used.
736  */
737 #define EMLXS_BUMP_RDIOCTR(port, cnt) \
738 { \
739         /* Use binary search to find the first high bit */ \
740         if (cnt & 0xffff0000) { \
741                 if (cnt & 0xff800000) { \
742                         TGTPORTSTAT.FctP2IORcnt[15]++; \
743                 } \
744                 else { \
745                         /* It must be 0x007f0000 */ \
746                         if (cnt & 0x00700000) { \
747                                 if (cnt & 0x00400000) { \
748                                         TGTPORTSTAT.FctP2IORcnt[14]++; \
749                                 } \
750                                 else { \
751                                         /* it must be 0x00300000 */ \
752                                         if (cnt & 0x00200000) { \
753                                                 TGTPORTSTAT.FctP2IORcnt[13]++; \
754                                         } \
755                                         else { \
756                                                 /* It must be 0x00100000 */ \
757                                                 TGTPORTSTAT.FctP2IORcnt[12]++; \
758                                         } \
759                                 } \
760                         } \
761                         else { \
762                                 /* It must be 0x000f0000 */ \
763                                 if (cnt & 0x000c0000) { \
764                                         if (cnt & 0x00080000) { \
765                                                 TGTPORTSTAT.FctP2IORcnt[11]++; \
766                                         } \
767                                         else { \
768                                                 /* It must be 0x00040000 */ \
769                                                 TGTPORTSTAT.FctP2IORcnt[10]++; \
770                                         } \
771                                 } \
772                                 else { \
773                                         /* It must be 0x00030000 */ \
774                                         if (cnt & 0x00020000) { \
775                                                 TGTPORTSTAT.FctP2IORcnt[9]++; \
776                                         } \
777                                         else { \
778                                                 /* It must be 0x00010000 */ \
779                                                 TGTPORTSTAT.FctP2IORcnt[8]++; \
780                                         } \
781                                 } \
782                         } \
783                 } \
784         } \
785         else { \
786                 if (cnt & 0x0000fe00) { \
787                         if (cnt & 0x0000f000) { \
```

```
788                                 if (cnt & 0x0000c000) { \
789                                         if (cnt & 0x00008000) { \
790                                                 TGTPORTSTAT.FctP2IORcnt[7]++; \
791                                         } \
792                                         else { \
793                                                 /* It must be 0x00004000 */ \
794                                                 TGTPORTSTAT.FctP2IORcnt[6]++; \
795                                         } \
796                                 } \
797                                 else { \
798                                         /* It must be 0x00000300 */ \
799                                         if (cnt & 0x00000200) { \
800                                                 TGTPORTSTAT.FctP2IORcnt[5]++; \
801                                         } \
802                                         else { \
803                                                 /* It must be 0x00000100 */ \
804                                                 TGTPORTSTAT.FctP2IORcnt[4]++; \
805                                         } \
806                                 } \
807                         } \
808                         else { \
809                                 /* It must be 0x00000e00 */ \
810                                 if (cnt & 0x00000800) { \
811                                         TGTPORTSTAT.FctP2IORcnt[3]++; \
812                                 } \
813                                 else { \
814                                         /* It must be 0x00000600 */ \
815                                         if (cnt & 0x00000400) { \
816                                                 TGTPORTSTAT.FctP2IORcnt[2]++; \
817                                         } \
818                                         else { \
819                                                 /* It must be 0x00000200 */ \
820                                                 TGTPORTSTAT.FctP2IORcnt[1]++; \
821                                         } \
822                                 } \
823                         } \
824                 } \
825                 else { \
826                         /* It must be 0x000001ff */ \
827                         TGTPORTSTAT.FctP2IORcnt[0]++; \
828                 } \
829         } \
830 }


833 #define EMLXS_BUMP_WRIOCTR(port, cnt) \
834 { \
835 /* Use binary search to find the first high bit */ \
836         if (cnt & 0xffff0000) { \
837                 if (cnt & 0xff800000) { \
838                         TGTPORTSTAT.FctP2IOWcnt[15]++; \
839                 } \
840                 else { \
841                         /* It must be 0x007f0000 */ \
842                         if (cnt & 0x00700000) { \
843                                 if (cnt & 0x00400000) { \
844                                         TGTPORTSTAT.FctP2IOWcnt[14]++; \
845                                 } \
846                                 else { \
847                                         /* It must be 0x00300000 */ \
848                                         if (cnt & 0x00200000) { \
849                                                 TGTPORTSTAT.FctP2IOWcnt[13]++; \
850                                         } \
851                                         else { \
852                                                 /* It must be 0x00100000 */ \
853                                                 TGTPORTSTAT.FctP2IOWcnt[12]++; \
```



```
854                                         } \
855                                 } \
856                         } \
857                         else { \
858                                 /* It must be 0x000f0000 */ \
859                                 if (cnt & 0x000c0000) { \
860                                         if (cnt & 0x00080000) { \
861                                                 TGTPORTSTAT.FctP2IOWcnt[11]++; \
862                                         } \
863                                         else { \
864                                                 /* it must be 0x00040000 */ \
865                                                 TGTPORTSTAT.FctP2IOWcnt[10]++; \
866                                         } \
867                                 } \
868                                 else { \
869                                         /* It must be 0x00030000 */ \
870                                         if (cnt & 0x00020000) { \
871                                                 TGTPORTSTAT.FctP2IOWcnt[9]++; \
872                                         } \
873                                         else { \
874                                                 /* It must be 0x00010000 */ \
875                                                 TGTPORTSTAT.FctP2IOWcnt[8]++; \
876                                         } \
877                                 } \
878                         } \
879                 } \
880         } \
881         else { \
882                 if (cnt & 0x0000fe00) { \
883                         if (cnt & 0x0000f000) { \
884                                 if (cnt & 0x0000c000) { \
885                                         if (cnt & 0x00008000) { \
886                                                 TGTPORTSTAT.FctP2IOWcnt[7]++; \
887                                         } \
888                                         else { \
889                                                 /* It must be 0x00004000 */ \
890                                                 TGTPORTSTAT.FctP2IOWcnt[6]++; \
891                                         } \
892                                 } \
893                                 else { \
894                                         /* It must be 0x00000300 */ \
895                                         if (cnt & 0x00000200) { \
896                                                 TGTPORTSTAT.FctP2IOWcnt[5]++; \
897                                         } \
898                                         else { \
899                                                 /* It must be 0x00000100 */ \
900                                                 TGTPORTSTAT.FctP2IOWcnt[4]++; \
901                                         } \
902                                 } \
903                         } \
904                         else { \
905                                 /* It must be 0x00000e00 */ \
906                                 if (cnt & 0x00000800) { \
907                                         TGTPORTSTAT.FctP2IOWcnt[3]++; \
908                                 } \
909                                 else { \
910                                         /* It must be 0x00000600 */ \
911                                         if (cnt & 0x00000400) { \
912                                                 TGTPORTSTAT.FctP2IOWcnt[2]++; \
913                                         } \
914                                         else { \
915                                                 /* It must be 0x00000200 */ \
916                                                 TGTPORTSTAT.FctP2IOWcnt[1]++; \
917                                         } \
918                                 } \
919                         } \
```

```
920                 } \
921                 else { \
922                         /* It must be 0x000001ff */ \
923                         TGTPORTSTAT.FctP2IOWcnt[0]++; \
924                 } \
925         } \
926 }

928 typedef struct emlxs_tgtport_stat
929 {
930         /* IO counters */
931         uint64_t        FctP2IOWcnt[MAX_TGTPORT_IOCNT]; /* Writes */
932         uint64_t        FctP2IORcnt[MAX_TGTPORT_IOCNT]; /* Reads  */
933         uint64_t        FctIOCmdCnt;                    /* Other, ie TUR */
934         uint64_t        FctCmdReceived;                 /* total IOs */
935         uint64_t        FctReadBytes;                   /* total read bytes */
936         uint64_t        FctWriteBytes;                  /* total write bytes */

938         /* IOCB handling counters */
939         uint64_t        FctEvent;       /* FctStray + FctCompleted */
940         uint64_t        FctCompleted;   /* FctCmplGood + FctCmplError */
941         uint64_t        FctCmplGood;

943         uint32_t        FctCmplError;
944         uint32_t        FctStray;

946         /* Fct event counters */
947         uint32_t        FctRcvDropped;
948         uint32_t        FctOverQDepth;
949         uint32_t        FctOutstandingIO;
950         uint32_t        FctFailedPortRegister;
951         uint32_t        FctPortRegister;
952         uint32_t        FctPortDeregister;

954         uint32_t        FctAbortSent;
955         uint32_t        FctNoBuffer;
956         uint32_t        FctScsiStatusErr;
957         uint32_t        FctScsiQfullErr;
958         uint32_t        FctScsiResidOver;
959         uint32_t        FctScsiResidUnder;
960         uint32_t        FctScsiSenseErr;

962         uint32_t        FctFiller1;
963 } emlxs_tgtport_stat_t;

965 #ifdef FCT_IO_TRACE
966 #define MAX_IO_TRACE    67
967 typedef struct emlxs_iotrace
968 {
969         fct_cmd_t       *fct_cmd;
970         uint32_t        xri;
971         uint8_t         marker;  /* 0xff */
972         uint8_t         trc[MAX_IO_TRACE]; /* trc[0] = index */
973 } emlxs_iotrace_t;
974 #endif /* FCT_IO_TRACE */
975 #endif /* SFCT_SUPPORT */


978 #include <emlxs_fcf.h>

980 /*
981  *     Port Information Data Structure
982  */

984 typedef struct emlxs_port
985 {
```



```
986         struct emlxs_hba        *hba;

988         /* Virtual port management */
989         struct VPIobj           VPIobj;
990         uint32_t                vpi;

992         uint32_t                flag;
993 #define EMLXS_PORT_ENABLE               0x00000001
994 #define EMLXS_PORT_BOUND                0x00000002

996 #define EMLXS_PORT_REG_VPI              0x00010000 /* SLI3 */
997 #define EMLXS_PORT_REG_VPI_CMPL         0x00020000 /* SLI3 */

999 #define EMLXS_PORT_IP_UP                0x00000010
1000 #define EMLXS_PORT_CONFIG               0x00000020
1001 #define EMLXS_PORT_RESTRICTED           0x00000040 /* Restrict logins */
1002 #define EMLXS_PORT_FLOGI_CMPL           0x00000080

1004 #define EMLXS_PORT_RESET_MASK           0x0000FFFF /* Flags to keep */
1005                                                    /* across hard reset */
1006 #define EMLXS_PORT_LINKDOWN_MASK        0xFFFFFF7F /* Flags to keep */
1007                                                    /* across link reset */

1009         uint32_t                options;
1010 #define EMLXS_OPT_RESTRICT              0x00000001 /* Force restricted */
1011                                                    /* logins */
1012 #define EMLXS_OPT_UNRESTRICT            0x00000002 /* Force Unrestricted */
1013                                                    /* logins */
1014 #define EMLXS_OPT_RESTRICT_MASK         0x00000003


1017         /* FC world wide names */
1018         NAME_TYPE               wwnn;
1019         NAME_TYPE               wwpn;
1020         char                    snn[256];
1021         char                    spn[256];

1023         /* Common service paramters */
1024         SERV_PARM               sparam;
1025         SERV_PARM               fabric_sparam;
1026         SERV_PARM               prev_fabric_sparam;

1028         /* fc_id management */
1029         uint32_t                did;
1030         uint32_t                prev_did;

1032         /* support FC_PORT_GET_P2P_INFO only */
1033         uint32_t                rdid;

1035         /* FC_AL management */
1036         uint8_t                 lip_type;
1037         uint8_t                 alpa_map[128];

1039         /* Node management */
1040         emlxs_node_t            node_base;
1041         uint32_t                node_count;
1042         krwlock_t               node_rwlock;
1043         emlxs_node_t            *node_table[EMLXS_NUM_HASH_QUES];

1045         /* Polled packet management */
1046         kcondvar_t              pkt_lock_cv;    /* pkt polling */
1047         kmutex_t                pkt_lock;       /* pkt polling */

1049         /* ULP */
1050         uint32_t                ulp_statec;
1051         void                    (*ulp_statec_cb) ();    /* Port state change */
```

```
1052                                                    /* callback routine */
1053         void                    (*ulp_unsol_cb) ();     /* unsolicited event */
1054                                                    /* callback routine */
1055         opaque_t                ulp_handle;

1057         /* ULP unsolicited buffers */
1058         kmutex_t                ub_lock;
1059         uint32_t                ub_count;
1060         emlxs_unsol_buf_t       *ub_pool;
1061         uint32_t                ub_post[MAX_CHANNEL];
1062         uint32_t                ub_timer;

1064         emlxs_ub_priv_t         *ub_wait_head;  /* Unsolicited IO received */
1065                                                 /* before link up */
1066         emlxs_ub_priv_t         *ub_wait_tail;  /* Unsolicited IO received */
1067                                                 /* before link up */


1070 #ifdef DHCHAP_SUPPORT
1071         emlxs_port_dhc_t        port_dhc;
1072 #endif  /* DHCHAP_SUPPORT */

1074         uint16_t                ini_mode;
1075         uint16_t                tgt_mode;

1077 #ifdef SFCT_SUPPORT

1079 #define FCT_BUF_COUNT_512               256
1080 #define FCT_BUF_COUNT_8K                128
1081 #define FCT_BUF_COUNT_64K               64
1082 #define FCT_BUF_COUNT_128K              64
1083 #define FCT_MAX_BUCKETS                 16
1084 #define FCT_DMEM_MAX_BUF_SIZE           131072   /* 128K */
1085 #define FCT_DMEM_MAX_BUF_SEGMENT        8388608  /* 8M */

1087         struct emlxs_fct_dmem_bucket dmem_bucket[FCT_MAX_BUCKETS];

1089         char                    cfd_name[24];
1090         stmf_port_provider_t    *port_provider;
1091         fct_local_port_t        *fct_port;
1092         uint32_t                fct_flags;

1094 #define FCT_STATE_PORT_ONLINE           0x00000001
1095 #define FCT_STATE_NOT_ACKED             0x00000002
1096 #define FCT_STATE_LINK_UP               0x00000010
1097 #define FCT_STATE_LINK_UP_ACKED         0x00000020

1099         emlxs_tgtport_stat_t    fct_stat;

1101         /* Used to save fct_cmd for deferred unsol ELS commands, except FLOGI */
1102         emlxs_buf_t             *fct_wait_head;
1103         emlxs_buf_t             *fct_wait_tail;

1105         /* Used to save context for deferred unsol FLOGIs */
1106         fct_flogi_xchg_t        fx;

1108 #ifdef FCT_IO_TRACE
1109         emlxs_iotrace_t         *iotrace;
1110         uint16_t                iotrace_cnt;
1111         uint16_t                iotrace_index;
1112         kmutex_t                iotrace_mtx;
1113 #endif /* FCT_IO_TRACE */

1115 #endif /* SFCT_SUPPORT */

1117 #ifdef SAN_DIAG_SUPPORT
```



```
1118         uint8_t                 sd_io_latency_state;
1119 #define SD_INVALID      0x00
1120 #define SD_COLLECTING   0x01
1121 #define SD_STOPPED      0x02

1123         /* SD event management list */
1124         uint32_t                sd_event_mask;   /* bit-mask */
1125         emlxs_dfc_event_t       sd_events[MAX_DFC_EVENTS];
1126 #endif

1128 } emlxs_port_t;


1131 /* Host Attn reg */
1132 #define FC_HA_REG(_hba)         ((volatile uint32_t *) \
1133                                     ((_hba)->sli.sli3.ha_reg_addr))

1135 /* Chip Attn reg */
1136 #define FC_CA_REG(_hba)         ((volatile uint32_t *) \
1137                                     ((_hba)->sli.sli3.ca_reg_addr))

1139 /* Host Status reg */
1140 #define FC_HS_REG(_hba)         ((volatile uint32_t *) \
1141                                     ((_hba)->sli.sli3.hs_reg_addr))

1143 /* Host Cntl reg */
1144 #define FC_HC_REG(_hba)         ((volatile uint32_t *) \
1145                                     ((_hba)->sli.sli3.hc_reg_addr))

1147 /* BIU Configuration reg */
1148 #define FC_BC_REG(_hba)         ((volatile uint32_t *) \
1149                                     ((_hba)->sli.sli3.bc_reg_addr))

1151 /* Used by SBUS adapter */
1152 /* TITAN Cntl reg */
1153 #define FC_SHC_REG(_hba)        ((volatile uint32_t *) \
1154                                     ((_hba)->sli.sli3.shc_reg_addr))

1156 /* TITAN Status reg */
1157 #define FC_SHS_REG(_hba)        ((volatile uint32_t *) \
1158                                     ((_hba)->sli.sli3.shs_reg_addr))

1160 /* TITAN Update reg */
1161 #define FC_SHU_REG(_hba)        ((volatile uint32_t *) \
1162                                     ((_hba)->sli.sli3.shu_reg_addr))

1164 /* MPU Semaphore reg */
1165 #define FC_SEMA_REG(_hba)       ((volatile uint32_t *)\
1166                                     ((_hba)->sli.sli4.MPUEPSemaphore_reg_addr))

1168 /* Bootstrap Mailbox Doorbell reg */
1169 #define FC_MBDB_REG(_hba)       ((volatile uint32_t *) \
1170                                     ((_hba)->sli.sli4.MBDB_reg_addr))

1172 /* MQ Doorbell reg */
1173 #define FC_MQDB_REG(_hba)       ((volatile uint32_t *) \
1174                                     ((_hba)->sli.sli4.MQDB_reg_addr))

1176 /* CQ Doorbell reg */
1177 #define FC_CQDB_REG(_hba)       ((volatile uint32_t *) \
1178                                     ((_hba)->sli.sli4.CQDB_reg_addr))

1180 /* WQ Doorbell reg */
1181 #define FC_WQDB_REG(_hba)       ((volatile uint32_t *) \
1182                                     ((_hba)->sli.sli4.WQDB_reg_addr))
```

```
/* RQ Doorbell reg */
#define FC_RQDB_REG(_hba)       ((volatile uint32_t *) \
                                    ((_hba)->sli.sli4.RQDB_reg_addr))


#define FC_SLIM2_MAILBOX(_hba)  ((MAILBOX *)(_hba)->sli.sli3.slim2.virt)

#define FC_SLIM1_MAILBOX(_hba)  ((MAILBOX *)(_hba)->sli.sli3.slim_addr)

#define FC_MAILBOX(_hba)        (((_hba)->flag & FC_SLIM2_MODE) ? \
        FC_SLIM2_MAILBOX(_hba) : FC_SLIM1_MAILBOX(_hba))

#define WRITE_CSR_REG(_hba, _regp, _value) ddi_put32(\
        (_hba)->sli.sli3.csr_acc_handle, (uint32_t *)(_regp), \
        (uint32_t)(_value))

#define READ_CSR_REG(_hba, _regp) ddi_get32(\
        (_hba)->sli.sli3.csr_acc_handle, (uint32_t *)(_regp))

#define WRITE_SLIM_ADDR(_hba, _regp, _value) ddi_put32(\
        (_hba)->sli.sli3.slim_acc_handle, (uint32_t *)(_regp), \
        (uint32_t)(_value))

#define READ_SLIM_ADDR(_hba, _regp) ddi_get32(\
        (_hba)->sli.sli3.slim_acc_handle, (uint32_t *)(_regp))

#define WRITE_SLIM_COPY(_hba, _bufp, _slimp, _wcnt) ddi_rep_put32(\
        (_hba)->sli.sli3.slim_acc_handle, (uint32_t *)(_bufp), \
        (uint32_t *)(_slimp), (_wcnt), DDI_DEV_AUTOINCR)

#define READ_SLIM_COPY(_hba, _bufp, _slimp, _wcnt) ddi_rep_get32(\
        (_hba)->sli.sli3.slim_acc_handle, (uint32_t *)(_bufp), \
        (uint32_t *)(_slimp), (_wcnt), DDI_DEV_AUTOINCR)

/* Used by SBUS adapter */
#define WRITE_SBUS_CSR_REG(_hba, _regp, _value) ddi_put32(\
        (_hba)->sli.sli3.sbus_csr_handle, (uint32_t *)(_regp), \
        (uint32_t)(_value))

#define READ_SBUS_CSR_REG(_hba, _regp) ddi_get32(\
        (_hba)->sli.sli3.sbus_csr_handle, (uint32_t *)(_regp))

#define SBUS_WRITE_FLASH_COPY(_hba, _offset, _value) ddi_put8(\
        (_hba)->sli.sli3.sbus_flash_acc_handle, \
        (uint8_t *)((volatile uint8_t *)(_hba)->sli.sli3.sbus_flash_addr + \
        (_offset)), (uint8_t)(_value))

#define SBUS_READ_FLASH_COPY(_hba, _offset) ddi_get8(\
        (_hba)->sli.sli3.sbus_flash_acc_handle, \
        (uint8_t *)((volatile uint8_t *)(_hba)->sli.sli3.sbus_flash_addr + \
        (_offset)))

/* SLI4 registers */
#define WRITE_BAR1_REG(_hba, _regp, _value) ddi_put32(\
        (_hba)->sli.sli4.bar1_acc_handle, (uint32_t *)(_regp), \
        (uint32_t)(_value))

#define READ_BAR1_REG(_hba, _regp) ddi_get32(\
        (_hba)->sli.sli4.bar1_acc_handle, (uint32_t *)(_regp))

#define WRITE_BAR2_REG(_hba, _regp, _value) ddi_put32(\
        (_hba)->sli.sli4.bar2_acc_handle, (uint32_t *)(_regp), \
        (uint32_t)(_value))

#define READ_BAR2_REG(_hba, _regp) ddi_get32(\
        (_hba)->sli.sli4.bar2_acc_handle, (uint32_t *)(_regp))
```



```


#define EMLXS_STATE_CHANGE(_hba, _state)\
{                                                                       \
        mutex_enter(&EMLXS_PORT_LOCK);                                  \
        EMLXS_STATE_CHANGE_LOCKED((_hba), (_state));                    \
        mutex_exit(&EMLXS_PORT_LOCK);                                   \
}

/* Used when EMLXS_PORT_LOCK is already held */
#define EMLXS_STATE_CHANGE_LOCKED(_hba, _state)                 \
{                                                                       \
        if ((_hba)->state != (_state))                                  \
        {                                                               \
                uint32_t _st = _state;                                  \
                EMLXS_MSGF(EMLXS_CONTEXT,                               \
                        &emlxs_state_msg, "%s --> %s",                  \
                        emlxs_ffstate_xlate((_hba)->state),             \
                        emlxs_ffstate_xlate(_state));                   \
                        (_hba)->state = (_state);                       \
                if ((_st) == FC_ERROR)                                  \
                {                                                       \
                        (_hba)->flag |= FC_HARDWARE_ERROR;              \
                }                                                       \
        }                                                               \
}

#ifdef FMA_SUPPORT
#define EMLXS_CHK_ACC_HANDLE(_hba, _acc) \
        if (emlxs_fm_check_acc_handle(_hba, _acc) != DDI_FM_OK) { \
                EMLXS_MSGF(EMLXS_CONTEXT, \
                    &emlxs_invalid_access_handle_msg, NULL); \
        }
#endif  /* FMA_SUPPORT */

/*
 * This is the HBA control area for the adapter
 */

#ifdef MODSYM_SUPPORT

typedef struct emlxs_modsym
{
        ddi_modhandle_t  mod_fctl;      /* For Leadville */

        /* Leadville (fctl) */
        int             (*fc_fca_attach)(dev_info_t *, fc_fca_tran_t *);
        int             (*fc_fca_detach)(dev_info_t *);
        int             (*fc_fca_init)(struct dev_ops *);

#ifdef SFCT_SUPPORT
        uint32_t        fct_modopen;
        uint32_t        reserved;  /* Padding for alignment */

        ddi_modhandle_t  mod_fct;       /* For Comstar */
        ddi_modhandle_t  mod_stmf;      /* For Comstar */

        /* Comstar (fct) */
        void*   (*fct_alloc)(fct_struct_id_t, int, int);
        void    (*fct_free)(void *);
        void*   (*fct_scsi_task_alloc)(void *, uint16_t, uint32_t, uint8_t *,
                        uint16_t, uint16_t);
        int     (*fct_register_local_port)(fct_local_port_t *);
        void    (*fct_deregister_local_port)(fct_local_port_t *);
        void    (*fct_handle_event)(fct_local_port_t *, int, uint32_t, caddr_t);
        void    (*fct_post_rcvd_cmd)(fct_cmd_t *, stmf_data_buf_t *);
```

```
1316         void    (*fct_ctl)(void *, int, void *);
1317         void    (*fct_queue_cmd_for_termination)(fct_cmd_t *, fct_status_t);
1318         void    (*fct_send_response_done)(fct_cmd_t *, fct_status_t, uint32_t);
1319         void    (*fct_send_cmd_done)(fct_cmd_t *, fct_status_t, uint32_t);
1320         void    (*fct_scsi_data_xfer_done)(fct_cmd_t *, stmf_data_buf_t *,
1321                         uint32_t);
1322         fct_status_t    (*fct_port_shutdown)
1323                                 (fct_local_port_t *, uint32_t, char *);
1324         fct_status_t    (*fct_port_initialize)
1325                                 (fct_local_port_t *, uint32_t, char *);
1326         void            (*fct_cmd_fca_aborted)
1327                                 (fct_cmd_t *, fct_status_t, int);
1328         fct_status_t    (*fct_handle_rcvd_flogi)
1329                                 (fct_local_port_t *, fct_flogi_xchg_t *);

1331         /* Comstar (stmf) */
1332         void*  (*stmf_alloc)(stmf_struct_id_t, int, int);
1333         void   (*stmf_free)(void *);
1334         void    (*stmf_deregister_port_provider) (stmf_port_provider_t *);
1335         int     (*stmf_register_port_provider) (stmf_port_provider_t *);
1336 #endif /* SFCT_SUPPORT */
1337 } emlxs_modsym_t;
1338 extern emlxs_modsym_t emlxs_modsym;

1340 #define MODSYM(_f)      emlxs_modsym._f

1342 #else

1344 #define MODSYM(_f)      _f

1346 #endif /* MODSYM_SUPPORT */



1350 typedef struct RPIHdrTmplate
1351 {
1352         uint32_t        Word[16];  /* 64 bytes */
1353 } RPIHdrTmplate_t;


1356 typedef struct EQ_DESC
1357 {
1358         uint16_t        host_index;
1359         uint16_t        max_index;
1360         uint16_t        qid;
1361         uint16_t        msix_vector;
1362         kmutex_t        lastwq_lock;
1363         uint16_t        lastwq;
1364         MBUF_INFO       addr;
1365 } EQ_DESC_t;


1368 typedef struct CQ_DESC
1369 {
1370         uint16_t        host_index;
1371         uint16_t        max_index;
1372         uint16_t        qid;
1373         uint16_t        eqid;
1374         uint16_t        type;
1375 #define EMLXS_CQ_TYPE_GROUP1    1  /* associated with a MQ and async events */
1376 #define EMLXS_CQ_TYPE_GROUP2    2  /* associated with a WQ and RQ */
1377         uint16_t        rsvd;

1379         MBUF_INFO       addr;
1380         CHANNEL         *channelp; /* ptr to CHANNEL associated with CQ */
```



```
1382 } CQ_DESC_t;


1385 typedef struct WQ_DESC
1386 {
1387         uint16_t        host_index;
1388         uint16_t        max_index;
1389         uint16_t        port_index;
1390         uint16_t        release_depth;
1391 #define WQE_RELEASE_DEPTH       (8 * EMLXS_NUM_WQ_PAGES)
1392         uint16_t        qid;
1393         uint16_t        cqid;
1394         MBUF_INFO       addr;
1395 } WQ_DESC_t;


1398 typedef struct RQ_DESC
1399 {
1400         uint16_t        host_index;
1401         uint16_t        max_index;
1402         uint16_t        qid;
1403         uint16_t        cqid;

1405         MBUF_INFO       addr;
1406         MBUF_INFO       rqb[RQ_DEPTH];

1408         kmutex_t        lock;

1410 } RQ_DESC_t;


1413 typedef struct RXQ_DESC
1414 {
1415         kmutex_t        lock;
1416         emlxs_queue_t   active;

1418 } RXQ_DESC_t;


1421 typedef struct MQ_DESC
1422 {
1423         uint16_t        host_index;
1424         uint16_t        max_index;
1425         uint16_t        qid;
1426         uint16_t        cqid;
1427         MBUF_INFO       addr;
1428 } MQ_DESC_t;


1431 /* Define the number of queues the driver will be using */
1432 #define EMLXS_MAX_EQS   EMLXS_MSI_MAX_INTRS
1433 #define EMLXS_MAX_WQS   EMLXS_MSI_MAX_INTRS
1434 #define EMLXS_MAX_RQS   2       /* ONLY 1 pair is allowed */
1435 #define EMLXS_MAX_MQS   1

1437 /* One CQ for each WQ & (RQ pair) plus one for the MQ */
1438 #define EMLXS_MAX_CQS   (EMLXS_MAX_WQS + (EMLXS_MAX_RQS/2) + 1)

1440 /* The First CQ created is ALWAYS for mbox / event handling */
1441 #define EMLXS_CQ_MBOX           0

1443 /* The Second CQ created is ALWAYS for unsol rcv handling */
1444 /* At this time we are allowing ONLY 1 pair of RQs */
1445 #define EMLXS_CQ_RCV            1

1447 /* The remaining CQs are for WQ completions */
```

```
1448 #define EMLXS_CQ_OFFSET_WQ      2


1451 /* FCFI RQ Configuration */
1452 #define EMLXS_FCFI_RQ0_INDEX    0
1453 #define EMLXS_FCFI_RQ0_RMASK    0 /* match all */
1454 #define EMLXS_FCFI_RQ0_RCTL     0 /* match all */
1455 #define EMLXS_FCFI_RQ0_TMASK    0 /* match all */
1456 #define EMLXS_FCFI_RQ0_TYPE     0 /* match all */

1458 /* Define the maximum value for a Queue Id */
1459 #define EMLXS_MAX_EQ_IDS        256
1460 #define EMLXS_MAX_CQ_IDS        1024
1461 #define EMLXS_MAX_WQ_IDS        1024
1462 #define EMLXS_MAX_RQ_IDS        4

1464 #define EMLXS_RXQ_ELS           0
1465 #define EMLXS_RXQ_CT            1
1466 #define EMLXS_MAX_RXQS          2

1468 #define PCI_CONFIG_SIZE   0x80

1470 typedef struct emlxs_sli3
1471 {
1472         /* SLIM management */
1473         MATCHMAP        slim2;

1475         /* HBQ management */
1476         uint32_t        hbq_count;      /* Total number of HBQs */
1477                                         /* configured */
1478         HBQ_INIT_t      hbq_table[EMLXS_NUM_HBQ];

1480         /* Adapter memory management */
1481         caddr_t         csr_addr;
1482         caddr_t         slim_addr;
1483         ddi_acc_handle_t csr_acc_handle;
1484         ddi_acc_handle_t slim_acc_handle;

1486         /* SBUS adapter management */
1487         caddr_t         sbus_flash_addr;        /* Virt addr of R/W */
1488                                                 /* Flash */
1489         caddr_t         sbus_core_addr;         /* Virt addr of TITAN */
1490                                                 /* CORE */
1491         caddr_t         sbus_csr_addr;          /* Virt addr of TITAN */
1492                                                 /* CSR */
1493         ddi_acc_handle_t sbus_flash_acc_handle;
1494         ddi_acc_handle_t sbus_core_acc_handle;
1495         ddi_acc_handle_t sbus_csr_handle;

1497         /* SLI 2/3 Adapter register management */
1498         uint32_t        *bc_reg_addr;   /* virtual offset for BIU */
1499                                         /* config reg */
1500         uint32_t        *ha_reg_addr;   /* virtual offset for host */
1501                                         /* attn reg */
1502         uint32_t        *hc_reg_addr;   /* virtual offset for host */
1503                                         /* ctl reg */
1504         uint32_t        *ca_reg_addr;   /* virtual offset for FF */
1505                                         /* attn reg */
1506         uint32_t        *hs_reg_addr;   /* virtual offset for */
1507                                         /* status reg */
1508         uint32_t        *shc_reg_addr;  /* virtual offset for SBUS */
1509                                         /* Ctrl reg */
1510         uint32_t        *shs_reg_addr;  /* virtual offset for SBUS */
1511                                         /* Status reg */
1512         uint32_t        *shu_reg_addr;  /* virtual offset for SBUS */
1513                                         /* Update reg */
```



```
1514         uint16_t        hgp_ring_offset;
1515         uint16_t        hgp_hbq_offset;
1516         uint16_t        iocb_cmd_size;
1517         uint16_t        iocb_rsp_size;
1518         uint32_t        hc_copy;        /* local copy of HC register */

1520         /* Ring management */
1521         uint32_t        ring_count;
1522         emlxs_ring_t    ring[MAX_RINGS];
1523         kmutex_t        ring_cmd_lock[MAX_RINGS];
1524         uint8_t         ring_masks[4];  /* number of masks/rings used */
1525         uint8_t         ring_rval[6];
1526         uint8_t         ring_rmask[6];
1527         uint8_t         ring_tval[6];
1528         uint8_t         ring_tmask[6];

1530         /* Protected by EMLXS_FCTAB_LOCK */
1531 #ifdef EMLXS_SPARC
1532         MEMSEG          fcp_bpl_seg;
1533         MATCHMAP        **fcp_bpl_table; /* iotag table for */
1534                                         /* bpl buffers */
1535 #endif  /* EMLXS_SPARC */
1536         uint32_t        mem_bpl_size;
1537 } emlxs_sli3_t;

1539 typedef struct emlxs_sli4
1540 {
1541         MATCHMAP        bootstrapmb;
1542         caddr_t         bar1_addr;
1543         caddr_t         bar2_addr;
1544         ddi_acc_handle_t bar1_acc_handle;
1545         ddi_acc_handle_t bar2_acc_handle;

1547         /* SLI4 Adapter register management */
1548         uint32_t        *MPUEPSemaphore_reg_addr;
1549         uint32_t        *MBDB_reg_addr;

1551         uint32_t        *CQDB_reg_addr;
1552         uint32_t        *MQDB_reg_addr;
1553         uint32_t        *WQDB_reg_addr;
1554         uint32_t        *RQDB_reg_addr;

1556         uint32_t        flag;
1557 #define EMLXS_SLI4_INTR_ENABLED         0x00000001
1558 #define EMLXS_SLI4_HW_ERROR             0x00000002
1559 #define EMLXS_SLI4_DOWN_LINK            0x00000004

1561         uint16_t        XRICount;
1562         uint16_t        XRIBase;
1563         uint16_t        RPICount;
1564         uint16_t        RPIBase;
1565         uint16_t        VPICount;
1566         uint16_t        VPIBase;
1567         uint16_t        VFICount;
1568         uint16_t        VFIBase;
1569         uint16_t        FCFICount;

1571         kmutex_t        fcf_lock;
1572         FCFTable_t      fcftab;
1573         VFIobj_t        *VFI_table;

1575         /* Save Config Region 23 info */
1576         tlv_fcoe_t      cfgFCOE;
1577         tlv_fcfconnectlist_t    cfgFCF;

1579         MBUF_INFO       slim2;
```

```
1580         MBUF_INFO       dump_region;
1581 #define EMLXS_DUMP_REGION_SIZE  1024

1583         RPIobj_t        *RPIp;
1584         MBUF_INFO       HeaderTmplate;
1585         XRIobj_t        *XRIp;

1587         /* Double linked list for available XRIs */
1588         XRIobj_t        *XRIfree_f;
1589         XRIobj_t        *XRIfree_b;
1590         uint32_t        xrif_count;
1591         uint32_t        mem_sgl_size;

1593         /* Double linked list for XRIs in use */
1594         XRIobj_t        *XRIinuse_f;
1595         XRIobj_t        *XRIinuse_b;
1596         uint32_t        xria_count;

1598         kmutex_t        que_lock[EMLXS_MAX_WQS];
1599         EQ_DESC_t       eq[EMLXS_MAX_EQS];
1600         CQ_DESC_t       cq[EMLXS_MAX_CQS];
1601         WQ_DESC_t       wq[EMLXS_MAX_WQS];
1602         RQ_DESC_t       rq[EMLXS_MAX_RQS];
1603         MQ_DESC_t       mq;

1605         /* Used to map a queue ID to a queue DESC_t */
1606         uint16_t        eq_map[EMLXS_MAX_EQ_IDS];
1607         uint16_t        cq_map[EMLXS_MAX_CQ_IDS];
1608         uint16_t        wq_map[EMLXS_MAX_WQ_IDS];
1609         uint16_t        rq_map[EMLXS_MAX_RQ_IDS];

1611         RXQ_DESC_t      rxq[EMLXS_MAX_RXQS];

1613         uint32_t        ue_mask_lo;
1614         uint32_t        ue_mask_hi;

1616 } emlxs_sli4_t;


1619 typedef struct emlxs_sli_api
1620 {
1621         int             (*sli_map_hdw)();
1622         void            (*sli_unmap_hdw)();
1623         int32_t         (*sli_online)();
1624         void            (*sli_offline)();
1625         uint32_t        (*sli_hba_reset)();
1626         void            (*sli_hba_kill)();
1627         void            (*sli_issue_iocb_cmd)();
1628         uint32_t        (*sli_issue_mbox_cmd)();
1629         uint32_t        (*sli_prep_fct_iocb)();
1630         uint32_t        (*sli_prep_fcp_iocb)();
1631         uint32_t        (*sli_prep_ip_iocb)();
1632         uint32_t        (*sli_prep_els_iocb)();
1633         uint32_t        (*sli_prep_ct_iocb)();
1634         void            (*sli_poll_intr)();
1635         int32_t         (*sli_intx_intr)();
1636         uint32_t        (*sli_msi_intr)();
1637         void            (*sli_disable_intr)();
1638         void            (*sli_timer)();
1639         void            (*sli_poll_erratt)();

1641 } emlxs_sli_api_t;


1644 typedef struct emlxs_hba
1645 {
```



```
1646         dev_info_t      *dip;
1647         int32_t         emlxinst;
1648         int32_t         ddiinst;
1649         uint8_t         pci_function_number;
1650         uint8_t         pci_device_number;
1651         uint8_t         pci_bus_number;
1652         uint8_t         pci_cap_offset[PCI_CAP_MAX_PTR];

1654 #ifdef FMA_SUPPORT
1655         int32_t         fm_caps;        /* FMA capabilities */
1656 #endif  /* FMA_SUPPORT */
1657         fc_fca_tran_t   *fca_tran;

1659         /* DMA attributes */
1660         ddi_dma_attr_t  dma_attr;
1661         ddi_dma_attr_t  dma_attr_ro;
1662         ddi_dma_attr_t  dma_attr_1sg;
1663         ddi_dma_attr_t  dma_attr_fcip_rsp;

1665         /* HBA Info */
1666         emlxs_model_t   model_info;
1667         emlxs_vpd_t     vpd;    /* vital product data */
1668         NAME_TYPE       wwnn;
1669         NAME_TYPE       wwpn;
1670         char            snn[256];
1671         char            spn[256];
1672         PROG_ID         load_list[MAX_LOAD_ENTRY];
1673         WAKE_UP_PARMS   wakeup_parms;
1674         uint32_t        max_nodes;
1675         uint32_t        io_throttle;
1676         uint32_t        io_active;
1677         uint32_t        bus_type;
1678 #define PCI_FC          0
1679 #define SBUS_FC         1

1681         /* Link management */
1682         uint32_t        link_event_tag;
1683         uint8_t         topology;
1684         uint8_t         linkspeed;
1685         uint16_t        qos_linkspeed;
1686         uint32_t        linkup_wait_flag;
1687         kcondvar_t      linkup_lock_cv;
1688         kmutex_t        linkup_lock;

1690         /* Memory Pool management */
1691         emlxs_memseg_t  memseg[FC_MAX_SEG];     /* memory for buffer */
1692                                                         /* structures */
1693         kmutex_t        memget_lock;    /* locks all memory pools get */
1694         kmutex_t        memput_lock;    /* locks all memory pools put */

1696         /* Fibre Channel Service Parameters */
1697         SERV_PARM       sparam;
1698         uint32_t        fc_edtov;       /* E_D_TOV timer value */
1699         uint32_t        fc_arbtov;      /* ARB_TOV timer value */
1700         uint32_t        fc_ratov;       /* R_A_TOV timer value */
1701         uint32_t        fc_rttov;       /* R_T_TOV timer value */
1702         uint32_t        fc_altov;       /* AL_TOV timer value */
1703         uint32_t        fc_crtov;       /* C_R_TOV timer value */
1704         uint32_t        fc_citov;       /* C_I_TOV timer value */

1706         /* Adapter State management */
1707         int32_t         state;
1708 #define FC_ERROR                0x01    /* Adapter shutdown */
1709 #define FC_KILLED               0x02    /* Adapter interlocked/killed */
1710 #define FC_WARM_START           0x03    /* Adapter reset, but not restarted */
1711 #define FC_INIT_START           0x10    /* Adapter restarted */
```

```
#define FC_INIT_NVPARAMS        0x11
#define FC_INIT_REV             0x12
#define FC_INIT_CFGPORT         0x13
#define FC_INIT_CFGRING         0x14
#define FC_INIT_INITLINK        0x15
#define FC_LINK_DOWN            0x20
#define FC_LINK_DOWN_PERSIST    0x21
#define FC_LINK_UP              0x30
#define FC_CLEAR_LA             0x31
#define FC_READY                0x40

        uint32_t        flag;
#define FC_ONLINING_MODE        0x00000001
#define FC_ONLINE_MODE          0x00000002
#define FC_OFFLINING_MODE       0x00000004
#define FC_OFFLINE_MODE         0x00000008

#define FC_NPIV_ENABLED         0x00000010      /* NPIV enabled on adapter    */
#define FC_NPIV_SUPPORTED       0x00000020      /* NPIV supported on fabric   */
#define FC_NPIV_UNSUPPORTED     0x00000040      /* NPIV unsupported on fabric */
#define FC_NPIV_LINKUP          0x00000100      /* NPIV enabled, supported, */
                                                /* and link is ready */
#define FC_NPIV_DELAY_REQUIRED  0x00000200      /* Delay issuing FLOGI/FDISC */
                                                /* and NameServer cmds */

#define FC_BOOTSTRAPMB_INIT     0x00000400
#define FC_FIP_SUPPORTED        0x00000800      /* FIP supported */

#define FC_FABRIC_ATTACHED      0x00001000
#define FC_PT_TO_PT             0x00002000
#define FC_BYPASSED_MODE        0x00004000
#define FC_MENLO_MODE           0x00008000      /* Menlo maintenance mode */

#define FC_DUMP_SAFE            0x00010000      /* Safe to DUMP */
#define FC_DUMP_ACTIVE          0x00020000      /* DUMP in progress */
#define FC_NEW_FABRIC           0x00040000

#define FC_SLIM2_MODE           0x00100000      /* SLIM in host memory */
#define FC_INTERLOCKED          0x00200000
#define FC_HBQ_ENABLED          0x00400000
#define FC_ASYNC_EVENTS         0x00800000

#define FC_ILB_MODE             0x01000000
#define FC_ELB_MODE             0x02000000
#define FC_LOOPBACK_MODE        0x03000000      /* Loopback Mode Mask */
#define FC_DUMP                 0x04000000      /* DUMP in progress */
#define FC_SHUTDOWN             0x08000000      /* SHUTDOWN in progress */

#define FC_OVERTEMP_EVENT       0x10000000      /* FC_ERROR reason: */
                                                /* over temperature event */
#define FC_MBOX_TIMEOUT         0x20000000      /* FC_ERROR reason: */
                                                /* mailbox timeout event */
#define FC_DMA_CHECK_ERROR      0x40000000      /* Shared memory (slim,..) */
                                                /* DMA handle went bad */
#define FC_HARDWARE_ERROR       0x80000000      /* FC_ERROR state triggered */

#define FC_RESET_MASK           0x00030C1F      /* Bits to protect during */
                                                /* a hard reset */
#define FC_LINKDOWN_MASK        0xFFF30C1F      /* Bits to protect during */
                                                /* a linkdown */

        uint32_t fw_timer;
        uint32_t fw_flag;
#define FW_UPDATE_NEEDED        0x00000001
#define FW_UPDATE_KERNEL        0x00000002
```



```
        uint32_t temperature;                   /* Last reported temperature */

        /* SBUS adapter management */
        caddr_t         sbus_pci_addr;          /* Virt addr of TITAN */
                                                /* pci config */
        ddi_acc_handle_t sbus_pci_handle;

        /* PCI BUS adapter management */
        caddr_t         pci_addr;
        ddi_acc_handle_t pci_acc_handle;

        uint32_t        sli_mode;
#define EMLXS_HBA_SLI1_MODE     1
#define EMLXS_HBA_SLI2_MODE     2
#define EMLXS_HBA_SLI3_MODE     3
#define EMLXS_HBA_SLI4_MODE     4

        /* SLI private data */
        union {
                emlxs_sli3_t sli3;
                emlxs_sli4_t sli4;
        } sli;

        /* SLI API entry point routines */
        emlxs_sli_api_t sli_api;

        uint32_t        io_poll_count;  /* Number of poll commands */
                                        /* in progress */

        /* IO Completion management */
        uint32_t        iodone_count;   /* Number of IO's on done Q */
        /* Protected by EMLXS_PORT_LOCK  */
        emlxs_buf_t     *iodone_list;   /* fc_packet being deferred */
        emlxs_buf_t     *iodone_tail;   /* fc_packet being deferred */
        emlxs_thread_t  iodone_thread;
        emlxs_thread_t  *spawn_thread_head;
        emlxs_thread_t  *spawn_thread_tail;
        kmutex_t        spawn_lock;
        uint32_t        spawn_open;

        /* IO Channel management */
        int32_t         chan_count;
        emlxs_channel_t chan[MAX_CHANNEL];
        kmutex_t        channel_tx_lock;
        uint8_t         channel_fcp;    /* Default channel to use for FCP IO */
#define CHANNEL_FCT channel_fcp
        uint8_t         channel_ip;     /* Default channel to use for IP IO */
        uint8_t         channel_els;    /* Default channel to use for ELS IO */
        uint8_t         channel_ct;     /* Default channel to use for CT IO */

        /* IOTag management */
        emlxs_buf_t     **fc_table;     /* sc_buf pointers indexed by */
                                        /* iotag */
        uint16_t        fc_iotag;       /* used to identify I/Os */
        uint16_t        fc_oor_iotag;   /* OutOfRange (fc_table) iotags */
                                        /* typically used for Abort/close */
#define EMLXS_MAX_ABORT_TAG     0x7fff
        uint16_t        max_iotag;      /* ALL IOCBs except aborts */
        kmutex_t        iotag_lock;
        uint32_t        io_count;               /* No of IO holding */
                                                /* regular iotag */
        uint32_t        channel_tx_count;       /* No of IO on tx Q */

        /* Mailbox Management */
        uint32_t        mbox_queue_flag;
        emlxs_queue_t   mbox_queue;
```

```
1844         void            *mbox_mqe;      /* active mbox mqe */
1845         void            *mbox_mbq;      /* active MAILBOXQ */
1846         kcondvar_t      mbox_lock_cv;   /* MBX_SLEEP */
1847         kmutex_t        mbox_lock;      /* MBX_SLEEP */
1848         uint32_t        mbox_timer;

1850         /* Interrupt management */
1851         void            *intr_arg;
1852         uint32_t        intr_unclaimed;
1853         uint32_t        intr_autoClear;
1854         uint32_t        intr_flags;
1855 #define EMLXS_INTX_INITED       0x0001
1856 #define EMLXS_INTX_ADDED        0x0002
1857 #define EMLXS_MSI_ENABLED       0x0010
1858 #define EMLXS_MSI_INITED        0x0020
1859 #define EMLXS_MSI_ADDED         0x0040
1860 #define EMLXS_INTR_INITED       (EMLXS_INTX_INITED|EMLXS_MSI_INITED)
1861 #define EMLXS_INTR_ADDED        (EMLXS_INTX_ADDED|EMLXS_MSI_ADDED)

1863 #ifdef MSI_SUPPORT
1864         ddi_intr_handle_t *intr_htable;
1865         uint32_t        *intr_pri;
1866         int32_t         *intr_cap;
1867         uint32_t        intr_count;
1868         uint32_t        intr_type;
1869         uint32_t        intr_cond;
1870         uint32_t        intr_map[EMLXS_MSI_MAX_INTRS];
1871         uint32_t        intr_mask;

1873         kmutex_t        msiid_lock; /* for last_msiid */
1874         int             last_msiid;

1876         kmutex_t        intr_lock[EMLXS_MSI_MAX_INTRS];
1877         int                     chan2msi[MAX_CHANNEL];
1878                                 /* Index is the channel id */
1879         int                     msi2chan[EMLXS_MSI_MAX_INTRS];
1880                                 /* Index is the MSX-X msg id */
1881 #endif  /* MSI_SUPPORT */

1883         uint32_t        heartbeat_timer;
1884         uint32_t        heartbeat_flag;
1885         uint32_t        heartbeat_active;

1887         /* IOCTL management */
1888         kmutex_t        ioctl_lock;
1889         uint32_t        ioctl_flags;
1890 #define EMLXS_OPEN              0x00000001
1891 #define EMLXS_OPEN_EXCLUSIVE    0x00000002

1893         /* Timer management */
1894         kcondvar_t      timer_lock_cv;
1895         kmutex_t        timer_lock;
1896         timeout_id_t    timer_id;
1897         uint32_t        timer_tics;
1898         uint32_t        timer_flags;
1899 #define EMLXS_TIMER_STARTED     0x0000001
1900 #define EMLXS_TIMER_BUSY        0x0000002
1901 #define EMLXS_TIMER_KILL        0x0000004
1902 #define EMLXS_TIMER_ENDED       0x0000008

1904         /* Misc Timers */
1905         uint32_t        linkup_timer;
1906         uint32_t        discovery_timer;
1907         uint32_t        pkt_timer;

1909         /* Power Management */
```



```
1910         uint32_t        pm_state;
1911         /* pm_state */
1912 #define EMLXS_PM_IN_ATTACH      0x00000001
1913 #define EMLXS_PM_IN_DETACH      0x00000002
1914 #define EMLXS_PM_IN_SOL_CB      0x00000010
1915 #define EMLXS_PM_IN_UNSOL_CB    0x00000020
1916 #define EMLXS_PM_IN_LINK_RESET  0x00000100
1917 #define EMLXS_PM_IN_HARD_RESET  0x00000200
1918 #define EMLXS_PM_SUSPENDED      0x01000000

1920         uint32_t        pm_level;
1921         /* pm_level */
1922 #define EMLXS_PM_ADAPTER_DOWN   0
1923 #define EMLXS_PM_ADAPTER_UP     1

1925         uint32_t        pm_busy;
1926         kmutex_t        pm_lock;
1927         uint8_t         pm_config[PCI_CONFIG_SIZE];
1928 #ifdef IDLE_TIMER
1929         uint32_t        pm_idle_timer;
1930         uint32_t        pm_active;      /* Only used by timer */
1931 #endif  /* IDLE_TIMER */

1933         /* Loopback management */
1934         uint32_t        loopback_tics;
1935         void            *loopback_pkt;

1937         /* Event management */
1938         emlxs_event_queue_t event_queue;
1939         uint32_t        event_mask;
1940         uint32_t        event_timer;
1941         emlxs_dfc_event_t dfc_event[MAX_DFC_EVENTS];
1942         emlxs_hba_event_t hba_event;

1944         /* Parameter management */
1945         emlxs_config_t  config[NUM_CFG_PARAM];

1947         /* Driver stat management */
1948         kstat_t         *kstat;
1949         emlxs_stats_t   stats;

1951         /* Log management */
1952         emlxs_msg_log_t log;

1954         /* Port managment */
1955         uint32_t        vpi_base;
1956         uint32_t        vpi_max;
1957         uint32_t        vpi_high;
1958         uint32_t        num_of_ports;

1960         kmutex_t        port_lock;      /* locks port, nodes, rings */
1961         emlxs_port_t    port[MAX_VPORTS + 1];   /* port specific info */
1962                                                 /* Last one is for */
1963                                                 /* NPIV ready test */

1965 #ifdef DHCHAP_SUPPORT
1966         kmutex_t        dhc_lock;
1967         kmutex_t        auth_lock;
1968         emlxs_auth_cfg_t        auth_cfg;       /* Default auth_cfg. */
1969                                                 /* Points to list of entries. */
1970                                                 /* Protected by auth_lock */
1971         uint32_t        auth_cfg_count;
1972         emlxs_auth_key_t        auth_key;       /* Default auth_key. */
1973                                                 /* Points to list of entries. */
1974                                                 /* Protected by auth_lock */
1975         uint32_t        auth_key_count;
```

```
1976         uint32_t        rdn_flag;
1977 #endif  /* DHCHAP_SUPPORT */

1979         uint16_t        ini_mode;
1980         uint16_t        tgt_mode;

1982 #ifdef TEST_SUPPORT
1983         uint32_t        underrun_counter;
1984 #endif /* TEST_SUPPORT */

1986 #ifdef MODFW_SUPPORT
1987         ddi_modhandle_t fw_modhandle;
1988 #endif /* MODFW_SUPPORT */

1990 #ifdef DUMP_SUPPORT
1991         emlxs_file_t    dump_txtfile;
1992         emlxs_file_t    dump_dmpfile;
1993         emlxs_file_t    dump_ceefile;
1994         kmutex_t        dump_lock;
1995 #define EMLXS_DUMP_LOCK         hba->dump_lock
1996 #define EMLXS_TXT_FILE          1
1997 #define EMLXS_DMP_FILE          2
1998 #define EMLXS_CEE_FILE          3

2000 #define EMLXS_DRV_DUMP          0
2001 #define EMLXS_TEMP_DUMP         1
2002 #define EMLXS_USER_DUMP         2

2004 #endif /* DUMP_SUPPORT */

2006 } emlxs_hba_t;

2008 #define EMLXS_SLI_MAP_HDW               (hba->sli_api.sli_map_hdw)
2009 #define EMLXS_SLI_UNMAP_HDW             (hba->sli_api.sli_unmap_hdw)
2010 #define EMLXS_SLI_ONLINE                (hba->sli_api.sli_online)
2011 #define EMLXS_SLI_OFFLINE               (hba->sli_api.sli_offline)
2012 #define EMLXS_SLI_HBA_RESET             (hba->sli_api.sli_hba_reset)
2013 #define EMLXS_SLI_HBA_KILL              (hba->sli_api.sli_hba_kill)
2014 #define EMLXS_SLI_ISSUE_IOCB_CMD        (hba->sli_api.sli_issue_iocb_cmd)
2015 #define EMLXS_SLI_ISSUE_MBOX_CMD        (hba->sli_api.sli_issue_mbox_cmd)
2016 #define EMLXS_SLI_PREP_FCT_IOCB         (hba->sli_api.sli_prep_fct_iocb)
2017 #define EMLXS_SLI_PREP_FCP_IOCB         (hba->sli_api.sli_prep_fcp_iocb)
2018 #define EMLXS_SLI_PREP_IP_IOCB          (hba->sli_api.sli_prep_ip_iocb)
2019 #define EMLXS_SLI_PREP_ELS_IOCB         (hba->sli_api.sli_prep_els_iocb)
2020 #define EMLXS_SLI_PREP_CT_IOCB          (hba->sli_api.sli_prep_ct_iocb)
2021 #define EMLXS_SLI_POLL_INTR             (hba->sli_api.sli_poll_intr)
2022 #define EMLXS_SLI_INTX_INTR             (hba->sli_api.sli_intx_intr)
2023 #define EMLXS_SLI_MSI_INTR              (hba->sli_api.sli_msi_intr)
2024 #define EMLXS_SLI_DISABLE_INTR          (hba->sli_api.sli_disable_intr)
2025 #define EMLXS_SLI_TIMER                 (hba->sli_api.sli_timer)
2026 #define EMLXS_SLI_POLL_ERRATT           (hba->sli_api.sli_poll_erratt)

2028 #define EMLXS_HBA_T  1  /* flag emlxs_hba_t is already typedefed */

2030 #ifdef MSI_SUPPORT
2031 #define EMLXS_INTR_INIT(_hba, _m)               emlxs_msi_init(_hba, _m)
2032 #define EMLXS_INTR_UNINIT(_hba)                 emlxs_msi_uninit(_hba)
2033 #define EMLXS_INTR_ADD(_hba)                    emlxs_msi_add(_hba)
2034 #define EMLXS_INTR_REMOVE(_hba)                 emlxs_msi_remove(_hba)
2035 #else
2036 #define EMLXS_INTR_INIT(_hba, _m)               emlxs_intx_init(_hba, _m)
2037 #define EMLXS_INTR_UNINIT(_hba)                 emlxs_intx_uninit(_hba)
2038 #define EMLXS_INTR_ADD(_hba)                    emlxs_intx_add(_hba)
2039 #define EMLXS_INTR_REMOVE(_hba)                 emlxs_intx_remove(_hba)
2040 #endif  /* MSI_SUPPORT */
```



```
2043 /* Power Management Component */
2044 #define EMLXS_PM_ADAPTER        0


2047 #define DRV_TIME        (uint32_t)((gethrtime() - emlxs_device.drv_timestamp) /
  26 #define DRV_TIME        (uint32_t)(ddi_get_time() - emlxs_device.drv_timestamp)

2049 #define HBA                     port->hba
2050 #define PPORT                   hba->port[0]
2051 #define VPORT(x)                hba->port[x]
2052 #define EMLXS_TIMER_LOCK        hba->timer_lock
2053 #define VPD                     hba->vpd
2054 #define CFG                     hba->config[0]
2055 #define LOG                     hba->log
2056 #define EVENTQ                  hba->event_queue
2057 #define EMLXS_MBOX_LOCK         hba->mbox_lock
2058 #define EMLXS_MBOX_CV           hba->mbox_lock_cv
2059 #define EMLXS_LINKUP_LOCK       hba->linkup_lock
2060 #define EMLXS_LINKUP_CV         hba->linkup_lock_cv
2061 #define EMLXS_TX_CHANNEL_LOCK   hba->channel_tx_lock    /* ring txq lock */
2062 #define EMLXS_MEMGET_LOCK       hba->memget_lock        /* mempool get lock */
2063 #define EMLXS_MEMPUT_LOCK       hba->memput_lock        /* mempool put lock */
2064 #define EMLXS_IOCTL_LOCK        hba->ioctl_lock         /* ioctl lock */
2065 #define EMLXS_SPAWN_LOCK        hba->spawn_lock         /* spawn lock */
2066 #define EMLXS_PM_LOCK           hba->pm_lock            /* pm lock */
2067 #define HBASTATS                hba->stats
2068 #define EMLXS_CMD_RING_LOCK(n)  hba->sli.sli3.ring_cmd_lock[n]

2070 #define EMLXS_QUE_LOCK(n)       hba->sli.sli4.que_lock[n]
2071 #define EMLXS_MSIID_LOCK        hba->msiid_lock

2073 #define EMLXS_FCTAB_LOCK        hba->iotag_lock

2075 #define EMLXS_FCF_LOCK          hba->sli.sli4.fcf_lock

2077 #define EMLXS_PORT_LOCK         hba->port_lock          /* locks ports, */
2078                                                         /* nodes, rings */
2079 #define EMLXS_INTR_LOCK(_id)    hba->intr_lock[_id]     /* locks intr threads */

2081 #define EMLXS_PKT_LOCK          port->pkt_lock          /* used for pkt */
2082                                                         /* polling */
2083 #define EMLXS_PKT_CV            port->pkt_lock_cv       /* Used for pkt */
2084                                                         /* polling */
2085 #define EMLXS_UB_LOCK           port->ub_lock           /* locks unsolicited */
2086                                                         /* buffer pool */

2088 /* These SWAPs will swap on any platform */
2089 #define SWAP32_BUFFER(_b, _c)           emlxs_swap32_buffer(_b, _c)
2090 #define SWAP32_BCOPY(_s, _d, _c)        emlxs_swap32_bcopy(_s, _d, _c)

2092 #define SWAP64(_x)      ((((uint64_t)(_x) & 0xFF)<<56) | \
2093                          (((uint64_t)(_x) & 0xFF00)<<40) | \
2094                          (((uint64_t)(_x) & 0xFF0000)<<24) | \
2095                          (((uint64_t)(_x) & 0xFF000000)<<8) | \
2096                          (((uint64_t)(_x) & 0xFF00000000)>>8) | \
2097                          (((uint64_t)(_x) & 0xFF0000000000)>>24) | \
2098                          (((uint64_t)(_x) & 0xFF000000000000)>>40) | \
2099                          (((uint64_t)(_x) & 0xFF00000000000000)>>56))

2101 #define SWAP32(_x)      ((((uint32_t)(_x) & 0xFF)<<24) | \
2102                          (((uint32_t)(_x) & 0xFF00)<<8) | \
2103                          (((uint32_t)(_x) & 0xFF0000)>>8) | \
2104                          (((uint32_t)(_x) & 0xFF000000)>>24))

2106 #define SWAP16(_x)      ((((uint16_t)(_x) & 0xFF)<<8) | \
```

```
2107                            (((uint16_t)(_x) & 0xFF00)>>8))

2109 #define SWAP24_LO(_x)   ((((uint32_t)(_x) & 0xFF)<<16) | \
2110                            ((uint32_t)(_x) & 0xFF00FF00) | \
2111                            (((uint32_t)(_x) & 0x00FF0000)>>16))

2113 #define SWAP24_HI(_x)   (((uint32_t)(_x) & 0x00FF00FF) | \
2114                            (((uint32_t)(_x) & 0x0000FF00)<<16) | \
2115                            (((uint32_t)(_x) & 0xFF000000)>>16))

2117 /* These LE_SWAPs will only swap on a LE platform */
2118 #ifdef EMLXS_LITTLE_ENDIAN
2119 #define LE_SWAP32_BUFFER(_b, _c)        SWAP32_BUFFER(_b, _c)
2120 #define LE_SWAP32_BCOPY(_s, _d, _c)     SWAP32_BCOPY(_s, _d, _c)
2121 #define LE_SWAP64(_x)                   SWAP64(_x)
2122 #define LE_SWAP32(_x)                   SWAP32(_x)
2123 #define LE_SWAP16(_x)                   SWAP16(_x)
2124 #define LE_SWAP24_LO(_x)                SWAP24_LO(X)
2125 #define LE_SWAP24_HI(_x)                SWAP24_HI(X)

2127 #if (EMLXS_MODREVX == EMLXS_MODREV2X)
2128 #undef  LE_SWAP24_LO
2129 #define LE_SWAP24_LO(_x)                (_x)
2130 #undef  LE_SWAP24_HI
2131 #define LE_SWAP24_HI(_x)                (_x)
2132 #endif  /* EMLXS_MODREV2X */

2134 #else /* BIG ENDIAN */
2135 #define LE_SWAP32_BUFFER(_b, _c)
2136 #define LE_SWAP32_BCOPY(_s, _d, _c)     bcopy(_s, _d, _c)
2137 #define LE_SWAP64(_x)                   (_x)
2138 #define LE_SWAP32(_x)                   (_x)
2139 #define LE_SWAP16(_x)                   (_x)
2140 #define LE_SWAP24_LO(_x)                (_x)
2141 #define LE_SWAP24_HI(_x)                (_x)
2142 #endif /* EMLXS_LITTLE_ENDIAN */

2144 /* These BE_SWAPs will only swap on a BE platform */
2145 #ifdef EMLXS_BIG_ENDIAN
2146 #define BE_SWAP32_BUFFER(_b, _c)        SWAP32_BUFFER(_b, _c)
2147 #define BE_SWAP32_BCOPY(_s, _d, _c)     SWAP32_BCOPY(_s, _d, _c)
2148 #define BE_SWAP64(_x)                   SWAP64(_x)
2149 #define BE_SWAP32(_x)                   SWAP32(_x)
2150 #define BE_SWAP16(_x)                   SWAP16(_x)
2151 #else /* LITTLE ENDIAN */
2152 #define BE_SWAP32_BUFFER(_b, _c)
2153 #define BE_SWAP32_BCOPY(_s, _d, _c)     bcopy(_s, _d, _c)
2154 #define BE_SWAP64(_x)                   (_x)
2155 #define BE_SWAP32(_x)                   (_x)
2156 #define BE_SWAP16(_x)                   (_x)
2157 #endif /* EMLXS_BIG_ENDIAN */

2159 #ifdef  __cplusplus
2160 }
```

_____***unchanged_portion_omitted_***