Appendix B: C++ Code

This appendix presents the source code of the research that had been developed in OPNET Modeler simulation software.

```
/* This variable carries the header into the object file */
const char mobile_ip_mn_pr_c [] = "MIL_3_Tfile_Hdr_ 30A op_runsim_dev 7
4F427647 4F427647 1 Khanista-PC Khanista 0 0 none none 0 0 none 0 0 0 0 0 0 0
0 4871 6
";
#include <string.h>

/* OPNET system definitions */
#include <opnet.h>

/* Header Block */

#include    "ip_rte_support.h"
#include    "mobile_ip_support.h"
#include    "ip_addr_v4.h"
#include    "ip_mcast_support.h"
#include    "mobility_support.h"
#include    "ip_igmp_support.h"
#include    "ip_pim_sm_support.h"
#include    "udp_api.h"
#include    "ip_icmp_pk.h"
#include    "ip_dgram_sup.h"
#include    "math.h"

#define     IP_PK                   is_ip_pk
#define     REG_PK                  is_reg_pk
#define     AD_RECEPTION            is_ad_reception
#define     HA_AD_RECEPTION         is_ha_ad_reception
#define     FA_AD_RECEPTION         is_fa_ad_reception
#define     SOLICITATION_TIME       is_solicitation_time
#define     VALID_FA_CANDIDATE      is_valid_fa_candidate
#define     HA_TIMEOUT              is_ha_timeout
#define     FA_TIMEOUT              is_fa_timeout
#define     TIMEOUT                 is_timeout
#define     RETRY                   is_retry
#define     FA_REG_SUCCESS          is_fa_reg_success //Me
#define     HA_REG_SUCCESS          is_ha_reg_success //Me
#define     OUT_OF_RETRIES          is_out_of_retries
#define     INVALID_REPLY           is_invalid_reply
#define     REREGISTER              is_reregister
#define     SWITCH_FA               is_switch_fa

#define     TIMER                   is_timer //Me
#define     SOLICIT                 is_solicit //Me

#define     MipC_MN_Rereg_Buffer    1.0

#define     MipC_MN_Solicit_Max_Interval  60.0
#define     MipC_MN_Solicit_Min_Interval  1.0
```

```
#define     IP_DEFAULT_TTL                32
#define     IPC_PIM_SM_RPF_TIMER_OFFSET   50
#define     IPC_PIM_SM_NOT_BSR_CAND       0
#define     IPC_PIM_SM_TOS                0
#define     IPC_PIM_SM_DATA_RATE_TIMER    1
#define     IPC_PIM_SM_START              2
#define     IPC_PIM_SM_RPF_UPDATE         3
#define     IPC_PIM_SM_SEND_JOIN_PRUNE_MSG      4
#define     IPC_PIM_SM_DR_TIMEOUT_OFFSET        1000
#define     IPC_PIM_SM_SEND_HELLO_MSG_OFFSET    2000

/****** Transition Macros ******/
#define     HELLO_MSG               (transition_code ==
IpC_Pim_Sm_Hello_Msg_Recvd)
#define JOIN_PRUNE_MSG              (transition_code ==
IpC_Pim_Sm_Join_Prune_Msg_Recvd)
#define DATA_PKT                    (transition_code ==
IpC_Pim_Sm_Data_Pkt_Recvd)
#define RTE_PLUS                    (transition_code == IpC_Pim_Sm_Rte_Plus)
#define RTE_MINUS                   (transition_code == IpC_Pim_Sm_Rte_Minus)
#define REGISTER_MSG                (transition_code ==
IpC_Pim_Sm_Register_Msg_Recvd)
#define REGISTER_STOP_MSG           (transition_code ==
IpC_Pim_Sm_Register_Stop_Msg_Recvd)
#define DATA_RATE_TIMER_EXPD  (transition_code ==
IpC_Pim_Sm_Data_Rate_Timer_Expd)
#define SEND_HELLO_MSG              (transition_code ==
IpC_Pim_Sm_Send_Hello_Msg)
#define SEND_JOIN_PRUNE_MSG         (transition_code ==
IpC_Pim_Sm_Send_Join_Prune_Msg)
#define DR_TIMEOUT                  (transition_code ==
IpC_Pim_Sm_Dr_Timeout)
#define RPF_UPDATE                  (transition_code ==
IpC_Pim_Sm_RPF_Update)
#define FAILURE_RECOVERY            (transition_code ==
IpC_Pim_Sm_Failure_Recovery)

#define START_INTERRUPT             (invmode == OPC_PROINV_DIRECT)

#define DELAY_TIMER_EXPD            (transition_code ==
IpC_Igmp_Host_Delay_Timer_Expd)
#define REPORT_RECVD                (transition_code ==
IpC_Igmp_Host_Report_Recvd)
#define QUERY_RECVD                 (transition_code ==
IpC_Igmp_Host_Query_Recvd)
#define LEAVE_GROUP                 (transition_code ==
IpC_Igmp_Host_Leave_Grp)
#define JOIN_GROUP                  (transition_code ==
IpC_Igmp_Host_Join_Grp)

#define ICMP_IP_PROCESS_INVOKE      -1
#define ECHO_REQUEST_GEN            (intrpt_type == OPC_INTRPT_SELF) &&
(pkt_from_ip == OPC_FALSE)
#define ECHO_REQUEST_RCVD     (icmp_message_type == IpC_Icmp_Echo_Request)
#define ECHO_REPLY_RCVD     (icmp_message_type == IpC_Icmp_Echo_Reply)
#define IP_ICMP_DEST_ADDR_UNSPECIFIED     "0.0.0.0"
```

```
#define IPC_ICMP_ECHO_PKSIZE_BITS         64

/****** Macro for Traces *******/
#define LTRACE_PIM_SM               (op_prg_odb_ltrace_active ("pim-sm") ||
op_prg_odb_trace_active ())
#define LTRACE_PIM_SM_ALL_BUT_DATA  (op_prg_odb_ltrace_active ("pim-
sm_all_but_data"))
#define LTRACE_PIM_SM_JOIN_PRUNE    (LTRACE_PIM_SM ||
LTRACE_PIM_SM_ALL_BUT_DATA || op_prg_odb_ltrace_active ("pim-sm_join_prune"))
#define LTRACE_PIM_SM_HELLO         (LTRACE_PIM_SM ||
LTRACE_PIM_SM_ALL_BUT_DATA || op_prg_odb_ltrace_active ("pim-sm_hello"))
#define LTRACE_PIM_SM_TIMERS        (LTRACE_PIM_SM ||
LTRACE_PIM_SM_ALL_BUT_DATA || op_prg_odb_ltrace_active ("pim-sm_timers"))
#define LTRACE_PIM_SM_DATA          (LTRACE_PIM_SM ||
op_prg_odb_ltrace_active ("pim-sm_data"))
#define LTRACE_PIM_SM_FAIL_RECOVER  (LTRACE_PIM_SM ||
op_prg_odb_ltrace_active ("pim-sm_fail_recover"))
#define LTRACE_IGMP           (op_prg_odb_ltrace_active ("igmp") ||
op_prg_odb_trace_active ())

static Boolean               log_call_scheduled = OPC_FALSE;
static int                   ip_pim_sm_efficiency_mode = -1;

/* Global RP lists.     */
List*       bootstrap_rp_lptr = OPC_NIL;
List*       auto_rp_lptr = OPC_NIL;
Boolean     boostrap_support = OPC_FALSE;
Boolean     auto_rp_agent_found = OPC_FALSE;
static Log_Handle       ip_igmp_host_config_warn_loghndl;
static Log_Handle       ip_igmp_host_lowlevel_error_loghndl;
static Boolean          ip_igmp_host_loghndls_init = OPC_FALSE;

typedef struct
      {
      int         interval;
      int         retry;
      int         req_lifetime;
      }
MipT_MN_Reg_Info;

typedef enum
      {
      MipC_MN_Timer_Rereg,
      MipC_MN_Timer_Agent,
      MipC_MN_Timer_Solicit,
      MipC_MN_Timer_Retry
      }
MipC_MN_Timer;

typedef enum IpT_Icmp_Echo_Message_Type
      {
      IpC_Icmp_Unspec = -1,
      IpC_Icmp_Echo_Reply = 0,
      IpC_Icmp_Echo_Request = 8
      } IpT_Icmp_Echo_Message_Type;
```

```
typedef struct
      {
      Stathandle        pkts_sent_stathandle;
      Stathandle        pkts_rcvd_stathandle;
      Stathandle        resp_time_stathandle;
      } IpT_Icmp_Stats;

typedef struct
      {
      IpT_Address       ip_grp_addr;
      int               interface;
      Evhandle          delay_timer_evh;
      int               delay_timer_id;
      Boolean           timer_on_flag;
      int               unsolicit_msg_count;
      Boolean           report_sent_flag;
      } IpT_Igmp_Host_Grp_Elem;

static Log_Handle       ip_igmp_host_config_warn_loghndl;
static Log_Handle       ip_igmp_host_lowlevel_error_loghndl;
static Boolean          ip_igmp_host_loghndls_init = OPC_FALSE;

typedef struct
      {
      InetT_Address     address;
      double            lifetime;
      int               pref_level;
      IpT_Interface_Info *incoming_intf_ptr;
      }
MipT_MN_Agent_Info;

typedef struct {

      Evhandle    expire_time;
      Objid       DUID;
      Objid       IAID;
      unsigned char     assign_type;

      InetT_Address_Range     assignment;             /* Holds the v4 addr or
v6 addr/prefix */
      InetT_Address                 link_local_addr;  /* The address used to
unicast back to the client */
} DhcpT_Srv_Assignment;

/* Function declarations. */
static void mip_mn_register (int, MipT_MN_Agent_Info, Boolean);
static void mip_mn_agent_timer_update (double);
static void mip_mn_tunneled_pk_stat_write (Packet*);
static void mip_mn_agent_solicit_pk_send (void);
static void mip_mn_agent_cache_update (MipT_MN_Agent_Info*, InetT_Address,
double, int, MipT_Invocation_Info*);
static void mip_mn_ip_pk_handle (MipT_Invocation_Info*);
static void mip_mn_ad_packet_parse (Packet*, int*, int*, InetT_Address*,
int*, int*);

void  dhcp_parse_msg();
int   dhcp_msg_server_duid_match(PrgT_List* rcv_pkt_opts);
```

```
int   dhcp_get_free_addr(int iface_index);
int   dhcp_get_free_prefix(int iface_index);
int   dhcp_get_iface_info_from_index(int index);

static void mip_mn_agent_solicit_pk_send_adv (void);//me

static void             ip_icmp_sv_init (void);
static void             ip_icmp_ping_specs_parse (IpT_Icmp_Temp_Ping_Specs*
ip_temp_ping_specs_ptr);
static void             ip_icmp_initial_echo_requests_schedule (void);
static Packet*          ip_icmp_echo_request_packet_create (int
req_index);//Me
static Packet*          ip_icmp_pkt_encapsulate (Packet* icmp_req_pkptr, int
req_index);
static void             ip_icmp_echo_reply_create (Packet* ip_dgram_pkptr,
Packet* ip_icmp_pkptr, Packet* icmp_reply_pkptr);
static void             ip_icmp_ip_process_invoke (Packet* ip_dgram_pkptr);
EXTERN_C_BEGIN
static void             ip_icmp_ip_process_invoker (void* state_ptr, int
code);//Me
EXTERN_C_END
static void             ip_icmp_ip_process_invoke_schedule (Packet*
ip_dgram_pkptr);
static double           ip_icmp_reply_stats_update (Packet* icmp_reply_pkptr,
int req_index);
static void             ip_icmp_next_echo_request_schedule (int req_index);
static void             ip_icmp_ping_stats_register (int stat_index, char*
dest_host_name);
static void             ip_icmp_sim_log_init ();
static void             ip_icmp_ip_dgram_discard (Packet* ip_dgram_pkptr);
EXTERN_C_BEGIN
static void             ip_icmp_request_timeout (void *state_ptr, int
index);//Me
EXTERN_C_END
static IpT_Icmp_Temp_Ping_Specs *
                        ip_icmp_ping_traffic_list_generate (Objid
node_objid);
static void             ip_icmp_dgram_fdstruct_update_for_reply
(IpT_Dgram_Fields* ip_dgram_fd_ptr);

static void             ip_pim_sm_do_init (void);
static void             ip_pim_sm_rte_plus (void);
static void             ip_pim_sm_data_pkt (void);
static void             ip_pim_sm_join_prune_msg (void);
static void             ip_pim_sm_data_rate_timer_expired (void);
static void             ip_pim_sm_register_msg (void);
static void             ip_pim_sm_register_stop_msg (void);
static void             ip_pim_sm_hello_msg (void);
static void             ip_pim_sm_rte_minus (void);
static void             ip_pim_sm_rpf_update (void);
static void             ip_pim_sm_fail_recover (void);

/****** Procedures *************/
static void ip_igmp_host_sv_init (void);
static IpT_Igmp_Host_Transition ip_igmp_host_get_transition_code (Packet**
igmp_pkt_pptr, Boolean* is_gs_query_msg_ptr);
```

```
static IpT_Igmp_Host_Grp_Elem*      ip_igmp_host_get_grp_elem (IpT_Address
ip_grp_addr, int interface);
static void ip_igmp_host_join_grp (IpT_Address ip_grp_addr, int interface);
//Me: Joinning multicast group by using CoA address.
static void ip_igmp_host_leave_grp (IpT_Address ip_grp_addr, int interface);
static void       ip_igmp_host_start_timer_for_grp (IpT_Igmp_Host_Grp_Elem*
grp_elem_ptr, double max_resp_time);
static void       ip_igmp_host_start_timer_for_grps_on_intf (int interface,
double max_resp_time);
static void       ip_igmp_host_cancel_timer_for_grp (IpT_Igmp_Host_Grp_Elem*
grp_elem_ptr);
static void       ip_igmp_host_timer_expired (int timer_code);
static IpT_Igmp_Host_Grp_Elem*      ip_igmp_host_grp_elem_alloc (void);
static void       ip_igmp_host_grp_elem_dealloc (IpT_Igmp_Host_Grp_Elem*
grp_elem_ptr);
static void       ip_igmp_host_error (const char* msg1, const char* msg2,
const char* msg3);
static void       ip_igmp_host_log_handles_init (void);
static void       ip_igmp_host_log_found_no_grp_info (const char*
ip_addr_str, int ip_intf_num);
static void       ip_igmp_host_grp_info_print (List* grp_lptr, Boolean
short_version);
static void       ip_igmp_host_igmp_msgs_sent_stat_update (OpT_Packet_Size
size); //Me

EXTERN_C_BEGIN
static void
      ip_igmp_host_ip_process_invoke (void *state_ptr, int
interface_to_send);
EXTERN_C_END

/* End of Header Block */

#if !defined (VOSD_NO_FIN)
#undef      BIN
#undef      BOUT
#define     BIN         FIN_LOCAL_FIELD(_op_last_line_passed) = __LINE__ -
_op_block_origin;
#define     BOUT  BIN
#define     BINIT FIN_LOCAL_FIELD(_op_last_line_passed) = 0; _op_block_origin
= __LINE__;
#else
#define     BINIT
#endif /* #if !defined (VOSD_NO_FIN) */

/* State variable definitions */
typedef struct
      {
      /* Internal state tracking for FSM */
      FSM_SYS_STATE
      /* State Variables */
      MipC_Node_Type                      mip_node_type
;     /* Type of MN I am (MN or MR) */
      MipT_Proc_Info*                     proc_info_struct_ptr
;     /* General info shared between parent and me */
      MipT_MN_Reg_Info                    reg_info
;     /* registration information for this MN or MR */
```

```
      InetT_Address                         subnet_bcast_addr
;     /* brodcast address of the mobile interface */
      InetT_Address                         ha_address
;     /* IP address of the home agent interface */
      InetT_Address                         home_address
;     /* local interface ip address */
      double                                time_to_reregister
;     /* sim time for next registration */
      Evhandle                              reregister_timer_ehndl
;     /* eventhandle for reregistration */
      int                                   reg_id
;     /* identification for pending registration */
      int                                   retry_counter
;     /* current number of registration retries */
      InetT_Address                         agent_address
;     /* Current agent address serving me */
      Boolean                               direct_reg
;     /* direct with HA or indirect through FA */
      MipT_MN_Agent_Info                    latest_fa_info
;     /* Last FA information on FA other than the currnt one */
      Evhandle                              agent_timer_ehndl
;     /* event handle for agent timeouts */
      MipT_MN_Agent_Info                    latest_ha_info
;     /* Latest info on the HA from advertisements */
      IpT_Rte_Module_Data*                  module_data
;     /* node wide IP module info */
      Stathandle                            tunneled_pk_rcvd_sec_sh
;
      Stathandle        tunneled_bit_rcvd_sec_sh                        ;
      Evhandle          solicit_timer_ehndl                             ;
      /* Event handle for sending solicitation packet. */
      Boolean           solicitation                                    ;
      /* Whether or not the MN/MR solicit when lost */
      int               solicit_count                                   ;
      /* number of times solicitation was sent out before getting an ad */
      Stathandle        irdp_sent_pkts_sh                               ;
      Stathandle        irdp_sent_bits_sh                               ;
      Objid             node_objid                                      ;
      Boolean           simultaneous_binding                            ;
      /* Flag indicating if this MN/MR will be asking for HA to keep */

      /* simulatneous binding.                                        */
      Evhandle          reg_retry_timer_ehndl                           ;
      /* Event handle for registration retry in case of failure. */
      Stathandle        g_irdp_sent_bits_sh                             ;
      Boolean           loopback_intf                                   ;
      /* flag indicating if this MN/MR is configured on the loopback
interface */
      int               current_agent_pref_level                        ;
      /* preference level of the current FA */
      IpT_Interface_Info*     current_roaming_intf                      ;
      Boolean           default_gateway                                 ;
      /* Flag to indicate, have a default gateway setup */
      IpT_Address       last_default_addr                               ;
      } mobile_ip_mn_state;

typedef struct
```

```
      {
      /* Internal state tracking for FSM */
      FSM_SYS_STATE
      /* State Variables */
      Objid                  module_objid                                    ;
      Objid                  node_objid                                      ;
      OmsT_Pr_Handle         my_proc_handle                                  ;
      Objid                  udp_objid                                       ;
      IpT_Rte_Module_Data*     ip_support_module_ptr                         ;
      Ici*                   command_ici_ptr                                 ;
      int                    num_dhcp_interfaces                             ;
      /* Number of interfaces DHCP will be listening on. */
      DhcpT_Srv_Interface*     interfaces_config                             ;
      int                      input_strm                                    ;
      int                      output_strm                                   ;
      InetT_Address           server_multicast_addr                    ;  // Me
      DhcpT_Stathandles*       global_stats                                  ;
      DhcpT_Stathandles*       local_stats                                  ;
      int                      intrpt_code                                   ;
      Boolean                  logging                                      ;
      /* Boolean to indicate if global DHCP logging is turned on/off */
      Log_Handle              new_log_handle                                 ;
      Log_Handle              renew_log_handle                               ;
      Log_Handle              error_log_handle                               ;
      Log_Handle              expire_log_handle                              ;
      } dhcp_server_state;

static enum
      {IpC_Pim_Grp_Tbl_Grp_Addr = 0, IpC_Pim_Grp_Tbl_RP_Addr,
      IpC_Pim_Grp_Tbl_Src_Addr, IpC_Pim_Grp_Type,
      IpC_Pim_Grp_Tbl_In_Iface, IpC_Pim_Grp_Tbl_Out_Iface,
      IpC_Pim_Grp_Tbl_Num_Columns
      } IpC_Pim_Grp_Tbl_Table_Column_Index;

typedef struct
      {
      OpT_Int8          pim_sm_status;
      double            hello_period;
      double            hello_holdtime;
      IpT_Pim_Intf_Stat_Handle*     stat_handle_ptr;
      Evhandle                      hello_evhandle;
      int   priority;
      } IpT_Pim_Intf;

#define mip_node_type                    op_sv_ptr->mip_node_type
#define proc_info_struct_ptr             op_sv_ptr->proc_info_struct_ptr
#define reg_info                         op_sv_ptr->reg_info
#define subnet_bcast_addr                op_sv_ptr->subnet_bcast_addr
#define ha_address                       op_sv_ptr->ha_address
#define home_address                     op_sv_ptr->home_address
#define time_to_reregister               op_sv_ptr->time_to_reregister //Me
#define reregister_timer_ehndl           op_sv_ptr->reregister_timer_ehndl
#define reg_id                           op_sv_ptr->reg_id
#define retry_counter                    op_sv_ptr->retry_counter
#define agent_address                    op_sv_ptr->agent_address
#define direct_reg                       op_sv_ptr->direct_reg
#define latest_fa_info                   op_sv_ptr->latest_fa_info
```

```
#define agent_timer_ehndl               op_sv_ptr->agent_timer_ehndl
#define latest_ha_info                  op_sv_ptr->latest_ha_info
#define module_data                     op_sv_ptr->module_data
#define tunneled_pk_rcvd_sec_sh         op_sv_ptr->tunneled_pk_rcvd_sec_sh
#define tunneled_bit_rcvd_sec_sh        op_sv_ptr->tunneled_bit_rcvd_sec_sh
#define solicit_timer_ehndl             op_sv_ptr->solicit_timer_ehndl
#define solicitation                    op_sv_ptr->solicitation
#define solicit_count                   op_sv_ptr->solicit_count
#define irdp_sent_pkts_sh               op_sv_ptr->irdp_sent_pkts_sh
#define irdp_sent_bits_sh               op_sv_ptr->irdp_sent_bits_sh
#define node_objid                      op_sv_ptr->node_objid
#define simultaneous_binding            op_sv_ptr->simultaneous_binding
#define reg_retry_timer_ehndl           op_sv_ptr->reg_retry_timer_ehndl
//Me
#define g_irdp_sent_bits_sh             op_sv_ptr->g_irdp_sent_bits_sh
#define loopback_intf                   op_sv_ptr->loopback_intf
#define current_agent_pref_level        op_sv_ptr->current_agent_pref_level
#define current_roaming_intf            op_sv_ptr->current_roaming_intf
#define default_gateway                 op_sv_ptr->default_gateway
#define last_default_addr               op_sv_ptr->last_default_addr

#define module_objid                    op_sv_ptr->module_objid
#define node_objid                      op_sv_ptr->node_objid
#define my_proc_handle                  op_sv_ptr->my_proc_handle
#define udp_objid                       op_sv_ptr->udp_objid
#define ip_support_module_ptr           op_sv_ptr->ip_support_module_ptr
#define command_ici_ptr                 op_sv_ptr->command_ici_ptr
#define max_sol_tmout                   op_sv_ptr->max_sol_tmout
#define max_req_tmout                   op_sv_ptr->max_req_tmout
#define max_req_retries                 op_sv_ptr->max_req_retries
#define init_renew_tmout                op_sv_ptr->init_renew_tmout
#define max_renew_tmout                 op_sv_ptr->max_renew_tmout
#define init_rebind_tmout               op_sv_ptr->init_rebind_tmout
#define max_rebind_tmout                op_sv_ptr->max_rebind_tmout
#define send_iface                      op_sv_ptr->send_iface
#define last_trans_id                   op_sv_ptr->last_trans_id
#define rapid_commit                    op_sv_ptr->rapid_commit
#define server_rapid_commit             op_sv_ptr->server_rapid_commit
#define server_id                       op_sv_ptr->server_id
#define srv_str                         op_sv_ptr->srv_str
#define interfaces_config               op_sv_ptr->interfaces_config
#define gateway_node                    op_sv_ptr->gateway_node
#define input_strm                      op_sv_ptr->input_strm
#define output_strm                     op_sv_ptr->output_strm
#define snd_pkt_ptr                     op_sv_ptr->snd_pkt_ptr // Me
#define snd_pkt_opts                    op_sv_ptr->snd_pkt_opts // Me
#define snd_msg_type                    op_sv_ptr->snd_msg_type // Me
#define rcv_pkt_ptr                     op_sv_ptr->rcv_pkt_ptr  // Me
#define rcv_pkt_opts                    op_sv_ptr->rcv_pkt_opts // Me
#define rcv_msg_type                    op_sv_ptr->rcv_msg_type // Me
#define rcv_msg_trans                   op_sv_ptr->rcv_msg_trans // Me
#define RTprev                          op_sv_ptr->RTprev
#define expire_time                     op_sv_ptr->expire_time
#define rebind_time                     op_sv_ptr->rebind_time
#define retrans_count                   op_sv_ptr->retrans_count
#define intrpt_type                     op_sv_ptr->intrpt_type
#define intrpt_code                     op_sv_ptr->intrpt_code
```

```
#define num_interfaces                  op_sv_ptr->num_interfaces
#define next_evh                        op_sv_ptr->next_evh
#define server_multicast_addr           op_sv_ptr->server_multicast_addr
//Me
#define node_ll_addr                    op_sv_ptr->node_ll_addr
#define global_stats                    op_sv_ptr->global_stats
#define local_stats                     op_sv_ptr->local_stats
#define transaction_time                op_sv_ptr->transaction_time
#define logging                         op_sv_ptr->logging
#define new_log_handle                  op_sv_ptr->new_log_handle
#define renew_log_handle                op_sv_ptr->renew_log_handle
#define error_log_handle                op_sv_ptr->error_log_handle
#define expire_log_handle               op_sv_ptr->expire_log_handle
#define server_inet_addr                op_sv_ptr->server_inet_addr


/* These macro definitions will define a local variable called    */
/* "op_sv_ptr" in each function containing a FIN statement. */
/* This variable points to the state variable data structure,     */
/* and can be used from a C debugger to display their values.     */
#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE
#define FIN_PREAMBLE_DEC      mobile_ip_mn_state *op_sv_ptr;
#define FIN_PREAMBLE_CODE     \
            op_sv_ptr = ((mobile_ip_mn_state *)(OP_SIM_CONTEXT_PTR-
>_op_mod_state_ptr));

#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE
#define FIN_PREAMBLE_DEC      dhcp_client_state *op_sv_ptr;
#define FIN_PREAMBLE_CODE     \
            op_sv_ptr = ((dhcp_client_state *)(OP_SIM_CONTEXT_PTR-
>_op_mod_state_ptr));

/* Function Block */

#if !defined (VOSD_NO_FIN)
enum { _op_block_origin = __LINE__ + 2};
#endif

/* Transitional Executives */
void SendMsg()
      {
      /* Build a DHCP message with
       * appropriate options as determined by the nodes
       * configuration, and send the packet to the well known
       * DHCP multicast address.
       */
      DhcpT_Opt*              tmp_dhcp_opt;
      DhcpT_Cli_Assignment*   iface_config;
      double                  retrans_time;
      int                     int_code;
      PrgT_List_Cell*         list_cell_ptr;
      int                     oro_length        = 0;
      Boolean                 retrans_msg       = OPC_FALSE;
      char                    log_str[1000];
```

```
      FIN(SendMsg());

      /* Double check if a result of receiving a message from potentially
      many servers, if we've already sent a request to a particular server,
      then ignore this recent message if it's not from the server we sent the
      request to. */
      if (intrpt_type == OPC_INTRPT_STRM)
            {
            tmp_dhcp_opt = dhcp_optlist_get_opt(DHCPC_OPT_SERVERID,
rcv_pkt_opts);
            if(tmp_dhcp_opt->simple_data != server_id)
                  {
                  Discard();
                  FOUT;
                  }
            }

      /* If retransmitting, determine if this is a normal retransmission,
       * or if we are switching from a Renew phase to a Rebind phase.  If
       * it is a normal retransmission, we only increment the retransmission
       * count.
       */
      if ((intrpt_type == OPC_INTRPT_SELF) && (intrpt_code ==
DHCPC_MSG_TIMEOUT))
            {
            if((rebind_time == 0) || ((rebind_time > 0) && (op_sim_time() <
rebind_time)))
                  {
                  retrans_count++;
                  retrans_msg = OPC_TRUE;

                  if(LTRACE_ACTIVE)
                        op_prg_odb_print_minor ("Client timed out waiting for
response. Retransmitting..." , OPC_NIL);
                  }
            else
                  {
                  snd_msg_type = DHCPC_MSG_REBIND;

                  sprintf(log_str, "Current server %s not responding to DHCP
messages. "
                        "Sending Rebind message instead of Renew message to
obtain service "
                        "from any available server.", srv_str);
                  LOG(renew_log_handle, log_str, PrgC_Log_Severity_Warning);
                  }
            }

      /* If we received an Advertise, increment the transaction ID and send a
Request: */
      if ((intrpt_type == OPC_INTRPT_STRM) && (rcv_msg_type ==
DHCPC_MSG_ADVERTISE))
            {
            snd_msg_type = DHCPC_MSG_REQUEST;
            last_trans_id++;
            }
```

```
      /* If our assignment is close to expiring, increment the transaction ID
and send a Renew: */
      if ((intrpt_type == OPC_INTRPT_SELF) && (intrpt_code ==
DHCPC_ASSIGN_TIMEOUT))
            {
            snd_msg_type = DHCPC_MSG_RENEW;
            last_trans_id++;

            sprintf(log_str, "Current assignment(s) getting stale. Initiating
Renew "
                  "message exchange with current server %s", srv_str);
            LOG(renew_log_handle, log_str, PrgC_Log_Severity_Information);
            }

      /* Now, determine the interval for the next retransmission attempt
        Set the time :: Me*/

      if((snd_msg_type == DHCPC_MSG_SOLICIT) || (snd_msg_type ==
DHCPC_MSG_SOLICIT_RAPID))
            {

            RTprev = dhcp_get_retrans_time(RTprev, DHCPC_TRP_SOL_TIMEOUT,
max_sol_tmout);
            retrans_time = op_sim_time() + RTprev;

            int_code = DHCPC_MSG_TIMEOUT;
            }

      if(snd_msg_type == DHCPC_MSG_REQUEST)
            {
            if (retrans_count < max_req_retries)
                  {
                  RTprev = dhcp_get_retrans_time(RTprev,
DHCPC_TRP_REQ_TIMEOUT, max_req_tmout);
                  retrans_time = op_sim_time() + RTprev;
                  int_code = DHCPC_MSG_TIMEOUT;
                  }
            else
                  {
                  /* Immediately fallback to Solicit message by going back to
"Begin" FSM state */
                  op_intrpt_schedule_self(op_sim_time(), DHCPC_MSG_FALLBACK);
                  FOUT;
                  }
            }

      if(snd_msg_type == DHCPC_MSG_RENEW)
            {
            RTprev = dhcp_get_retrans_time(RTprev, init_renew_tmout,
max_renew_tmout);
            if((op_sim_time() + RTprev) > rebind_time)
                  {
                  /* Another scheduled retransmission would exceeded the
Renew phase, so we must switch to the Rebind phase at the rebind_time */
                  retrans_time = rebind_time;
                  }
```

```
            else
                  retrans_time = op_sim_time() + RTprev;

            int_code = DHCPC_MSG_TIMEOUT;
            }

      if(snd_msg_type == DHCPC_MSG_REBIND)
            {
            RTprev = dhcp_get_retrans_time(RTprev, init_rebind_tmout,
max_rebind_tmout);
            if((op_sim_time() + RTprev) > expire_time)
                  {
                  /* Another scheduled retransmission would exceed the valid
lifetime of the assignment.  */
                  retrans_time = expire_time;
                  int_code = DHCPC_MSG_FALLBACK;
                  }
            else
                  {
                  retrans_time = op_sim_time() + RTprev;
                  int_code = DHCPC_MSG_TIMEOUT;
                  }
            }

      /* Schedule the next timer self-interrupt: */
      if (op_ev_pending(next_evh))
            op_ev_cancel(next_evh);
      next_evh = op_intrpt_schedule_self(retrans_time, int_code);

      /* Now that we know we'll be sending a message (and not falling back,
       * create the message):
       */
      snd_pkt_ptr = dhcp_msg_create(snd_msg_type, last_trans_id);
      snd_pkt_opts = dhcp_optlist_create();

      prg_list_insert(snd_pkt_opts,
                        dhcp_opt_create_id(DHCPC_OPT_CLIENTID, node_objid),
                              PRGC_LISTPOS_TAIL);

      /* If the message have reached a server, include its Server ID, as long
as this message is NOT a Rebind message: */
      if((server_id != 0) && (snd_msg_type != DHCPC_MSG_REBIND))
            prg_list_insert(snd_pkt_opts,
                                    dhcp_opt_create_id(DHCPC_OPT_SERVERID,
server_id),
                                    PRGC_LISTPOS_TAIL);
      /* If this message is a rapid commit Solicit, add the option: */
      if(snd_msg_type == DHCPC_MSG_SOLICIT_RAPID)
            prg_list_insert(snd_pkt_opts,
                                    dhcp_opt_create(DHCPC_OPT_RAPID_COMMIT),
                                    PRGC_LISTPOS_TAIL);

      /* Add an option for every interface we are requesting configuration
info for: */
      if((snd_msg_type == DHCPC_MSG_REQUEST) || (snd_msg_type ==
DHCPC_MSG_RENEW)
            || (snd_msg_type == DHCPC_MSG_REBIND)
```

```
            || (snd_msg_type == DHCPC_MSG_SOLICIT_RAPID))
            {
            for(list_cell_ptr = prg_list_head_cell_get(&interfaces_config);
                  list_cell_ptr != PRGC_NIL;
                  list_cell_ptr = prg_list_cell_next_get(list_cell_ptr))
                  {
                  iface_config = (DhcpT_Cli_Assignment
*)prg_list_cell_data_get(list_cell_ptr);

                  if(iface_config->assign_type == DHCPC_ASSIGN_ADDR)
                        /* We're adding an option to request an address: */
                        tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_IA_NA);
                  else
                        /* We're adding an option to request a prefix: */
                        tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_IA_PD);

                  tmp_dhcp_opt->simple_data = iface_config->iface_index;


                  if(iface_config->configured == OPC_FALSE)
                        tmp_dhcp_opt->length = 12;
                  else
                        {
                        if(iface_config->assign_type == DHCPC_ASSIGN_ADDR)
                              tmp_dhcp_opt->length =
DHCPC_OPT_IA_NA_DATA_LEN;
                        else
                              tmp_dhcp_opt->length =
DHCPC_OPT_IA_PD_DATA_LEN;
                        }
                  /* Add this option to the total length count of the Option
Request option: */
                  oro_length += 2;

                  prg_list_insert(snd_pkt_opts, tmp_dhcp_opt,
PRGC_LISTPOS_TAIL);
                  }

            tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_ORO);
            tmp_dhcp_opt->length = oro_length;
            prg_list_insert(snd_pkt_opts, tmp_dhcp_opt,
      PRGC_LISTPOS_TAIL);

            }

      /* Finalize the packet and send it to the UDP module for delivery.*/
      dhcp_msg_finalize(snd_pkt_ptr, snd_pkt_opts);
      op_ici_install(command_ici_ptr);
      op_pk_send(snd_pkt_ptr, output_strm);
      op_ici_install(OPC_NIL);

      /* Update the DHCP stats for the message type sent: */
      if(retrans_msg == OPC_TRUE)
            dhcp_update_cli_stat(DHCPC_MSG_RETRANSMIT, 1);
      else
            /* If this is not a retransmission, reset the transaction time:
*/
```

```
            transaction_time = op_sim_time();

      dhcp_update_cli_stat(snd_msg_type, 1);

      /* Free the received packet memory: */
      if(intrpt_type == OPC_INTRPT_STRM)
            {
            op_pk_destroy(rcv_pkt_ptr);
            dhcp_optlist_destroy(rcv_pkt_opts);
            }

      FOUT;
      }

void Config()
      {
      /* Me:Once a mobile node has received a Reply message from access point
        which includes address: CoA address, this function is called to
      configure the assignments on the nodes interfaces on mobile node.
       */
      DhcpT_Opt*                    dhcp_opt_ptr;
      InetT_Address_Range*    inet_range_ptr;
      DhcpT_Cli_Assignment*   assign_ptr;
      int                           timers_configured = OPC_FALSE;
      int                           new_addr          = 0;
      int                           new_prefix        = 0;
      int                           renew_addr        = 0;
      int                           renew_prefix      = 0;
      char        log_str[1000], tmp_str[100], val_str[100], val2_str[100];
      PrgT_List_Cell*               list_cell_opt_ptr;
      PrgT_List_Cell*               list_cell_assign_ptr;
      double                        trans_delay;

      PrgT_List                     *dyn_assignment_lptr = OPC_NIL;
      int                           dyn_assignment_count = 0;
      IpT_Dynamic_Assignment  *dyn_assignment_ptr;
      IpT_Dynamic_Assignment_Array  *dyn_array_ptr;
      IpT_Dynamic_Assignment_Type   dyn_assignment_type;
      int                                 default_route_action =
DHCPC_DEFAULT_ROUTE_UNCHANGED;

      Ici                           *rcvd_ici_ptr;
      InetT_Address                 *rcvd_inet_address;

      FIN(Config());

      /* Me: Calculate the transaction delay for this message exchange:- some
      factor of assigning CoA address delay */
      trans_delay = op_sim_time() - transaction_time;

      /* Me: Get the server address from access point:- the address of
      foreign agent */
      rcvd_ici_ptr = op_ev_ici (op_ev_current ());
      op_ici_attr_get (rcvd_ici_ptr, "rem_addr", &rcvd_inet_address);

      /* Cycle through the options to find the IA_NA and IA_PD options: */
      for(list_cell_opt_ptr = prg_list_head_cell_get(rcv_pkt_opts);
```

```
            list_cell_opt_ptr != PRGC_NIL;
            list_cell_opt_ptr = prg_list_cell_next_get(list_cell_opt_ptr))
            {
            dyn_assignment_type = IpC_Dynamic_Assignment_None;

            dhcp_opt_ptr = (DhcpT_Opt
*)prg_list_cell_data_get(list_cell_opt_ptr);

            if((dhcp_opt_ptr->code != DHCPC_OPT_IA_NA) && (dhcp_opt_ptr->code
!= DHCPC_OPT_IA_PD))
                  continue;

            if(dhcp_opt_ptr->simple_data2 == DHCPC_NO_ADDR_AVAIL)
                  {
                  /* If this option indicated no address available, skip it
to go to the next option: */
                  sprintf(log_str, "Option received from server %s indicated
no address available.", srv_str);
                  LOG(error_log_handle, log_str, PrgC_Log_Severity_Notice);
                  continue;
                  }
            else if(dhcp_opt_ptr->simple_data2 == DHCPC_NO_PREFIX_AVAIL)
                  {
                  /* If this option indicated no prefix available, skip it to
go to the next option: */
                  sprintf(log_str, "Option received from server %s indicated
no prefix available.", srv_str);
                  LOG(error_log_handle, log_str, PrgC_Log_Severity_Notice);
                  continue;
                  }

            /* Me: In this point, the mobile node received the valid address
in the complex_data  */
            inet_range_ptr = (InetT_Address_Range *)(dhcp_opt_ptr-
>complex_data);

            /* Me: Store all info for this particular assignment, have to
check interface by cycling through the array of interfaces and comparing the
interface index with the value we received as the IAID.*/
            for(list_cell_assign_ptr =
prg_list_head_cell_get(&interfaces_config);
                  list_cell_assign_ptr != PRGC_NIL;
                  list_cell_assign_ptr =
prg_list_cell_next_get(list_cell_assign_ptr))
                  {
                  assign_ptr = (DhcpT_Cli_Assignment
*)prg_list_cell_data_get(list_cell_assign_ptr);
                  if(assign_ptr->iface_index == dhcp_opt_ptr->simple_data)
                        break;
                  }
            if(list_cell_assign_ptr == PRGC_NIL)
                  op_sim_end("Error while attempting to configure client:",
                                    "Unable to determine interface for
assignment.", OPC_NIL, OPC_NIL);
```

```
            /* Me: If mobile node already have an assignment on this
interface, determine if it's a renewal,or a different assignment (potentially
from a different server or access point). */
            if(assign_ptr->configured == OPC_TRUE)
                  {
                  if(inet_address_range_equal(&(assign_ptr->assignment),
inet_range_ptr) != OPC_TRUE)
                        {
                        /* Me: Set the IP notification to update */
                        dyn_assignment_type =
IpC_Dynamic_Assignment_Addr_Update;

                        /* Me: Just for logging: */
                        inet_address_range_print(val_str, &(assign_ptr-
>assignment));
                        inet_address_range_print(val2_str, inet_range_ptr);
                        if(assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                              sprintf(tmp_str, "address");
                        else
                              sprintf(tmp_str, "prefix");
                        sprintf(log_str, "Received new %s assignment for %s:
%s\n Old assignment: %s\n "
                              "New server: %s\n Lifetime: %d\n Transaction
time: %f sec",
                              tmp_str, assign_ptr->iface_name, val2_str,
val_str, srv_str,
                              dhcp_opt_ptr->simple_data2,   trans_delay);
                        LOG(new_log_handle, log_str,
PrgC_Log_Severity_Notice);

                        /* This is a different assignment from what we
already have destroy the existing assignment before installing the new one:*/
                        inet_address_range_destroy(&(assign_ptr-
>assignment));
                        assign_ptr->assignment = inet_address_range_ptr_copy(

      (InetT_Address_Range *)dhcp_opt_ptr->complex_data);

                        /* Me: Increment the count of configured
addresses/prefixes CoAs: */
                        if(dhcp_opt_ptr->code == DHCPC_OPT_IA_NA)
                              new_addr++;
                        else if(dhcp_opt_ptr->code == DHCPC_OPT_IA_PD)
                              {
                              new_prefix++;

                              /* Also inform IP that the default route will
need to be updated */
                              default_route_action =
DHCPC_DEFAULT_ROUTE_UPDATE;
                              }

                        }
                  else
                        {
                  /* This is a successful renew of an existing assignment. */
```

```
                        /* No IP notification needed */

                        /* For logging: */
                        inet_address_range_print(val_str, &(assign_ptr-
>assignment));
                        if(assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                              sprintf(tmp_str, "address");
                        else
                              sprintf(tmp_str, "prefix");
                        sprintf(log_str, "Renewing existing %s assignment for
%s: %s\n "
                              "From server: %s\n Lifetime: %d\n Transaction
time: %f sec",
                              tmp_str, assign_ptr->iface_name, val_str,
srv_str, dhcp_opt_ptr->simple_data2, trans_delay);
                        LOG(renew_log_handle, log_str,
PrgC_Log_Severity_Information);

                        /* Write appropriate stats: */
                        if(dhcp_opt_ptr->code == DHCPC_OPT_IA_NA)
                              renew_addr++;
                        else if(dhcp_opt_ptr->code == DHCPC_OPT_IA_PD)
                              renew_prefix++;

                        }
                  }

            /* If this interface has not yet been configured, then copy the
option data containing the assignment into our local assignment array.
             */
            if(assign_ptr->configured != OPC_TRUE)
                  {
                  /* Set IP notification to create a new assignment */
                  if (assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                        dyn_assignment_type =
IpC_Dynamic_Assignment_Addr_Create;
                  else
                        dyn_assignment_type =
IpC_Dynamic_Assignment_Prefix_Create;

                  assign_ptr->assignment = inet_address_range_ptr_copy(

      (InetT_Address_Range *)dhcp_opt_ptr->complex_data);

                  /* For logging: */
                  inet_address_range_print(val_str, &(assign_ptr-
>assignment));
                  if(assign_ptr->assign_type == DHCPC_ASSIGN_ADDR)
                        sprintf(tmp_str, "address");
                  else
                        sprintf(tmp_str, "prefix");

            sprintf(log_str, "Obtained initial %s assignment for %s: %s\n "
            "From server: %s\n Lifetime: %d\n Transaction time: %f sec",
            tmp_str, assign_ptr->iface_name, val_str, srv_str, dhcp_opt_ptr-
>simple_data2, trans_delay);
            LOG(new_log_handle, log_str, PrgC_Log_Severity_Information);
```

```
                  /* Write appropriate stats: */
                  if(dhcp_opt_ptr->code == DHCPC_OPT_IA_NA)
                        new_addr++;
                  else if(dhcp_opt_ptr->code == DHCPC_OPT_IA_PD)
                        {
                        new_prefix++;

                        /* Inform IP to set up a new default route */
                        default_route_action = DHCPC_DEFAULT_ROUTE_ADD;
                        }
                  }

            /* This interface is now configured: */
            assign_ptr->configured = OPC_TRUE;

            /* Me : Set DHCP timers, the timers are only set once per
message. The protocol allows for each individual assignment to have it's
             *  own timers/expiration times. */
            if(timers_configured == OPC_FALSE)
                  {
                  /* Cancel any pending events: */
                  if(op_ev_pending(next_evh)) { op_ev_cancel(next_evh); }

                  /* Store times needed to properly implement protocol
timers: */
                  expire_time = (op_sim_time()) + dhcp_opt_ptr->simple_data2;
                  rebind_time = (op_sim_time()) + (.8 * dhcp_opt_ptr-
>simple_data2);

                  /* Schedule the interrupt to start sending renews: */
                  next_evh = op_intrpt_schedule_self(
                        op_sim_time() + (.5 * dhcp_opt_ptr->simple_data2),
DHCPC_ASSIGN_TIMEOUT);

                  /* Reset the retransmission timer: */
                  RTprev = 0;

                  timers_configured = OPC_TRUE;
                  }

            /* Notify IP process of interface assignments: */
            if (dyn_assignment_type != IpC_Dynamic_Assignment_None)
                  {
                  /* - Build the IpT_Dynamic_Assignment structure */
                  dyn_assignment_ptr = (IpT_Dynamic_Assignment *)
op_prg_mem_alloc (sizeof (IpT_Dynamic_Assignment));
                  dyn_assignment_ptr->assignment_type       = (assign_ptr-
>assign_type == DHCPC_ASSIGN_ADDR) ? IpC_Dynamic_Assignment_Addr_Create :
IpC_Dynamic_Assignment_Prefix_Create;
                  dyn_assignment_ptr->intf_index                  =
assign_ptr->iface_index;
                  dyn_assignment_ptr->dynamic_addr_range    =
                        inet_address_range_ptr_copy (inet_range_ptr);

                  /* - Append to list of dynamic assignments */
```

```
                  if (dyn_assignment_lptr == OPC_NIL)
                        dyn_assignment_lptr = op_prg_list_create ();
                  op_prg_list_insert (dyn_assignment_lptr,
dyn_assignment_ptr, OPC_LISTPOS_TAIL);
                  }

            }

      /* If we were unable to configure ANY interfaces, fallback to
       * the begin state, after a waiting period.  Otherwise, calculate the
       * total time of this transaction and record it in the stats.
       */
      if(new_addr + new_prefix + renew_addr + renew_prefix == 0)
            {
            next_evh = op_intrpt_schedule_self(op_sim_time() + max_sol_tmout,
DHCPC_MSG_FALLBACK);

            sprintf(log_str, "The message from server %s did not contain any
configuration information. "
                  "Soliciting service from any available server", srv_str);
            LOG(error_log_handle, log_str, PrgC_Log_Severity_Error);
            }
      else
            {
            /* Record the stats for the number of addr/prefixes new or
renewed,
             * and the total transaction time:
             */
            if(new_addr)
                  dhcp_update_cli_stat(DHCPC_COUNT_NEW_ADDR, new_addr);
            if(new_prefix)
                  dhcp_update_cli_stat(DHCPC_COUNT_NEW_PREFIX, new_prefix);
            if(renew_addr)
                  dhcp_update_cli_stat(DHCPC_COUNT_RENEW_ADDR, renew_addr);
            if(renew_prefix)
                  dhcp_update_cli_stat(DHCPC_COUNT_RENEW_PREFIX,
renew_prefix);

            dhcp_update_cli_stat(DHCPC_TRANSACTION_DELAY, trans_delay);
            }

      /* Notify IP by sending a remote interrupt with all assignments: */
      if (dyn_assignment_lptr != OPC_NIL)
            {
            void *prev_state;
            int i;

            /* Remove an old default route */
            if (default_route_action == DHCPC_DEFAULT_ROUTE_UPDATE)
                  {
                  /* - Build the IpT_Dynamic_Assignment structure */
                  dyn_assignment_ptr = (IpT_Dynamic_Assignment *)
op_prg_mem_alloc (sizeof (IpT_Dynamic_Assignment));
                  dyn_assignment_ptr->assignment_type       =
IpC_Dynamic_Assignment_Default_Route_Add;
                  dyn_assignment_ptr->intf_index                  =
send_iface;
```

```
                  dyn_assignment_ptr->dynamic_addr_range    =
inet_address_range_create (server_inet_addr, 0);

                  /* - Append to list of dynamic assignments */
                  op_prg_list_insert (dyn_assignment_lptr,
dyn_assignment_ptr, OPC_LISTPOS_TAIL);
                  }

            /* Add the default route if necessary */
            if (default_route_action >= DHCPC_DEFAULT_ROUTE_ADD)
                  {
                  /* Make the received address the new server address */
                  server_inet_addr = inet_address_copy (*rcvd_inet_address);

                  /* - Build the IpT_Dynamic_Assignment structure */
                  dyn_assignment_ptr = (IpT_Dynamic_Assignment *)
op_prg_mem_alloc (sizeof (IpT_Dynamic_Assignment));
                  dyn_assignment_ptr->assignment_type       =
IpC_Dynamic_Assignment_Default_Route_Add;
                  dyn_assignment_ptr->intf_index                  =
send_iface;
                  dyn_assignment_ptr->dynamic_addr_range    =
inet_address_range_create (server_inet_addr, 0);

                  /* - Append to list of dynamic assignments */
                  op_prg_list_insert (dyn_assignment_lptr,
dyn_assignment_ptr, OPC_LISTPOS_TAIL);
                  }

            /* - Convert list of dynamic assignments to
IpT_Dynamic_Assignment_Array */
            dyn_array_ptr = (IpT_Dynamic_Assignment_Array*) op_prg_mem_alloc
(sizeof (IpT_Dynamic_Assignment_Array));
            dyn_array_ptr->assignment_count = op_prg_list_size
(dyn_assignment_lptr);
            dyn_array_ptr->assignments = (IpT_Dynamic_Assignment **)
                  op_prg_mem_alloc (dyn_array_ptr->assignment_count * sizeof
(IpT_Dynamic_Assignment *));

            for (i = 0; i < dyn_array_ptr->assignment_count; i++)
                  {
                  dyn_array_ptr->assignments [i] = (IpT_Dynamic_Assignment *)
                        op_prg_list_remove (dyn_assignment_lptr,
OPC_LISTPOS_HEAD);
                  }

            /* - Set assignment array as event state */
            prev_state = op_ev_state_install (dyn_array_ptr, OPC_NIL);

            /* - Schedule a remote interrupt for ip_dispatch */
            op_intrpt_schedule_remote (op_sim_time (),
IPC_DYNAMIC_ASSIGNMENTS_INTRPT_CODE,
                  ip_support_module_ptr->module_id);

            op_ev_state_install (prev_state, OPC_NIL);

            /* Destroy the list structure */
```

```
            prg_list_destroy (dyn_assignment_lptr, OPC_FALSE);
            }

      /* Free all memory associated with this packet: */
      op_pk_destroy(rcv_pkt_ptr);
      dhcp_optlist_destroy(rcv_pkt_opts);

      FOUT;
      }

void
dhcp_get_packet(void) /* Me: When mobile node received DHCP message*/
      {

      FIN(dhcp_get_packet());
      /* Consume the packet from the input stream: */
      rcv_pkt_ptr   = op_pk_get(input_strm);

      /* Parse the message to obtain values for the state variables: */
      dhcp_msg_parse(rcv_pkt_ptr, &rcv_msg_type, &rcv_msg_trans);

      /* Get the options from the received packet: */
      op_pk_fd_get_ptr (rcv_pkt_ptr, DHCPC_PK_FIELD_OPTIONS,
(void**)&rcv_pkt_opts);

      /* If this is a Reply message with a Rapid Commit option
       * change the received message type:*/
      if((rcv_msg_type == DHCPC_MSG_REPLY)
            && (dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts))
!= OPC_NIL)
            rcv_msg_type = DHCPC_MSG_REPLY_RAPID;

      /* Update the local stats for the message type received: */
      dhcp_update_cli_stat(rcv_msg_type, 1);

      FOUT;
      }

/* End of Function Block */

void
dhcp_client (OP_SIM_CONTEXT_ARG_OPT)
      {
#if !defined (VOSD_NO_FIN)
      int _op_block_origin = 0;
#endif
      FIN_MT (dhcp_client ());

            {
            /* Temporary Variables */
            Objid cli_params_objid, timers_objid, timers_row_objid, objid1;
            char  tmp_str[TMP_STR_SIZE];
            char  val_str[TMP_STR_SIZE];
            int         tmp_int;
            double      initial_solicit_delay;

            /* IP address manipulations: */
```

```
            InetT_Addr_Family addr_fam;

            DhcpT_Opt*  dhcp_opt_ptr;

            /* End of Temporary Variables */

            FSM_ENTER ("dhcp_client")

            FSM_BLOCK_SWITCH
                  {
                  /** state (Begin) enter executives **/
                  FSM_STATE_ENTER_FORCED (0, "Begin", state0_enter_exec,
"dhcp_client [Begin enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Begin enter
execs]", state0_enter_exec)
                        {
                        /* Me: Initialize protocol related info that is reset
every time we restart a Solicit message exchange:*/
                        server_rapid_commit     = OPC_TRUE;
                        retrans_count           = 0;
                        server_id               = 0;
                        RTprev                        = 0;
                        rebind_time             = 0;
                        expire_time             = 0;
                        rcv_pkt_ptr             = OPC_NIL;
                        rcv_pkt_opts            = OPC_NIL;

                        /* Create the DHCP message to be sent: */
                        if(Rapid_Commit)
                              snd_msg_type = DHCPC_MSG_SOLICIT_RAPID;
                        else
                              snd_msg_type = DHCPC_MSG_SOLICIT;

            snd_pkt_ptr = dhcp_msg_create(snd_msg_type, last_trans_id++);

                        /* Initialize the DHCP options list to be sent: */
                        snd_pkt_opts = dhcp_optlist_create();

                        /* Always insert our Client ID in every message: */
                        prg_list_insert(snd_pkt_opts,

      dhcp_opt_create_id(DHCPC_OPT_CLIENTID, node_objid),
                                                PRGC_LISTPOS_TAIL);

                        if(Rapid_Commit)
                              {
                              DhcpT_Opt*                    tmp_dhcp_opt
      = OPC_NIL;
                              DhcpT_Cli_Assignment*   iface_config      =
OPC_NIL;
                              int                                   oro_length
      = 0;
                              int                                   rep;

                              /* Insert the Rapid Commit option: */
                              prg_list_insert(snd_pkt_opts,
```

```
      dhcp_opt_create(DHCPC_OPT_RAPID_COMMIT),
                                                      PRGC_LISTPOS_TAIL);

                              /* Insert an option for every interface we're
requesting configuration for: */
                              for(rep = 0; rep < num_interfaces; rep++)
                                    {
                                    iface_config = (DhcpT_Cli_Assignment
*)prg_list_access(&interfaces_config, rep);
                                    if(iface_config->assign_type ==
DHCPC_ASSIGN_ADDR)
                                          /* Requesting an address: */
                                          tmp_dhcp_opt =
dhcp_opt_create(DHCPC_OPT_IA_NA);
                                    else
                                          /* Requesting a prefix: */
                                          tmp_dhcp_opt =
dhcp_opt_create(DHCPC_OPT_IA_PD);


      tmp_dhcp_opt->simple_data = iface_config->iface_index;

                                    /* Set the length of this option: */
                                    tmp_dhcp_opt->length = 12;

                                    /* Add this option to the total length
count of the Option Request option: */
                                    oro_length += 2;

                                    prg_list_insert(snd_pkt_opts,
tmp_dhcp_opt, PRGC_LISTPOS_TAIL);
                                    }

                              tmp_dhcp_opt = dhcp_opt_create(DHCPC_OPT_ORO);
                              tmp_dhcp_opt->length = oro_length;
                              prg_list_insert(snd_pkt_opts, tmp_dhcp_opt,
      PRGC_LISTPOS_TAIL);
                              }

                        dhcp_msg_finalize(snd_pkt_ptr, snd_pkt_opts);

      /* Schedule a timer self-interrupt in case we need to retransmit: */
                        RTprev = dhcp_get_retrans_time(RTprev,
DHCPC_TRP_SOL_TIMEOUT, max_sol_tmout);
                        next_evh = op_intrpt_schedule_self(op_sim_time() +
RTprev, DHCPC_MSG_TIMEOUT);

                        /* Me: Send the message to the multicast address,
with an initial delay for this first Solicit message:*/
                        initial_solicit_delay =
op_dist_uniform(DHCPC_TRP_SOL_MAX_DELAY);
                        op_ici_install(command_ici_ptr);
                        op_pk_send_delayed(snd_pkt_ptr, output_strm,
initial_solicit_delay);
                        op_ici_install(OPC_NIL);
```

```
                        /* Log this initial transmission: */
                        if(logging == OPC_TRUE)
                              {
                              op_prg_log_handle_severity_set(&new_log_handle,
PrgC_Log_Severity_Information);
                              op_prg_log_entry_write_t(new_log_handle,
op_sim_time() + initial_solicit_delay,
                                    "Soliciting any server for new
configuration information");
                              }

                        /* Note this delayed transmission in the stats: */
                        if(Rapid_Commit)
                              {
                              op_stat_write_t(global_stats-
>msg_count_solicit_rapid, 1.0, op_sim_time() + initial_solicit_delay);
                              op_stat_write_t(local_stats-
>msg_count_solicit_rapid, 1.0, op_sim_time() + initial_solicit_delay);
                              }
                        else
                              {
                              op_stat_write_t(global_stats-
>msg_count_solicit, 1.0, op_sim_time() + initial_solicit_delay);
                              op_stat_write_t(local_stats->msg_count_solicit,
1.0, op_sim_time() + initial_solicit_delay);
                              }

                        /* Record the start time of this transaction: */
                        transaction_time = op_sim_time();
                        }
                        FSM_PROFILE_SECTION_OUT (state0_enter_exec)

                  /** state (Begin) exit executives **/
                  FSM_STATE_EXIT_FORCED (0, "Begin", "dhcp_client [Begin exit
execs]")

                  /** state (Begin) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Begin trans
conditions]", state0_trans_conds)
                  FSM_INIT_COND (Rapid_Commit)
                  FSM_TEST_COND (!Rapid_Commit)
                  FSM_TEST_LOGIC ("Begin")
                  FSM_PROFILE_SECTION_OUT (state0_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;,
"Rapid_Commit", "", "Begin", "Wait_Reply", "tr_12", "dhcp_client [Begin ->
Wait_Reply : Rapid_Commit / ]")
                        FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;,
"!Rapid_Commit", "", "Begin", "Wait_Advertise", "tr_13", "dhcp_client [Begin
-> Wait_Advertise : !Rapid_Commit / ]")
                        }
            /*---------------------------------------------------------*/
```

```
                  /** state (Wait_Reply) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (1, "Wait_Reply",
state1_enter_exec, "dhcp_client [Wait_Reply enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (3,"dhcp_client")

                  /** state (Wait_Reply) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (1, "Wait_Reply", "dhcp_client
[Wait_Reply exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply exit
execs]", state1_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();

                              if(server_id == 0)
                                    {
                                    /* If Mobile nodes have not yet
discovered a server, remember this new server's identification:*/
                                    dhcp_opt_ptr =
dhcp_optlist_get_opt(DHCPC_OPT_SERVERID, rcv_pkt_opts);
                                    server_id = dhcp_opt_ptr->simple_data;
                                    op_ima_obj_hname_get(server_id, srv_str,
200);
      /* Turn off rapid commit if the server didn't indicate support: */

      if(!(dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts)))
                                          server_rapid_commit = OPC_FALSE;

                                    }
                              }
                        else
                              intrpt_code = op_intrpt_code();
                        }
                        FSM_PROFILE_SECTION_OUT (state1_exit_exec)

                  /** state (Wait_Reply) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply trans
conditions]", state1_trans_conds)
                  FSM_INIT_COND (Rcv_Reply)
                  FSM_TEST_COND (Rcv_NonReply & !Rcv_Advertise)
                  FSM_TEST_COND (Msg_Tmout)
                  FSM_TEST_COND (Msg_Fail)
                  FSM_TEST_COND (Rcv_Advertise)
                  FSM_TEST_LOGIC ("Wait_Reply")
                  FSM_PROFILE_SECTION_OUT (state1_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec, Config();,
"Rcv_Reply", "Config()", "Wait_Reply", "Idle", "tr_21", "dhcp_client
[Wait_Reply -> Idle : Rcv_Reply / Config()]")
```

```
                        FSM_CASE_TRANSIT (1, 1, state1_enter_exec,
Discard();, "Rcv_NonReply & !Rcv_Advertise", "Discard()", "Wait_Reply",
"Wait_Reply", "tr_24", "dhcp_client [Wait_Reply -> Wait_Reply : Rcv_NonReply
& !Rcv_Advertise / Discard()]")
                        FSM_CASE_TRANSIT (2, 1, state1_enter_exec,
SendMsg();, "Msg_Tmout", "SendMsg()", "Wait_Reply", "Wait_Reply", "tr_26",
"dhcp_client [Wait_Reply -> Wait_Reply : Msg_Tmout / SendMsg()]")
                        FSM_CASE_TRANSIT (3, 0, state0_enter_exec, ;,
"Msg_Fail", "", "Wait_Reply", "Begin", "tr_46", "dhcp_client [Wait_Reply ->
Begin : Msg_Fail / ]")
                        FSM_CASE_TRANSIT (4, 1, state1_enter_exec,
SendMsg();, "Rcv_Advertise", "SendMsg()", "Wait_Reply", "Wait_Reply",
"tr_51", "dhcp_client [Wait_Reply -> Wait_Reply : Rcv_Advertise /
SendMsg()]")
                        }
            /*---------------------------------------------------------*/

                  /** state (Wait_Advertise) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (2, "Wait_Advertise",
state2_enter_exec, "dhcp_client [Wait_Advertise enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (5,"dhcp_client")

                  /** state (Wait_Advertise) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (2, "Wait_Advertise", "dhcp_client
[Wait_Advertise exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Advertise
exit execs]", state2_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();

                              if(server_id == 0)
                                    {
      /* If the mobile nodes have not yet discovered a server, remember this
new server's identification:*/
                                    dhcp_opt_ptr =
dhcp_optlist_get_opt(DHCPC_OPT_SERVERID, rcv_pkt_opts);
                                    server_id = dhcp_opt_ptr->simple_data;
                                    op_ima_obj_hname_get(server_id, srv_str,
200);
      /* Turn off rapid commit if the server didn't indicate support: */

      if(!(dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts)))
                                          server_rapid_commit = OPC_FALSE;

                                    }
                              }
                        else
                              intrpt_code = op_intrpt_code();
                        }
                        FSM_PROFILE_SECTION_OUT (state2_exit_exec)
```

```
                  /** state (Wait_Advertise) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Advertise trans
conditions]", state2_trans_conds)
                  FSM_INIT_COND (Rcv_Advertise)
                  FSM_TEST_COND (Rcv_NonAdvertise)
                  FSM_TEST_COND (Msg_Tmout)
                  FSM_TEST_LOGIC ("Wait_Advertise")
                  FSM_PROFILE_SECTION_OUT (state2_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 1, state1_enter_exec,
SendMsg();, "Rcv_Advertise", "SendMsg()", "Wait_Advertise", "Wait_Reply",
"tr_16", "dhcp_client [Wait_Advertise -> Wait_Reply : Rcv_Advertise /
SendMsg()]")
                        FSM_CASE_TRANSIT (1, 2, state2_enter_exec,
Discard();, "Rcv_NonAdvertise", "Discard()", "Wait_Advertise",
"Wait_Advertise", "tr_18", "dhcp_client [Wait_Advertise -> Wait_Advertise :
Rcv_NonAdvertise / Discard()]")
                        FSM_CASE_TRANSIT (2, 2, state2_enter_exec,
SendMsg();, "Msg_Tmout", "SendMsg()", "Wait_Advertise", "Wait_Advertise",
"tr_19", "dhcp_client [Wait_Advertise -> Wait_Advertise : Msg_Tmout /
SendMsg()]")
                        }
      /*---------------------------------------------------------*/

                  /** state (Idle) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (3, "Idle", state3_enter_exec,
"dhcp_client [Idle enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (7,"dhcp_client")

                  /** state (Idle) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (3, "Idle", "dhcp_client [Idle exit
execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Idle exit
execs]", state3_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();
                              }
                        else
                              intrpt_code = op_intrpt_code();
                        }
                        FSM_PROFILE_SECTION_OUT (state3_exit_exec)

                  /** state (Idle) transition processing **/
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Idle trans
conditions]", state3_trans_conds)
                  FSM_INIT_COND (Msg_Rcv)
                  FSM_TEST_COND (Assign_Tmout)
                  FSM_TEST_COND (Msg_Fail)
```

```
                  FSM_DFLT_COND
                  FSM_TEST_LOGIC ("Idle")
                  FSM_PROFILE_SECTION_OUT (state3_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec,
Discard();, "Msg_Rcv", "Discard()", "Idle", "Idle", "tr_27", "dhcp_client
[Idle -> Idle : Msg_Rcv / Discard()]")
                        FSM_CASE_TRANSIT (1, 4, state4_enter_exec,
SendMsg();, "Assign_Tmout", "SendMsg()", "Idle", "Wait_Reply_Conf", "tr_28",
"dhcp_client [Idle -> Wait_Reply_Conf : Assign_Tmout / SendMsg()]")
                        FSM_CASE_TRANSIT (2, 0, state0_enter_exec, ;,
"Msg_Fail", "", "Idle", "Begin", "tr_55", "dhcp_client [Idle -> Begin :
Msg_Fail / ]")
                        FSM_CASE_TRANSIT (3, 3, state3_enter_exec, ;,
"default", "", "Idle", "Idle", "tr_56", "dhcp_client [Idle -> Idle : default
/ ]")
                        }
            /*---------------------------------------------------------*/
                  /** state (Wait_Reply_Conf) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (4, "Wait_Reply_Conf",
state4_enter_exec, "dhcp_client [Wait_Reply_Conf enter execs]")

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (9,"dhcp_client")


                  /** state (Wait_Reply_Conf) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (4, "Wait_Reply_Conf", "dhcp_client
[Wait_Reply_Conf exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply_Conf
exit execs]", state4_exit_exec)
                        {
                        intrpt_type = op_intrpt_type();

                        if (intrpt_type == OPC_INTRPT_STRM)
                              {
                              dhcp_get_packet();

                              /* If mobile nodes have not yet discovered a
server, or if we are in the Rebind phase,remember this new server's
identification:*/
                              if((server_id == 0) || (snd_msg_type ==
DHCPC_MSG_REBIND))
                                    {
                                    dhcp_opt_ptr =
dhcp_optlist_get_opt(DHCPC_OPT_SERVERID, rcv_pkt_opts);
                                    server_id = dhcp_opt_ptr->simple_data;
                                    op_ima_obj_hname_get(server_id, srv_str,
200);
      /* Turn off rapid commit if the server didn't indicate support: */

      if(!(dhcp_optlist_get_opt(DHCPC_OPT_RAPID_COMMIT, rcv_pkt_opts)))
                                          server_rapid_commit = OPC_FALSE;

                                    }
```

```
                        }
                  else
                        intrpt_code = op_intrpt_code();
                  }
                  FSM_PROFILE_SECTION_OUT (state4_exit_exec)

            /** state (Wait_Reply_Conf) transition processing **/
            FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_Reply_Conf trans
conditions]", state4_trans_conds)
            FSM_INIT_COND (Rcv_Reply)
            FSM_TEST_COND (Rcv_NonReply)
            FSM_TEST_COND (Msg_Tmout)
            FSM_TEST_COND (Msg_Fail)
            FSM_TEST_LOGIC ("Wait_Reply_Conf")
            FSM_PROFILE_SECTION_OUT (state4_trans_conds)

            FSM_TRANSIT_SWITCH
                  {
                  FSM_CASE_TRANSIT (0, 3, state3_enter_exec, Config();,
"Rcv_Reply", "Config()", "Wait_Reply_Conf", "Idle", "tr_29", "dhcp_client
[Wait_Reply_Conf -> Idle : Rcv_Reply / Config()]")
                  FSM_CASE_TRANSIT (1, 4, state4_enter_exec,
Discard();, "Rcv_NonReply", "Discard()", "Wait_Reply_Conf",
"Wait_Reply_Conf", "tr_32", "dhcp_client [Wait_Reply_Conf -> Wait_Reply_Conf
: Rcv_NonReply / Discard()]")
                  FSM_CASE_TRANSIT (2, 4, state4_enter_exec,
SendMsg();, "Msg_Tmout", "SendMsg()", "Wait_Reply_Conf", "Wait_Reply_Conf",
"tr_34", "dhcp_client [Wait_Reply_Conf -> Wait_Reply_Conf : Msg_Tmout /
SendMsg()]")
                  FSM_CASE_TRANSIT (3, 0, state0_enter_exec,
Unconfig();, "Msg_Fail", "Unconfig()", "Wait_Reply_Conf", "Begin", "tr_50",
"dhcp_client [Wait_Reply_Conf -> Begin : Msg_Fail / Unconfig()]")
                  }
      /*---------------------------------------------------------*/

            /** state (Init) enter executives **/
            FSM_STATE_ENTER_UNFORCED_NOLABEL (5, "Init", "dhcp_client
[Init enter execs]")
                  FSM_PROFILE_SECTION_IN ("dhcp_client [Init enter
execs]", state5_enter_exec)
                  {
                  if (LTRACE_ACTIVE)
                        op_prg_odb_print_major ("DHCP Client: Begin
simulation" , OPC_NIL);

                  /* Initialize some variables: */
                  module_objid                  = op_id_self();
                  node_objid                    =
op_topo_parent(module_objid);
                  ip_support_module_ptr   = ip_support_module_data_get
(node_objid);
                  gateway_node                  =
ip_rte_node_is_gateway(ip_support_module_ptr);
                  last_trans_id                 = 1;
                  server_inet_addr        = INETC_ADDRESS_INVALID;

                  prg_list_init(&interfaces_config);
```

```
                        /* Determine if the nodes are logging DHCP: */
            op_ima_sim_attr_get(OPC_IMA_TOGGLE, "DHCP Logging", &logging);
                        if(LOGGING_ACTIVE)
                              {
                              new_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "New
Configuration", 100);
                              renew_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "Renewal", 100);
                              error_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "Protocol Error",
100);
                              expire_log_handle =
op_prg_log_handle_create(OpC_Log_Category_Protocol, "DHCP", "Expiration",
100);
                              }

                        /* Register process in the Process Registry: */
            op_ima_obj_attr_get(module_objid, "process model", tmp_str);

      my_proc_handle = oms_pr_process_register(node_objid, module_objid,
                              op_pro_self(), tmp_str);
oms_pr_attr_set(my_proc_handle,
                              "protocol", OMSC_PR_STRING, "dhcp",
                              OPC_NIL);

                  /* Schedule a self interrupt to wait for lower layers */
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state5_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (11,"dhcp_client")

                  /** state (Init) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (5, "Init", "dhcp_client [Init exit
execs]")

                  /** state (Init) transition processing **/
                  FSM_TRANSIT_FORCE (8, state8_enter_exec, ;, "default", "",
"Init", "Wait_0", "tr_19_0", "dhcp_client [Init -> Wait_0 : default / ]")
            /*---------------------------------------------------------*/

                  /** state (Wait_1) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (6, "Wait_1", state6_enter_exec,
"dhcp_client [Wait_1 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_1 enter
execs]", state6_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state6_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (13,"dhcp_client")
```

```
                  /** state (Wait_1) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (6, "Wait_1", "dhcp_client [Wait_1
exit execs]")

                  /** state (Wait_1) transition processing **/
                  FSM_TRANSIT_FORCE (9, state9_enter_exec, ;, "default", "",
"Wait_1", "Wait_2", "tr_19_4", "dhcp_client [Wait_1 -> Wait_2 : default / ]")
            /*---------------------------------------------------------*/

                  /** state (Wait_3) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (7, "Wait_3", state7_enter_exec,
"dhcp_client [Wait_3 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_3 enter
execs]", state7_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state7_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (15,"dhcp_client")

                  /** state (Wait_3) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (7, "Wait_3", "dhcp_client [Wait_3
exit execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_3 exit
execs]", state7_exit_exec)
                        {

                        /* Read in all of our attribute values: */
                        op_ima_obj_attr_get(module_objid, "DHCPv6 Client
Parameters", &cli_params_objid);
                        objid1 = op_topo_child(cli_params_objid,
OPC_OBJTYPE_GENERIC, 0);


                        op_ima_obj_attr_get_str(objid1, "Sending Interface",
TMP_STR_SIZE, val_str);
                        send_iface =
dhcp_get_ipv6_table_index_from_name(val_str, ip_support_module_ptr);
                        if(send_iface == -1)
                              {
                              sprintf(tmp_str, "Unable to get interface index
for interface '%s'", val_str);
                              op_sim_end("Error resolving interface index:",
                                    tmp_str, OPC_NIL, OPC_NIL);
                              }
                        else
                              {
                  /* Remember the link layer address of this interface: */
                              IpT_Interface_Info*     ip_iface_elem_ptr =
inet_rte_intf_tbl_access(ip_support_module_ptr, send_iface);
                              node_ll_addr =
ip_rte_intf_link_local_addr_get(ip_iface_elem_ptr);
                              }
```

```
                        op_ima_obj_attr_get(objid1, "Rapid Commit",
&rapid_commit);

                        op_ima_obj_attr_get(objid1, "Timers", &timers_objid);
                        timers_row_objid = op_topo_child(timers_objid,
OPC_OBJTYPE_GENERIC, 0);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Solicit Timeout", &max_sol_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Request Timeout", &max_req_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Request Retries", &max_req_retries);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Initial
Renew Timeout", &init_renew_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Renew Timeout", &max_renew_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Initial
Rebind Timeout", &init_rebind_tmout);
                        op_ima_obj_attr_get_int32(timers_row_objid, "Max
Rebind Timeout", &max_rebind_tmout);

                        /* Create the ICI to be used with UDP: */
                        command_ici_ptr = op_ici_create ("udp_command_inet");

                        /* Create a receive port for this application: */
                        tmp_int = dhcp_connect_to_udp(DHCPC_PORT_CLIENT,
module_objid,
                              &udp_objid, &input_strm, &output_strm,
command_ici_ptr);

                        if (tmp_int != UDPC_IND_SUCCESS)
                              {
                              sprintf (tmp_str, "%d in response to
CREATE_PORT command", tmp_int);
                              op_sim_end ("Error: process model dhcp received
error", tmp_str, "", "");
                              }

            /* Me: Since we are a client, mobile node will always send to the
well known multicast address for DHCP servers.  Set this address on the ICI
here: */
                        addr_fam = InetC_Addr_Family_v6;
                        server_multicast_addr =
inet_address_create(DHCPC_ADDR_ALL_AGENTS_AND_SERVERS, addr_fam);
                        op_ici_attr_set_ptr(command_ici_ptr, "rem_addr",
&server_multicast_addr);

            /* Me: For multicast to work, mobile node need a setting into the
"strm_index" field of the UDP ici - this setting will eventually be written
into the "multicast_major_port" of the IP layer ICI.*/
                        op_ici_attr_set (command_ici_ptr, "strm_index",
IPC_MCAST_ALL_MAJOR_PORTS);
                        server_multicast_addr =
inet_address_create(DHCPC_ADDR_ALL_AGENTS_AND_SERVERS, addr_fam);
```

```
                  /* Mobile node always send to the DHCP server port: */
                        op_ici_attr_set (command_ici_ptr, "rem_port",
DHCPC_PORT_SERVER);

                        /* Also, since we are a client, mobile node will
always send using our link local address: */
                        op_ici_attr_set (command_ici_ptr, "src_addr",
&node_ll_addr);


/* Build the list to contain configuration information for all existing
interfaces. For both routers and hosts.*/
                        if(gateway_node)
                              {
                              int         rep;
                              int         num_total_interfaces = 0;
                              Objid ip_group_objid, iface_info_objid,
iface_row_objid;
                              Objid gaddr_info_objid, gaddr_row_objid;
                              DhcpT_Cli_Assignment * iface_config;

                              /* First get the total number of interfaces on
this node.  We're only dealing with physical interfaces for now.*/
                              op_ima_obj_attr_get(node_objid, "IPv6
Parameters", &ip_group_objid);
                              objid1 = op_topo_child(ip_group_objid,
OPC_OBJTYPE_GENERIC, 0);
                              op_ima_obj_attr_get(objid1, "Interface
Information", &iface_info_objid);
                              num_total_interfaces =
op_topo_child_count(iface_info_objid, OPC_OBJTYPE_GENERIC);

                              for (rep = 0; rep < num_total_interfaces;
rep++)
                                    {

                                    iface_row_objid =
op_topo_child(iface_info_objid, OPC_OBJTYPE_GENERIC, rep);


                                    op_ima_obj_attr_get(iface_row_objid,
"Global Address(es)", &gaddr_info_objid);
                                    gaddr_row_objid =
op_topo_child(gaddr_info_objid, OPC_OBJTYPE_GENERIC, 0);

                                    /* Get the "Address": */
                                    op_ima_obj_attr_get_str(gaddr_row_objid,
"Address", 100, val_str);
                                    if(strstr(val_str, "DHCP"))
                                          {
                                          int  intf_table_index;
                                          char iface_str [8];

      /* First confirm that the interface has been added into the interface
table */
                                          op_ima_obj_attr_get_str
(iface_row_objid, "Name", 7, iface_str);
```

```
                                        if ((intf_table_index =
dhcp_get_ipv6_table_index_from_name (iface_str, ip_support_module_ptr)) == -
1)
                                            {
                                            char log_str [500];

                  printf (log_str, "Interface %s has DHCP specified, but is
not being used in the simulation. Confirm that it is connected, has a link-
local address, and no other global addresses are specified.", iface_str);
                                            LOG(error_log_handle,
log_str, PrgC_Log_Severity_Error);

                                            continue;
                                            }
                                        else
                                            {
            /* This interface is using DHCP for either address or prefix
assignment. */
                                            iface_config =
(DhcpT_Cli_Assignment *)op_prg_mem_alloc(sizeof(DhcpT_Cli_Assignment));
                                            iface_config->configured =
OPC_FALSE;
                                            iface_config->iface_index =
rep;
                                            if(strstr(val_str, "Prefix
Delegation"))
                                                  iface_config-
>assign_type = DHCPC_ASSIGN_PREFIX;
                                            else
                                                  iface_config-
>assign_type = DHCPC_ASSIGN_ADDR;


      op_ima_obj_attr_get_str(iface_row_objid, "Name", TMP_STR_SIZE,
tmp_str);
                                            iface_config->iface_name =
prg_string_copy(tmp_str);


      prg_list_insert(&interfaces_config, iface_config, PRGC_LISTPOS_TAIL);
                                            }
                                        }
                                    }
                                }
                           else
                                {
      /* Initialize the configuration data for only a single interface: */
                                DhcpT_Cli_Assignment * iface_config =
                                      (DhcpT_Cli_Assignment
*)op_prg_mem_alloc(sizeof(DhcpT_Cli_Assignment));
                                iface_config->configured = OPC_FALSE;
                                iface_config->iface_index = send_iface;
                                iface_config->assign_type = DHCPC_ASSIGN_ADDR;

                                iface_config->iface_name = prg_string_copy("N/A
(single interface host)");
```

```
                              prg_list_insert(&interfaces_config,
iface_config, PRGC_LISTPOS_TAIL);
                              }
                        num_interfaces = prg_list_size(&interfaces_config);

                        if(num_interfaces < 1)
                              {
                              LOG(error_log_handle, "No interfaces are
requesting configuration information. "
                                    "Exiting DHCP client.",
PrgC_Log_Severity_Error);
                              op_pro_destroy(op_pro_self());
                              }
                        /* Get the handles to the DHCP statistics: */
                        global_stats = dhcp_get_global_stathandles();
                        local_stats  =
dhcp_get_local_stathandles(DHCPC_CLIENT);
                        }
                        FSM_PROFILE_SECTION_OUT (state7_exit_exec)

                  /* Me: Register this address for multicast: */

      Inet_Address_Multicast_Register(server_multicast_addr, tmp_int,

      DHCPC_PORT_SERVER, ip_support_module_ptr);

                  /** state (Wait_3) transition processing **/
                  FSM_TRANSIT_FORCE (0, state0_enter_exec, ;, "default", "",
"Wait_3", "Begin", "tr_19_2", "dhcp_client [Wait_3 -> Begin : default / ]")
            /*---------------------------------------------------------*/

/** state (Wait_0) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (8, "Wait_0", state8_enter_exec,
"dhcp_client [Wait_0 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_0 enter
execs]", state8_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state8_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (17,"dhcp_client")


                  /** state (Wait_0) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (8, "Wait_0", "dhcp_client [Wait_0
exit execs]")


                  /** state (Wait_0) transition processing **/
                  FSM_TRANSIT_FORCE (6, state6_enter_exec, ;, "default", "",
"Wait_0", "Wait_1", "tr_19_3", "dhcp_client [Wait_0 -> Wait_1 : default / ]")
      /*---------------------------------------------------------*/
```

```
                  /** state (Wait_2) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (9, "Wait_2", state9_enter_exec,
"dhcp_client [Wait_2 enter execs]")
                        FSM_PROFILE_SECTION_IN ("dhcp_client [Wait_2 enter
execs]", state9_enter_exec)
                        {
                        op_intrpt_schedule_self(op_sim_time(), 0);
                        }
                        FSM_PROFILE_SECTION_OUT (state9_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (19,"dhcp_client")

                  /** state (Wait_2) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (9, "Wait_2", "dhcp_client [Wait_2
exit execs]")

                  /** state (Wait_2) transition processing **/
                  FSM_TRANSIT_FORCE (7, state7_enter_exec, ;, "default", "",
"Wait_2", "Wait_3", "tr_19_1", "dhcp_client [Wait_2 -> Wait_3 : default / ]")
            /*---------------------------------------------------------*/
                  }
            FSM_EXIT (5,"dhcp_client")
            }
      }

/* Undefine shortcuts to state variables to avoid */
#undef module_objid
#undef node_objid
#undef my_proc_handle
#undef udp_objid
#undef ip_support_module_ptr
#undef command_ici_ptr
#undef max_sol_tmout
#undef max_req_tmout
#undef max_req_retries
#undef init_renew_tmout
#undef max_renew_tmout
#undef init_rebind_tmout
#undef max_rebind_tmout
#undef send_iface
#undef last_trans_id
#undef rapid_commit
#undef server_rapid_commit
#undef server_id
#undef srv_str
#undef interfaces_config
#undef gateway_node
#undef input_strm
#undef output_strm
#undef snd_pkt_ptr
#undef snd_pkt_opts
#undef snd_msg_type
#undef rcv_pkt_ptr
#undef rcv_pkt_opts
#undef rcv_msg_type
```

```
#undef rcv_msg_trans
#undef RTprev
#undef expire_time
#undef rebind_time
#undef retrans_count
#undef intrpt_type
#undef intrpt_code
#undef num_interfaces
#undef next_evh
#undef server_multicast_addr
#undef node_ll_addr
#undef global_stats
#undef local_stats
#undef transaction_time
#undef logging
#undef new_log_handle
#undef renew_log_handle
#undef error_log_handle
#undef expire_log_handle
#undef server_inet_addr

#undef FIN_PREAMBLE_DEC
#undef FIN_PREAMBLE_CODE

#define FIN_PREAMBLE_DEC
#define FIN_PREAMBLE_CODE

VosT_Obtype
_op_dhcp_client_init (int * init_block_ptr)
	{
	VosT_Obtype obtype = OPC_NIL;
	FIN_MT (_op_dhcp_client_init (init_block_ptr))

	obtype = Vos_Define_Object_Prstate ("proc state vars (dhcp_client)",
		sizeof (dhcp_client_state));
	*init_block_ptr = 10;

	FRET (obtype)
	}

VosT_Address
_op_dhcp_client_alloc (VosT_Obtype obtype, int init_block)
	{
#if !defined (VOSD_NO_FIN)
	int _op_block_origin = 0;
#endif
	dhcp_client_state * ptr;
	FIN_MT (_op_dhcp_client_alloc (obtype))

	ptr = (dhcp_client_state *)Vos_Alloc_Object (obtype);
	if (ptr != OPC_NIL)
		{
		ptr->_op_current_block = init_block;
#if defined (OPD_ALLOW_ODB)
		ptr->_op_current_state = "dhcp_client [Init enter execs]";
#endif
		}
```

```
	FRET ((VosT_Address)ptr)
	}

void
_op_dhcp_client_svar (void * gen_ptr, const char * var_name, void **
var_p_ptr)
	{
	dhcp_client_state	*prs_ptr;

	FIN_MT (_op_dhcp_client_svar (gen_ptr, var_name, var_p_ptr))

	if (var_name == OPC_NIL)
		{
		*var_p_ptr = (void *)OPC_NIL;
		FOUT
		}
	prs_ptr = (dhcp_client_state *)gen_ptr;

	if (strcmp ("module_objid" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->module_objid);
		FOUT
		}
	if (strcmp ("node_objid" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->node_objid);
		FOUT
		}
	if (strcmp ("my_proc_handle" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->my_proc_handle);
		FOUT
		}
	if (strcmp ("udp_objid" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->udp_objid);
		FOUT
		}
	if (strcmp ("ip_support_module_ptr" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->ip_support_module_ptr);
		FOUT
		}
	if (strcmp ("command_ici_ptr" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->command_ici_ptr);
		FOUT
		}
	if (strcmp ("max_sol_tmout" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->max_sol_tmout);
		FOUT
		}
	if (strcmp ("max_req_tmout" , var_name) == 0)
		{
		*var_p_ptr = (void *) (&prs_ptr->max_req_tmout);
		FOUT
```

```
      }
if (strcmp ("max_req_retries" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->max_req_retries);
      FOUT
      }
if (strcmp ("init_renew_tmout" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->init_renew_tmout);
      FOUT
      }
if (strcmp ("max_renew_tmout" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->max_renew_tmout);
      FOUT
      }
if (strcmp ("init_rebind_tmout" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->init_rebind_tmout);
      FOUT
      }
if (strcmp ("max_rebind_tmout" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->max_rebind_tmout);
      FOUT
      }
if (strcmp ("send_iface" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->send_iface);
      FOUT
      }
if (strcmp ("last_trans_id" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->last_trans_id);
      FOUT
      }
if (strcmp ("rapid_commit" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->rapid_commit);
      FOUT
      }
if (strcmp ("server_rapid_commit" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->server_rapid_commit);
      FOUT
      }
if (strcmp ("server_id" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->server_id);
      FOUT
      }
if (strcmp ("srv_str" , var_name) == 0)
      {
      *var_p_ptr = (void *) (prs_ptr->srv_str);
      FOUT
      }
if (strcmp ("interfaces_config" , var_name) == 0)
```

```
      {
      *var_p_ptr = (void *) (&prs_ptr->interfaces_config);
      FOUT
      }
if (strcmp ("gateway_node" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->gateway_node);
      FOUT
      }
if (strcmp ("input_strm" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->input_strm);
      FOUT
      }
if (strcmp ("output_strm" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->output_strm);
      FOUT
      }
if (strcmp ("snd_pkt_ptr" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->snd_pkt_ptr);
      FOUT
      }
if (strcmp ("snd_pkt_opts" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->snd_pkt_opts);
      FOUT
      }
if (strcmp ("snd_msg_type" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->snd_msg_type);
      FOUT
      }
if (strcmp ("rcv_pkt_ptr" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->rcv_pkt_ptr);
      FOUT
      }
if (strcmp ("rcv_pkt_opts" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->rcv_pkt_opts);
      FOUT
      }
if (strcmp ("rcv_msg_type" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->rcv_msg_type);
      FOUT
      }
if (strcmp ("rcv_msg_trans" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->rcv_msg_trans);
      FOUT
      }
if (strcmp ("RTprev" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->RTprev);
```

```
      FOUT
      }
if (strcmp ("expire_time" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->expire_time);
      FOUT
      }
if (strcmp ("rebind_time" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->rebind_time);
      FOUT
      }
if (strcmp ("retrans_count" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->retrans_count);
      FOUT
      }
if (strcmp ("intrpt_type" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->intrpt_type);
      FOUT
      }
if (strcmp ("intrpt_code" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->intrpt_code);
      FOUT
      }
if (strcmp ("num_interfaces" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->num_interfaces);
      FOUT
      }
if (strcmp ("next_evh" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->next_evh);
      FOUT
      }
if (strcmp ("server_multicast_addr" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->server_multicast_addr);
      FOUT
      }
if (strcmp ("node_ll_addr" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->node_ll_addr);
      FOUT
      }
if (strcmp ("global_stats" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->global_stats);
      FOUT
      }
if (strcmp ("local_stats" , var_name) == 0)
      {
      *var_p_ptr = (void *) (&prs_ptr->local_stats);
      FOUT
      }
```

```
      if (strcmp ("transaction_time" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->transaction_time);
            FOUT
            }
      if (strcmp ("logging" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->logging);
            FOUT
            }
      if (strcmp ("new_log_handle" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->new_log_handle);
            FOUT
            }
      if (strcmp ("renew_log_handle" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->renew_log_handle);
            FOUT
            }
      if (strcmp ("error_log_handle" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->error_log_handle);
            FOUT
            }
      if (strcmp ("expire_log_handle" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->expire_log_handle);
            FOUT
            }
      if (strcmp ("server_inet_addr" , var_name) == 0)
            {
            *var_p_ptr = (void *) (&prs_ptr->server_inet_addr);
            FOUT
            }
      *var_p_ptr = (void *)OPC_NIL;

      FOUT
      }

static void /* Me: Joinning multicast tree by using CoA address*/
ip_igmp_host_join_grp (IpT_Address ip_grp_addr, int interface)
      {
      IpT_Igmp_Host_Grp_Elem*       grp_elem_ptr;
      List*                               ip_grp_intf_list_ptr;
      Packet*                                   igmp_msg_pkptr;
      Packet*                                   ip_dgram_pkptr;
      char                                msg0 [256], msg1 [256], msg2 [256];
      char                                ip_addr_str [IPC_ADDR_STR_LEN];

      FIN (ip_igmp_host_join_grp (ip_grp_addr, interface));

      /* Generate a trace message   */
      if (LTRACE_IGMP)
            {
            ip_address_print (ip_addr_str, ip_grp_addr);
```

```
            sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
            sprintf (msg1, "IP Interface              :           %d",
interface);
            op_prg_odb_print_major ("Received a Join request from an
application for: ", msg0, msg1, OPC_NIL);
            }

      /* Check if the group membership information for the given IP group
address already exists */
      grp_elem_ptr = ip_igmp_host_get_grp_elem (ip_grp_addr, interface);
      if (grp_elem_ptr == OPC_NIL)
            {

            grp_elem_ptr = ip_igmp_host_grp_elem_alloc ();

            /* Set the fields of the data structure */
            grp_elem_ptr->ip_grp_addr = ip_address_copy (ip_grp_addr);
            grp_elem_ptr->interface   = interface;

            /* Until now only one application on this node has joined this
group */
            grp_elem_ptr->num_of_apps = 1;

            /* Assign a timer ID to this group's delay timer */
            grp_elem_ptr->delay_timer_id = timer_id++;

            /* Initialize this field to the value of Robustness Variable
attribute minus 1 */
            grp_elem_ptr->unsolicit_msg_count = robustness_variable - 1;

            /* Since we are going to send an Unsolicited Membership Report
message, set this field to OPC_TRUE*/
                        */
            grp_elem_ptr->report_sent_flag = OPC_TRUE;

            /* Start the delay timer to send the next Unsolicited Membership
Report message. */

            if (igmp_sim_efficiency == OPC_FALSE)
                  {
                  /* Set the delay timer to OPC_TRUE, as we are going to
start */
                  /* the timer for sending the next Unsolicited Report
message */
                  grp_elem_ptr->timer_on_flag = OPC_TRUE;

                  /* Start the delay timer      */
                  grp_elem_ptr->delay_timer_evh = op_intrpt_schedule_self
(op_sim_time () +
                              unsolicited_report_interval, grp_elem_ptr-
>delay_timer_id);
                  }
            else
                  {
                  /* Set the unsolicited messages count to zero   */
                  grp_elem_ptr->unsolicit_msg_count = 0;
```

```
                }

            /** Add this group membership information to the corresponding
list      **/

            /* Generate a trace message   */
            if (LTRACE_IGMP)
                  {
                  ip_address_print (ip_addr_str, ip_grp_addr);
                  sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
                  sprintf (msg1, "IP Interface              :           %d",
interface);
                  strcpy (msg2, "to the table, which is maintained by IGMP
Host process.");
                  op_prg_odb_print_major ("Adding group membership
information for: ", msg0, msg1, msg2, OPC_NIL);
                  }

            /* Get the list for the interface on which this node has joined
the group */
            ip_grp_intf_list_ptr = (List *) op_prg_list_access
(ip_grp_list_ptr, interface);

            /* Add the group membership information data structure to the
list        */
            op_prg_list_insert (ip_grp_intf_list_ptr, grp_elem_ptr,
OPC_LISTPOS_TAIL);

            /** Create and send an Unsolicited Membership Report message **/

            /* Generate a trace message   */
            if (LTRACE_IGMP)
                  {
                  ip_address_print (ip_addr_str, ip_grp_addr);
                  sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
                  sprintf (msg1, "IP Interface              :           %d",
interface);
                  op_prg_odb_print_major ("Sending an IGMP Unsolicited
Membership Report message for: ", msg0, msg1, OPC_NIL);
                  op_prg_odb_print_major ("Starting Delay timer for the above
group.", OPC_NIL);
                  }

            /* Create an IGMP Report message  */
            igmp_msg_pkptr = ip_igmp_create_igmp_msg
(IpC_Igmp_Membership_Report_Msg, 0, ip_grp_addr);

            /* Update IGMP messages sent statistics   */
            ip_igmp_host_igmp_msgs_sent_stat_update (op_pk_total_size_get
(igmp_msg_pkptr));

            /* Create an IP datagram for transmitting the IGMP message */
            ip_dgram_pkptr = ip_igmp_create_ipdgram (igmp_msg_pkptr,
ip_grp_addr);
```

```
            /* Schedule a procedure interrupt for the current simulation time
to invoke the IP process */
            op_intrpt_schedule_call (op_sim_time (), interface,
ip_igmp_host_ip_process_invoke, ip_dgram_pkptr);
            }
      else
            {
            /* Group membership information already exists. Another
application on this node has joined */
      /* this group. Increment the num_of_apps field's value by one */
            grp_elem_ptr->num_of_apps++;
            }
      FOUT;
      }

static void /* Using IGMP protocol to leave multicast group*/
ip_igmp_host_leave_grp (IpT_Address ip_grp_addr, int interface)
      {
      IpT_Igmp_Host_Grp_Elem*       grp_elem_ptr;
    List*                           ip_grp_intf_list_ptr;
      int                           num_grp_elems, i;
      Packet*                       igmp_msg_pkptr;
      Packet*                       ip_dgram_pkptr;
      char                          ip_addr_str [IPC_ADDR_STR_LEN];
      char                          msg0 [256], msg1 [256], msg2 [256];

      FIN (ip_igmp_host_leave_grp (ip_grp_addr, interface));
      /* Generate a trace message   */
      if (LTRACE_IGMP)
            {
            ip_address_print (ip_addr_str, ip_grp_addr);
            sprintf (msg0, "IP Group Address          :           %s",
ip_addr_str);
            sprintf (msg1, "IP Interface              :           %d",
interface);
            op_prg_odb_print_major ("Received a Leave request from an
application for: ", msg0, msg1, OPC_NIL);
            }

      /* Get the list corresponding to the given interface */
      ip_grp_intf_list_ptr = (List *) op_prg_list_access (ip_grp_list_ptr,
interface);

    /* Determine the number of elements in the list */
      num_grp_elems = op_prg_list_size (ip_grp_intf_list_ptr);

/* Access the group membership information data structure for the given */
      /* IP group address     from the list
                        */
      for (i=0; i<num_grp_elems; i++)
            {
            /* Access ith element from the list */
            grp_elem_ptr = (IpT_Igmp_Host_Grp_Elem *)
                  op_prg_list_access (ip_grp_intf_list_ptr, i);

/* If this is the list element we are looking for, break from the loop */
            if (ip_address_equal (grp_elem_ptr->ip_grp_addr, ip_grp_addr))
```

```
                  {
                  break;
                  }
            }

      /* Sanity check */
      if (i == num_grp_elems)
            {
            /** The group membership information for the given group address
and interface doesn't exist in the list **/

            /* Report a log message */
            ip_address_print (ip_addr_str, ip_grp_addr);
            ip_igmp_host_log_found_no_grp_info (ip_addr_str, interface);

            /* Generate a trace message   */
            if (LTRACE_IGMP)
                  {
sprintf (msg0, "IP Group Address          :           %s", ip_addr_str);
sprintf (msg1, "IP Interface              :           %d", interface);
strcpy (msg2, "doesn't exist in the list, which is maintained by IGMP Host
process.");
op_prg_odb_print_major ("Group membership information for: ", msg0, msg1,
msg2, OPC_NIL);
                  }
            }
      else
            {
            /* An application on the node has left this group. Decrement
num_of_apps field */
            grp_elem_ptr->num_of_apps--;
            if (grp_elem_ptr->num_of_apps == 0)
                  {
                  /* Generate a trace message   */
                  if (LTRACE_IGMP)
                        {
                        ip_address_print (ip_addr_str, ip_grp_addr);
                        sprintf (msg0, "IP Group Address          :
      %s", ip_addr_str);
                        sprintf (msg1, "IP Interface              :
      %d", interface);
                        strcpy (msg2, "from the table, which is maintained by
IGMP Host process.");
                        op_prg_odb_print_major ("Removing the group
membership information for: ", msg0, msg1, msg2, OPC_NIL);
                        }

                  /* Remove the group membership information from the list */
                  op_prg_list_remove (ip_grp_intf_list_ptr, i);

                  /* Cancel the delay timer for this group membership, if its
running */
                  if (grp_elem_ptr->timer_on_flag == OPC_TRUE)
                        {
                        op_ev_cancel (grp_elem_ptr->delay_timer_evh);
                        }
```

```
                  /* If this is the last node to send a Report message, send
a Leave message */
                  if (grp_elem_ptr->report_sent_flag == OPC_TRUE)
                        {
                        /* Generate a trace message   */
                        if (LTRACE_IGMP)
                              {
                              ip_address_print (ip_addr_str, ip_grp_addr);
                              sprintf (msg0, "IP Group Address          :
      %s", ip_addr_str);
                              sprintf (msg1, "IP Interface              :
      %d", interface);
                              op_prg_odb_print_major ("Sending an IGMP Leave
Group message for: ", msg0, msg1, OPC_NIL);
                              }

                        /* Create an IGMP Leave Group message */
                        igmp_msg_pkptr = ip_igmp_create_igmp_msg
(IpC_Igmp_Leave_Group_Msg, 0, grp_elem_ptr->ip_grp_addr);

                        /* Update IGMP messages sent statistics   */
                        ip_igmp_host_igmp_msgs_sent_stat_update
(op_pk_total_size_get (igmp_msg_pkptr));

                        /* Create an IP datagram to transmit the Leave group
message */
                        ip_dgram_pkptr = ip_igmp_create_ipdgram
(igmp_msg_pkptr, ip_address_create (IPC_ALL_ROUTERS_MULTICAST_ADDR));

                        /* Schedule a procedure interrupt for the current
simulation time to invoke the IP process */
                        op_intrpt_schedule_call (op_sim_time (),
grp_elem_ptr->interface, &ip_igmp_host_ip_process_invoke, ip_dgram_pkptr);
                        }
                  ip_igmp_host_grp_elem_dealloc (grp_elem_ptr);
                  }
            }

      FOUT;
      }

static void
mip_mn_register (int lifetime, MipT_MN_Agent_Info agent_info, Boolean direct)
      {
      IpT_Port_Info     port_info;
      double                        retry_timer;
      Ici*              reg_ici_ptr;

      /** PURPOSE: Registers with home agent.**/
      /** REQUIRES: lifetime it is requesting and care of address to use.
      **/
      /** EFFECTS: An interrupt sent to the reg manager.**/
      FIN (mip_mn_register (lifetime, agent_info, direct));

      /* Create Ici for mobile IP module. */
      reg_ici_ptr = op_ici_create ("mobile_ip_reg_ici");
```

```
      /* Set the foreign agent address for later reference. */
      agent_address = agent_info.address;
      current_agent_pref_level = agent_info.pref_level;
      current_roaming_intf = agent_info.incoming_intf_ptr;

      /* Set the appropriate values on the ici. */
      op_ici_attr_set (reg_ici_ptr, "reg_type", MipC_Reg_Type_Req);
      op_ici_attr_set (reg_ici_ptr, "home_address", inet_ipv4_address_get
(home_address));
      op_ici_attr_set (reg_ici_ptr, "home_agent", inet_ipv4_address_get
(ha_address));
      op_ici_attr_set (reg_ici_ptr, "lifetime_req", lifetime);
      op_ici_attr_set (reg_ici_ptr, "dest_address", inet_ipv4_address_get
(agent_address));
      op_ici_attr_set (reg_ici_ptr, "identification", ++reg_id);
      op_ici_attr_set (reg_ici_ptr, "s", simultaneous_binding);

      /* What is the type of the current registration? */
      if (direct)
            {
            /* Diect to the home agent. */
            direct_reg = OPC_TRUE;
            op_ici_attr_set (reg_ici_ptr, "care_of_address", OPC_NIL);
            }
      else
            {
            direct_reg = OPC_FALSE;
            op_ici_attr_set (reg_ici_ptr, "care_of_address",
inet_ipv4_address_get (agent_address));
            }

      /* Send the registrtation packet out. */
      op_ici_install (reg_ici_ptr);
      op_intrpt_schedule_process (proc_info_struct_ptr->mip_reg_mgr_phndl,
op_sim_time (), 0);
      op_ici_install (OPC_NIL);

      /* need to reset the default gateway to the agent, that are registering
with. */
      port_info = ip_rte_port_info_from_tbl_index (module_data,
ip_rte_intf_index_get (agent_info.incoming_intf_ptr));

      if (!default_gateway)
            {
            Ip_Cmn_Rte_Table_Entry_Add (module_data->ip_route_table, OPC_NIL,
                  IpI_Default_Addr, IpI_Default_Addr,
agent_info.incoming_intf_ptr->addr_range_ptr->address, port_info,
                  0, IP_CMN_RTE_TABLE_UNIQUE_ROUTE_PROTO_ID
(IPC_DYN_RTE_MOBILE_IP, IPC_NO_MULTIPLE_PROC), 0, OPC_NIL);

            /* Cache info. */
            default_gateway = OPC_TRUE;
            }
      else
            {
            Ip_Cmn_Rte_Table_Entry_Update (module_data->ip_route_table,
IpI_Default_Addr, IpI_Default_Addr,
```

```
                  last_default_addr, IP_CMN_RTE_TABLE_UNIQUE_ROUTE_PROTO_ID
(IPC_DYN_RTE_MOBILE_IP, IPC_NO_MULTIPLE_PROC),
                  agent_info.incoming_intf_ptr->addr_range_ptr->address,
port_info, 0, OPC_NIL);
            }

      /* Cache info. */
      last_default_addr = ip_rte_intf_addr_get
(agent_info.incoming_intf_ptr);

      /*  need to schedule retry in case that do not get the answer back. */
      retry_timer = reg_info.interval * pow (2.0, (double) retry_counter);
      if (retry_timer > (double) (reg_info.req_lifetime))
            {
            retry_timer = (double) (reg_info.req_lifetime);
            }

      reg_retry_timer_ehndl = op_intrpt_schedule_self (op_sim_time () +
retry_timer, MipC_MN_Timer_Retry);
      retry_counter++;

      if (MIP_TRACE)
            {
            op_prg_odb_print_major ("Trying registering with HA.", OPC_NIL);
            }

      FOUT;
      }

static void
mip_mn_agent_timer_update (double new_lifetime)
      {
      /** PURPOSE: Update the timer for agent timeouts.**/
      /** REQUIRES: new lifetime value from the last agent ad.    **/
      /** EFFECTS: timer event handle gets updated.**/
      FIN (mip_mn_agent_timer_update (new_lifetime));

      /* Cancel the current handle. */
      op_ev_cancel_if_pending (agent_timer_ehndl);

      /* Schedule new one. */
      agent_timer_ehndl = op_intrpt_schedule_self (new_lifetime,
MipC_MN_Timer_Agent);

      FOUT;
      }

static Packet* /* Switch the Active/Standby mode. */
ip_icmp_echo_request_packet_create (int req_index)
      {
      Packet*                 req_pkptr;


      FIN (ip_icmp_echo_request_packet_create (req_index));

      /* Create an ICMP packet to switch Active/Standby mode.     */
      req_pkptr = op_pk_create_fmt ("ip_icmp_echo");
```

```
      /* Set the "type" field to indicate that this is a    */
      /* request packet.                                                */
      op_pk_nfd_set (req_pkptr, "type", IpC_Icmp_Echo_Request);

      op_pk_nfd_set (req_pkptr, "identifier", req_index);

      op_pk_nfd_set (req_pkptr, "sequence number", ping_specs_ptr
[req_index].seq_number);

      /* set the source node object id in the packet.                   */
      op_pk_nfd_set (req_pkptr, "source module objid", (double) my_objid);

/* Increment the sequence number for the next message send operation.   */
      ping_specs_ptr [req_index].seq_number++;

      if (ping_specs_ptr [req_index].ping_pattern_ptr->pkt_size > 0)
            {

            op_pk_bulk_size_set (req_pkptr, ping_specs_ptr
[req_index].ping_pattern_ptr->pkt_size * 8);
            }

      FRET (req_pkptr);
      }

static void
ip_icmp_request_timeout (void *state_ptr, int index)/* Me:Check timeout for
the connection*/
      {
      IpT_Icmp_Ping_Specs*    ping_spec_ptr;
      char                          dest_host_name [OMSC_HNAME_MAX_LEN];

      FIN (ip_icmp_request_timeout (void *state_ptr, int index));

      /* Get the ping_spec_ptr from the state ptr.    */
      ping_spec_ptr = (IpT_Icmp_Ping_Specs *)state_ptr;

      oms_tan_hname_get (ping_spec_ptr->dest_objid, dest_host_name);

      op_prg_log_entry_write (ip_icmp_timeout_log_handle,
            "ERROR(S):\n"
            " The echo request for destination (%s) \n"
            " sent at (%.2f) seconds failed to \n"
            " receive a response before the timeout \n"
            " interval of (%.5f) seconds. \n"
            "\n",

            dest_host_name, ping_spec_ptr [index].start_time,
            ping_spec_ptr [index].ping_pattern_ptr->timeout, ping_spec_ptr
[index].ping_pattern_ptr->timeout);

      FOUT;

      }

static void
```

```
ip_pim_sm_join_prune_msg (void) /* Me:Function Join/Standby message for
keeping route */
      {
      IpT_Pim_Sm_Rte_Entry*               rpt_rte_entry_ptr;
      IpT_Pim_Sm_Rte_Entry*               rte_entry_ptr;
      IpT_Address                         rp_ip_addr;
      IpT_Pim_Sm_Msg*                     pim_sm_msg_ptr;
      IpT_Pim_Sm_Join_Prune_Msg*          join_prune_msg_ptr;
      IpT_Pim_Sm_Join_Prune_List_Elem*    join_prune_lelem_ptr;
      IpT_Pim_Sm_Join_Prune_Src*          join_prune_src_ptr;
      int                                 i, j;
      Boolean                             send_join_prune = OPC_FALSE;

      FIN (ip_pim_sm_join_prune_msg ());

      /* Fist check that PIM-SM is supported on the interface.    */
      if (intf_array [pkt_recvd_intf].pim_sm_status == OPC_FALSE)
            {
            /* The interface does not support PIM-SM. */

            op_pk_destroy (pimsm_pkptr);
            op_pk_destroy (ip_dgram_pkptr);

            FOUT;
            }

      /* Obtain the Join/Prune message from the PIM-SM packet     */
      op_pk_nfd_access (pimsm_pkptr, "message", &pim_sm_msg_ptr);
      join_prune_msg_ptr = (IpT_Pim_Sm_Join_Prune_Msg *) pim_sm_msg_ptr-
>msg_ds_ptr;

      if (ip_pim_sm_is_my_address (join_prune_msg_ptr-
>upstream_neighbor_addr) == OPC_TRUE)
            {
            /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
            if (LTRACE_PIM_SM_JOIN_PRUNE)
                  {
                  char                                          ip_addr_str
[IPC_ADDR_STR_LEN];
                  char                                          info0
[256], info1 [256];
                  char                                          msg0 [256];
                  ip_address_print (ip_addr_str, join_prune_msg_ptr-
>upstream_neighbor_addr);
                  sprintf (info0, "Upstream Neighbor Address      :
      %s", ip_addr_str);
                  sprintf (info1, "Number of Groups               :
      %d", join_prune_msg_ptr->num_grps);
                  sprintf (msg0, "Received a PIM-SM Join/Prune message on
interface, %d with: ", pkt_recvd_intf);
                  op_prg_odb_print_major (msg0, info0, info1, OPC_NIL);
                  }
#endif

      /* For each IP group in the list, process the Join and Prune list */
            for (i=0; i<join_prune_msg_ptr->num_grps; i++)
```

```
                  {
                  /* Obtain ith element from the Join/Prune list  */
                  join_prune_lelem_ptr = (IpT_Pim_Sm_Join_Prune_List_Elem *)
op_prg_list_access (join_prune_msg_ptr->join_prune_lptr, i);

                  /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
                  if (LTRACE_PIM_SM_JOIN_PRUNE)
                        {
                        char        info0 [256];
                        char        grp_addr_str [IPC_ADDR_STR_LEN];
                        ip_address_print (grp_addr_str, join_prune_lelem_ptr-
>grp_addr);
                        sprintf (info0, "IP Group Address         :
      %s", grp_addr_str);
                        op_prg_odb_print_major ("Processing Join/Prune list
in the PIM-SM Join/Prune message for the group: ", info0, OPC_NIL);
                        }
#endif

                  /* Process each source in the Join list   */
                  for (j=0; j<join_prune_lelem_ptr->num_join_src; j++)
                        {
                        /* Obtain the jth element from the Join list    */
                        join_prune_src_ptr = (IpT_Pim_Sm_Join_Prune_Src *)
op_prg_list_access (join_prune_lelem_ptr->join_src_lptr, j);

                        /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
                        if (LTRACE_PIM_SM_JOIN_PRUNE)
                              {
                              char
      ip_addr_str [IPC_ADDR_STR_LEN];
                              char
      info0 [256], info1 [256], info2 [256];
                              ip_address_print (ip_addr_str,
join_prune_src_ptr->src_addr);
                              sprintf (info0, "Source IP Address        :
      %s", ip_addr_str);
                              sprintf (info1, "WC-bit                   :
      %d", join_prune_src_ptr->wc_bit);
                              sprintf (info2, "RPT-bit                  :
      %d", join_prune_src_ptr->rpt_bit);
                              op_prg_odb_print_major ("Processing the
following Source element in the Join list: ", info0, info1, info2, OPC_NIL);
                              }
#endif

                        /* Get the route entry for this group     */
                        rte_entry_ptr = ip_pim_sm_mcast_rte_entry_get
(join_prune_lelem_ptr->grp_addr, join_prune_src_ptr->src_addr,

                  join_prune_src_ptr->wc_bit);

                        /* Print out information about the route entry that
was found.  */
#ifndef OPD_NO_DEBUG
```

```
                        ip_pim_sm_entry_exists_odb_print (rte_entry_ptr);
#endif

                        if ((join_prune_src_ptr->wc_bit == OPC_TRUE) &&
(join_prune_src_ptr->rpt_bit == OPC_TRUE))
                              {

                              if (rte_entry_ptr == OPC_NIL)
                                    {

                                    rte_entry_ptr =
ip_pim_sm_rte_entry_wc_create (join_prune_src_ptr->src_addr,

      join_prune_lelem_ptr->grp_addr);

                                    /* Add this route entry to the multicast
route table */
                  ip_pim_sm_mcast_rte_entry_add (rte_entry_ptr, OPC_TRUE);

                                    /* This case require a join/prune message
to be sent. */
                  send_join_prune = OPC_TRUE;
                                    }

                              if (rte_entry_ptr->in_intf_addr !=
inet_ipv4_address_get (ip_dgram_fdptr->src_addr))
                                    ip_pim_sm_oif_table_add (rte_entry_ptr,
pkt_recvd_intf, inet_ipv4_address_get (ip_dgram_fdptr->src_addr),
                                          join_prune_msg_ptr->hold_time,
OPC_FALSE);

      ip_pim_sm_all_spt_rte_entries_out_intf_add (rte_entry_ptr,

                  pkt_recvd_intf,

                  inet_ipv4_address_get (ip_dgram_fdptr->src_addr),

                  join_prune_msg_ptr->hold_time,

                  OPC_FALSE,

                  join_prune_lelem_ptr->prune_src_lptr);
                              }
                        else if ((join_prune_src_ptr->wc_bit == OPC_FALSE) &&
(join_prune_src_ptr->rpt_bit == OPC_FALSE))
                              {

                              if (rte_entry_ptr == OPC_NIL)
                                    {

                                    rp_ip_addr = ip_pim_sm_rp_addr_get
(rp_hash_table_ptr, rp_lptr, join_prune_lelem_ptr->grp_addr,
bsr_hash_mask_length);

                        /* If RP information is not found, its an error */
                                    if (rp_ip_addr == IPC_ADDR_INVALID)
                                          {
```

```
                                        /* Report a log message */

      ipnl_protwarn_mcast_rp_unknown_log_add (join_prune_lelem_ptr->grp_addr,
ip_module_data_ptr->node_id);
                                        }

                                    rpt_rte_entry_ptr =
ip_pim_sm_mcast_rte_entry_get (join_prune_lelem_ptr->grp_addr, rp_ip_addr,
OPC_TRUE);

      rte_entry_ptr = ip_pim_sm_rte_entry_spt_create (join_prune_src_ptr-
>src_addr, join_prune_lelem_ptr->grp_addr,
                                        rpt_rte_entry_ptr, OPC_FALSE);
                                    if (rpt_rte_entry_ptr != OPC_NIL)
                                        {
      op_prg_list_elems_copy (rpt_rte_entry_ptr->out_intf_table_lptr,
rte_entry_ptr->out_intf_table_lptr);
                                        }

                  /* Add this route entry to the multicast route table  */
                                    ip_pim_sm_mcast_rte_entry_add
(rte_entry_ptr, OPC_FALSE);

            /* This case requires sending a join/prune message.         */
                                    send_join_prune = OPC_TRUE;
                                    }
                               else
                                    {

                                    if (rte_entry_ptr->rpt_flag == OPC_TRUE)
                                        {
                                        /* Clear the RPT-bit    */
                                        ip_pim_sm_entry_clear_rpt_bit
(rte_entry_ptr, OPC_FALSE);

                  /* This case requires sending a join/prune message    */
                                        send_join_prune = OPC_TRUE;
                                        }
                                    }

                               if (inet_ipv4_address_get (ip_dgram_fdptr-
>src_addr) != rte_entry_ptr->in_intf_addr)
                                    ip_pim_sm_oif_table_add (rte_entry_ptr,
pkt_recvd_intf, inet_ipv4_address_get (ip_dgram_fdptr->src_addr),
                                        join_prune_msg_ptr->hold_time,
OPC_FALSE);
                               }
                          if (send_join_prune == OPC_TRUE)
                               ip_pim_sm_send_join_prune_msg (rte_entry_ptr,
OPC_TRUE);

                          /* Reset the flag to FALSE.              */
                          send_join_prune = OPC_FALSE;
                          }

                  /* Process each source in the Prune list  */
                  for (j=0; j<join_prune_lelem_ptr->num_prune_src; j++)
```

```
                        {
                        /* Obtain the jth element from the Prune list   */
                        join_prune_src_ptr = (IpT_Pim_Sm_Join_Prune_Src *)
op_prg_list_access (join_prune_lelem_ptr->prune_src_lptr, j);

                        /* Get the route entry for this group     */
                        rte_entry_ptr = ip_pim_sm_mcast_rte_entry_get
(join_prune_lelem_ptr->grp_addr, join_prune_src_ptr->src_addr,

                  join_prune_src_ptr->wc_bit);

                        /* Generate trace messages    */
#ifndef OPD_NO_DEBUG
                        if (LTRACE_PIM_SM_JOIN_PRUNE)
                              {
                              char
      ip_addr_str [IPC_ADDR_STR_LEN];
                              char
      info0 [256], info1 [256], info2 [256];
                              ip_address_print (ip_addr_str,
join_prune_src_ptr->src_addr);
                              sprintf (info0, "Source IP Address        :
      %s", ip_addr_str);
                              sprintf (info1, "WC-bit                   :
      %d", join_prune_src_ptr->wc_bit);
                              sprintf (info2, "RPT-bit                  :
      %d", join_prune_src_ptr->rpt_bit);
                              op_prg_odb_print_major ("Processing the
following Source element in the Prune list: ", info0, info1, info2, OPC_NIL);
                              }
#endif

#ifndef OPD_NO_DEBUG
                        ip_pim_sm_entry_exists_odb_print (rte_entry_ptr);
#endif

                        if ((join_prune_src_ptr->wc_bit == OPC_FALSE) &&
(join_prune_src_ptr->rpt_bit == OPC_TRUE))
                              {
      if (rte_entry_ptr == OPC_NIL)
                                    {
                                    /* Get the RP address for the group */
                                    rp_ip_addr = ip_pim_sm_rp_addr_get
(rp_hash_table_ptr, rp_lptr, join_prune_lelem_ptr->grp_addr,
bsr_hash_mask_length);

                        /* If RP information is not found, its an error */
                                    if (rp_ip_addr == IPC_ADDR_INVALID)
                                          {
      ipnl_protwarn_mcast_rp_unknown_log_add (join_prune_lelem_ptr->grp_addr,
ip_module_data_ptr->node_id);
                                          }
                        /* Get the (*, G) entry */
                                    rpt_rte_entry_ptr =
ip_pim_sm_mcast_rte_entry_get (join_prune_lelem_ptr->grp_addr, rp_ip_addr,
OPC_TRUE);
```

```
                                    if (rpt_rte_entry_ptr != OPC_NIL)
                                          {
                                          rte_entry_ptr =
ip_pim_sm_rte_entry_spt_create (join_prune_src_ptr->src_addr,

                                    join_prune_lelem_ptr->grp_addr,

                                    rpt_rte_entry_ptr,

                                    OPC_FALSE);

                                          ip_pim_sm_entry_set_rpt_bit
(rte_entry_ptr, OPC_FALSE);

                                          rte_entry_ptr->in_intf =
rpt_rte_entry_ptr->in_intf;
                                          rte_entry_ptr->in_intf_addr =
ip_address_copy(rpt_rte_entry_ptr->in_intf_addr);
                                          op_prg_list_elems_copy
(rpt_rte_entry_ptr->out_intf_table_lptr, rte_entry_ptr->out_intf_table_lptr);

                  /* Add this route entry to the multicast route table  */
                                          ip_pim_sm_mcast_rte_entry_add
(rte_entry_ptr, OPC_FALSE);
                                          }
                                    }
                              }

                        if (rte_entry_ptr != OPC_NIL)
                              {

                              if (op_prg_list_size (rte_entry_ptr-
>out_intf_table_lptr) == 0)
                                    {
                        /* Check if the data timer should be reset.     */
                                    if ((join_prune_src_ptr->wc_bit ==
OPC_FALSE) && (join_prune_src_ptr->rpt_bit == OPC_TRUE))
                                          {

            ip_pim_sm_reset_data_timer (rte_entry_ptr,
LTRACE_PIM_SM_JOIN_PRUNE);
                                          }

            /* Nothing more needs to be done.  Just continue.           */
                                    continue;
                                    }

ip_pim_sm_oif_table_remove (rte_entry_ptr->out_intf_table_lptr,
pkt_recvd_intf, OPC_FALSE);

                              if ((join_prune_src_ptr->wc_bit == OPC_TRUE) &&
(join_prune_src_ptr->rpt_bit == OPC_TRUE))
                                    {

      ip_pim_sm_all_spt_rte_entries_out_intf_remove (join_prune_lelem_ptr-
>grp_addr, pkt_recvd_intf, OPC_FALSE);
                                    }
```

```
                                if (op_prg_list_size (rte_entry_ptr-
>out_intf_table_lptr) == 0)
                                        {

                                        ip_pim_sm_send_join_prune_msg
(rte_entry_ptr, OPC_FALSE);

                                        if (((join_prune_src_ptr->wc_bit ==
OPC_FALSE) && (join_prune_src_ptr->rpt_bit == OPC_TRUE)) &&
                                                ((rte_entry_ptr->wc_flag ==
OPC_FALSE) && (rte_entry_ptr->rpt_flag == OPC_FALSE)))
                                                {
                        /* Set the RPT bit to TRUE for this route entry.      */
                                                ip_pim_sm_entry_set_rpt_bit
(rte_entry_ptr, OPC_FALSE);

                        /* Set the SPT bit to FALSE for this route entry.     */
                                                ip_pim_sm_entry_clear_spt_bit
(rte_entry_ptr, OPC_FALSE);
                                                }

                        /* Remove the route entry from the multicast route
table,only if its is not a (S, G)RPT-bit entry  */
                                        if (!((rte_entry_ptr->wc_flag ==
OPC_FALSE) && (rte_entry_ptr->rpt_flag == OPC_TRUE)))
                                                {
#ifndef OPD_NO_DEBUG
                                                if (LTRACE_PIM_SM_JOIN_PRUNE)
                                                        {
                                                        op_prg_odb_print_major ("The
Out Interface table size of the route entry became zero.", OPC_NIL);
                                                        }
#endif
ip_pim_sm_mcast_rte_entry_remove (rte_entry_ptr->grp_addr, rte_entry_ptr-
>src_addr, rte_entry_ptr->wc_flag);

                                                continue;
                                                }
                                        }

                                if ((join_prune_src_ptr->wc_bit == OPC_FALSE)
&& (join_prune_src_ptr->rpt_bit == OPC_TRUE))
                                        {
                                        ip_pim_sm_reset_data_timer
(rte_entry_ptr, LTRACE_PIM_SM_JOIN_PRUNE);
                                        }
                                }
                        }
                }
            }

      /* We no longer need the PIM-SM packet and the  */
      /* IP datagram. Destroy them  */
      op_pk_destroy (pimsm_pkptr);
      op_pk_destroy (ip_dgram_pkptr);
```

```
	FOUT;
	}

static void
mip_mn_tunneled_pk_stat_write (Packet*	pk_ptr)
	{
	/** PURPOSE: Write statistic for tunneld packets received.**/
	FIN (mip_mn_tunneled_pk_stat_write (pk_ptr));

	/* Write the stats. */
	op_stat_write (tunneled_pk_rcvd_sec_sh, 1.0);
	op_stat_write (tunneled_bit_rcvd_sec_sh, op_pk_total_size_get
(pk_ptr));
	op_stat_write (tunneled_pk_rcvd_sec_sh, 0.0);
	op_stat_write (tunneled_bit_rcvd_sec_sh, 0.0);

	FOUT;
	}

static void
mip_mn_agent_solicit_pk_send (void)
	{
	Packet*		solicit_pkptr;
	double		solicit_interval;

	/** PURPOSE: Send the ICMP agent solicitation packet.**/
	/** REQUIRES: none.	**/
	/** EFFECTS: Packet will be given to IP to handle.**/
	FIN (mip_mn_agent_solicit_pk_send (void));

	/* Time to send out the solicitation. */
	solicit_pkptr = op_pk_create_fmt ("mobile_ip_irdp_solicit");

	/* Send the packet out. */
	module_data->ip_ptc_mem.child_pkptr = mip_sup_irdp_pkt_encapsulate
		(solicit_pkptr, home_address, subnet_bcast_addr,
IcmpC_Type_IRDP_Sol);

	/* Record some stats. */
	op_stat_write (irdp_sent_pkts_sh, 1.0);
	op_stat_write (irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
	op_stat_write (g_irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
	op_stat_write (g_irdp_sent_bits_sh, 0.0);

	/* Invoke IP to handle the packet. */
	op_pro_invoke (proc_info_struct_ptr->ip_phndl, OPC_NIL);

	/* Schedule the next transmission. */
	if (++solicit_count > 3)
		{
		solicit_interval = MipC_MN_Solicit_Min_Interval * pow (2.0,
(double) (solicit_count - 3));

		if (solicit_interval > MipC_MN_Solicit_Max_Interval)
```

```
                  {
                  solicit_interval = MipC_MN_Solicit_Max_Interval;
                  }
            }
      else
            {
            solicit_interval = MipC_MN_Solicit_Min_Interval;
            }

      solicit_timer_ehndl = op_intrpt_schedule_self (op_sim_time () +
solicit_interval, MipC_MN_Timer_Solicit);

      FOUT;
      }

static void
mip_mn_agent_cache_update (MipT_MN_Agent_Info* agent_info, InetT_Address
      new_agent_address,
      double life_time, int pref_level, MipT_Invocation_Info*
invoke_info_ptr)
      {
      /** PURPOSE: Update Agent cache based on rule. (higher pref level or
same or lower if the current one expired) **/
      /** REQUIRES: new FA address and its lifetime and pref level.     **/
      /** EFFECTS: the cache will be updated if it matches the criteria.**/
      FIN (mip_mn_agent_cache_update (agent_info, new_agent_address,
life_time, pref_level, invoke_info_ptr));

      if ((pref_level >= agent_info->pref_level) ||
            (agent_info->lifetime < op_sim_time ()) ||
            inet_address_equal (agent_info->address, new_agent_address))
            {
            /* Update the FA cache information. */
            agent_info->address = new_agent_address;
            agent_info->lifetime = op_sim_time () + life_time;
            agent_info->pref_level = pref_level;
            agent_info->incoming_intf_ptr = ip_rte_intf_tbl_access
                  (module_data, invoke_info_ptr->rte_info_ici_ptr-
>intf_recvd_index);
            }

      FOUT;
      }

static void
mip_mn_ip_pk_handle (MipT_Invocation_Info* invoke_info_ptr)
      {
      Prohandle   tmp_phndl;
      FIN (mip_mn_ip_pk_handle (invoke_info_ptr));

      /* See if there are any visiting MN with the address. */
      if (mip_sup_visitor_search_by_addr (proc_info_struct_ptr-
>node_visitor_list_lptr,
            invoke_info_ptr->rte_info_ici_ptr->dest_addr, &tmp_phndl) ==
OPC_COMPCODE_SUCCESS)
            {
            /*  will let the FA agent to handle this packet. */
```

```
            op_pro_invoke (tmp_phndl, invoke_info_ptr);
            }
      else
            {
            if ((inet_address_range_check (invoke_info_ptr->rte_info_ici_ptr-
>dest_addr,
                  &proc_info_struct_ptr->intf_info_ptr->inet_addr_range)) &&
                  !inet_address_equal (invoke_info_ptr->rte_info_ici_ptr-
>dest_addr, home_address) &&
                  !inet_address_equal (invoke_info_ptr->rte_info_ici_ptr-
>dest_addr, ha_address))
                  {
                  /* This address falls in the same range but not for me.
Destroy packet. */
                  mip_sup_pk_cleanup (invoke_info_ptr);
                  }
            else
                  {
                  if (current_roaming_intf)
                        {
                        if (current_roaming_intf == invoke_info_ptr-
>interface_ptr)
                              {
                              /* This is going out on the roaming interface.
Send to the current agent. */
                              if (inet_address_valid (agent_address))
                                    invoke_info_ptr->rte_info_ici_ptr-
>next_addr = agent_address;
                              }
                        else
                              {
                              /* Unknown address. Let IP handle it
generically. */
                              }
                        }
                  }
            }

      FOUT;
      }

static void
mip_mn_ad_packet_parse (Packet* pkptr, int* h, int* f, InetT_Address*
agent_addr,
      int* lifetime, int* pref)
      {
      IpT_Address       tmp_addr;

      /* Helper macro to parse information from the advertisement packet
received. */
      FIN (mip_mn_ad_packet_parse (...));

      /* Access the information from the packet received. */
      op_pk_nfd_access (pkptr, "H", h);
      op_pk_nfd_access (pkptr, "F", f);
      op_pk_nfd_access (pkptr, "Agent Address", &tmp_addr);
      op_pk_nfd_access (pkptr, "lifetime", lifetime);
```

```
      op_pk_nfd_access (pkptr, "Preference Level", pref);

      /* Convert the V4 address to v6 for internal reference. */
      *agent_addr = inet_address_from_ipv4_address_create (tmp_addr);

      FOUT;
      }

static void
mip_mn_agent_solicit_pk_send_adv (void) /*Me: Sending Mobile IP to join in
advance */
      {
      Packet*           solicit_pkptr_adv;
      double            solicit_interval;

      /** PURPOSE: Send the ICMP agent solicitation packet.**/
      /** REQUIRES: none.     **/
      /** EFFECTS: Packet will be given to IP to handle.**/
      FIN (mip_mn_agent_solicit_pk_send (void));

      /* Time to send out the solicitation. */

      usleep(10000000); // 10 secs
      solicit_pkptr_adv = op_pk_create_fmt ("mobile_ip_irdp_solicit");

      /* Send the packet out. */
      module_data->ip_ptc_mem.child_pkptr = mip_sup_irdp_pkt_encapsulate
            (solicit_pkptr_adv, home_address, subnet_bcast_addr,
IcmpC_Type_IRDP_Sol);

      /* Record some stats. */
      op_stat_write (irdp_sent_pkts_sh, 1.0);
      op_stat_write (irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
      op_stat_write (g_irdp_sent_bits_sh, op_pk_total_size_get (module_data-
>ip_ptc_mem.child_pkptr));
      op_stat_write (g_irdp_sent_bits_sh, 0.0);

      /* Invoke IP to handle the packet. */
      op_pro_invoke (proc_info_struct_ptr->ip_phndl, OPC_NIL);

      /* Schedule the next transmission. */
      if (++solicit_count > 3)
            {
            solicit_interval = MipC_MN_Solicit_Min_Interval * pow (2.0,
(double) (solicit_count - 3));

            if (solicit_interval > MipC_MN_Solicit_Max_Interval)
                  {
                  solicit_interval = MipC_MN_Solicit_Max_Interval;
                  }
            }
      else
            {
            solicit_interval = MipC_MN_Solicit_Min_Interval;
            }
```

```
      solicit_timer_ehndl = op_intrpt_schedule_self (op_sim_time () +
solicit_interval, MipC_MN_Timer_Solicit);

      FOUT;
      }

//me

/* End of Function Block */

/* Undefine optional tracing in FIN/FOUT/FRET */
/* The FSM has its own tracing code and the other */
/* functions should not have any tracing.         */
#undef FIN_TRACING
#define FIN_TRACING

#undef FOUTRET_TRACING
#define FOUTRET_TRACING

#if defined (__cplusplus)
extern "C" {
#endif
      void mobile_ip_mn (OP_SIM_CONTEXT_ARG_OPT);
      VosT_Obtype _op_mobile_ip_mn_init (int * init_block_ptr);
      void _op_mobile_ip_mn_diag (OP_SIM_CONTEXT_ARG_OPT);
      void _op_mobile_ip_mn_terminate (OP_SIM_CONTEXT_ARG_OPT);
      VosT_Address _op_mobile_ip_mn_alloc (VosT_Obtype, int);
      void _op_mobile_ip_mn_svar (void *, const char *, void **);


#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif

/* Process model interrupt handling procedure */

void
mobile_ip_mn (OP_SIM_CONTEXT_ARG_OPT)
      {
#if !defined (VOSD_NO_FIN)
      int _op_block_origin = 0;
#endif
      FIN_MT (mobile_ip_mn ());

            {
            /* Temporary Variables */
            Boolean     is_ip_pk = OPC_FALSE;
            Boolean     is_ad_reception = OPC_FALSE;
            Boolean     is_ha_ad_reception = OPC_FALSE;
            Boolean     is_fa_ad_reception = OPC_FALSE;
            Boolean     is_solicitation_time = OPC_FALSE;
            Boolean     is_valid_fa_candidate = OPC_FALSE;
            Boolean     is_ha_timeout = OPC_FALSE;
            Boolean     is_fa_timeout = OPC_FALSE;
            Boolean     is_timeout = OPC_FALSE;
            Boolean     is_retry = OPC_FALSE;
            Boolean     is_ha_reg_success = OPC_FALSE;
```

```
            Boolean     is_fa_reg_success = OPC_FALSE;
            Boolean     is_reg_pk = OPC_FALSE;
            Boolean     is_out_of_retries = OPC_FALSE;
            Boolean     is_invalid_reply = OPC_FALSE;
            Boolean     is_reregister = OPC_FALSE;
            Boolean     is_switch_fa = OPC_FALSE;

            Objid mip_reg_cfg_objid;
            char  ha_address_str[64];
            int         h_bit,f_bit, reply_code, lifetime_grant, tmp_reg_id,
                              irdp_lifetime, inv_mode, intrpt_code,
pref_level;
            Packet                  *irdp_pkptr, *encap_pk_ptr;
            InetT_Address           tmp_agent_address;
            MipT_Invocation_Info*   invoke_info_ptr;
            Ici*  reg_ici_ptr;
            /* End of Temporary Variables */


            FSM_ENTER ("mobile_ip_mn")

            FSM_BLOCK_SWITCH
                  {
            /*---------------------------------------------------------*/
                  /** state (Init) enter executives **/
                  FSM_STATE_ENTER_FORCED_NOLABEL (0, "Init", "mobile_ip_mn
[Init enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Init enter
execs]", state0_enter_exec)
                        {
                        /* Access the parent memory. */
                        proc_info_struct_ptr = (MipT_Proc_Info*)
op_pro_parmem_access ();

                        /* Get the type of agent I am. (MN or MR)*/
                        mip_node_type = proc_info_struct_ptr->node_type;

                        /* Access the module wide memory. */
                        module_data = (IpT_Rte_Module_Data*)
op_pro_modmem_access ();

                        /* Keep the subnet bcast address for later use. */
                        subnet_bcast_addr = inet_rte_intf_broadcast_addr_get
                              (proc_info_struct_ptr->intf_info_ptr,
InetC_Addr_Family_v4);

                        /* Parse the home agent interface address. */
                        op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Home Agent IP Address",
                              &ha_address_str);
                        ha_address = inet_address_create (ha_address_str,
InetC_Addr_Family_v4);
                        if (inet_address_equal (ha_address,
InetI_Invalid_v4_Addr))
                              {
                              op_sim_end ("An invalid address was configured
as Home Agent.","","","");
```

```
                              }

                        /* local interface address. */
                        home_address = inet_rte_intf_addr_get
(proc_info_struct_ptr->intf_info_ptr, InetC_Addr_Family_v4);
                        agent_address = InetI_Invalid_v4_Addr;

                        /* Parse the registration related information. */
                        op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Registration Parameters",
                              &mip_reg_cfg_objid);
                        mip_reg_cfg_objid = op_topo_child (mip_reg_cfg_objid,
OPC_OBJTYPE_GENERIC, 0);

                        op_ima_obj_attr_get (mip_reg_cfg_objid, "Interval",
&(reg_info.interval));
                        op_ima_obj_attr_get (mip_reg_cfg_objid, "Retry",
&(reg_info.retry));
                        op_ima_obj_attr_get (mip_reg_cfg_objid, "Lifetime
Request", &(reg_info.req_lifetime));

                        /* Initialize state vars. */
                        reg_id = 0;
                        retry_counter = 0;
                        latest_fa_info.address = InetI_Invalid_v4_Addr;
                        latest_fa_info.lifetime = 0.0;
                        latest_fa_info.pref_level = 0;
                        current_roaming_intf = OPC_NIL;

                        /* Register some stats. */
                        tunneled_bit_rcvd_sec_sh = op_stat_reg ("Mobile
IP.Tunneled Traffic Received (bits/sec)", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);
                        tunneled_pk_rcvd_sec_sh = op_stat_reg ("Mobile
IP.Tunneled Traffic Received (packets/sec)", OPC_STAT_INDEX_NONE,
OPC_STAT_LOCAL);

                        /* Find out if  need to send out solicitation when
lost. */
                        op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Agent Solicitation",
                              &solicitation);

                        /* See if I am configured on a loopback interface. */
                        loopback_intf = ip_rte_intf_is_loopback
(proc_info_struct_ptr->intf_info_ptr);

                        /* Schedule interrupt to send solicitation if
enabled. */
                        if (solicitation)
                              {
                              if (loopback_intf)
                                    {
                                    /*  will not solicitate if on a loopback
interface. */
                                    op_sim_end ("Mobile IP currently cannot
support solicitation when configured on loopback interface.",
```

```
                                        OPC_NIL, OPC_NIL, OPC_NIL);
                                  }

                              /* Schedule an interrupt for the first
solicitation. */
                              solicit_count = 0;
                              solicit_timer_ehndl = op_intrpt_schedule_self
(mip_sup_activation_time_calculate (), MipC_MN_Timer_Solicit);

                              /* Register some stats. */
                              irdp_sent_pkts_sh = op_stat_reg ("Mobile
IP.IRDP Traffic Sent (packets)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                              irdp_sent_bits_sh = op_stat_reg ("Mobile
IP.IRDP Traffic Sent (bits)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
                              g_irdp_sent_bits_sh = op_stat_reg ("Mobile
IP.IRDP Traffic Sent (bits/sec)", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
                              }

                        /* Find out if need to ask for simultaneous binding.
*/
                        op_ima_obj_attr_get (proc_info_struct_ptr->cfg_objid,
"Simultaneous Binding Support", &simultaneous_binding);

                        /* Cache the node objid info. */
                        node_objid = op_topo_parent (op_id_self ());

                        /* To handle default gateway. */
                        default_gateway = OPC_FALSE;

                        /* Animation. */
                        if (op_sim_anim ())
                              {
                              /* Initialize the view. */
                              mip_sup_prepare_animation ();
                              }

                        }
                        FSM_PROFILE_SECTION_OUT (state0_enter_exec)

                  /** state (Init) exit executives **/
                  FSM_STATE_EXIT_FORCED (0, "Init", "mobile_ip_mn [Init exit
execs]")


                  /** state (Init) transition processing **/
                  FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "",
"Init", "Lost", "tr_-1", "mobile_ip_mn [Init -> Lost : default / ]")
            /*---------------------------------------------------------*/

                  /** state (Lost) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (1, "Lost", state1_enter_exec,
"mobile_ip_mn [Lost enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Lost enter
execs]", state1_enter_exec)
                        {
                        /* Update the status for debugging. */
```

```
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Lost,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state1_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (3,"mobile_ip_mn")

                  /** state (Lost) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (1, "Lost", "mobile_ip_mn [Lost
exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Lost exit
execs]", state1_exit_exec)
                        {
                        /* Who invoked me? */
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              is_solicitation_time = OPC_TRUE;

                              if (op_intrpt_code () == MipC_MN_Timer_Solicit)
                                    {
                                    mip_mn_agent_solicit_pk_send ();
                                    }
                              }
                        else
                              {
                              /* See if  are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                        /*  have an invocation from IP.  What kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
                        /*  have an IRDP packet.  But which kind thou? */
                                                if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                      {
                        /*   do not want to deal with this packet. */
                                                      mip_sup_pk_cleanup
(invoke_info_ptr);

                                                      is_ip_pk = OPC_TRUE;
                                                      break;
                                                      }

                        /* This must be an advertisement from an agent. */
                                                op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
```

```
                                                mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffice. */
                                                if
(inet_address_equal(tmp_agent_address, ha_address))
                                                      {
                                                      if (loopback_intf)
                                                            {
                                          /* This is not supported. */
                                                            op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                OPC_NIL, OPC_NIL, OPC_NIL);
                                                            }

                                          /* Ad from my home agent. */
                                                is_ad_reception = OPC_TRUE;

                           /* Initialize counter before start reg process. */
                                                      retry_counter = 0;

                        /* Update the latest ha info structure for later. */

      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                            (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                          /* Deregister with HA. */
                                                      mip_mn_register (0,
latest_ha_info, OPC_TRUE);

                                          /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                      }
                                                else
                                                      {
                                                      if (f_bit)
                                                            {
                                                /* A foreign agent ad. */
                                                is_ad_reception = OPC_TRUE;

                           /* Initialize counter before start reg process. */
                                                retry_counter = 0;

                           /* Update the latest fa info structure for later. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                           (double) irdp_lifetime, pref_level, invoke_info_ptr);

                                    /* Register with HA using the FA address. */
      mip_mn_register (reg_info.req_lifetime, latest_fa_info, OPC_FALSE);

                                          /* Update timer for HA timeout. */
```

```
      mip_mn_agent_timer_update (latest_fa_info.lifetime);
                                                              }
                                                         else
                                                              {
/* cannot do anything with this agent who is only HA for other group. */
                                                    is_ip_pk = OPC_TRUE;
                                                              }
                                                         }
                                       /* Clean up solicitation if any. */
      if (solicitation && is_ad_reception)
                                                         {
      op_ev_cancel_if_pending (solicit_timer_ehndl);
                                                         }

                                                    /* Clean up. */
      mip_sup_pk_cleanup (invoke_info_ptr);
      op_pk_destroy (irdp_pkptr);

      break;
                                                    }

                                               case MipC_Invoke_Type_Tunnel_Check:
                                                    {
                                                    is_ip_pk = OPC_TRUE;

                  /*   should try to decapsulte packet if in MR mode. */
                                 if (mip_node_type == MipC_Node_Type_MR)
                                                         {
      /*   will check first if this packet is tunneling other IP packet */
                                                         if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
                                                              ==
OPC_COMPCODE_SUCCESS)
                                                              {
                                               /* Sanity check on the packet */

                                    /* Invoke IP delayed to handle the packet */

      mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

                           /* Write stats for the received tunneled packet. */

      mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

                  /* Let IP caller know that   are handling the packet */

      mip_sup_pk_cleanup (invoke_info_ptr);
                                                              }
                                                         }

                                                    break;
                                                    }

                                               case MipC_Invoke_Type_IP_Datagram:
                                                    {
                                                    is_ip_pk = OPC_TRUE;
```

```
/* The destination address that is going out on this interface when in MR
mode should be forwarded to either HA or FA.  Whoever already registered or
trying to. */
                                    if (mip_node_type == MipC_Node_Type_MR)
                                                {
                                /* Handle packet if I am a MR. */
                                    mip_mn_ip_pk_handle (invoke_info_ptr);
                                                }
                                          else
                                                {
                              /* Unknown address.  Send to the current agent. */
                                    if (inet_address_valid (agent_address))
                                                      invoke_info_ptr-
>rte_info_ici_ptr->next_addr = agent_address;
                                                }

                                          break;
                                          }
                                     } /* switch (invoke_info_ptr-
>invocation_type) */
                                    }
                               else
                                    {
                                    /* Registration arrival. */
                                    op_ici_destroy (op_intrpt_ici ());
                                    is_reg_pk = OPC_TRUE;
                                    }
                               }
                         }
                         FSM_PROFILE_SECTION_OUT (state1_exit_exec)

                   /** state (Lost) transition processing **/
                   FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Lost trans
conditions]", state1_trans_conds)
                   FSM_INIT_COND (IP_PK)
                   FSM_TEST_COND (SOLICITATION_TIME)
                   FSM_TEST_COND (REG_PK)
                   FSM_TEST_COND (AD_RECEPTION)
                   FSM_TEST_LOGIC ("Lost")
                   FSM_PROFILE_SECTION_OUT (state1_trans_conds)

                   FSM_TRANSIT_SWITCH
                         {
                         FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;,
"IP_PK", "", "Lost", "Lost", "tr_8", "mobile_ip_mn [Lost -> Lost : IP_PK /
]")
                         FSM_CASE_TRANSIT (1, 1, state1_enter_exec, ;,
"SOLICITATION_TIME", "", "Lost", "Lost", "tr_32", "mobile_ip_mn [Lost -> Lost
: SOLICITATION_TIME / ]")
                         FSM_CASE_TRANSIT (2, 1, state1_enter_exec, ;,
"REG_PK", "", "Lost", "Lost", "tr_36", "mobile_ip_mn [Lost -> Lost : REG_PK /
]")
                         FSM_CASE_TRANSIT (3, 4, state4_enter_exec, ;,
"AD_RECEPTION", "", "Lost", "Pending registration", "tr_1", "mobile_ip_mn
[Lost -> Pending registration : AD_RECEPTION / ]")
                         }
            /*---------------------------------------------------------*/
```

```
                  /** state (At home) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (2, "At home", state2_enter_exec,
"mobile_ip_mn [At home enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [At home enter
execs]", state2_enter_exec)
                        {
                        /* Update the status for debugging. */
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Home,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state2_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (5,"mobile_ip_mn")


                  /** state (At home) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (2, "At home", "mobile_ip_mn [At
home exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [At home exit
execs]", state2_exit_exec)
                        {
                        /* Who invoked me? */
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              is_ha_timeout = OPC_TRUE;

                              if (MIP_TRACE)
                                    {
                                    op_prg_odb_print_major ("Trying
reregister for timeout occurred.", OPC_NIL);
                                    }
                              }
                        else
                              {
                              /* See if   are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                        /*   have an invocation from IP.  What kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
                        /*   have an IRDP packet.  But which kind thou? */
                                                if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                      {
                        /*   do not want to deal with this packet. */
```

```
                                                        mip_sup_pk_cleanup
(invoke_info_ptr);

                                                        is_ip_pk = OPC_TRUE;
                                                        break;
                                                        }

                        /* This must be an advertisement from an agent. */
                                                is_ad_reception = OPC_TRUE;

                    /*   have an agent advertisement.  But which kind thou? */
                                                op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
                                                mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffice. */
                                                if
(inet_address_equal(tmp_agent_address, ha_address))
                                                        {
                                                        if (loopback_intf)
                                                              {
                                    /* This is not supported. */
                                                        op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                                    OPC_NIL,
OPC_NIL, OPC_NIL);
                                                        }

                                            /* Ad from my home agent. */

                        /* Update the latest ha info structure for later. */

      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                        (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                    /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                        }
                                                else
                                                        {
                                                        if (f_bit)
                                                              {
                                                /* A foreign agent ad. */

                        /* Update the latest fa info structure for later. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                                                                  (double)
irdp_lifetime, pref_level, invoke_info_ptr);
                                                        }
                                                else
                                                        {
```

```
/* cannot do anything with this agent who is only HA for other group. */
                                                                    }
                                                                }

                                                    /* Clean up. */
                                                    mip_sup_pk_cleanup
(invoke_info_ptr);
                                                    op_pk_destroy (irdp_pkptr);

                                                    break;
                                                    }

                                                case
MipC_Invoke_Type_Tunnel_Check:
                                                    {
                                                    is_ip_pk = OPC_TRUE;

                        /*   should try to decapsulte packet if in MR mode. */
                                                    if (mip_node_type ==
MipC_Node_Type_MR)
                                                        {
            /*   will check first if this packet is tunneling other IP packet */
                                                        if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
                                                            ==
OPC_COMPCODE_SUCCESS)
                                                            {
                                        /* Sanity check on the packet */

                                    /* Invoke IP delayed to handle the packet */

            mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

            /* Write stats for the received tunneled packet. */

            mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

                  /* Let IP caller know that   are handling the packet */

            mip_sup_pk_cleanup (invoke_info_ptr);
                                                            }
                                                        }

                                                    break;
                                                    }

                                               case MipC_Invoke_Type_IP_Datagram:
                                                    {
                                                    is_ip_pk = OPC_TRUE;

                                                    /* The destination address
that is going out on this interface when in MR mode should be forwarded to
either HA or FA.  Whoever already registered or trying to. */
                                        if (mip_node_type == MipC_Node_Type_MR)
                                                        {
                                                  /* Handle packet if I am a MR. */
                                        mip_mn_ip_pk_handle (invoke_info_ptr);
```

```
                                                }
                                          else
                                                {
                        /* Unknown address.  Send to the current agent. */
            invoke_info_ptr->rte_info_ici_ptr->next_addr = agent_address;
                                                }

                                          break;
                                          }
                                    }/* switch (invoke_info_ptr-
>invocation_type) */
                              }
                        else
                              {
                              /* Registration arrival. */
                              op_ici_destroy (op_intrpt_ici ());
                              is_reg_pk = OPC_TRUE;
                              }
                        }
                  }
                  FSM_PROFILE_SECTION_OUT (state2_exit_exec)

            /** state (At home) transition processing **/
            FSM_PROFILE_SECTION_IN ("mobile_ip_mn [At home trans
conditions]", state2_trans_conds)
            FSM_INIT_COND (HA_TIMEOUT)
            FSM_TEST_COND (IP_PK)
            FSM_TEST_COND (AD_RECEPTION)
            FSM_TEST_COND (REG_PK)
            FSM_TEST_LOGIC ("At home")
            FSM_PROFILE_SECTION_OUT (state2_trans_conds)

            FSM_TRANSIT_SWITCH
                  {
                  FSM_CASE_TRANSIT (0, 5, state5_enter_exec, ;,
"HA_TIMEOUT", "", "At home", "Check FA cache", "tr_5", "mobile_ip_mn [At home
-> Check FA cache : HA_TIMEOUT / ]")
                  FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;,
"IP_PK", "", "At home", "At home", "tr_9", "mobile_ip_mn [At home -> At home
: IP_PK / ]")
                  FSM_CASE_TRANSIT (2, 2, state2_enter_exec, ;,
"AD_RECEPTION", "", "At home", "At home", "tr_26", "mobile_ip_mn [At home ->
At home : AD_RECEPTION / ]")
                  FSM_CASE_TRANSIT (3, 2, state2_enter_exec, ;,
"REG_PK", "", "At home", "At home", "tr_35", "mobile_ip_mn [At home -> At
home : REG_PK / ]")
                  }
            /*---------------------------------------------------------*/

            /** state (Away) enter executives **/
            FSM_STATE_ENTER_UNFORCED (3, "Away", state3_enter_exec,
"mobile_ip_mn [Away enter execs]")
                  FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Away enter
execs]", state3_enter_exec)
                  {
                  /* Update the status for debugging. */
```

```
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Foreign,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state3_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (7,"mobile_ip_mn")

                  /** state (Away) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (3, "Away", "mobile_ip_mn [Away
exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Away exit
execs]", state3_exit_exec)
                        {
                        /* Who invoked me? */
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              intrpt_code = op_intrpt_code ();
                              switch (intrpt_code)
                                    {
                                    case MipC_MN_Timer_Agent:
                                          {
                                          is_fa_timeout = OPC_TRUE;
                                          break;
                                          }

                                      case MipC_MN_Timer_Rereg:
                                          {
                                          is_reregister = OPC_TRUE;
                                          break;
                                          }
                                    }

                              if (MIP_TRACE)
                                    {
                                    op_prg_odb_print_major ("Trying
reregister for timeout occurred.", OPC_NIL);
                                    }
                              }
                        else
                              {
                              /* See if   are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                                    /*   have an invocation from IP.  What
kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
```

```
                /*   have an IRDP packet.  But which kind thou? */
                                        if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                {
                        /*   do not want to deal with this packet. */
                                                mip_sup_pk_cleanup
(invoke_info_ptr);

                                                is_ip_pk = OPC_TRUE;
                                                break;
                                                }

                        /* This must be an advertisement from an agent. */
                                        op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
                                        mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffices. */
                                        if
(inet_address_equal(tmp_agent_address, ha_address))
                                                {
                                                if (loopback_intf)
                                                      {
                                    /* This is not supported. */
                                                      op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                            OPC_NIL,
OPC_NIL, OPC_NIL);
                                                      }

                                    /* Ad from my home agent. */
                                                is_ha_ad_reception =
OPC_TRUE;

                        /* Initialize counter before start reg process. */
                                                retry_counter = 0;

                        /* Update the latest ha info structure for later. */

      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                      (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                    /* Deregister with HA. */
                                                mip_mn_register (0,
latest_ha_info, OPC_TRUE);

                                          /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                }
                                        else
                                                {
                                                if (f_bit)
```

```
                                                        {
                                                    /* A foreign agent ad. */
                                                        if
(current_agent_pref_level < pref_level)
                                                            {
            /* New FA agent advertising has higher preference.  Switch. */

      is_switch_fa = OPC_TRUE;
                                                            }
                                                        else
                                                            {

      is_fa_ad_reception = OPC_TRUE;
                                                            }

                                    /* Update the FA cache information. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                                                                (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                /* Check to see if it is the same agent. */
                                                        if
(inet_address_equal (agent_address, tmp_agent_address))
                                                            {
                                            /* Update timer for FA timeout. */

      mip_mn_agent_timer_update (op_sim_time () + (double) irdp_lifetime);
                                                            }
                                                        }
                                                    else
                                                        {
/* cannot do anything with this agent who is only HA for other group. */
                                                        is_ip_pk =
OPC_TRUE;
                                                        }
                                                    }

                                                /* Clean up. */
                                                mip_sup_pk_cleanup
(invoke_info_ptr);
                                                op_pk_destroy (irdp_pkptr);

                                                break;
                                                }

                                            case MipC_Invoke_Type_Tunnel_Check:
                                                {
                                                is_ip_pk = OPC_TRUE;

                    /*   should try to decapsulte packet if in MR mode. */
                                                if (mip_node_type ==
MipC_Node_Type_MR)
                                                    {
      /*   will check first if this packet is tunneling other IP packet */
                                                    if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
```

```
                                                              ==
OPC_COMPCODE_SUCCESS)
                                                              {
                                          /* Sanity check on the packet */

                                    /* Invoke IP delayed to handle the packet */

      mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

                              /* Write stats for the received tunneled packet. */

      mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

                        /* Let IP caller know that   are handling the packet */

      mip_sup_pk_cleanup (invoke_info_ptr);
                                                              }
                                                        }

                                                 break;
                                                 }

                                          case MipC_Invoke_Type_IP_Datagram:
                                                 {
                                                 is_ip_pk = OPC_TRUE;

                                                 /* The destination address
that is going out on this interface when in MR mode should be forwarded to
either HA or FA.  Whoever already registered or trying to. */
                                       if (mip_node_type == MipC_Node_Type_MR)
                                                 {
                                                 /* Handle packet if I am a MR. */
                                       mip_mn_ip_pk_handle (invoke_info_ptr);
                                                 }
                                                 else
                                                 {
                              /* Unknown address.  Send to the current agent. */
                                                 invoke_info_ptr-
>rte_info_ici_ptr->next_addr = agent_address;
                                                 }

                                                 break;
                                                 }
                                          }/* switch (invoke_info_ptr-
>invocation_type) */
                                       }
                                 else
                                       {
                                       /* Reg packet arrival. */
                                       op_ici_destroy (op_intrpt_ici ());
                                       is_reg_pk = OPC_TRUE;
                                       }
                                 }
                           }
                           FSM_PROFILE_SECTION_OUT (state3_exit_exec)
```

```
                  /** state (Away) transition processing **/
                  FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Away trans
conditions]", state3_trans_conds)
                  FSM_INIT_COND (FA_TIMEOUT || REREGISTER || SWITCH_FA)
                  FSM_TEST_COND (IP_PK)
                  FSM_TEST_COND (FA_AD_RECEPTION)
                  FSM_TEST_COND (REG_PK)
                  FSM_TEST_COND (HA_AD_RECEPTION)
                  FSM_TEST_LOGIC ("Away")
                  FSM_PROFILE_SECTION_OUT (state3_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 5, state5_enter_exec, ;,
"FA_TIMEOUT || REREGISTER || SWITCH_FA", "", "Away", "Check FA cache",
"tr_4", "mobile_ip_mn [Away -> Check FA cache : FA_TIMEOUT || REREGISTER ||
SWITCH_FA / ]")
                        FSM_CASE_TRANSIT (1, 3, state3_enter_exec, ;,
"IP_PK", "", "Away", "Away", "tr_10", "mobile_ip_mn [Away -> Away : IP_PK /
]")
                        FSM_CASE_TRANSIT (2, 3, state3_enter_exec, ;,
"FA_AD_RECEPTION", "", "Away", "Away", "tr_27", "mobile_ip_mn [Away -> Away :
FA_AD_RECEPTION / ]")
                        FSM_CASE_TRANSIT (3, 3, state3_enter_exec, ;,
"REG_PK", "", "Away", "Away", "tr_37", "mobile_ip_mn [Away -> Away : REG_PK /
]")
                        FSM_CASE_TRANSIT (4, 4, state4_enter_exec, ;,
"HA_AD_RECEPTION", "", "Away", "Pending registration", "tr_45", "mobile_ip_mn
[Away -> Pending registration : HA_AD_RECEPTION / ]")
                        }
            /*-------------------------------------------------------*/

                  /** state (Pending registration) enter executives **/
                  FSM_STATE_ENTER_UNFORCED (4, "Pending registration",
state4_enter_exec, "mobile_ip_mn [Pending registration enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Pending
registration enter execs]", state4_enter_exec)
                        {
                        /* Update the status for debugging. */
                        mip_sup_mn_mr_status_update (node_objid,
home_address, MipC_Mn_Mr_Status_Pending,
                              ha_address, agent_address);
                        }
                        FSM_PROFILE_SECTION_OUT (state4_enter_exec)

                  /** blocking after enter executives of unforced state. **/
                  FSM_EXIT (9,"mobile_ip_mn")


                  /** state (Pending registration) exit executives **/
                  FSM_STATE_EXIT_UNFORCED (4, "Pending registration",
"mobile_ip_mn [Pending registration exit execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Pending
registration exit execs]", state4_exit_exec)
                        {
                        /* Who invoked me? */
```

```
                        op_pro_invoker (proc_info_struct_ptr->pro_hndl,
&inv_mode);

                        if (inv_mode == OPC_PROINV_DIRECT)
                              {
                              /* one of those timer went off. */
                              is_timeout = OPC_TRUE;
                              }
                        else
                              {
                              /* See if   are getting an IP packet. */
                              invoke_info_ptr = (MipT_Invocation_Info*)
op_pro_argmem_access ();
                              if (invoke_info_ptr != OPC_NIL)
                                    {
                        /*   have an invocation from IP.  What kind thou? */
                                    switch (invoke_info_ptr->invocation_type)
                                          {
                                          case MipC_Invoke_Type_IRDP:
                                                {
                        /*   have an IRDP packet.  But which kind thou? */
                                                if (invoke_info_ptr-
>irdp_type == IcmpC_Type_IRDP_Sol)
                                                      {
                              /*   do not want to deal with this packet. */
                                                      mip_sup_pk_cleanup
(invoke_info_ptr);

                                                      is_ip_pk = OPC_TRUE;
                                                      break;
                                                      }

                        /* This must be an advertisement from an agent. */
                                                is_ad_reception = OPC_TRUE;

                  /*   have an agent advertisement.  But which kind thou? */
                                                op_pk_nfd_get
(invoke_info_ptr->pk_ptr, "data", &irdp_pkptr);
                                                mip_mn_ad_packet_parse
(irdp_pkptr, &h_bit, &f_bit, &tmp_agent_address, &irdp_lifetime,
&pref_level);

      /* For now, comaprison of the only address in the packet suffice. */
                                                if
(inet_address_equal(tmp_agent_address, ha_address))
                                                      {
                                                      if (loopback_intf)
                                                            {
                                                /* This is not supported. */
                                                            op_sim_end
("MR/MN when configured on a loopback interface, cannot directly communicate
with HA.",
                                                                  OPC_NIL,
OPC_NIL, OPC_NIL);
                                                            }

                        /* Update the latest ha info structure for later. */
```

```
      mip_mn_agent_cache_update (&latest_ha_info, tmp_agent_address,
                                                              (double)
irdp_lifetime, pref_level, invoke_info_ptr);

      /* Check if   are currently trying to register directly with HA. */
                                                        if (!direct_reg)
                                                              {

      op_ev_cancel_if_pending (reg_retry_timer_ehndl);
                                                    /* Deregister with HA. */
                                                              mip_mn_register
(0, latest_ha_info, OPC_TRUE);
                                                              }

                                              /* Update timer for HA timeout. */

      mip_mn_agent_timer_update (latest_ha_info.lifetime);
                                                        }
                                                    else
                                                        {
                                                        if (f_bit)
                                                              {
                                        /* Update the FA cache information. */

      mip_mn_agent_cache_update (&latest_fa_info, tmp_agent_address,
                                                                    (double)
irdp_lifetime, pref_level, invoke_info_ptr);

                                  /* Check to see if it is the same agent. */
                                                              if
(inet_address_equal (agent_address, tmp_agent_address))
                                                                    {
                                              /* Update timer for FA timeout. */

      mip_mn_agent_timer_update (op_sim_time () + (double) irdp_lifetime);
                                                                    }
                                                              }
                                                        else
                                                              {
/* cannot do anything with this agent who is only HA for other group. */
                                                              }
                                                        }

                                                    /* Clean up. */
                                                    mip_sup_pk_cleanup
(invoke_info_ptr);
                                                    op_pk_destroy (irdp_pkptr);

                                                    break;
                                                    }

                                              case MipC_Invoke_Type_Tunnel_Check:
                                                    {
                                                    is_ip_pk = OPC_TRUE;

                        /*   should try to decapsulte packet if in MR mode. */
```

```
                                                if (mip_node_type ==
MipC_Node_Type_MR)
                                                      {
      /*   will check first if this packet is tunneling other IP packet */
                                                      if
(mip_sup_ip_in_ip_decapsulate (invoke_info_ptr->pk_ptr, &encap_pk_ptr)
                                                            ==
OPC_COMPCODE_SUCCESS)
                                                      {
                                          /* Sanity check on the packet */

                                 /* Invoke IP delayed to handle the packet */

      mip_sup_packet_send_to_ip (module_data, encap_pk_ptr);

                    /* Write stats for the received tunneled packet. */

      mip_mn_tunneled_pk_stat_write (invoke_info_ptr->pk_ptr);

                                                      /* Let IP caller
know that   are handling the packet */

      mip_sup_pk_cleanup (invoke_info_ptr);
                                                      }
                                                }

                                                break;
                                                }

                                          case MipC_Invoke_Type_IP_Datagram:
                                                {
                                                is_ip_pk = OPC_TRUE;

                                                /* The destination address
that is going out on this interface when in MR mode should be forwarded to
either HA or FA.  Whoever already registered or trying to. */
                                                if (mip_node_type ==
MipC_Node_Type_MR)
                                                      {
                    /* Handle packet if I am a MR. */
                                                      mip_mn_ip_pk_handle
(invoke_info_ptr);
                                                      }
                                                else
                                                      {
                           /* Unknown address.  Send to the current agent. */
                                                      invoke_info_ptr-
>rte_info_ici_ptr->next_addr = agent_address;
                                                      }

                                                break;
                                                }
                                          } /* switch (invoke_info_ptr-
>invocation_type) */
                                    }
                                else
                                    {
```

```
                              /* Registration packet arrival. */
                              is_reg_pk = OPC_TRUE;
                              }
                        }
                  }
                  FSM_PROFILE_SECTION_OUT (state4_exit_exec)

            /** state (Pending registration) transition processing **/
            FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Pending registration
trans conditions]", state4_trans_conds)
            FSM_INIT_COND ( REG_PK)
            FSM_TEST_COND (TIMEOUT)
            FSM_TEST_COND (AD_RECEPTION)
            FSM_TEST_COND (IP_PK)
            FSM_TEST_LOGIC ("Pending registration")
            FSM_PROFILE_SECTION_OUT (state4_trans_conds)

            FSM_TRANSIT_SWITCH
                  {
                  FSM_CASE_TRANSIT (0, 6, state6_enter_exec, ;, "
REG_PK", "", "Pending registration", "Handle registration", "tr_15",
"mobile_ip_mn [Pending registration -> Handle registration :  REG_PK / ]")
                  FSM_CASE_TRANSIT (1, 7, state7_enter_exec, ;,
"TIMEOUT", "", "Pending registration", "Handle timeout", "tr_19",
"mobile_ip_mn [Pending registration -> Handle timeout : TIMEOUT / ]")
                  FSM_CASE_TRANSIT (2, 4, state4_enter_exec, ;,
"AD_RECEPTION", "", "Pending registration", "Pending registration", "tr_28",
"mobile_ip_mn [Pending registration -> Pending registration : AD_RECEPTION /
]")
                  FSM_CASE_TRANSIT (3, 4, state4_enter_exec, ;,
"IP_PK", "", "Pending registration", "Pending registration", "tr_11",
"mobile_ip_mn [Pending registration -> Pending registration : IP_PK / ]")
                  }
      /*---------------------------------------------------------*/

            /** state (Check FA cache) enter executives **/
            FSM_STATE_ENTER_FORCED (5, "Check FA cache",
state5_enter_exec, "mobile_ip_mn [Check FA cache enter execs]")
                  FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Check FA cache
enter execs]", state5_enter_exec)
                  {
                  /* See if   can use the cached FA information. */
                  if (inet_address_valid (latest_fa_info.address))
                        {
                        if (latest_fa_info.lifetime > op_sim_time () )
                              {
                  /* Initialize counter before start reg process. */
                              retry_counter = 0;

                  /* Register with the latest advertised FA. */
                              mip_mn_register (reg_info.req_lifetime,
                                    latest_fa_info, OPC_FALSE);

                              /* Update the timer timeout. */
                              if (!HA_TIMEOUT && !FA_TIMEOUT)
                                    {
                  mip_mn_agent_timer_update (latest_fa_info.lifetime);
```

```
                                    }
                                else
                                    {
                                    /* cannot cancel the current event. */
      agent_timer_ehndl = op_intrpt_schedule_self (latest_fa_info.lifetime,
                                                   MipC_MN_Timer_Agent);
                                    }

                                is_valid_fa_candidate = OPC_TRUE;
                                }
                            }

                        if (!is_valid_fa_candidate)
                            {
                            if (solicitation)
                                {
                    /* Schedule an interrupt to send solicitation packet. */
              op_intrpt_schedule_self (op_sim_time (), MipC_MN_Timer_Solicit);
                                }

                    /* Cancel reregister timer first. */
                        op_ev_cancel_if_pending (reregister_timer_ehndl);

                            if (op_sim_anim ())
                                {
                                /* Erase the existing tunnel. */
                                mip_sup_draw_tunnel (node_objid,
ha_address, ha_address, ((mip_node_type == MipC_Node_Type_MR) ? OPC_TRUE :
OPC_FALSE));
                                }
                            }

                        }
                        FSM_PROFILE_SECTION_OUT (state5_enter_exec)

                    /** state (Check FA cache) exit executives **/
                    FSM_STATE_EXIT_FORCED (5, "Check FA cache", "mobile_ip_mn
[Check FA cache exit execs]")

                    /** state (Check FA cache) transition processing **/
                    FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Check FA cache trans
conditions]", state5_trans_conds)
                    FSM_INIT_COND (!VALID_FA_CANDIDATE)
                    FSM_TEST_COND (VALID_FA_CANDIDATE)
                    FSM_TEST_LOGIC ("Check FA cache")
                    FSM_PROFILE_SECTION_OUT (state5_trans_conds)

                    FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;,
"!VALID_FA_CANDIDATE", "", "Check FA cache", "Lost", "tr_6", "mobile_ip_mn
[Check FA cache -> Lost : !VALID_FA_CANDIDATE / ]")
                        FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ;,
"VALID_FA_CANDIDATE", "", "Check FA cache", "Pending registration", "tr_7",
"mobile_ip_mn [Check FA cache -> Pending registration : VALID_FA_CANDIDATE /
]")
                        }
```

```
            /*-------------------------------------------------------*/

                  /** state (Handle registration) enter executives **/
                  FSM_STATE_ENTER_FORCED (6, "Handle registration",
state6_enter_exec, "mobile_ip_mn [Handle registration enter execs]")
                        FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Handle
registration enter execs]", state6_enter_exec)
                        {
                        /* Access ICI from mobile ip module. */
                        reg_ici_ptr = op_intrpt_ici ();

                        /* Get the ICI values. */
                        op_ici_attr_get (reg_ici_ptr, "reply_code",
&reply_code);
                        op_ici_attr_get (reg_ici_ptr, "lifetime_grant",
&lifetime_grant);
                        op_ici_attr_get (reg_ici_ptr, "identification",
&tmp_reg_id);

                        if (tmp_reg_id == reg_id)
                              {
                              /* Did it go through? */
                              if ((reply_code == MipC_Reg_Reply_Code_Accept)
||
                                    (reply_code ==
MipC_Reg_Reply_Code_Accept_No_Simultaneous_Binding))
                                    {
                                    /* Cancel the retry timer first. */
                                    op_ev_cancel_if_pending
(reg_retry_timer_ehndl);

                                    if (direct_reg)
                                          {
                                          is_ha_reg_success = OPC_TRUE;

                                          if (MIP_TRACE)
                                                {
                                                op_prg_odb_print_major
("Registering directly with HA successful.", OPC_NIL);
                                                }

                                          if (op_sim_anim ())
                                                {
                                                /* Draw tunnel to the HA. */
                                                mip_sup_draw_tunnel
(node_objid, ha_address, agent_address, ((mip_node_type == MipC_Node_Type_MR)
? OPC_TRUE : OPC_FALSE));
                                                }
                                          }
                                    else
                                          {
                                          is_fa_reg_success = OPC_TRUE;

                                    /* Cancel reregister timer first. */
                                          op_ev_cancel_if_pending
(reregister_timer_ehndl);
```

```
                                        /* Update the lifetime. */
                                        time_to_reregister = op_sim_time ()
+ (double) lifetime_grant - MipC_MN_Rereg_Buffer;
                                        reregister_timer_ehndl =
op_intrpt_schedule_self (time_to_reregister,
                                              MipC_MN_Timer_Rereg);

                                        if (MIP_TRACE)
                                              {
                                              op_prg_odb_print_major
("Registering via a FA successful.", OPC_NIL);
                                              }

                                        if (op_sim_anim ())
                                              {
                                        /* Draw tunnel to the HA through FA. */
                                              mip_sup_draw_tunnel
(node_objid, ha_address, agent_address, ((mip_node_type == MipC_Node_Type_MR)
? OPC_TRUE : OPC_FALSE));
                                              }
                                        }
                                  }
                            else
                                  {
                                  /* have to retry. */
                                  if (retry_counter <=  reg_info.retry)
                                        {
                                        is_retry = OPC_TRUE;
                                        }
                                  else
                                        {
                                        op_ev_cancel_if_pending
(reg_retry_timer_ehndl);
                                        is_out_of_retries = OPC_TRUE;
                                        }
                                  }
                            }
                      else
                            {
                            /* Identification mismatch. */
                            is_invalid_reply = OPC_TRUE;
                            }

                      /* Clean up. */
                      op_ici_destroy (reg_ici_ptr);
                      }
                      FSM_PROFILE_SECTION_OUT (state6_enter_exec)

                /** state (Handle registration) exit executives **/
                FSM_STATE_EXIT_FORCED (6, "Handle registration",
"mobile_ip_mn [Handle registration exit execs]")

                /** state (Handle registration) transition processing **/
                FSM_PROFILE_SECTION_IN ("mobile_ip_mn [Handle registration
trans conditions]", state6_trans_conds)
                FSM_INIT_COND (FA_REG_SUCCESS)
                FSM_TEST_COND (OUT_OF_RETRIES)
```

```
                  FSM_TEST_COND (RETRY || INVALID_REPLY)
                  FSM_TEST_COND (HA_REG_SUCCESS)
                  FSM_TEST_LOGIC ("Handle registration")
                  FSM_PROFILE_SECTION_OUT (state6_trans_conds)

                  FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;,
"FA_REG_SUCCESS", "", "Handle registration", "Away", "tr_2", "mobile_ip_mn
[Handle registration -> Away : FA_REG_SUCCESS / ]")
                        FSM_CASE_TRANSIT (1, 5, state5_enter_exec, ;,
"OUT_OF_RETRIES", "", "Handle registration", "Check FA cache", "tr_12",
"mobile_ip_mn [Handle registration -> Check FA cache : OUT_OF_RETRIES / ]")
                        FSM_CASE_TRANSIT (2, 4, state4_enter_exec, ;, "RETRY
|| INVALID_REPLY", "", "Handle registration", "Pending registration",
"tr_18", "mobile_ip_mn [Handle registration -> Pending registration : RETRY
|| INVALID_REPLY / ]")
                        FSM_CASE_TRANSIT (3, 2, state2_enter_exec, ;,
"HA_REG_SUCCESS", "", "Handle registration", "At home", "tr_44",
"mobile_ip_mn [Handle registration -> At home : HA_REG_SUCCESS / ]")
                        }
            /*---------------------------------------------------------*/
```