```
/*************************************************************************/
        /*HEADFILE McrDef.h*/
/*************************************************************************/
#ifndef __MCRDEF__H__
#define __MCRDEF__H__

/* --- traffic class types --- */
#define  AUDIO_TRAFFIC              1
#define  VIDEO_TRAFFIC              0

/* --- retried user request  --- */
#define  MAX_RETRY_COUNT            3
#define  MEAN_RETRY_INTVAL          15

/* --- network environment --- */
#define  MAX_ROUTE_LEN              10
#define  MAX_ROUTE_NUM              50
#define  MAX_TRUNK_NUM              40
#define  MAX_PLINK_NUM              38
#define  MAX_GATEWAY_NUM            150
#define  MAX_ROUTER_NUM             15
#define  MAX_LINK_NUM               19
#define  MAX_USR_NUM                8000
#define  MAX_ROUTER_LINK            4
#define  MAX_HOLD_NUM               1000

/* --- user request result --- */
#define  ADMITTED                   0
#define  BLOCKED                    1
#define  HOLD_ACCEPT                2
#define  HOLD_REJECT                3
#define  HOLD_ADMITTED              4
#define  BLOCKED_i                  5
#define  ADMITTED_i                 6

/* --- mark the policy server as a special gateway with the id 0 ---*/
#define  POLICY_SERVER              15
#define  POLICY_SERVER_ROUTER       6

/* --- message types --- */
#define  USR_REQ                    1
#define  USR_REQ_ADMITTED           2
#define  USR_PKT                    3
#define  USR_PKT_END                4
#define  TRK_CFG_REQ                5
#define  TRK_CFG_CNF                6
#define  TRK_RSC_RLS                7
#define  VIP_EST                    8
#define  VIP_RLS                    9
#define  SETUP_TIME                 10
#define  PROVLINK_BUSY              11
#define  PROVLINK_NOTBUSY           12
#define  USR_DATA                   13
```

```
#define  CreateNewUser                  100
#define  DESTROY_USR_PROC               101
#define  EDGE_HLD_ADMITTED              102
#define  EDGE_HLD_REP                   103
#define  RESPONSE_TIME                  105

/* --- packet types --- */
#define  QOS_FIFO_QUEUE                 "QoS_fifo_queue"
#define  QOS_RESPONSE_TIME              "QoS_response_time"
#define  QOS_EDGE_HLD_ADM               "QOS_edge_hld_adm"
#define  QOS_USR_REQ                    "QoS_usr_req"
#define  QOS_USR_REQ_ADM                "QoS_usr_req_adm"
#define  QOS_USR_PKT                    "QoS_usr_pkt"
#define  QOS_USR_PKT_END                "QoS_usr_pkt_end"
#define  QOS_TRK_CFG_REQ                "QoS_trk_cfg_req"
#define  QOS_TRK_CFG_CNF                "QoS_trk_cfg_cnf"
#define  QOS_TRK_RSC_RLS                "QoS_trk_rsc_rls"
#define  QOS_VIP_EST                    "QoS_vip_est"
#define  QOS_VIP_RLS                    "QoS_vip_rls"
#define  QOS_SETUP_TIME                 "QoS_setup_time"
#define  QOS_PROVLINK_BUSY              "QoS_provlink_busy"
#define  QOS_PROVLINK_NOTBUSY           "QoS_provlink_notbusy"
#define  QOS_DESTROY_USR_PROC           "QoS_destroy_usr_proc"
#define  QOS_DATA                       "QoS_data"

/* --- user request profile directory --- */
#define  PROFILE_PATH           "C:\\OpNet\\PS\\OpNet\\Environmentfile\\"
/* --- network topology and configuration files --- */
#define  ROUTE_FILE             "C:\\OpNet\\PS\\OpNet\\Environmentfile\\QoSROUTE"
#define  TRUNK_FILE             "C:\\OpNet\\PS\\OpNet\\Environmentfile\\QoSTrunk"
#define  FORWARD_FILE           "C:\\OpNet\\PS\\OpNet\\Environmentfile\\FORWARD_FILE"
#define  LINK_FILE              "C:\\OpNet\\PS\\OpNet\\Environmentfile\\Link"
#define  PROVISIONED_LINK_FILE      "C:\\OpNet\\PS\\OpNet\\Environmentfile\\ProLink"
#define  RUNTIME_FILE           "C:\\OpNet\\PS\\OpNet\\Environmentfile\\Runtime\\QoSRuntime"
#define  RUNTIME_PATH
         "C:\\OpNet\\PS\\OpNet\\Environmentfile\\Runtime\\QoSRuntime_Router_"
/* --- variables used by the gateway to report excess resource --- */
#define  LIFETIME               240
#define  Released_Period        10
#define  CHECK_PERIOD           100
#define  EXCESS_THRESHOLD       3
#define  RELEASE_PERCENT        3
#define  INCREASING_TIMES       2
#define  BROADCAST              100
#endif
```

```
/*************************************************************************/
        /*HEADFILE McrDS.h*/
/*************************************************************************/
#ifndef __POLICYSVR__MCRDS__H__
#define __POLICYSVR__MCRDS__H__

#include ".\McrDef.h"
typedef struct {
        int                     RouteID;
        int                     Type;
        int                     Src_router;
        int                     Dest_router;
        int                     Length;
        int                     TrunkList[MAX_ROUTE_LEN];
} QoSRoute;

typedef struct {
        int                     TrunkID;
        int                     RsvBdw;
        int                     UsdBdw;
        int                     PLinkID;
} QoSTrunk;

typedef struct {
        int                     Dest;
        int                     Outgoing_intf;
} QoSFwd;

struct RouteDef {
int                             RouteID;
int                             Length;
int                             TrunkList[MAX_ROUTE_LEN];
struct RouteDef *       Next;
};

#endif
```

```
/*************************************************************************/
        /*HEADFILE PolicySvr_gateway.h*/
/*************************************************************************/
#ifndef __POLICYSVR__GATEWAY__H__
#define __POLICYSVR__GATEWAY__H__

#include ".\McrDef.h"
#include ".\McrDS.h"
/* --- default increasing times for trunk reconfiguration --- */
/* --- allocated QoS routes information table --- */
/* 1. route description */
typedef  struct RouteDef RouteRec;
/* 2. structure to separate QoS routes for audio and video */
typedef struct {
RouteRec *              ptr;
//RouteRec *            Audio_ptr;
//RouteRec *            Video_ptr;
} RouteItem;

/* --- allocated trunks information table --- */
/* 1. one VIP going through the trunk */
struct VipDef {
int                     VipID;
int                     UsdBdw;
struct VipDef *         Next;
};

/* 2. one item in the trunk information table, indexed by the trunk ID */
struct TrunkDef {
int                     TrunkID;
int                     RsvBdw;
int                     UsdBdw;
int                     PLinkID;
int                     Busy;
struct VipDef *         VipPnt;
};

typedef struct TrunkDef TrunkRec;

/* --- one item in forwarding table --- */
typedef QoSFwd          FwdDef;

#endif
```

```
/*************************************************************************/
        /*HEADFILE PolicySvr_router.h*/
/*************************************************************************/
#ifndef __POLICYSVR__ROUTER__H__
#define __POLICYSVR__ROUTER__H__

#include ".\McrDef.h"
#include ".\McrDS.h"

/* forwarding table */
typedef QoSFwd      RouteItem;

/* resource usage information table */
struct VipRec {
int                 Src_gtw;
int                 Dest_gtw;
int                 Route_id;
int                 Bdw_used;
int                 Vip_id;
struct VipRec *     Next;
};

typedef struct VipRec VipItem;

#endif
```

```
/*************************************************************************/
        /*HEADFILE PolicySvr_Runtime.h*/
/*************************************************************************/
#ifndef __POLICYSVR__RUNTIME__H__
#define __POLICYSVR__RUNTIME__H__

#include ".\McrDef.h"
#include <stdio.h>
#include <STDLIB.H>
#include <math.h>

struct NMS_KEEP {
int                         GatewayID;
int                         Vip_id;
int                         Class;
int                         DestRouter;
//int                       TrunkID;
int                         UsdBdw;
int                         Finish_time;
struct NMS_KEEP *           KeepPtn;
};

typedef struct NMS_KEEP NMS_REC;

typedef struct {
int                         GatewayID;
int                         Vip_id;
int                         Class;
int                         DestRouter;
//int                       TrunkID;
int                         UsdBdw;
int                         Finish_time;
} QoSRuntime;

typedef struct {
int                         UsrID;
int                         VipID;
int                         ReqType;
int                         ProcID;
int                         SrcRouter;
int                         SrcGtw;
int                         DestRouter;
int                         DestGtw;
int                         BdwReq;
int                         CallDuration;
int                         RecallTime;
int                         Reqtime;
int                         QueueID;
} FlowData;
typedef FlowDataFlowHold;
typedef struct {
int                         QueueID;
} ReleasedRecord;
typedef struct {
int                         ProcID;
double                      TimeOut;
} TimeOut;
#endif
```

```
/*************************************************************************/
        /*HEADFILE PolicySvr_server.h*/
/*************************************************************************/
#ifndef __POLICYSVR__SERVER__H__
#define __POLICYSVR__SERVER__H__
#include ".\McrDef.h"
#include ".\McrDS.h"
/* --- all QoS routes information table see McrDS.h QoSRoute --- */
/* --- trunk information --- */
typedef QoSTrunk TrunkItem;

/* --- link information --- */
typedef struct {
int                             LinkID;
int                             Capacity;
} LinkItem;

/* --- provisioned link description --- */
typedef struct {
int                             PLinkID;
long                            RsvBdw;
long                            UsdBdw;
int                             LinkID;
int                             Busy;
} PLinkItem;

int     all_req_num=0;
int     admit_req_num=0;
int     overhead_msg_num=0;
int     block_req_num=0;
int     recall_overhead_num=0;
int     all_hold_num=0;
int     all_incoming_num=0;
int     total_CAC=0;
int     queue_id=0;
int     queue_id1=0;
#endif
```

```
/*************************************************************************/
        /*HEADFILE PolicySvr_user.h*/
/*************************************************************************/
#ifndef __POLICYSVR__USER__H__
#define __POLICYSVR__USER__H__
#include ".\McrDef.h"
/* --- ratio of audio and video traffic --- */
#define  AUDIO_VIDEO                    83

/* --- single user request --- */
#define  AUDIO_BDW_REQUIRE                      64      /* default unit is kbits/s */
#define  VIDEO_BDW_REQUIRE          3000    /* default unit is kbits/s */
#define  MEAN_AUDIO_SESSION_LEN     300     /* default unit is second(s) */
#define  MEAN_VIDEO_SESSION_LEN     1152    /* default unit is second(s) */
#define  MEAN_RETRY_COUNT           50
/* --- parameters to define the user traffic --- */
#define PKT_INTARVL_TIME            "constant (10.0)"
#define PKT_SIZE                            "constant (1024)"
#define PKT_FORMAT                  QOS_USR_REQ
#define audio_interarrival_str      "constant(0.003)"
#define video_interarrival_str      "constant(0.125)"
//#define MAX_NUM                   15000

/* --- self-defined structure used between QoSpGtw_pkt_gen(parent process) and QoSpGtw_pkt(child
process) --- */
typedef struct {
int                                 Para_src_gtw;
int                                 Para_dest_gtw;
int                                 Para_route_id;
int                                 Para_vip_id;
double                              Para_start_time;
double                              Para_stop_time;
} ParaRec;

/* --- structure describing the user request arrival distribution pattern, used by QoSpGtw_req_gen ---*/
struct PtnDef {
char                                intarvl_ptn [128];
char                                size_ptn [128];
double                              ptn_start_time;
double                              ptn_stop_time;
};

struct UsrReqPtn {
double                              ptn_chg_time;
struct PtnDef                       ptn_item;
};

typedef  struct {
struct PtnDef                       para_info;
struct UsrReqRec *                  next;
} UsrReqRec;

/* --- structure describing the user traffic distribution pattern --- */
typedef struct {
int                                 GatewayID;
int                                 DistProb;
} UsrTraDist;

/* --- structure used between QoS_usr_usr_gen(parent process) and QoS_usr(child) --- */
typedef struct {
```

```
int                                     Msg_type;
int                                     Src_router;
int                                     Src_gtw;
int                                     Dest_router;
int                                     Dest_gtw;
int                                     Route_id;
int                                     Vip_id;
int                                     Decision;
int                                     Usr_id;
int                                     Proc_id;
int                                     Req_type;
int                                     Session_len;
int                                     Retry_count;
int                                     Mean_retry_intval;
int                                     dest_queue_id;
int                                             Reqtime;
int                                             Destination;
int                                             Time_est;
} UsrPtn;

typedef struct {
int                                             Queue_id;
int                                             Route_id;
int                                             Usr_id;
int                                             Proc_id;
int                                             Vip_id;
int                                             BW_used;
double                                          Finish_time;
} KeepPtn[MAX_NUM];

/* --- structure used between QoS_usr_proc and QoS_usr --- */
typedef struct {
int                                             IntID;
Objid                                           OpID;
} IdRec;

/* --- structure used between QoS_usr_node(parent process) and QoS_usr(child) --- */
typedef struct {
int                                             ProcID;
Prohandle                                       Pro_handle;
} ProcRec;

#endif
```

```
/*****************************************************************************/
/*USER NODE*/
/*****************************************************************************/
    /*usr_gen module*/
/*****************************************************************************/
/* Process model C form file: QoS_usr_usr_gen.pr.c */
/* This variable carries the header into the object file */
static const char QoS_usr_usr_gen_pr_c [] = "MIL_3_Tfile_Hdr_ 80C 30A op_runsim 7 43F415E3
43F415E3 1 KiT KiT 0 0 none none 0 0 none 0 0 0 0 0 0";

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

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

/* Header Block */
/* --- include file made by myself --- */
#include   "..\headfile\McrDef.h"
#include   "..\headfile\PolicySvr_user.h"

/* Include files.*/
#include <oms_dist_support.h>

/* Special attribute values.*/
#define         SSC_INFINITE_TIME           -1.0

/* Interrupt code values.*/
#define         SSC_PTN_START               0
#define         SSC_GENERATE                    1
#define         SSC_PTN_STOP                2
#define         SSC_CHANGE                  3

/* Node configuration constants.*/
#define         SSC_STRM_TO_LOW             0

/* Macro definitions for state transitions.*/
#define         PTN_START                   (op_intrpt_type () == OPC_INTRPT_SELF &&
op_intrpt_code() == SSC_PTN_START)
#define         PTN_DISABLED                        (op_intrpt_type () ==
OPC_INTRPT_SELF && op_intrpt_code() == SSC_PTN_STOP)
#define         PTN_STOP                    (op_intrpt_type () == OPC_INTRPT_SELF &&
op_intrpt_code() == SSC_PTN_STOP)
#define         USR_STOP                    (op_intrpt_type () == OPC_INTRPT_SELF &&
op_intrpt_code() == SSC_USR_STOP)
#define         USR_START                   (op_intrpt_type () == OPC_INTRPT_SELF &&
op_intrpt_code() == SSC_USR_START)
#define         USR_GENERATE                (op_intrpt_type () == OPC_INTRPT_SELF &&
op_intrpt_code() == SSC_GENERATE)
#define         PATTERN_CHANGE              (op_intrpt_type () == OPC_INTRPT_SELF &&
op_intrpt_code() == SSC_CHANGE)
#define    PKT_ARRIVAL                      ( op_intrpt_type () == OPC_INTRPT_STRM )
```

```
/* Function prototypes.*/
static void                     ss_usr_generate (void);
/* End of Header Block */
#if !defined (VOSD_NO_FIN)
#undef  BIN
#undef  BOUT
#define BIN     FIN_LOCAL_FIELD(last_line_passed) = __LINE__ - _block_origin;
#define BOUT    BIN
#define BINIT   FIN_LOCAL_FIELD(last_line_passed) = 0; _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 */
        Objid                   own_id;
        char                    format_str [64];
        double                  start_time;
        double                  stop_time;
        OmsT_Dist_Handle        interarrival_dist_ptr;
        OmsT_Dist_Handle        pksize_dist_ptr;
        Boolean                 generate_unformatted;
        Evhandle                next_pk_evh;
        double                  next_intarr_time;
        Stathandle              bits_sent_hndl;
        Stathandle              packets_sent_hndl;
        Stathandle              packet_size_hndl;
        Stathandle              interarrivals_hndl;
        char                    rkey;
        int                     pk_count;
        UsrReqRec*              ptn_table;
        UsrReqRec*              ptn_list;
        UsrReqRec*              current_rec;
        Distribution *          retry_count;
        int                     mean_retry_intval;
        Distribution *          router_addr_seed;
        UsrTraDist              tra_dist_table [MAX_GATEWAY_NUM];
        UsrPtn                  para_rec;
        Objid                   parent_id;
        char                    node_name [15];
        int                     tmp_queue_Objid;
        ProcRec                 prorec [MAX_USR_NUM];
        int                     procnum;
        Distribution *          audio_session_duration;
        Distribution *          video_session_duration;
        Distribution *          req_type;
        Distribution *          gtw_addr_seed;
        UsrPtn                  para_hold;
        int                     tmp_q_Objid;
        } QoS_usr_usr_gen_state;
#define pr_state_ptr            ((QoS_usr_usr_gen_state*) SimI_Mod_State_Ptr)
#define own_id                  pr_state_ptr->own_id
#define format_str              pr_state_ptr->format_str
#define start_time              pr_state_ptr->start_time
#define stop_time               pr_state_ptr->stop_time
#define interarrival_dist_ptr   pr_state_ptr->interarrival_dist_ptr
```

```
#define pksize_dist_ptr                 pr_state_ptr->pksize_dist_ptr
#define generate_unformatted            pr_state_ptr->generate_unformatted
#define next_pk_evh                     pr_state_ptr->next_pk_evh
#define next_intarr_time                pr_state_ptr->next_intarr_time
#define bits_sent_hndl                  pr_state_ptr->bits_sent_hndl
#define packets_sent_hndl               pr_state_ptr->packets_sent_hndl
#define packet_size_hndl                pr_state_ptr->packet_size_hndl
#define interarrivals_hndl              pr_state_ptr->interarrivals_hndl
#define rkey                            pr_state_ptr->rkey
#define pk_count                        pr_state_ptr->pk_count
#define ptn_table                       pr_state_ptr->ptn_table
#define ptn_list                        pr_state_ptr->ptn_list
#define current_rec                     pr_state_ptr->current_rec
#define retry_count                     pr_state_ptr->retry_count
#define mean_retry_intval               pr_state_ptr->mean_retry_intval
#define router_addr_seed                pr_state_ptr->router_addr_seed
#define tra_dist_table                  pr_state_ptr->tra_dist_table
#define para_rec                        pr_state_ptr->para_rec
#define parent_id                       pr_state_ptr->parent_id
#define node_name                       pr_state_ptr->node_name
#define tmp_queue_Objid                 pr_state_ptr->tmp_queue_Objid
#define prorec                          pr_state_ptr->prorec
#define procnum                         pr_state_ptr->procnum
#define audio_session_duration          pr_state_ptr->audio_session_duration
#define video_session_duration          pr_state_ptr->video_session_duration
#define req_type                        pr_state_ptr->req_type
#define gtw_addr_seed                   pr_state_ptr->gtw_addr_seed
#define para_hold                       pr_state_ptr->para_hold
#define tmp_q_Objid                     pr_state_ptr->tmp_q_Objid

/* This macro definition 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
#define FIN_PREAMBLE QoS_usr_usr_gen_state *op_sv_ptr = pr_state_ptr;
/* Function Block */
enum { _block_origin = __LINE__ };
static void
ss_usr_generate (void)
        {
        Prohandle       pro_handle;
        int             i;
        int             dest_router;
        int             dest_gtw;
        int             pro_id;
        int         Session_len;
        /** This function creates a usr based on the usr generation          **/
        /** specifications of the source model and sends it to the lower layer.   **/
        FIN (ss_usr_generate ());

        dest_router = (int)op_dist_outcome (router_addr_seed);
        dest_gtw=(int)op_dist_outcome (gtw_addr_seed);

        i = (int)op_dist_outcome (req_type);
   if ( i<AUDIO_VIDEO ) {
        Session_len = (int)op_dist_outcome (audio_session_duration);
        }
   else
        {
```

```
		Session_len = (int)op_dist_outcome (video_session_duration);
		}
		para_rec.Dest_router = dest_router;
		para_rec.Dest_gtw = dest_gtw;
		para_rec.Session_len =Session_len;     // (int)op_dist_outcome (session_duration);
		para_rec.Retry_count = MEAN_RETRY_COUNT;
		para_rec.Mean_retry_intval = MEAN_RETRY_INTVAL;
		para_rec.dest_queue_id=tmp_queue_Objid;
		para_rec.Msg_type=CreateNewUser;
		para_rec.Req_type=i;
		pro_handle = op_pro_create ("QoS_usr", OPC_NIL);
		pro_id=op_pro_id(pro_handle);
		prorec[procnum].ProcID=pro_id;
		prorec[procnum].Pro_handle=pro_handle;
		procnum=procnum+1;
		op_pro_invoke (pro_handle,&para_rec);

		//next user create intr schedule
		next_intarr_time = oms_dist_outcome (interarrival_dist_ptr);
		if(next_intarr_time<(double)1)
		next_intarr_time=(double)1;
   next_pk_evh= op_intrpt_schedule_self (op_sim_time () + next_intarr_time, SSC_GENERATE);
		FOUT;
		}
/* 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 QoS_usr_usr_gen (void);
		Compcode QoS_usr_usr_gen_init (void **);
		void QoS_usr_usr_gen_diag (void);
		void QoS_usr_usr_gen_terminate (void);
		void QoS_usr_usr_gen_svar (void *, const char *, char **);
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif

/* Process model interrupt handling procedure */
void
QoS_usr_usr_gen (void)
		{
		int _block_origin = 0;
		FIN (QoS_usr_usr_gen ());
		if (1)
			{
			/* Variables used in the "ptninit" state.                */
			char                    interarrival_str [128];
			char                    size_str [128];
			Prg_List*               pk_format_names_lptr;
			char*                   found_format_str;
			int                     low, high;
```

```
Boolean                         format_found;
int                             i;

/* Variables used in state transitions.*/
int                             intrpt_code;
int                             intrpt_type;

/* Variables used in the "init" state */
FILE*                           profile_hndl;
struct UsrReqPtn        ptn_rec;
int                             count;
UsrReqRec*                      tmp_ptr;
char                            file_name [128];
UsrTraDist                      dist_rec;
Packet*                 pkptr;
// Variables used in pktdeliver
int                             msg_type;
int                             src_gtw;
int                             src_router;
int                             dest_router;
int                             dest_gtw;
int                             route_id;
int                             vip_id;

int                             usr_id;
int                             proc_id;
int                             decision;
Prohandle               pro_handle;
int                             curr;
int                             tmp_proc_id;
BOOL                    found;
int                             bdw_used;
int                             req_class;
int                             Destination;
int                             time_est;
int                             response_time;
int                             diff_time;
Packet*                         hold_admit_pkptr;
Packet*                         pkptr1;
int                             req_time;

FSM_ENTER (QoS_usr_usr_gen)
FSM_BLOCK_SWITCH
        {
        /*-------------------------------------------------------*/
        /** state (ptninit) enter executives **/
        FSM_STATE_ENTER_UNFORCED (0, state0_enter_exec, "ptninit",
"QoS_usr_usr_gen () [ptninit enter execs]")
                {
        /* - set the state variables describing the packet generation parameters for later usage - */
                strcpy (interarrival_str, (current_rec->para_info).intarvl_ptn);
                strcpy (size_str, (current_rec->para_info).size_ptn);
                strcpy (format_str, PKT_FORMAT);
                start_time = (current_rec->para_info).ptn_start_time;
                stop_time = (current_rec->para_info).ptn_stop_time;
                /* --- move the current_rec forward by one step --- */
                current_rec = current_rec->next;
                /* Set the values of the packet generation parameters, i.e. the   */
                /* attribute values of the surrounding module.*/
```

```
                op_ima_obj_attr_set (own_id, "Packet Interarrival Time",
interarrival_str);
                op_ima_obj_attr_set (own_id, "Packet Size", size_str);
                op_ima_obj_attr_set (own_id, "Packet Format", format_str);
                op_ima_obj_attr_set (own_id, "Start Time", &start_time);
                op_ima_obj_attr_set (own_id, "Stop Time", &stop_time);

                /* Load the PDFs that will be used in computing the packet*/
                /* interarrival times and packet sizes.*/
                interarrival_dist_ptr = oms_dist_load_from_string
(interarrival_str);
                pksize_dist_ptr       = oms_dist_load_from_string (size_str);
                /* Verify the existence of the packet format to be used for*/
                /* generated packets.*/
                if (strcmp (format_str, "NONE") == 0)
                        {
                        /* We will generate unformatted packets. Set the flag.*/
                        generate_unformatted = OPC_TRUE;
                        }
                else
                        {
                        /* We will generate formatted packets. Turn off the
flag.*/
                        generate_unformatted = OPC_FALSE;

                        /* Get the list of all available packet formats.*/
                        pk_format_names_lptr = prg_tfile_name_list_get
(PrgC_Tfile_Type_Packet_Format);

                        /* Search the list for the requested packet format.*/
                        format_found = OPC_FALSE;
                        for (i = prg_list_size (pk_format_names_lptr);
((format_found == OPC_FALSE) && (i > 0)); i--)
                                {
                        /* Access the next format name and compare with */
                        /* requested format name.*/
                                found_format_str = (char *) prg_list_access
(pk_format_names_lptr, i - 1);
                                if (strcmp (found_format_str, format_str) == 0)
                                        format_found = OPC_TRUE;
                                }

                    if (format_found == OPC_FALSE)
                                {
                        /* The requested format does not exist. Generate*/
                        /* unformatted packets.*/
                                generate_unformatted = OPC_TRUE;

                        /* Display an appropriate warning.*/
                                op_prg_odb_print_major ("Warning from simple
packet generator model (simple_source):", "The specified packet format", format_str, "is not found.
Generating unformatted packets instead.", OPC_NIL);
                                }
                        /*Destroy the lits and its elements since we don't need it*/
                        /* anymore.*/
                        prg_list_free (pk_format_names_lptr);
                        prg_mem_free  (pk_format_names_lptr);
                        }
```

```
/* Make sure we have valid start and stop times, i.e. stop time is*/
/* not earlier than start time.*/
if ((stop_time <= start_time) && (stop_time != SSC_INFINITE_TIME))
	{
/* Stop time is earlier than start time. Disable the source.*/
	start_time = SSC_INFINITE_TIME;

/* Display an appropriate warning.*/
	op_prg_odb_print_major ("Warning from simple packet generator model (simple_source):", "Although the generator is not disabled (start time is set to a finite value),", "a stop time that is not later than the start time is specified.", "Disabling the generator.", OPC_NIL);
	}

/* Schedule a self interrupt that will indicate our start time for*/
/* packet generation activities. If the source is disabled, */
/* schedule it at current time with the appropriate code value.*/
if (start_time == SSC_INFINITE_TIME)
	{
	op_intrpt_schedule_self (op_sim_time (), SSC_PTN_STOP);
	}
else
	{
	op_intrpt_schedule_self (start_time, SSC_PTN_START);



/* In this case, also schedule the interrupt when we will stop*/
/* generating packets, unless we are configured to run until*/
/* the end of the simulation.*/
	if (stop_time != SSC_INFINITE_TIME)
		{
		op_intrpt_schedule_self (stop_time, SSC_PTN_STOP);
		}
	}
/* Register the statistics that will be maintained by this model.*/
	bits_sent_hndl	= op_stat_reg ("Generator.Traffic Sent (bits/sec)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
	packets_sent_hndl	= op_stat_reg ("Generator.Traffic Sent (packets/sec)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
	packet_size_hndl	= op_stat_reg ("Generator.Packet Size (bits)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
	interarrivals_hndl	= op_stat_reg ("Generator.Packet Interarrival Time (secs)", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
	printf("I'm QoS_usr_Usr_gen, ptnchange exit\n");
	//scanf("%i",&rkey);
	}
/** blocking after enter executives of unforced state. **/
FSM_EXIT (1,QoS_usr_usr_gen)
/** state (ptninit) exit executives **/
FSM_STATE_EXIT_UNFORCED (0, "ptninit", "QoS_usr_usr_gen () [ptninit exit execs]")
	{
/* set up the process attributes according to the no. of time event*/
/* Determine the code of the interrupt, which is used in evaluating*/
/* state transition conditions.*/
```

```
intrpt_code = op_intrpt_code ();
printf("Hello, I'm QoSpGtw_req_gen ptninit exit\n");
printf("current simulation time is %lf\n", op_sim_time());
//scanf("%i",&rkey);
}

/** state (ptninit) transition processing **/
FSM_INIT_COND (PTN_START)
FSM_TEST_COND (PTN_DISABLED)
FSM_TEST_LOGIC ("ptninit")
FSM_TRANSIT_SWITCH
{
FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;, "PTN_START", "",
"ptninit", "generatefirst")
FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ;, "PTN_DISABLED", "",
"ptninit", "wait")
}
/** state (generatefirst) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "generatefirst",
"QoS_usr_usr_gen () [generatefirst enter execs]")
{
/* At the enter execs of the "generate" state we schedule the*/
/* arrival of the next packet.*/
next_intarr_time = oms_dist_outcome (interarrival_dist_ptr);
next_pk_evh      = op_intrpt_schedule_self (op_sim_time () +
next_intarr_time, SSC_GENERATE);
own_id = op_id_self ();
parent_id = op_topo_parent (own_id);
op_ima_obj_attr_get (parent_id,
"queue_Objid",&tmp_queue_Objid);
}

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

/** state (generatefirst) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, "generatefirst", "QoS_usr_usr_gen ()
[generatefirst exit execs]")
{
/* Determine the code of the interrupt, which is used in */
/* evaluating state transition conditions.*/
intrpt_code = op_intrpt_code ();
}
/** state (generatefirst) transition processing **/
FSM_INIT_COND (USR_GENERATE)
FSM_TEST_COND (PTN_STOP)
FSM_TEST_LOGIC ("generatefirst")
FSM_TRANSIT_SWITCH
{
FSM_CASE_TRANSIT (0, 4, state4_enter_exec, ss_usr_generate();;,
"USR_GENERATE", "ss_usr_generate();", "generatefirst", "wait")
FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ;, "PTN_STOP", "",
"generatefirst", "wait")
}
/** state (init) enter executives **/
FSM_STATE_ENTER_UNFORCED_NOLABEL (2, "init",
"QoS_usr_usr_gen () [init enter execs]")
{
/* --- initializing state variables --- */
```

```
                                    procnum=0;
                                    for(i=0;i<MAX_USR_NUM;i++)
                                            {prorec[i].ProcID=0;}

                                    for (i=1; i<15; i++)
                                            node_name[i] = '\0';
                                    router_addr_seed = op_dist_load ("uniform_int",1,15);
                                    gtw_addr_seed    = op_dist_load ("uniform_int",0,9);
                                    retry_count = op_dist_load ("uniform_int",1,
MAX_RETRY_COUNT);
                                    req_type = op_dist_load ("uniform_int",0, 100);
                                    audio_session_duration = op_dist_load
("exponential",MEAN_AUDIO_SESSION_LEN,0);
                                    video_session_duration = op_dist_load
("exponential",MEAN_VIDEO_SESSION_LEN,0);
                                    pk_count = 0;
                                    /* Obtain the object id of the surrounding module.*/
                                    own_id = op_id_self ();
                                    parent_id = op_topo_parent (own_id);
                                    op_ima_obj_attr_get (parent_id, "name", node_name);
                                    strcpy (file_name, PROFILE_PATH);

                                    if
((node_name=="Users_1_0")||(node_name=="Users_5_0")||(node_name=="Users_11_0"))
                                         strcat (file_name, "HotSpot");
                                    else strcat (file_name, "Normalspot");
                            /* --- read in the user request pattern profile to form the pattern list --- */
                                    count = 0;
                                    if( (profile_hndl = fopen( file_name, "r" )) != NULL )
                                    {
                                    i = fread ((char *)&ptn_rec, sizeof( struct UsrReqPtn ), 1,
profile_hndl);
                                            while ( !feof (profile_hndl) ) {
                                                    tmp_ptr = (UsrReqRec *) malloc
( sizeof(UsrReqRec) );
                                                    count ++;
                            /* -- form the pattern list --- */
                            memcpy ( &(tmp_ptr->para_info), &(ptn_rec.ptn_item), sizeof( struct
PtnDef));
                                                    tmp_ptr->next = NULL;
                                                    if ( count == 1 ) {
                                                            ptn_list = tmp_ptr;
                                                            current_rec = tmp_ptr;
                                                    }
                                                    else
                                                            {
                                                            current_rec->next = tmp_ptr;
                                                            current_rec = tmp_ptr;
                                                            }

if((node_name=="Users_1_0")||(node_name=="Users_5_0")||(node_name=="Users_11_0"))
                                    {printf("interarrival pattern is %s\n", ptn_rec.ptn_item.intarvl_ptn);
                                    printf("size pattern is %s\n", ptn_rec.ptn_item.size_ptn);
                                    printf("start time is %lf\n", ptn_rec.ptn_item.ptn_start_time);
                                    printf("stop time is %lf\n", ptn_rec.ptn_item.ptn_stop_time);
                                    printf("pattern change time is %lf\n", ptn_rec.ptn_chg_time);
                                    printf("current simulation time is %lf\n", op_sim_time());
                                    scanf("%i", &rkey);
                                    }
```

```
                                        /* -- schedual the pattern change interruption --- */
                                        op_intrpt_schedule_self (ptn_rec.ptn_chg_time, SSC_CHANGE);
                                        i = fread ((char *)&ptn_rec, sizeof( struct UsrReqPtn ), 1,
profile_hndl);
                                        }
                                        current_rec = ptn_list;
                                        fclose(profile_hndl);
                                        }
                                        printf("usr gen init exit");
                                        //scanf("%i", &rkey);
                                        }
                                /** blocking after enter executives of unforced state. **/
                                FSM_EXIT (5,QoS_usr_usr_gen)
                                /** state (init) exit executives **/
                                FSM_STATE_EXIT_UNFORCED (2, "init", "QoS_usr_usr_gen () [init exit
execs]")
                                        {
                                /* Determine the code of the interrupt, which is used in evaluating*/
                                /* state transition conditions.*/
                                        intrpt_code = op_intrpt_code ();
                                        printf("Hello, I'm QoSpGtw_req_gen init exit\n");
                                        printf("current simulation time is %lf\n", op_sim_time());
                                        }
                                /** state (init) transition processing **/
                                FSM_TRANSIT_ONLY ((PATTERN_CHANGE), 0, state0_enter_exec, ;,
"init", "PATTERN_CHANGE", "", "init", "ptninit")
                                /*---------------------------------------------------------*/
                                /** state (pktdeliver) enter executives **/
                                FSM_STATE_ENTER_FORCED (3, state3_enter_exec, "pktdeliver",
"QoS_usr_usr_gen () [pktdeliver enter execs]")
                                        {
                                        pkptr = op_pk_get (op_intrpt_strm ());
                                        op_pk_nfd_get (pkptr, "Msg_type", &msg_type);
                                        switch(msg_type)
                                                {
                                                case DESTROY_USR_PROC:
                                                        op_pk_nfd_get (pkptr, "Proc_id", &tmp_proc_id);
                                                        curr=0;
                                                        found=FALSE;

                                                        while((found==FALSE) && (curr<procnum))
                                                                { if (prorec[curr].ProcID!=tmp_proc_id)
                                                                        curr++;
                                                                  else found=TRUE;
                                                                }
                                                        if (found==TRUE)
                                                         { for (i=curr;i<=procnum;i++)
        { prorec[i].ProcID=prorec[i+1].ProcID;
                                                           prorec[i].Pro_handle=prorec[i+1].Pro_handle;
                                                         }
                                                         procnum--;
                                                         }
                                                         else
                                                         { printf("error occured no this proc to destroy");
                                                           scanf("%i",&rkey);
                                                         }
                                                         op_pk_destroy (pkptr);
                                                break;
```

```
                                        case EDGE_HLD_ADMITTED:
                                        op_ima_obj_attr_get (parent_id,
"queue_Objid",&tmp_q_Objid);
                                        op_pk_nfd_get (pkptr, "Src_gtw", &src_gtw);
                                        op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
                                        op_pk_nfd_get (pkptr, "Vip_id",  &vip_id);
                                        op_pk_nfd_get (pkptr, "Decision",  &decision);
                                        op_pk_nfd_get (pkptr, "Usr_id",  &usr_id);
                                        op_pk_nfd_get (pkptr, "Proc_id", &proc_id);
                                        op_pk_nfd_get (pkptr, "Req_type", &req_class);
                                        op_pk_nfd_get (pkptr, "Req_time", &req_time);
                                        para_hold.Src_gtw=src_gtw;
                                        para_hold.Dest_gtw=dest_gtw;
                                        para_hold.Vip_id=vip_id;
                                        para_hold.Usr_id=usr_id;
                                        para_hold.Proc_id=proc_id;
                                        para_hold.Decision=decision;
                                        para_hold.Req_type=req_class;
                                        para_hold.Reqtime=req_time;

                                        i=0;
                                        while(i<procnum)
                                        {//printf("prorec %i: %i",i,prorec[i].ProcID);
                                         if (prorec[i].ProcID==proc_id)
                                                    { pro_handle=prorec[i].Pro_handle;
                                                    break;}
                                        else i++;
                                                     }
                                        para_hold.Msg_type=msg_type;
                                        op_pk_destroy (pkptr);
                                        op_pro_invoke (pro_handle, &para_hold);
                                        hold_admit_pkptr = op_pk_create_fmt
(QOS_EDGE_HLD_ADM);
                                        op_pk_total_size_set (hold_admit_pkptr, 1024);
                                        op_pk_nfd_set (hold_admit_pkptr, "Msg_type",
EDGE_HLD_REP);
                                        op_pk_nfd_set (hold_admit_pkptr, "Usr_id",
para_hold.Usr_id);
                                        op_pk_nfd_set (hold_admit_pkptr, "Vip_id",
para_hold.Vip_id);
                                        op_pk_nfd_set (hold_admit_pkptr, "Proc_id",
para_hold.Proc_id);
                                        op_pk_nfd_set (hold_admit_pkptr, "Req_type",
para_hold.Req_type);
                                        op_pk_nfd_set (hold_admit_pkptr, "Decision",
HOLD_ACCEPT);
                                        op_pk_deliver (hold_admit_pkptr, tmp_q_Objid, 0);
                                        break;

                                         default:
                                        op_pk_nfd_get (pkptr, "Src_router", &src_router);
                                        op_pk_nfd_get (pkptr, "Src_gtw", &src_gtw);
                                        op_pk_nfd_get (pkptr, "Dest_router", &dest_router);
                                        op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
                                        op_pk_nfd_get (pkptr, "Route_id", &route_id);
                                        op_pk_nfd_get (pkptr, "Vip_id",  &vip_id);
                                        op_pk_nfd_get (pkptr, "Decision",  &decision);
                                        op_pk_nfd_get (pkptr, "Usr_id",  &usr_id);
                                        op_pk_nfd_get (pkptr, "Proc_id", &proc_id);
                                        op_pk_nfd_get (pkptr, "Req_time", &req_time);
```

```
                                        i=0;
                                        while(i<procnum)
                                                {
                                        if (prorec[i].ProcID==proc_id)
                                                { pro_handle=prorec[i].Pro_handle;
                                                break;
                                                }
                                        else i++;
                                                }
                                        para_rec.Msg_type=msg_type;
                                        para_rec.Src_router=src_router;
                                        para_rec.Src_gtw=src_gtw;
                                        para_rec.Dest_router=dest_router;
                                        para_rec.Dest_gtw=dest_gtw;
                                        para_rec.Route_id=route_id;
                                        para_rec.Vip_id=vip_id;
                                        para_rec.Usr_id=usr_id;
                                        para_rec.Proc_id=proc_id;
                                        para_rec.Decision=decision;
                                        para_rec.Reqtime=req_time;
                                        op_pk_destroy (pkptr);
                                        op_pro_invoke (pro_handle, &para_rec);
                                        break;
                                }
                                }
                        /** state (pktdeliver) exit executives **/
                        FSM_STATE_EXIT_FORCED (3, "pktdeliver", "QoS_usr_usr_gen ()
[pktdeliver exit execs]")
                                {
                                }

                        /** state (pktdeliver) transition processing **/
                        FSM_TRANSIT_FORCE (4, state4_enter_exec, ;, "default", "",
"pktdeliver", "wait")
                        /*---------------------------------------------------------*/
                        /** state (wait) enter executives **/
                        FSM_STATE_ENTER_UNFORCED (4, state4_enter_exec, "wait",
"QoS_usr_usr_gen () [wait enter execs]")
                                {
                                if(op_intrpt_type () == OPC_INTRPT_SELF && op_intrpt_code()
== SSC_PTN_STOP)
                                {//printf("pattern stop");
                                 // scanf("%i", &rkey);
                                  if (op_ev_valid (next_pk_evh) == OPC_TRUE)
                                        {
                                        op_ev_cancel (next_pk_evh);
                                        }
                                  }
                                if(op_intrpt_type () == OPC_INTRPT_SELF && op_intrpt_code()
== SSC_CHANGE)
                                { //printf("pattern change");
                                  //scanf("%i", &rkey);
                                  strcpy (interarrival_str, (current_rec->para_info).intarvl_ptn);
                                  strcpy (size_str, (current_rec->para_info).size_ptn);
                                  strcpy (format_str, PKT_FORMAT);
                                  start_time = (current_rec->para_info).ptn_start_time;
                                  stop_time = (current_rec->para_info).ptn_stop_time;
                                  /* --- move the current_rec forward by one step --- */
                                  current_rec = current_rec->next;
```

```
                                /* Set the values of the packet generation parameters, i.e. the*/
                                /* attribute values of the surrounding module.          */
                                op_ima_obj_attr_set (own_id, "Packet Interarrival Time",
interarrival_str);
                                op_ima_obj_attr_set (own_id, "Packet Size",             size_str);
                                op_ima_obj_attr_set (own_id, "Packet Format",
format_str);
                                op_ima_obj_attr_set (own_id, "Start Time",
&start_time);
                                op_ima_obj_attr_set (own_id, "Stop Time",
&stop_time);
                              /* Load the PDFs that will be used in computing the packet*/
                              /* interarrival times and packet sizes.*/
                              interarrival_dist_ptr = oms_dist_load_from_string
(interarrival_str);
                              pksize_dist_ptr       = oms_dist_load_from_string (size_str);
                              /* Verify the existence of the packet format to be used for*/
                              /* generated packets.*/
                              if (strcmp (format_str, "NONE") == 0)
                                      {
                              /* We will generate unformatted packets. Set the flag.*/
                                      generate_unformatted = OPC_TRUE;
                                      }
                              else
                                      {
                              /* We will generate formatted packets. Turn off the flag.*/
                                      generate_unformatted = OPC_FALSE;
                              /* Get the list of all available packet formats.*/
                                      pk_format_names_lptr = prg_tfile_name_list_get
(PrgC_Tfile_Type_Packet_Format);
                              /* Search the list for the requested packet format.*/
                                      format_found = OPC_FALSE;
                                      for (i = prg_list_size (pk_format_names_lptr);
((format_found == OPC_FALSE) && (i > 0)); i--)
                                              {
                              /* Access the next format name and compare with requested*/
                              /* format name.*/
                                      found_format_str = (char *) prg_list_access
(pk_format_names_lptr, i - 1);
                                              if (strcmp (found_format_str, format_str) == 0)
                                                      format_found = OPC_TRUE;
                                              }

                                    if (format_found == OPC_FALSE)
                                              {
                              /* The requested format does not exist. Generate*/
                              /* unformatted packets.*/
                                              generate_unformatted = OPC_TRUE;
                              /* Display an appropriate warning.*/
                                              op_prg_odb_print_major ("Warning from simple
packet generator model (simple_source):", "The specified packet format", format_str, "is not found.
Generating unformatted packets instead.", OPC_NIL);
      }
                              /* Destroy the lits and its elements since we don't need */
                              /* it anymore.*/
                                      prg_list_free (pk_format_names_lptr);
                                      prg_mem_free  (pk_format_names_lptr);
                                      }
                              /* Make sure we have valid start and stop times,*/
```

```
/* i.e. stop time is not earlier than start time.*/
if ((stop_time <= start_time) && (stop_time != SSC_INFINITE_TIME))
	{
/* Stop time is earlier than start time. Disable the source.*/
	start_time = SSC_INFINITE_TIME;
/* Display an appropriate warning.*/
	op_prg_odb_print_major ("Warning from simple packet generator model (simple_source):", "Although the generator is not disabled (start time is set to a finite value),", "a stop time that is not later than the start time is specified.", "Disabling the generator.", OPC_NIL);}
/* Schedule a self interrupt that will indicate our start time for*/
/* packet generation activities. If the source is disabled,*/
/* schedule it at current time with the appropriate code value.*/
if (start_time == SSC_INFINITE_TIME)
	{
	op_intrpt_schedule_self (op_sim_time (), SSC_PTN_STOP);
	}
else
	{
next_intarr_time = oms_dist_outcome (interarrival_dist_ptr);
next_pk_evh      = op_intrpt_schedule_self (op_sim_time () + next_intarr_time, SSC_GENERATE);
/* In this case, also schedule the interrupt when we will */
/* stop generating packets, unless we are configured to */
/* run until the end of the simulation.*/
	if (stop_time != SSC_INFINITE_TIME)
		{
op_intrpt_schedule_self (stop_time, SSC_PTN_STOP);
		}
	}
  }
}
/** blocking after enter executives of unforced state. **/
FSM_EXIT (9,QoS_usr_usr_gen)
/** state (wait) exit executives **/
FSM_STATE_EXIT_UNFORCED (4, "wait", "QoS_usr_usr_gen () [wait exit execs]")
	{
	}
/** state (wait) transition processing **/
FSM_INIT_COND (PKT_ARRIVAL)
FSM_TEST_COND (USR_GENERATE)
FSM_TEST_COND (PTN_STOP)
FSM_TEST_COND (PATTERN_CHANGE)
FSM_TEST_LOGIC ("wait")
FSM_TRANSIT_SWITCH
	{
	FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;, "PKT_ARRIVAL", "", "wait", "pktdeliver")
	FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ss_usr_generate();;, "USR_GENERATE", "ss_usr_generate();", "wait", "wait")
	FSM_CASE_TRANSIT (2, 4, state4_enter_exec, ;, "PTN_STOP", "", "wait", "wait")
	FSM_CASE_TRANSIT (3, 4, state4_enter_exec, ;, "PATTERN_CHANGE", "", "wait", "wait")
	}
	/*---------------------------------------------------------*/
```

```
                    }
                FSM_EXIT (2,QoS_usr_usr_gen)
                }
        }
#if defined (__cplusplus)
        extern "C" {
#endif
        extern VosT_Fun_Status Vos_Catmem_Register (const char * , int , VosT_Void_Null_Proc,
VosT_Address *);
        extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
        extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
#if defined (__cplusplus)
        }
#endif
Compcode
QoS_usr_usr_gen_init (void ** gen_state_pptr)
        {
        int _block_origin = 0;
        static VosT_Address         obtype = OPC_NIL;
        FIN (QoS_usr_usr_gen_init (gen_state_pptr))
        if (obtype == OPC_NIL)
                {
                /* Initialize memory management */
                if (Vos_Catmem_Register ("proc state vars (QoS_usr_usr_gen)",
                        sizeof (QoS_usr_usr_gen_state), Vos_Vnop, &obtype) ==
VOSC_FAILURE)
                        {
                        FRET (OPC_COMPCODE_FAILURE)
                        }
                }
        *gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
        if (*gen_state_pptr == OPC_NIL)
                {
                FRET (OPC_COMPCODE_FAILURE)
                }
        else
                {
                /* Initialize FSM handling */
                ((QoS_usr_usr_gen_state *)(*gen_state_pptr))->current_block = 4;
                FRET (OPC_COMPCODE_SUCCESS)
                }
        }
void
QoS_usr_usr_gen_diag (void)
        {
        /* No Diagnostic Block */
        }
void
QoS_usr_usr_gen_terminate (void)
        {
        int _block_origin = __LINE__;
        FIN (QoS_usr_usr_gen_terminate (void))
        if (1)
                {
                char                    interarrival_str [128];
                char                    size_str [128];
                Prg_List*               pk_format_names_lptr;
                char*                   found_format_str;
                int                     low, high;
                Boolean                 format_found;
```

```
                int                     i;
                int                     intrpt_code;
                int                     intrpt_type;
                FILE*                   profile_hndl;
                struct UsrReqPtn        ptn_rec;
                int                     count;
                UsrReqRec*              tmp_ptr;
                char                    file_name [128];
                UsrTraDist              dist_rec;
                Packet*                 pkptr;
                int                     msg_type;
                int                     src_gtw;
                int                     src_router;
                int                     dest_router;
                int                     dest_gtw;
                int                     route_id;
                int                     vip_id;
                int                     usr_id;
                int                     proc_id;
                int                     decision;
                Prohandle               pro_handle;
                int                     curr;
                int                     tmp_proc_id;
                BOOL                    found;
                int                     bdw_used;
                int                     req_class;
                int                     Destination;
                int                     time_est;
                int                     response_time;
                int                     diff_time;
                Packet*                 hold_admit_pkptr;
                Packet*                 pkptr1;
                int                     req_time;
                /* No Termination Block */
                }
        Vos_Catmem_Dealloc (pr_state_ptr);
        FOUT;
        }
/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in QoS_usr_usr_gen_svar function. */
#undef own_id
#undef format_str
#undef start_time
#undef stop_time
#undef interarrival_dist_ptr
#undef pksize_dist_ptr
#undef generate_unformatted
#undef next_pk_evh
#undef next_intarr_time
#undef bits_sent_hndl
#undef packets_sent_hndl
#undef packet_size_hndl
#undef interarrivals_hndl
#undef rkey
#undef pk_count
#undef ptn_table
#undef ptn_list
#undef current_rec
#undef retry_count
```

```
#undef mean_retry_intval
#undef router_addr_seed
#undef tra_dist_table
#undef para_rec
#undef parent_id
#undef node_name
#undef tmp_queue_Objid
#undef prorec
#undef procnum
#undef audio_session_duration
#undef video_session_duration
#undef req_type
#undef gtw_addr_seed
#undef para_hold
#undef tmp_q_Objid
void
QoS_usr_usr_gen_svar (void * gen_ptr, const char * var_name, char ** var_p_ptr)
	{
	QoS_usr_usr_gen_state		*prs_ptr;

	FIN (QoS_usr_usr_gen_svar (gen_ptr, var_name, var_p_ptr))

	if (var_name == OPC_NIL)
		{
		*var_p_ptr = (char *)OPC_NIL;
		FOUT;
		}
	prs_ptr = (QoS_usr_usr_gen_state *)gen_ptr;

	if (strcmp ("own_id" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->own_id);
		FOUT;
		}
	if (strcmp ("format_str" , var_name) == 0)
		{
		*var_p_ptr = (char *) (prs_ptr->format_str);
		FOUT;
		}
	if (strcmp ("start_time" , var_name) == 0)

		{
		*var_p_ptr = (char *) (&prs_ptr->start_time);
		FOUT;
		}
	if (strcmp ("stop_time" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->stop_time);
		FOUT;
		}
	if (strcmp ("interarrival_dist_ptr" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->interarrival_dist_ptr);
		FOUT;
		}
	if (strcmp ("pksize_dist_ptr" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->pksize_dist_ptr);
		FOUT;
```

```
        }
if (strcmp ("generate_unformatted" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->generate_unformatted);
        FOUT;
        }
if (strcmp ("next_pk_evh" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->next_pk_evh);
        FOUT;
        }
if (strcmp ("next_intarr_time" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->next_intarr_time);
        FOUT;
        }
if (strcmp ("bits_sent_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->bits_sent_hndl);
        FOUT;
        }
if (strcmp ("packets_sent_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->packets_sent_hndl);
        FOUT;
        }
if (strcmp ("packet_size_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->packet_size_hndl);
        FOUT;
        }
if (strcmp ("interarrivals_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->interarrivals_hndl);
        FOUT;
        }
if (strcmp ("rkey" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->rkey);
        FOUT;
        }
if (strcmp ("pk_count" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->pk_count);
        FOUT;
        }
if (strcmp ("ptn_table" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->ptn_table);
        FOUT;
        }
if (strcmp ("ptn_list" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->ptn_list);
        FOUT;
        }
if (strcmp ("current_rec" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->current_rec);
```

```
            FOUT;
            }
if (strcmp ("retry_count" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->retry_count);
            FOUT;
            }
if (strcmp ("mean_retry_intval" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->mean_retry_intval);
            FOUT;
            }
if (strcmp ("router_addr_seed" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->router_addr_seed);
            FOUT;
            }
if (strcmp ("tra_dist_table" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->tra_dist_table);
            FOUT;
            }
if (strcmp ("para_rec" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->para_rec);
            FOUT;
            }
if (strcmp ("parent_id" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->parent_id);
            FOUT;
            }
if (strcmp ("node_name" , var_name) == 0)

            {
            *var_p_ptr = (char *) (prs_ptr->node_name);
            FOUT;
            }
if (strcmp ("tmp_queue_Objid" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->tmp_queue_Objid);
            FOUT;
            }
if (strcmp ("prorec" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->prorec);
            FOUT;
            }
if (strcmp ("procnum" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->procnum);
            FOUT;
            }
if (strcmp ("audio_session_duration" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->audio_session_duration);
            FOUT;
            }
if (strcmp ("video_session_duration" , var_name) == 0)
```

```
        {
        *var_p_ptr = (char *) (&prs_ptr->video_session_duration);
        FOUT;
        }
if (strcmp ("req_type" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->req_type);
        FOUT;
        }
if (strcmp ("gtw_addr_seed" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->gtw_addr_seed);
        FOUT;
        }
if (strcmp ("para_hold" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->para_hold);
        FOUT;
        }
if (strcmp ("tmp_q_Objid" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->tmp_q_Objid);
        FOUT;
        }
*var_p_ptr = (char *)OPC_NIL;
FOUT;
}
```

```
/**************************************************************************/
     /*queue module*/
/**************************************************************************/
/* This variable carries the header into the object file */
static const char acb_fifo_pr_c [] = "MIL_3_Tfile_Hdr_ 80C 30A op_runsim 7 441CF089
441CF089 1 KiT KiT 0 0 none none 0 0 none 0 0 0 0 0 0";
#include <string.h>
/* OPNET system definitions */
#include <opnet.h>
#if defined (__cplusplus)
extern "C" {
#endif
FSM_EXT_DECS
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Header Block */
#define QUEUE_EMPTY     (op_q_empty ())
#define SVC_COMPLETION  op_intrpt_type () == OPC_INTRPT_SELF
#define ARRIVAL         op_intrpt_type () == OPC_INTRPT_STRM
/* End of Header Block */

#if !defined (VOSD_NO_FIN)
#undef   BIN
#undef   BOUT
#define  BIN                FIN_LOCAL_FIELD(last_line_passed) = __LINE__ - _block_origin;
#define  BOUT   BIN
#define  BINIT  FIN_LOCAL_FIELD(last_line_passed) = 0; _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 */
        int                     server_busy;
        double                  service_rate;
        Objid                   own_id;
        } acb_fifo_state;
#define pr_state_ptr            ((acb_fifo_state*) SimI_Mod_State_Ptr)
#define server_busy             pr_state_ptr->server_busy
#define service_rate            pr_state_ptr->service_rate
#define own_id                  pr_state_ptr->own_id
/* This macro definition 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
#define FIN_PREAMBLE acb_fifo_state *op_sv_ptr = pr_state_ptr;
/* No Function Block */
enum { _block_origin = __LINE__ };
/* 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 acb_fifo (void);
	Compcode acb_fifo_init (void **);
	void acb_fifo_diag (void);
	void acb_fifo_terminate (void);
	void acb_fifo_svar (void *, const char *, char **);
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Process model interrupt handling procedure */
void
acb_fifo (void)
	{
	int _block_origin = 0;
	FIN (acb_fifo ());
	if (1)
		{
		Packet*	pkptr;
		int	pk_len;
		double	pk_svc_time;
		int	insert_ok;
		FSM_ENTER (acb_fifo)
		FSM_BLOCK_SWITCH
			{
			/*---------------------------------------------------------*/
			/** state (init) enter executives **/
			FSM_STATE_ENTER_FORCED_NOLABEL (0, "init", "acb_fifo () [init
enter execs]")
				{
				/* initially the server is idle*/
				server_busy = 0;

				/* get queue module's own object id*/
				own_id = op_id_self ();

				/* get assigned value of server*/
				/* processing rate*/
				op_ima_obj_attr_get (own_id, "service_rate", &service_rate);

				}
			/** state (init) exit executives **/
			FSM_STATE_EXIT_FORCED (0, "init", "acb_fifo () [init exit execs]")
				{
				}
			/** state (init) transition processing **/
			FSM_INIT_COND (ARRIVAL)
			FSM_DFLT_COND
			FSM_TEST_LOGIC ("init")
			FSM_TRANSIT_SWITCH
				{
				FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;, "ARRIVAL",
"", "init", "arrival")
				FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;, "default", "",
"init", "idle")
				}
```

```
/*---------------------------------------------------------*/
/** state (arrival) enter executives **/
FSM_STATE_ENTER_FORCED (1, state1_enter_exec, "arrival",
"acb_fifo () [arrival enter execs]")
        {
/* acquire the arriving packet*/
/* multiple arriving streams are supported. */
        pkptr = op_pk_get (op_intrpt_strm ());

/* attempt to enqueue the packet at tail*/
/* of subqueue 0.*/
        if (op_subq_pk_insert (0, pkptr, OPC_QPOS_TAIL) !=
OPC_QINS_OK)
          {
/* the inserton failed (due to to a*/
/* full queue) deallocate the packet.*/
          op_pk_destroy (pkptr);
/* set flag indicating insertion fail this flag is used to determine*/
/* transition out of this state*/
          insert_ok = 0;
          }
        else{
/* insertion was successful*/
          insert_ok = 1;
          }
        }
/** state (arrival) exit executives **/
FSM_STATE_EXIT_FORCED (1, "arrival", "acb_fifo () [arrival exit
execs]")
        {
        }
/** state (arrival) transition processing **/
FSM_INIT_COND (!server_busy && insert_ok)
FSM_DFLT_COND
FSM_TEST_LOGIC ("arrival")
FSM_TRANSIT_SWITCH
        {
FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;, "!server_busy &&
insert_ok", "", "arrival", "svc_start")
FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;, "default", "", "arrival",
"idle")
        }
/*---------------------------------------------------------*/
/** state (idle) enter executives **/
FSM_STATE_ENTER_UNFORCED (2, state2_enter_exec, "idle",
"acb_fifo () [idle enter execs]")
        {
        }
/** blocking after enter executives of unforced state. **/
FSM_EXIT (5,acb_fifo)
/** state (idle) exit executives **/
FSM_STATE_EXIT_UNFORCED (2, "idle", "acb_fifo () [idle exit
execs]")
        {
        }
/** state (idle) transition processing **/
FSM_INIT_COND (ARRIVAL)
FSM_TEST_COND (SVC_COMPLETION)
FSM_TEST_LOGIC ("idle")
```

```
                        FSM_TRANSIT_SWITCH
                                {
                        FSM_CASE_TRANSIT (0, 1, state1_enter_exec, ;, "ARRIVAL", "", "idle",
"arrival")
                        FSM_CASE_TRANSIT (1, 4, state4_enter_exec, ;,
"SVC_COMPLETION", "", "idle", "svc_compl")
                                }
                        /*---------------------------------------------------------*/
                        /** state (svc_start) enter executives **/
                        FSM_STATE_ENTER_FORCED (3, state3_enter_exec, "svc_start",
"acb_fifo () [svc_start enter execs]")
                                {
                        /* get a handle on packet at head of subqueue 0 */
                        /* (this does not remove the packet)*/
                                pkptr = op_subq_pk_access (0, OPC_QPOS_HEAD);
                        /* determine the packets length (in bits)*/
                                pk_len = op_pk_total_size_get (pkptr);
                        /* determine the time required to complete*/
                        /* service of the packet      */
                                pk_svc_time = pk_len / service_rate;
                        /* schedule an interrupt for this process        */
                        /* at the time where service ends.*/
                                op_intrpt_schedule_self (op_sim_time () + pk_svc_time, 0);
                        /* the server is now busy.*/
                                server_busy = 1;
                                }
                        /** state (svc_start) exit executives **/
                        FSM_STATE_EXIT_FORCED (3, "svc_start", "acb_fifo () [svc_start exit
execs]")
                                {
                                }
                        /** state (svc_start) transition processing **/
                        FSM_TRANSIT_FORCE (2, state2_enter_exec, ;, "default", "", "svc_start",
"idle")
                        /*---------------------------------------------------------*/
                        /** state (svc_compl) enter executives **/
                        FSM_STATE_ENTER_FORCED (4, state4_enter_exec, "svc_compl",
"acb_fifo () [svc_compl enter execs]")
                                {
                        /* extract packet at head of queue; this         */
                        /* is the packet just finishing service          */
                                pkptr = op_subq_pk_remove (0, OPC_QPOS_HEAD);
                        /* forward the packet on stream 0, causing   */
                        /* an immediate interrupt at destination.       */
                                op_pk_send_forced (pkptr, 0);
                        /* server is idle again.*/
                                server_busy = 0;
                                }
                        /** state (svc_compl) exit executives **/
                        FSM_STATE_EXIT_FORCED (4, "svc_compl", "acb_fifo () [svc_compl
exit execs]")
                                {
                                }

                        /** state (svc_compl) transition processing **/
                        FSM_INIT_COND (!QUEUE_EMPTY)
                        FSM_DFLT_COND
                        FSM_TEST_LOGIC ("svc_compl")
```

```
                        FSM_TRANSIT_SWITCH
                                {
                        FSM_CASE_TRANSIT (0, 3, state3_enter_exec, ;, "!QUEUE_EMPTY",
"", "svc_compl", "svc_start")
                        FSM_CASE_TRANSIT (1, 2, state2_enter_exec, ;, "default", "",
"svc_compl", "idle")
                                }
                        /*--------------------------------------------------------*/
                        }
                FSM_EXIT (0,acb_fifo)
                }
        }
#if defined (__cplusplus)
        extern "C" {
#endif
        extern VosT_Fun_Status Vos_Catmem_Register (const char * , int , VosT_Void_Null_Proc,
VosT_Address *);
        extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
        extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
#if defined (__cplusplus)
        }
#endif
Compcode
acb_fifo_init (void ** gen_state_pptr)
        {
        int _block_origin = 0;
        static VosT_Address         obtype = OPC_NIL;
        FIN (acb_fifo_init (gen_state_pptr))
        if (obtype == OPC_NIL)
                {
                /* Initialize memory management */
                if (Vos_Catmem_Register ("proc state vars (acb_fifo)",
                        sizeof (acb_fifo_state), Vos_Vnop, &obtype) == VOSC_FAILURE)
                        {
                        FRET (OPC_COMPCODE_FAILURE)
                        }
                }

        *gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
        if (*gen_state_pptr == OPC_NIL)
                {
                FRET (OPC_COMPCODE_FAILURE)
                }
        else
                {
                /* Initialize FSM handling */
                ((acb_fifo_state *)(*gen_state_pptr))->current_block = 0;

                FRET (OPC_COMPCODE_SUCCESS)
                }
        }
void
acb_fifo_diag (void)
        {
        /* No Diagnostic Block */
        }
void
acb_fifo_terminate (void)
        {
        int _block_origin = __LINE__;
```

```
	FIN (acb_fifo_terminate (void))
	if (1)
		{
		Packet*		pkptr;
		int		pk_len;
		double		pk_svc_time;
		int		insert_ok;
		/* No Termination Block */
		}
	Vos_Catmem_Dealloc (pr_state_ptr);
	FOUT;
	}
/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in acb_fifo_svar function. */
#undef server_busy
#undef service_rate
#undef own_id
void
acb_fifo_svar (void * gen_ptr, const char * var_name, char ** var_p_ptr)
	{
	acb_fifo_state		*prs_ptr;
	FIN (acb_fifo_svar (gen_ptr, var_name, var_p_ptr))
	if (var_name == OPC_NIL)
		{
		*var_p_ptr = (char *)OPC_NIL;
		FOUT;
		}
	prs_ptr = (acb_fifo_state *)gen_ptr;
	if (strcmp ("server_busy" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->server_busy);
		FOUT;
		}
	if (strcmp ("service_rate" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->service_rate);
		FOUT;
		}
	if (strcmp ("own_id" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->own_id);
		FOUT;
		}
	*var_p_ptr = (char *)OPC_NIL;
	FOUT;
	}
```

```
/**************************************************************************/
     /*proc module*/
/**************************************************************************/
/* This variable carries the header into the object file */
static const char QoS_usr_proc_pr_c [] = "MIL_3_Tfile_Hdr_ 80C 30A op_runsim 7 441CF08B
441CF08B 1 KiT KiT 0 0 none none 0 0 none 0 0 0 0 0 0";
#include <string.h>
/* OPNET system definitions */
#include <opnet.h>
#if defined (__cplusplus)
extern "C" {
#endif
FSM_EXT_DECS
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Header Block */
/* --- include the head files create by myself --- */
#include "../headfile/McrDef.h"
#include "../headfile/PolicySvr_user.h"
/* transition macros */
#define PKT_ARRVL ( op_intrpt_type () == OPC_INTRPT_STRM )
/* output stream */
#define XMT_OUT   0
/* global variables
IdRec               IdRec_table[MAX_USR_NUM];
int                 curr_num;
*/
/* End of Header Block */
#if !defined (VOSD_NO_FIN)
#undef   BIN
#undef   BOUT
#define  BIN      FIN_LOCAL_FIELD(last_line_passed) = __LINE__ - _block_origin;
#define  BOUT     BIN
#define  BINIT    FIN_LOCAL_FIELD(last_line_passed) = 0; _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 */
          Stathandle                    ete_gsh;
          Objid                         own_id;
          int                           pkt_count;
          Stathandle                    connection_setup_hndl;
          Objid                         parent_id;
          char                          node_name [15];
          Stathandle                    local_connection_setup_hndl;
          Stathandle                    response_time_hndl;
          } QoS_usr_proc_state;
#define pr_state_ptr                    ((QoS_usr_proc_state*) SimI_Mod_State_Ptr)
#define ete_gsh                         pr_state_ptr->ete_gsh
#define own_id                          pr_state_ptr->own_id
#define pkt_count                       pr_state_ptr->pkt_count
#define connection_setup_hndl           pr_state_ptr->connection_setup_hndl
#define parent_id                       pr_state_ptr->parent_id
```

```
#define node_name                       pr_state_ptr->node_name
#define local_connection_setup_hndl     pr_state_ptr->local_connection_setup_hndl
#define response_time_hndl              pr_state_ptr->response_time_hndl
/* This macro definition 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
#define FIN_PREAMBLE QoS_usr_proc_state *op_sv_ptr = pr_state_ptr;
/* No Function Block */
enum { _block_origin = __LINE__ };
/* 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 QoS_usr_proc (void);
        Compcode QoS_usr_proc_init (void **);
        void QoS_usr_proc_diag (void);
        void QoS_usr_proc_terminate (void);
        void QoS_usr_proc_svar (void *, const char *, char **);
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Process model interrupt handling procedure */
void
QoS_usr_proc (void)
        {
        int _block_origin = 0;
        FIN (QoS_usr_proc ());
        if (1)
                {
                Packet*                 pkptr;
                int                     msg_type;
                double                  ete_delay;
                int                     rkey;
                int                     count;
                int                     i;
                int                     j;
                int                     usr_id;
                Boolean                 id_found;
                Objid                   usr_op_id;
                Objid                   tmp_queue_Objid;
                double                  connection_time;
                FSM_ENTER (QoS_usr_proc)
                FSM_BLOCK_SWITCH
                        {
                        /*---------------------------------------------------------*/
                        /** state (init) enter executives **/
                        FSM_STATE_ENTER_FORCED_NOLABEL (0, "init", "QoS_usr_proc ()
[init enter execs]")
                                {
                                for (i=0; i<15; i++)
                                node_name[i]='\0';
```

```
own_id = op_id_self ();
parent_id = op_topo_parent (own_id);
op_ima_obj_attr_get (parent_id, "name", node_name);
pkt_count = 0;
response_time_hndl = op_stat_reg ("Response
Time",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);                    //KiT
connection_setup_hndl = op_stat_reg ("Connection Setup
Time",OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
local_connection_setup_hndl = op_stat_reg ("Local Connection
Setup Time",OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
printf("I'm QoS_usr_proc exit\n ");
}
/** state (init) exit executives **/
FSM_STATE_EXIT_FORCED (0, "init", "QoS_usr_proc () [init exit
execs]")
{
}
/** state (init) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "init",
"dispatch")
/*---------------------------------------------------------*/
/** state (dispatch) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "dispatch",
"QoS_usr_proc () [dispatch enter execs]")
{
}
/** blocking after enter executives of unforced state. **/
FSM_EXIT (3,QoS_usr_proc)
/** state (dispatch) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, "dispatch", "QoS_usr_proc ()
[dispatch exit execs]")
{
/* --- get the packet from the input stream --- */
//printf("I'm QoS_usr_proc on node %s, processing the packet
received\n", node_name);
//printf("current sim time %lf\n", op_sim_time());
//scanf("%i",&rkey);
pkptr = op_pk_get (op_intrpt_strm ());
op_pk_nfd_get (pkptr, "Msg_type", &msg_type);
switch ( msg_type ) {
case USR_REQ:
case USR_PKT_END:
case USR_DATA:
case EDGE_HLD_REP:
        op_pk_send (pkptr, XMT_OUT);
        break;
case USR_REQ_ADMITTED:
case EDGE_HLD_ADMITTED:
        op_pk_nfd_get (pkptr, "Usr_id", &usr_id);
        op_pk_deliver (pkptr, usr_id, 1);
        break;

case SETUP_TIME:
        op_pk_nfd_get (pkptr, "Connection_time",
&connection_time);
        op_stat_write (connection_setup_hndl, connection_time);
        op_pk_destroy(pkptr);
        break;
case RESPONSE_TIME:
```

```
                                        op_pk_nfd_get (pkptr, "Connection_time",
&connection_time);
                                        op_stat_write (response_time_hndl, connection_time);
                                        op_pk_destroy(pkptr);
                                    break;
                                }
                                }
                            /** state (dispatch) transition processing **/
                            FSM_TRANSIT_ONLY ((PKT_ARRVL), 1, state1_enter_exec, ;,
"dispatch", "PKT_ARRVL", "", "dispatch", "dispatch")
                            /*---------------------------------------------------------*/
                            }
                    FSM_EXIT (0,QoS_usr_proc)
                    }
            }
#if defined (__cplusplus)
            extern "C" {
#endif
            extern VosT_Fun_Status Vos_Catmem_Register (const char * , int , VosT_Void_Null_Proc,
VosT_Address *);
            extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
            extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
#if defined (__cplusplus)
            }
#endif
Compcode
QoS_usr_proc_init (void ** gen_state_pptr)
            {
            int _block_origin = 0;
            static VosT_Address         obtype = OPC_NIL;
            FIN (QoS_usr_proc_init (gen_state_pptr))
            if (obtype == OPC_NIL)
                    {
                    /* Initialize memory management */
                    if (Vos_Catmem_Register ("proc state vars (QoS_usr_proc)",
                            sizeof (QoS_usr_proc_state), Vos_Vnop, &obtype) == VOSC_FAILURE)
                            {
                            FRET (OPC_COMPCODE_FAILURE)
                            }
                    }
            *gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
            if (*gen_state_pptr == OPC_NIL)
                    {
                    FRET (OPC_COMPCODE_FAILURE)
                    }
            else
                    {
                    /* Initialize FSM handling */
                    ((QoS_usr_proc_state *)(*gen_state_pptr))->current_block = 0;
                    FRET (OPC_COMPCODE_SUCCESS)
                    }
            }
void
QoS_usr_proc_diag (void)
            {
            /* No Diagnostic Block */
            }
void
QoS_usr_proc_terminate (void)
            {
```

```
	int _block_origin = __LINE__;
	FIN (QoS_usr_proc_terminate (void))
	if (1)
		{
		Packet*			pkptr;
		int			msg_type;
		double			ete_delay;
		int			rkey;
		int			count;
		int			i;
		int			j;
		int			usr_id;
		Boolean			id_found;
		Objid			usr_op_id;
		Objid			tmp_queue_Objid;
		double			connection_time;
		/* No Termination Block */
		}
	Vos_Catmem_Dealloc (pr_state_ptr);
	FOUT;
	}
/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in QoS_usr_proc_svar function. */
#undef ete_gsh
#undef own_id
#undef pkt_count
#undef connection_setup_hndl
#undef parent_id
#undef node_name
#undef local_connection_setup_hndl
#undef response_time_hndl
void
QoS_usr_proc_svar (void * gen_ptr, const char * var_name, char ** var_p_ptr)
	{
	QoS_usr_proc_state		*prs_ptr;
	FIN (QoS_usr_proc_svar (gen_ptr, var_name, var_p_ptr))
	if (var_name == OPC_NIL)
		{
		*var_p_ptr = (char *)OPC_NIL;
		FOUT;
		}
	prs_ptr = (QoS_usr_proc_state *)gen_ptr;
	if (strcmp ("ete_gsh" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->ete_gsh);
		FOUT;
		}
	if (strcmp ("own_id" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->own_id);
		FOUT;
		}
	if (strcmp ("pkt_count" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->pkt_count);
		FOUT;
		}
	if (strcmp ("connection_setup_hndl" , var_name) == 0)
```

```
        {
        *var_p_ptr = (char *) (&prs_ptr->connection_setup_hndl);
        FOUT;
        }
if (strcmp ("parent_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->parent_id);
        FOUT;
        }
if (strcmp ("node_name" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->node_name);
        FOUT;
        }
if (strcmp ("local_connection_setup_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->local_connection_setup_hndl);
        FOUT;
        }
if (strcmp ("response_time_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->response_time_hndl);
        FOUT;
        }
*var_p_ptr = (char *)OPC_NIL;

FOUT;
}
```

```
/*****************************************************************************/
/*EDGE ROUTER NODE*/
/*****************************************************************************/
    /*proc module*/
/*****************************************************************************/
/* This variable carries the header into the object file */
static const char QoSpGtw_proc_pr_c [] = "MIL_3_Tfile_Hdr_ 80C 30A op_runsim 7 441CF08B
441CF08B 1 KiT KiT 0 0 none none 0 0 none 0 0 0 0 0 0";
#include <string.h>
/* OPNET system definitions */
#include <opnet.h>
#if defined (__cplusplus)
extern "C" {
#endif
FSM_EXT_DECS
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Header Block */
/* --- include the head files create by myself --- */
#include "..\headfile\McrDef.h"
#include "..\headfile\McrDS.h"
#include "..\headfile\PolicySvr_gateway.h"
#include "..\headfile\PolicySvr_Runtime.h"
/* packet stream definitions */
#define XMT_OUT_TO_USER                         0
#define XMT_OUT_TO_ROUTER           1
/* interruption code definition */
#define  PACKET_RECEIVE      0
#define  RESOURCE_REPORT     1
#define HOLD_FLOW            2
#define RECALL_FLOW          3
/* transition macros */
//#define PKT_ARRVL ( (op_intrpt_type () == OPC_INTRPT_STRM) && ( op_intrpt_code() ==
PACKET_RECEIVE ) )
#define PKT_ARRVL   (op_intrpt_type () == OPC_INTRPT_STRM)        //modified by YM
#define  TIME_OUT  ( (op_intrpt_type () == OPC_INTRPT_SELF) && ( op_intrpt_code() ==
RESOURCE_REPORT ) )
#define HOLD      ( (op_intrpt_type () == OPC_INTRPT_SELF) && ( op_intrpt_code() ==
HOLD_FLOW ) )
#define RECALL   ( (op_intrpt_type () == OPC_INTRPT_SELF) && ( op_intrpt_code() ==
RECALL_FLOW ) )
//int      LookUpFwdTbl (int, int);
extern    all_req_num;
extern    admit_req_num;
extern    overhead_msg_num;
extern    block_req_num;
extern      recall_overhead_num;
extern      all_hold_num;
extern      all_incoming_num;
extern      total_CAC;
extern      queue_id;
extern      queue_id1;
//extern   Stathandle  blocking_rate_hndl;
//extern   Stathandle  trunk_rsv_bdw_hndl[41];
//extern   Stathandle  trunk_bdw_eff_hndl[41];
//extern   Stathandle  packet_loss_rate_hndl;
//extern   received_packets;
```

```
//extern   send_out_packets;
void timeout_queue0(void);
void timeout_queue1(void);
void set_timeout0(void);
void set_timeout1(void);
/* End of Header Block */
#if !defined (VOSD_NO_FIN)
#undef   BIN
#undef   BOUT
#define  BIN      FIN_LOCAL_FIELD(last_line_passed) = __LINE__ - _block_origin;
#define  BOUT   BIN
#define  BINIT   FIN_LOCAL_FIELD(last_line_passed) = 0; _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 */
        Stathandle              ete_gsh;
        Objid                   own_id;
        int                     pkt_count;
        int                     vip_no;
        RouteItem               route_table [MAX_ROUTER_NUM];
        TrunkRec                trunk_table [MAX_TRUNK_NUM];
        FwdDef                  forward_table [MAX_ROUTE_NUM*2];
        Objid                   parent_id;
        int                     rsv_route_id;
        int                     rsv_intf;
        int                     fwd_tbl_len;
        Stathandle              overhead_hndl;
        Stathandle              blocking_rate_hndl;
        int                     local_all_req_num;
        Stathandle              local_overhead_hndl;
        int                     local_overhead_num;
        int                     local_admit_req_num;
        char                    node_name[15];
        Stathandle              audio_e2edelay_hndl;
        Stathandle              video_e2edelay_hndl;
        int                     Gtw_ID;
        int                     Router_ID;
        Stathandle              trunk_rsvbdw_hndl[39];
        Stathandle              trunk_usdbdw_hndl[39];
        NMS_REC                 keep_table[MAX_TRUNK_NUM];
        char                    file_name[128];
        int                     buffer;
        char                    xx;
        FlowData                flowrec;
        int                     hold_queue_id;
        FlowHold                hold_table[MAX_HOLD_NUM];
        FlowHold                hold_table1[MAX_HOLD_NUM];
        int                     hold_queue_id1;
        List*                   data_ptr;
        char                    edge_name[128];
        FlowData*               tmp;
        int                     counter;
        FlowHold                recall_table;
```

```
        List*                   hold_rec0;
        List*                   hold_rec1;
        FlowData*               tmp1;
        Stathandle              recall_overhead_hndl;
        Stathandle              total_CAC_hndl;
        List*                   delete_info0;
        List*                   delete_info1;
        int                     frag;
        List*                   LifeTime0;
        List*                   LifeTime1;
        int                     queueC_id;
        int                     queueC_id1;
        double                  next_timeout0;
        double                  next_timeout1;
        int                     stat_count;
        int                     sent_code0;
        int                     sent_code1;
        } QoSpGtw_proc_state;
#define pr_state_ptr            ((QoSpGtw_proc_state*) SimI_Mod_State_Ptr)
#define ete_gsh                 pr_state_ptr->ete_gsh
#define own_id                  pr_state_ptr->own_id
#define pkt_count               pr_state_ptr->pkt_count
#define vip_no                  pr_state_ptr->vip_no
#define route_table             pr_state_ptr->route_table
#define trunk_table             pr_state_ptr->trunk_table
#define forward_table           pr_state_ptr->forward_table
#define parent_id               pr_state_ptr->parent_id
#define rsv_route_id            pr_state_ptr->rsv_route_id
#define rsv_intf                pr_state_ptr->rsv_intf
#define fwd_tbl_len             pr_state_ptr->fwd_tbl_len
#define overhead_hndl           pr_state_ptr->overhead_hndl
#define blocking_rate_hndl      pr_state_ptr->blocking_rate_hndl
#define local_all_req_num       pr_state_ptr->local_all_req_num
#define local_overhead_hndl     pr_state_ptr->local_overhead_hndl
#define local_overhead_num      pr_state_ptr->local_overhead_num
#define local_admit_req_num     pr_state_ptr->local_admit_req_num
#define node_name               pr_state_ptr->node_name
#define audio_e2edelay_hndl     pr_state_ptr->audio_e2edelay_hndl
#define video_e2edelay_hndl     pr_state_ptr->video_e2edelay_hndl
#define Gtw_ID                  pr_state_ptr->Gtw_ID
#define Router_ID               pr_state_ptr->Router_ID
#define trunk_rsvbdw_hndl       pr_state_ptr->trunk_rsvbdw_hndl
#define trunk_usdbdw_hndl       pr_state_ptr->trunk_usdbdw_hndl
#define keep_table              pr_state_ptr->keep_table
#define file_name               pr_state_ptr->file_name
#define buffer                  pr_state_ptr->buffer
#define xx                      pr_state_ptr->xx
#define flowrec                 pr_state_ptr->flowrec
#define hold_queue_id           pr_state_ptr->hold_queue_id
#define hold_table              pr_state_ptr->hold_table
#define hold_table1             pr_state_ptr->hold_table1
#define hold_queue_id1          pr_state_ptr->hold_queue_id1
#define data_ptr                pr_state_ptr->data_ptr
#define edge_name               pr_state_ptr->edge_name
#define tmp                     pr_state_ptr->tmp
#define counter                 pr_state_ptr->counter
#define recall_table            pr_state_ptr->recall_table
#define hold_rec0               pr_state_ptr->hold_rec0
#define hold_rec1               pr_state_ptr->hold_rec1
```

```
#define tmp1                    pr_state_ptr->tmp1
#define recall_overhead_hndl    pr_state_ptr->recall_overhead_hndl
#define total_CAC_hndl          pr_state_ptr->total_CAC_hndl
#define delete_info0            pr_state_ptr->delete_info0
#define delete_info1            pr_state_ptr->delete_info1
#define frag                    pr_state_ptr->frag
#define LifeTime0               pr_state_ptr->LifeTime0
#define LifeTime1               pr_state_ptr->LifeTime1
#define queueC_id               pr_state_ptr->queueC_id
#define queueC_id1              pr_state_ptr->queueC_id1
#define next_timeout0           pr_state_ptr->next_timeout0
#define next_timeout1           pr_state_ptr->next_timeout1
#define stat_count              pr_state_ptr->stat_count
#define sent_code0              pr_state_ptr->sent_code0
#define sent_code1              pr_state_ptr->sent_code1
/* This macro definition 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
#define FIN_PREAMBLE QoSpGtw_proc_state *op_sv_ptr = pr_state_ptr;
/* Function Block */
enum { _block_origin = __LINE__ };
 GetIdFrmName ( char name[15] )
{
        char *          sptr1;
        char *          sptr2;

        /* --- the format of node name is node_11_1 ---*/
        /* --- we need to get 11 out of the name attribute as router ID--- */
        /* --- we need to get 1 out of the name attribute as gateway ID--- */
        /* --- use the number as the id of the gateway  --- */
        sptr1 = strchr(name, '_');
        sptr2=strrchr(name,'_');
        sptr1 ++;
        *sptr2='\0';
        sptr2++;
        Router_ID=atoi(sptr1);
        Gtw_ID = atoi(sptr2);
}
/*********************************************************************/
/*FUNCTION compare for op_prg_list_insert_sorted()*/
int     stoptime_comp (const void* ptr1, const void* ptr2)
{
QoSRuntime *tptr1, *tptr2;

tptr1 = (QoSRuntime*) ptr1;
tptr2 = (QoSRuntime*) ptr2;
if (tptr1->Finish_time < tptr2->Finish_time)
        { return 1; }
else if (tptr1->Finish_time > tptr2->Finish_time)
        { return -1; }
else
        { return 0; }
}
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int     recalltime_comp (const void* ptr1, const void* ptr2)
{
```

```
FlowData *tptr1, *tptr2;
tptr1 = (FlowData*) ptr1;
tptr2 = (FlowData*) ptr2;
if (tptr1->RecallTime < tptr2->RecallTime)
        { return 1; }
else if (tptr1->RecallTime > tptr2->RecallTime)
        { return -1; }
else
        { return 0; }
}
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int     queue_comp (const void* ptr1, const void* ptr2)
{
FlowData *tptr1, *tptr2;
tptr1 = (FlowData*) ptr1;
tptr2 = (FlowData*) ptr2;
if (tptr1->QueueID < tptr2->QueueID)
        { return 1; }
else if (tptr1->QueueID > tptr2->QueueID)
        { return -1; }
else
        { return 0; }
}
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
int     id_comp (const void* ptr1, const void* ptr2)
{
ReleasedRecord *tptr1, *tptr2;
tptr1 = (ReleasedRecord*) ptr1;
tptr2 = (ReleasedRecord*) ptr2;
if (tptr1->QueueID < tptr2->QueueID)
        { return 1; }
else if (tptr1->QueueID > tptr2->QueueID)
        { return -1; }
else
        { return 0; }
}
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void set_timeout0 (void)
        {
        //printf("SET TIMEOUT 0\n");
        op_intrpt_schedule_call (next_timeout0, sent_code0, timeout_queue0, OPC_NIL);
        }
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
void set_timeout1 (void)
        {
        //printf("SET TIMEOUT 1\n");
        op_intrpt_schedule_call (next_timeout1, sent_code1, timeout_queue1, OPC_NIL);
        }
/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/
/*---------------------------------FUNCTION TIMEOUT 0 ------------------------------------*/
void
timeout_queue0(FlowData * hold_ptr, int sent)
{
        int                     a;
        int                     b;
        int                     skip;
        TimeOut*                timeout0;
        FlowData*               check_timeout0;
```

```
	int			recall0_size;
	int			timeout0_size;
	Packet*			req_adm_pkptr;
	FIN (timeout_queue0(hold_ptr, sent));
	skip = 0;
	timeout0_size = op_prg_list_size (LifeTime0);
	if (timeout0_size != 0)
		{
		for (a=0; a<timeout0_size; a++)
		{
		if (skip == 0)
		 {
		timeout0 = (TimeOut*) op_prg_mem_alloc (sizeof (TimeOut));
		timeout0 = ((char *) op_prg_list_access (LifeTime0, a));
		if (sent == timeout0->ProcID)
			{
			 op_prg_list_remove(LifeTime0, a);
			 skip = 1;
			}
		 }}
		}
	recall0_size = op_prg_list_size (hold_rec0);
	if (recall0_size != 0)
		{
		for(b=0; b<recall0_size; b++)
		{
		check_timeout0 = (FlowData*) op_prg_mem_alloc (sizeof (FlowData));
		check_timeout0 = ((char *) op_prg_list_access (hold_rec0,b));
		if (sent == check_timeout0->ProcID)
			{
			/*+++++++++++++++SEND DENY+++++++++++++++*/
			block_req_num++;
			req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
			op_pk_total_size_set (req_adm_pkptr, 1024);
			op_pk_nfd_set (req_adm_pkptr, "Msg_type", USR_REQ_ADMITTED);
			op_pk_nfd_set (req_adm_pkptr, "Usr_id", check_timeout0->UsrID);
			op_pk_nfd_set (req_adm_pkptr, "Proc_id", check_timeout0->ProcID);
			op_pk_nfd_set (req_adm_pkptr, "Src_gtw", check_timeout0->SrcGtw);
			op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", check_timeout0->DestGtw);
			op_pk_nfd_set (req_adm_pkptr, "Decision", BLOCKED);
			printf("Timeout=>BLOCKED! at router %i gtw %i dest :%i req-type: %i at
time %lf\n",Router_ID,Gtw_ID,check_timeout0->DestRouter,check_timeout0-
>ReqType,op_sim_time());
			op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
	op_stat_write(blocking_rate_hndl,(double)((double)block_req_num/(double)all_req_num));
			total_CAC = total_CAC + (1*1024);
			op_stat_write(total_CAC_hndl,total_CAC);
			/*+++++++++++++++++SEND DENY++++++++++++++++++*/
			op_prg_list_remove(hold_rec0, b);
			FOUT;
			}
		}
		}
	FOUT;
	}
/*-----------------------------------FUNCTION TIMEOUT 0 -------------------------------------*/
/*-----------------------------------FUNCTION TIMEOUT 1 -------------------------------------*/
void
```

```
timeout_queue1(FlowData * hold_ptr, int sent)
{
	int			a;
	int			b;
	int			skip;
	TimeOut*		timeout1;
	FlowData*		check_timeout1;
	int			recall1_size;
	int			timeout1_size;
	Packet*			req_adm_pkptr;
	FIN (timeout_queue1(hold_ptr, sent));
	skip = 0;
	timeout1_size = op_prg_list_size (LifeTime1);
	//printf("PROCESS ID = %i\n", sent);
	if (timeout1_size != 0)
		{
		for (a=0; a<timeout1_size; a++)
		{
		if (skip == 0)
		 {
		timeout1 = (TimeOut*) op_prg_mem_alloc (sizeof (TimeOut));
		timeout1 = ((char *) op_prg_list_access (LifeTime1, a));
		if (sent == timeout1->ProcID)
			{
			 op_prg_list_remove(LifeTime1, a);
			 skip = 1;
			}
		 }}
		}
	recall1_size = op_prg_list_size (hold_rec1);
	if (recall1_size != 0)
		{
		for(b=0; b<recall1_size; b++)
		{
		check_timeout1 = (FlowData*) op_prg_mem_alloc (sizeof (FlowData));
		check_timeout1 = ((char *) op_prg_list_access (hold_rec1,b));
		if (sent == check_timeout1->ProcID)
			{
			/*++++++++++++++++++SEND DENY++++++++++++++++++*/
			block_req_num++;
			req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
			op_pk_total_size_set (req_adm_pkptr, 1024);
			op_pk_nfd_set (req_adm_pkptr, "Msg_type", USR_REQ_ADMITTED);
			op_pk_nfd_set (req_adm_pkptr, "Usr_id", check_timeout1->UsrID);
			op_pk_nfd_set (req_adm_pkptr, "Proc_id", check_timeout1->ProcID);
			op_pk_nfd_set (req_adm_pkptr, "Src_gtw", check_timeout1->SrcGtw);
			op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", check_timeout1->DestGtw);
			op_pk_nfd_set (req_adm_pkptr, "Decision", BLOCKED);
			printf("Timeout=>BLOCKED! at router %i gtw %i dest :%i req-type: %i at
time %lf\n",Router_ID,Gtw_ID,check_timeout1->DestRouter,check_timeout1-
>ReqType,op_sim_time());
			op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
	op_stat_write(blocking_rate_hndl,(double)((double)block_req_num/(double)all_req_num));
			total_CAC = total_CAC + (1*1024);
			op_stat_write(total_CAC_hndl,total_CAC);
			/*++++++++++++++++SEND DENY++++++++++++++++++*/
			op_prg_list_remove(hold_rec1, b);
			FOUT;
```

```
                        }
                }
                }
        FOUT;
        }
/*---------------------------------FUNCTION TIMEOUT 1 ---------------------------------------*/
/* 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 QoSpGtw_proc (void);
        Compcode QoSpGtw_proc_init (void **);
        void QoSpGtw_proc_diag (void);
        void QoSpGtw_proc_terminate (void);
        void QoSpGtw_proc_svar (void *, const char *, char **);
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Process model interrupt handling procedure */
void
QoSpGtw_proc (void)
        {
        int _block_origin = 0;
        FIN (QoSpGtw_proc ());
        if (1)
                {
                int                     intrpt_code;
                int                     qos_type;
                struct TrunkDef         trunk_pnt;
                struct RouteDef         route_pnt;
                struct RouteDef *       tmp_route_pnt;
                FwdDef                  fwd_pnt;
                QoSRoute                q_route;
                QoSTrunk                q_trunk;
                Packet*                 pkptr;
                Packet*                 req_adm_pkptr;
                int                     msg_type;
                int                     req_type;
                int                     user_id;
                int                     dest_gtw;
                int                     dest_router;
                int                     bdw_req;
                int                     usd_bdw;
                int                     call_duration;
                int                     req_time;
                int                     proc_id;
                int                     route_id;
                int                     trunk_id;
                int                     vip_id;
                int                     out_index;
                Packet*                 trk_cfg_pkptr;
                struct RouteDef *       route_ptr;
                struct RouteDef *       need_reconfig_route_ptr;
```

```
struct RouteDef *       widest_route_ptr;
struct RouteDef *       narrowest_route_ptr;
struct TrunkDef         tmp_trunk_ptr;
int                     widest_route_id;
int                     narrowest_route_id;
int                     tmp_avl_bdw;
int                     route_avl_bdw;
int                     max_avl_bdw;
int                     min_avl_bdw;
struct VipDef *         vip_ptr;
struct VipDef *         tmp_vip_ptr;
double                  e2edelay;
double                  sendout_time;
int                     rkey;
int                     count;
int                     i;
int                     j;
Packet*                 trk_rsc_pkptr;
int                     packet_sent;
FILE*                   profile_hndl;
int                     outgoing;
int                     notfeasible;
int                     lowest_threshhold;
int                     plink_id;
int                     length;
int                     src_addr;
int                     dest_addr;
long                    packet_loss_rate;
struct NMS_KEEP *       tmp_keep_table;
QoSRuntime              q_runtime;
int                     calltime;
int                     requesttime;
FILE*                   profile;
char                    name[128];
FILE*                   hold_profile;
char                    hold_name[128];
int                     time_req0;
int                     time_req1;
QoSRuntime*             runtime;
QoSRuntime*             tmp_ptr;
QoSRuntime*             temp_ptr;
char                    msg0[128],msg1[128];
int                     time;
int                     list_size;
int                     n;
int                     m;
FlowData*               temp_hold0;
FlowData*               temp_hold1;
FlowData*               tmp_hold0;
FlowData*               tmp_hold1;
FlowData*               hold_reply;
int                     recall0_size;
int                     recall1_size;
int                     re_time;
Packet*                 hold_admit_pkptr;
int                     decision;
int                     time_est;
int                     hold_size;
QoSRuntime*             del_ptr;
int                     del_size;
```

```
int                     del;
struct RouteDef *       flowrec_route_ptr;
struct RouteDef *       tmp_route_ptr;
int                     pass;
int                     check_count;
int                     Length_count;
int                     Length_count1;
int                     tmp_Length;
int                     ii;
int                     step;
int                     x;
ReleasedRecord*         temp_record0;
ReleasedRecord*         temp_record1;
ReleasedRecord*         temps_record0;
ReleasedRecord*         temps_record1;
int                     delete0_size;
int                     delete1_size;
int                     position0;
int                     position1;
int                     num;
FlowData*               temps_hold0;
FlowData*               temps_hold1;
int                     queue_check0;
int                     queue_check1;
int                     list_buff;
int                     diff;
Packet*                 hold_ptr;
TimeOut*                info_timeout0;
TimeOut*                info_timeout1;
int                     sent;
FSM_ENTER (QoSpGtw_proc)
FSM_BLOCK_SWITCH
        {
        /*---------------------------------------------------------*/
        /** state (init) enter executives **/
        FSM_STATE_ENTER_FORCED_NOLABEL (0, "init", "QoSpGtw_proc
() [init enter execs]")
                {
                printf("I'm QoSpGtw_proc\n");
                own_id = op_id_self ();
                parent_id = op_topo_parent (own_id);
                for (i=0; i<15; i++)
                node_name[i] = '\0';
                hold_queue_id  = 0;
                hold_queue_id1 = 0;
                queue_id = 0;
                queue_id1 = 0;
                frag = 0;
                queueC_id = 0;
                queueC_id1 = 0;
                counter  = 0;
                all_hold_num = 0;
                all_incoming_num = 0;
                vip_no = 0;
                pkt_count = 0;
                stat_count = 0;
                data_ptr = op_prg_list_create();
                hold_rec0 = op_prg_list_create();
```

```
                                    hold_rec1 = op_prg_list_create();
                                    delete_info0 = op_prg_list_create();
                                    delete_info1 = op_prg_list_create();
                                    LifeTime0 = op_prg_list_create();
                                    LifeTime1 = op_prg_list_create();
                                    /*blocking rate metric init */
                                    total_CAC_hndl = op_stat_reg ("CAC Signal Traffic",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
                                    recall_overhead_hndl =op_stat_reg ("Recall Overhead",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
                                    overhead_hndl =op_stat_reg ("Overhead",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
                                    blocking_rate_hndl =op_stat_reg ("Blocking Rate",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
                                    /*trunk_rsvbdw_hndl[1]=op_stat_reg ("Trunk 1 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[2]=op_stat_reg ("Trunk 2 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[3]=op_stat_reg ("Trunk 3 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[4]=op_stat_reg ("Trunk 4 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[5]=op_stat_reg ("Trunk 5 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[6]=op_stat_reg ("Trunk 6 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[7]=op_stat_reg ("Trunk 7 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[8]=op_stat_reg ("Trunk 8 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[9]=op_stat_reg ("Trunk 9 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[10]=op_stat_reg ("Trunk 10 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[11]=op_stat_reg ("Trunk 11 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[12]=op_stat_reg ("Trunk 12 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[13]=op_stat_reg ("Trunk 13 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[14]=op_stat_reg ("Trunk 14 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[15]=op_stat_reg ("Trunk 15 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[16]=op_stat_reg ("Trunk 16 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[17]=op_stat_reg ("Trunk 17 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[18]=op_stat_reg ("Trunk 18 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[19]=op_stat_reg ("Trunk 19 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );*/
                                    trunk_rsvbdw_hndl[20]=op_stat_reg ("Trunk 20 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[21]=op_stat_reg ("Trunk 21 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[22]=op_stat_reg ("Trunk 22 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
```

```
                                    trunk_rsvbdw_hndl[23]=op_stat_reg ("Trunk 23 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[24]=op_stat_reg ("Trunk 24 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[25]=op_stat_reg ("Trunk 25 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[26]=op_stat_reg ("Trunk 26 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[27]=op_stat_reg ("Trunk 27 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[28]=op_stat_reg ("Trunk 28 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[29]=op_stat_reg ("Trunk 29 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[30]=op_stat_reg ("Trunk 30 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[31]=op_stat_reg ("Trunk 31 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[32]=op_stat_reg ("Trunk 32 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[33]=op_stat_reg ("Trunk 33 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[34]=op_stat_reg ("Trunk 34 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[35]=op_stat_reg ("Trunk 35 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[36]=op_stat_reg ("Trunk 36 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[37]=op_stat_reg ("Trunk 37 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_rsvbdw_hndl[38]=op_stat_reg ("Trunk 38 reserved
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    /*trunk_usdbdw_hndl[1]=op_stat_reg ("Trunk 1 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[2]=op_stat_reg ("Trunk 2 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[3]=op_stat_reg ("Trunk 3 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[4]=op_stat_reg ("Trunk 4 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[5]=op_stat_reg ("Trunk 5 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[6]=op_stat_reg ("Trunk 6 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[7]=op_stat_reg ("Trunk 7 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[8]=op_stat_reg ("Trunk 8 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[9]=op_stat_reg ("Trunk 9 used bandwidth",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[10]=op_stat_reg ("Trunk 10 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[11]=op_stat_reg ("Trunk 11 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[12]=op_stat_reg ("Trunk 12 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    trunk_usdbdw_hndl[13]=op_stat_reg ("Trunk 13 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
```

```
                                trunk_usdbdw_hndl[14]=op_stat_reg ("Trunk 14 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[15]=op_stat_reg ("Trunk 15 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[16]=op_stat_reg ("Trunk 16 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[17]=op_stat_reg ("Trunk 17 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[18]=op_stat_reg ("Trunk 18 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[19]=op_stat_reg ("Trunk 19 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );*/
                                trunk_usdbdw_hndl[20]=op_stat_reg ("Trunk 20 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[21]=op_stat_reg ("Trunk 21 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[22]=op_stat_reg ("Trunk 22 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[23]=op_stat_reg ("Trunk 23 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[24]=op_stat_reg ("Trunk 24 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[25]=op_stat_reg ("Trunk 25 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[26]=op_stat_reg ("Trunk 26 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[27]=op_stat_reg ("Trunk 27 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[28]=op_stat_reg ("Trunk 28 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[29]=op_stat_reg ("Trunk 29 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[30]=op_stat_reg ("Trunk 30 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[31]=op_stat_reg ("Trunk 31 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[32]=op_stat_reg ("Trunk 32 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[33]=op_stat_reg ("Trunk 33 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[34]=op_stat_reg ("Trunk 34 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[35]=op_stat_reg ("Trunk 35 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[36]=op_stat_reg ("Trunk 36 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[37]=op_stat_reg ("Trunk 37 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                trunk_usdbdw_hndl[38]=op_stat_reg ("Trunk 38 used
bandwidth", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                local_all_req_num =0;
                                local_admit_req_num=0;
                                local_overhead_num=0;
                                for (i=0;i<MAX_TRUNK_NUM;i++)
                                        {trunk_table[i].RsvBdw=0;
                                         trunk_table[i].UsdBdw=0;
                                         trunk_table[i].PLinkID=0;
                                         trunk_table[i].TrunkID=0;
```

```
                                        trunk_table[i].Busy = 0;
                                        }
                                /* --- define the reserved QoS route for the policy server --- */
                                rsv_route_id = 0;
                                /* --- set the variables that identify the gateway --- */
                                op_ima_obj_attr_get (parent_id, "name", node_name);
                                strcpy(edge_name,node_name);
                                //printf("%s\n",edge_name);
                                GetIdFrmName (node_name);
                                /* -- schedule the resource check and excess resource release
period -- */
                                op_intrpt_schedule_self (op_sim_time () + CHECK_PERIOD,
RESOURCE_REPORT);
                                /* --- read in the allocated trunk information --- */
                                if( (profile_hndl = fopen( TRUNK_FILE, "r" )) != NULL )
                                {
                                        i = fread ((char *)&q_trunk, sizeof( QoSTrunk ), 1,
profile_hndl);
                                        while ( !feof (profile_hndl) ) {
                                                /* -- fill in the trunk table --- */
                                                        trunk_table[q_trunk.TrunkID].TrunkID
= q_trunk.TrunkID;
                                                        if(q_trunk.TrunkID<=19)
                                                        trunk_table[q_trunk.TrunkID].RsvBdw
= 3000;
                                                        else
trunk_table[q_trunk.TrunkID].RsvBdw = 64;
                                                        trunk_table[q_trunk.TrunkID].UsdBdw
= 0;
                                                        trunk_table[q_trunk.TrunkID].PLinkID
=q_trunk.PLinkID;
                                                        trunk_table[q_trunk.TrunkID].VipPnt =
NULL;
                                                        if(q_trunk.TrunkID>=20)

        { op_stat_write(trunk_rsvbdw_hndl[q_trunk.TrunkID],trunk_table[q_trunk.TrunkID].RsvBdw
);

op_stat_write(trunk_usdbdw_hndl[q_trunk.TrunkID],q_trunk.UsdBdw);
                                                                }
                                        i = fread ((char *)&q_trunk, sizeof( QoSTrunk ), 1,
profile_hndl);
                                        }
                                        fclose(profile_hndl);
                                }
                                /* --- read in the allocated QoS routes information--- */
                                if( (profile_hndl = fopen( ROUTE_FILE, "r" )) != NULL )
                                {
                                        i = fread ((char *)&q_route, sizeof( QoSRoute ), 1,
profile_hndl);
                                        while ( !feof (profile_hndl) ) {
                                                if ( q_route.Src_router == Router_ID )
                                                  {
                                                route_ptr = route_table[q_route.Dest_router].ptr;
                                                tmp_route_pnt = (RouteRec *) malloc
( sizeof(RouteRec) );
                                                tmp_route_pnt->RouteID = q_route.RouteID;
                                                tmp_route_pnt->Length = q_route.Length;
```

```
                                                    //printf(" dest router %i route
id %i\n",q_route.Dest_router,q_route.RouteID);
                                                    for(i=0;i<q_route.Length;i++)
                                                    {tmp_route_pnt-
>TrunkList[i]=q_route.TrunkList[i];
                                                    //printf("%i  ",tmp_route_pnt->TrunkList[i]);
                                                            }
                                                    //printf("\n");
                                                    //scanf("%i", &rkey);
                                                    tmp_route_pnt->Next = NULL;
                                                    if (route_ptr == NULL)

        {    route_table[q_route.Dest_router].ptr=tmp_route_pnt;
                                                             }
                                                    else
                                                            {
                                                    while (route_ptr->Next != NULL)
                                                    route_ptr = route_ptr->Next;
                                                    route_ptr->Next = tmp_route_pnt;
                                                            }
                                                            }
                                                    if ( q_route.Dest_router == Router_ID )
                                                            {
                                                    route_ptr = route_table[q_route.Src_router].ptr;
                                                    tmp_route_pnt = (RouteRec *) malloc
( sizeof(RouteRec) );
                                                    tmp_route_pnt->RouteID = q_route.RouteID;
                                                    tmp_route_pnt->Length = q_route.Length;
                                                    // printf(" dest router %i route
id %i\n",q_route.Src_router,q_route.RouteID);
                                                    for(i=0;i<q_route.Length;i++)
                                                    { tmp_route_pnt-
>TrunkList[i]=q_route.TrunkList[i];
                                                    //printf("%i  ",tmp_route_pnt->TrunkList[i]);
                                                            }
                                                    //printf("\n");
                                                    //scanf("%i", &rkey);
                                                    tmp_route_pnt->Next = NULL;
                                                    if (route_ptr == NULL)
                                            {route_table[q_route.Src_router].ptr=tmp_route_pnt;
                                                             }
                                                    else
                                                            {
                                                    while (route_ptr->Next != NULL)
                                                    route_ptr = route_ptr->Next;
                                                    route_ptr->Next = tmp_route_pnt;
                                                            }
                                                            }
                                                    i = fread ((char *)&q_route, sizeof( QoSRoute ),
1, profile_hndl);
                                            }
                                            fclose(profile_hndl);
                                            }
                                    }
                            /** state (init) exit executives **/
                            FSM_STATE_EXIT_FORCED (0, "init", "QoSpGtw_proc () [init exit
execs]")
                                    {
                                    }
```

```
                    /** state (init) transition processing **/
                    FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "init",
"dispatch")
                    /*---------------------------------------------------------*/
                    /** state (dispatch) enter executives **/
                    FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "dispatch",
"QoSpGtw_proc () [dispatch enter execs]")
                        {
                        }
                    /** blocking after enter executives of unforced state. **/
                    FSM_EXIT (3,QoSpGtw_proc)
                    /** state (dispatch) exit executives **/
                    FSM_STATE_EXIT_UNFORCED (1, "dispatch", "QoSpGtw_proc ()
[dispatch exit execs]")
                        {
                    /*Determine the code of the interrupt, which is used in evaluating */
                    /* state transition conditions.*/
                        //intrpt_code = op_intrpt_code ();
                        }
                    /** state (dispatch) transition processing **/
                    FSM_INIT_COND (PKT_ARRVL)
                    FSM_TEST_COND (TIME_OUT)
                    FSM_TEST_COND (HOLD)
                    FSM_TEST_COND (RECALL)
                    FSM_TEST_LOGIC ("dispatch")
                    FSM_TRANSIT_SWITCH
                        {
                        FSM_CASE_TRANSIT (0, 2, state2_enter_exec, ;,
"PKT_ARRVL", "", "dispatch", "pktproc")
                        FSM_CASE_TRANSIT (1, 3, state3_enter_exec, ;, "TIME_OUT",
"", "dispatch", "rpt")
                        FSM_CASE_TRANSIT (2, 4, state4_enter_exec, ;, "HOLD", "",
"dispatch", "HOLD")
                        FSM_CASE_TRANSIT (3, 5, state5_enter_exec, ;, "RECALL",
"", "dispatch", "RECALL")
                        }
                        /*---------------------------------------------------------*/
                    /** state (pktproc) enter executives **/
                    FSM_STATE_ENTER_FORCED (2, state2_enter_exec, "pktproc",
"QoSpGtw_proc () [pktproc enter execs]")
                        {
                        /* --- get the packet from the input stream --- */
                        //printf("I'm QoSpGtw_proc, processing the packet received\n");
                        pkptr = op_pk_get (op_intrpt_strm ());
                        op_pk_nfd_get (pkptr, "Msg_type", &msg_type);
                        switch ( msg_type ) {
                        case USR_REQ:
                        /* -- get the destination router ID and bandwidth requirement --- */
                        op_pk_nfd_get (pkptr, "Req_type", &req_type);
                        op_pk_nfd_get (pkptr, "Usr_id", &user_id);
                        op_pk_nfd_get (pkptr, "Dest_router", &dest_router);
                        op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
                        op_pk_nfd_get (pkptr, "Bdw_req", &bdw_req);
                        op_pk_nfd_get (pkptr, "Call_duration", &call_duration);
                        op_pk_nfd_get (pkptr, "Req_time", &req_time);
                        op_pk_nfd_get (pkptr, "Proc_id", &proc_id);
                        if(dest_router==Router_ID)
                        {
                        /* --- send out the blocked user request to the user --- */
```

```
                                    req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
                                    op_pk_total_size_set (req_adm_pkptr, 1024);
                                    op_pk_nfd_set (req_adm_pkptr, "Msg_type",
USR_REQ_ADMITTED);
                                    op_pk_nfd_set (req_adm_pkptr, "Usr_id", user_id);
                                    op_pk_nfd_set (req_adm_pkptr, "Proc_id", proc_id);
                                    op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                                    op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", dest_gtw);
                                    op_pk_nfd_set (req_adm_pkptr, "Decision", BLOCKED_i);
                                    op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
                                    }
                                    else {
                                     /* calc the all user req*/
                                    all_incoming_num++;
                                    if (op_sim_time() >= 4990)
                                    {printf("number income =%i\n",all_incoming_num);}
                                    if (req_time==1)
                                     {
                                     all_req_num = all_req_num + 1;
                                     total_CAC = total_CAC + (1*1024);
                                                }
                                    else
                                                {
                                    total_CAC = total_CAC + (1*1024);
                                    recall_overhead_num = recall_overhead_num+1;
                                    op_stat_write(recall_overhead_hndl,recall_overhead_num);
                                           }
                                    op_stat_write(total_CAC_hndl,total_CAC);
                                    /* --- look up the QoS route table and trunk table to find the
widest and narrowest one for the user request --- */
                                    route_ptr = route_table [dest_router].ptr;
                                    max_avl_bdw = 0;
                                    widest_route_ptr = NULL;
                                    narrowest_route_ptr=NULL;
                                    min_avl_bdw = 10000;
                                    narrowest_route_id = -1;
                                    need_reconfig_route_ptr=NULL;
                                    if ( route_ptr != NULL ) {
                                    count = 0;
                                    /*  check whether the QoS route is qualified for the user request */
                                    route_avl_bdw = 10000;
                                    while ( count < route_ptr->Length )
                                    {
                                    if(req_type==VIDEO_TRAFFIC)
                                    tmp_trunk_ptr = trunk_table[route_ptr->TrunkList[count]];
                                    else
                tmp_trunk_ptr = trunk_table[route_ptr->TrunkList[count]+MAX_LINK_NUM];
                tmp_avl_bdw = tmp_trunk_ptr.RsvBdw - tmp_trunk_ptr.UsdBdw;
                if(tmp_trunk_ptr.RsvBdw<bdw_req)
                need_reconfig_route_ptr=route_ptr;
                if ( route_avl_bdw > tmp_avl_bdw )
                route_avl_bdw = tmp_avl_bdw;
                count++;
                }
                if ( route_avl_bdw >= bdw_req )
                /* --- current QoS route is qualified, check to see whether it's the widest one --- */
                 {
                idest_route_ptr = route_ptr;
```

```
        if (route_avl_bdw < 2*bdw_req)
        need_reconfig_route_ptr=route_ptr;
        }
        else {
        /*current QoS route isn't qualified, check to see whether it's the narrowest one*/
        narrowest_route_ptr = route_ptr;need_reconfig_route_ptr=route_ptr;}
        }
        if ( widest_route_ptr != NULL )
        {
                        /* --- there is one qualified, generate the user_req_admitted packet --- */
                                widest_route_id = widest_route_ptr->RouteID;
                                //printf("qualified route ID: %i",widest_route_id);
                        /* --- allocate a VIP for the user request --- */
                                count = 0;
                                runtime = (QoSRuntime *) op_prg_mem_alloc (sizeof
(QoSRuntime));
                                time = ceil(op_sim_time() + call_duration + 1);
                                runtime->GatewayID = edge_name;
                                runtime->Vip_id = vip_no;
                                runtime->Class = req_type;
                                runtime->DestRouter = dest_router;
                                runtime->UsdBdw = bdw_req;
                                runtime->Finish_time = time;
                                op_prg_list_insert_sorted (data_ptr, runtime, stoptime_comp);
                                while ( count < widest_route_ptr->Length )
                                {
                                /* --- generate a new VIP record --- */
                                tmp_vip_ptr = ( struct VipDef * ) malloc ( sizeof (struct VipDef) );
                                tmp_vip_ptr->VipID = vip_no;
                                tmp_vip_ptr->UsdBdw = bdw_req;
                                /* --- update the trunk resource usage information --- */
                                if(req_type==VIDEO_TRAFFIC)
                        {trunk_table[widest_route_ptr->TrunkList[count]].UsdBdw += bdw_req;
                                }
                                else
                        {trunk_table[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM].UsdBdw += bdw_req;
                        if (widest_route_ptr->TrunkList[count]+MAX_LINK_NUM>=20)
                        op_stat_write(trunk_usdbdw_hndl[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM],(double)trunk_table[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM].UsdBdw);
                        }
                        count++;
                        }
                        /* --- generate the usr_req_adm --- */
                        req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
                        op_pk_total_size_set (req_adm_pkptr, 1024);
                        op_pk_nfd_set (req_adm_pkptr, "Msg_type", USR_REQ_ADMITTED);
                        op_pk_nfd_set (req_adm_pkptr, "Usr_id", user_id);
                        op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                        op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", dest_gtw);
                        op_pk_nfd_set (req_adm_pkptr, "Route_id", widest_route_id);
                        op_pk_nfd_set (req_adm_pkptr, "Vip_id", vip_no);
                        op_pk_nfd_set (req_adm_pkptr, "Decision", ADMITTED);
                        op_pk_nfd_set (req_adm_pkptr, "Proc_id", proc_id);
                        op_pk_nfd_set (req_adm_pkptr, "Req_time", req_time);
                        op_pk_nfd_set (req_adm_pkptr, "Src_router", Router_ID);
                        op_pk_nfd_set (req_adm_pkptr, "Req_type", req_type);
                        op_pk_nfd_set (req_adm_pkptr, "Destination", dest_router);
                        op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
```

```
                    total_CAC = total_CAC + (1*1024);
                    op_stat_write(total_CAC_hndl,total_CAC);
                    /* calc the admit_req_num and blocking rate*/
                    /* --- send out the new vip information along the chosen QoS route --- */
                    req_adm_pkptr = op_pk_create_fmt (QOS_VIP_EST);
                    op_pk_total_size_set (req_adm_pkptr, 1024);
                    op_pk_nfd_set (req_adm_pkptr, "Msg_type", VIP_EST);
                    op_pk_nfd_set (req_adm_pkptr, "Src_router", Router_ID);
                    op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                    op_pk_nfd_set (req_adm_pkptr, "Dest_router", dest_router);
                    op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", dest_gtw);
                    op_pk_nfd_set (req_adm_pkptr, "Route_id", widest_route_id);
                    op_pk_nfd_set (req_adm_pkptr, "Vip_id", vip_no);
                    op_pk_nfd_set (req_adm_pkptr, "Bdw_used", bdw_req);
                    /* --- increase the vip_no for future use --- */
                    vip_no++;
                    }
                    else {
                    all_hold_num++;             //KiT
                    if (op_sim_time() >= 4990)
                    {printf("number hold =%i\n",all_hold_num);}
                    flowrec.UsrID               = user_id;
                    flowrec.VipID               = vip_no;
                    flowrec.ReqType             = req_type;
                    flowrec.ProcID              = proc_id;
                    flowrec.SrcRouter           = Router_ID;
                    flowrec.SrcGtw              = Gtw_ID;
                    flowrec.DestRouter          = dest_router;
                    flowrec.DestGtw             = dest_gtw;
                    flowrec.BdwReq              = bdw_req;
                    flowrec.CallDuration        = call_duration;
                    flowrec.Reqtime             = req_time;
                    flowrec.RecallTime          = 0;
                    frag                        = 1;
                    if ((frag == 1) && (stat_count == 0))
                    {
                    op_intrpt_schedule_self (op_sim_time () + Released_Period,
RECALL_FLOW);
                    stat_count = 1;
                    }
                    hold_admit_pkptr = op_pk_create_fmt (QOS_EDGE_HLD_ADM);
                    op_pk_total_size_set (hold_admit_pkptr, 1024);
                    op_pk_nfd_set (hold_admit_pkptr, "Msg_type",
EDGE_HLD_ADMITTED);
                    op_pk_nfd_set (hold_admit_pkptr, "Usr_id", flowrec.UsrID);
                    op_pk_nfd_set (hold_admit_pkptr, "Vip_id", flowrec.VipID);
                    op_pk_nfd_set (hold_admit_pkptr, "Proc_id", flowrec.ProcID);
                    op_pk_nfd_set (hold_admit_pkptr, "Src_gtw", flowrec.SrcGtw);
                    op_pk_nfd_set (hold_admit_pkptr, "Dest_gtw", flowrec.DestGtw);
                    op_pk_nfd_set (hold_admit_pkptr, "Req_type", flowrec.ReqType);
                    op_pk_nfd_set (hold_admit_pkptr, "Req_time", flowrec.Reqtime);
                    op_pk_nfd_set (hold_admit_pkptr, "Decision", HOLD_ADMITTED);
                    op_pk_send (hold_admit_pkptr, XMT_OUT_TO_USER);
                    total_CAC = total_CAC + (1*1024);
                    op_stat_write(total_CAC_hndl,total_CAC);
                    op_intrpt_schedule_self (op_sim_time (), HOLD_FLOW);
                    }
                    if ( need_reconfig_route_ptr != NULL )
                    {
                    trk_cfg_pkptr = op_pk_create_fmt (QOS_TRK_CFG_REQ);
```

```
                        op_pk_total_size_set (trk_cfg_pkptr, 1024);
                        op_pk_nfd_set (trk_cfg_pkptr, "Msg_type", TRK_CFG_REQ);
                        op_pk_nfd_set (trk_cfg_pkptr, "Src_router", Router_ID);
                        op_pk_nfd_set (trk_cfg_pkptr, "Src_gtw",    Gtw_ID);
                        op_pk_nfd_set (trk_cfg_pkptr, "Dest_router",
POLICY_SERVER_ROUTER);
                        op_pk_nfd_set (trk_cfg_pkptr, "Dest_gtw", POLICY_SERVER);
                        op_pk_nfd_set (trk_cfg_pkptr, "Bdw_req",
bdw_req*INCREASING_TIMES);
                        op_pk_nfd_set (trk_cfg_pkptr, "Route_id", rsv_route_id);
                        count = 0;
                        i = 0;
                        notfeasible=0;
                        while ( count < need_reconfig_route_ptr->Length )
                        {
                        if(req_type==VIDEO_TRAFFIC)
                        tmp_trunk_ptr = trunk_table[need_reconfig_route_ptr->TrunkList[count]];
                        else
                        tmp_trunk_ptr = trunk_table[need_reconfig_route_ptr-
>TrunkList[count]+MAX_LINK_NUM];
                        tmp_avl_bdw = tmp_trunk_ptr.RsvBdw - tmp_trunk_ptr.UsdBdw;
                        if ( tmp_avl_bdw < bdw_req )
                        {
                        op_pk_fd_set (trk_cfg_pkptr, i+5 , OPC_FIELD_TYPE_INTEGER,
tmp_trunk_ptr.TrunkID, 8);
                        i++;
                        }
                        count++;
                        }
                        if(notfeasible==0)
                        {
                        op_pk_nfd_set (trk_cfg_pkptr, "Length", i);
                        op_pk_send (trk_cfg_pkptr, XMT_OUT_TO_ROUTER);
                        }
                        else op_pk_destroy (trk_cfg_pkptr);
                        }}
                        op_pk_destroy (pkptr);
                        break;
                        case USR_PKT_END:
                        op_pk_nfd_get (pkptr, "Req_type", &req_type);
                        op_pk_nfd_get (pkptr, "Dest_router", &dest_router);
                        op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
                        op_pk_nfd_get (pkptr, "Route_id", &route_id);
                        op_pk_nfd_get (pkptr, "Vip_id", &vip_id);
                        op_pk_nfd_get (pkptr, "Usd_bdw", &usd_bdw);
                        /* --- locate the QoS route --- */
                        route_ptr = route_table[dest_router].ptr;
                        while ( (route_ptr->RouteID != route_id)&&(route_ptr != NULL) )
                        route_ptr = route_ptr->Next;
                        if ( route_ptr->RouteID == route_id )
                        {
                        count = 0;
                        while ( count < route_ptr->Length )
                        {
                        if (req_type==VIDEO_TRAFFIC)
                        trunk_table[route_ptr->TrunkList[count]].UsdBdw -= usd_bdw;
                        else
                        {trunk_table[route_ptr->TrunkList[count]+MAX_LINK_NUM].UsdBdw -=
usd_bdw;
```

```
                        if((route_ptr->TrunkList[count]+MAX_LINK_NUM>=20)&&(route_ptr-
>TrunkList[count]+MAX_LINK_NUM<=30))
                        op_stat_write(trunk_usdbdw_hndl[route_ptr-
>TrunkList[count]+MAX_LINK_NUM],(double)trunk_table[route_ptr-
>TrunkList[count]+MAX_LINK_NUM].UsdBdw);
                         }
                        count++;
                        }}
                        else printf("error occur; no such a QoS route for the destination
gateway %i\n", dest_gtw);
                        del_size = op_prg_list_size (data_ptr);
                        del_size = del_size - 1;
                        for (del = 0; del < del_size; del++)
                        {del_ptr = (QoSRuntime*) op_prg_mem_alloc (sizeof (QoSRuntime));
                        del_ptr = ((char *) op_prg_list_access (data_ptr,del));
                        if ((del_ptr->Class == req_type) && (del_ptr->DestRouter == dest_router)
&& (del_ptr->Vip_id == vip_id) && (del_ptr->UsdBdw == usd_bdw))
                        {
                        op_prg_list_remove(data_ptr,del);
                        }}
                        op_pk_destroy (pkptr);
                        req_adm_pkptr = op_pk_create_fmt (QOS_VIP_RLS);
                        op_pk_total_size_set (req_adm_pkptr, 1024);
                        op_pk_nfd_set (req_adm_pkptr, "Msg_type", VIP_RLS);
                        op_pk_nfd_set (req_adm_pkptr, "Src_router", Router_ID);
                        op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                        op_pk_nfd_set (req_adm_pkptr, "Dest_router", dest_router);
                        op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", dest_gtw);
                        op_pk_nfd_set (req_adm_pkptr, "Route_id", route_id);
                        op_pk_nfd_set (req_adm_pkptr, "Vip_id", vip_id);
                        op_pk_nfd_set (req_adm_pkptr, "Usd_bdw", usd_bdw);
                        //op_pk_send (req_adm_pkptr, XMT_OUT_TO_ROUTER);
                        break;
                        case TRK_CFG_CNF:
                        //printf("received reconfig confirm in router %i gtw %i \n", Router_ID,
Gtw_ID);
                        //scanf("%i",&rkey);
                        op_pk_nfd_get (pkptr, "Length", &length);
                        count = 0;
                        for(i=0;i<length;i++)
                        {
                        op_pk_fd_get (pkptr, count+4, &trunk_id);
                        op_pk_fd_get (pkptr, count+5, &bdw_req);
                        j = 0;
                        while ( (j<MAX_TRUNK_NUM) && (trunk_table[j].TrunkID !=
trunk_id) )
                        j++;
                        if ( trunk_table[j].TrunkID == trunk_id )
                        { trunk_table[j].RsvBdw += bdw_req;
                        //printf("trunk %i +bdw %i ", trunk_id, bdw_req);
                        //scanf("%i",&rkey);
                        if(trunk_id>=20)
                        op_stat_write(trunk_rsvbdw_hndl[j],trunk_table[j].RsvBdw);
                         }
                        count=count+2;
                         }
                        op_pk_destroy (pkptr);
                        overhead_msg_num=overhead_msg_num +1;
                        op_stat_write(overhead_hndl,overhead_msg_num);
```

```
local_overhead_num++;
break;
case VIP_EST:
case VIP_RLS:
op_pk_destroy (pkptr);
break;
case PROVLINK_BUSY:
op_pk_nfd_get (pkptr, "Length", &i);
count = 0;
while ( count < i )
{
op_pk_fd_get (pkptr, count+3, &plink_id);
j = 0;
while  (j<MAX_TRUNK_NUM)
{ if (trunk_table[j].PLinkID==plink_id)
trunk_table[j].Busy=1;
j++;
}
count++;
}
overhead_msg_num=overhead_msg_num +1;
op_stat_write(overhead_hndl,overhead_msg_num);
local_overhead_num++;
op_pk_destroy(pkptr);
break;
case PROVLINK_NOTBUSY:
op_pk_nfd_get (pkptr, "Length", &i);
count = 0;
while ( count < i )
{
op_pk_fd_get (pkptr, count+3, &plink_id);
j = 0;
while  (j<MAX_TRUNK_NUM)
{ if (trunk_table[j].PLinkID == plink_id)
trunk_table[j].Busy=0;
j++;
}
count++;
}
overhead_msg_num=overhead_msg_num +1;
op_stat_write(overhead_hndl,overhead_msg_num);
local_overhead_num++;
op_pk_destroy(pkptr);
break;
case EDGE_HLD_REP:
op_pk_nfd_get (pkptr, "Decision", &decision);
op_pk_nfd_get (pkptr, "Usr_id", &user_id);
op_pk_nfd_get (pkptr, "Vip_id", &vip_id);
op_pk_nfd_get (pkptr, "Proc_id", &proc_id);
op_pk_nfd_get (pkptr, "Req_type", &req_type);
switch ( decision ) {
case HOLD_ACCEPT:
total_CAC = total_CAC + (1*1024);
op_stat_write(total_CAC_hndl,total_CAC);
break;
case HOLD_REJECT:
total_CAC = total_CAC + (1*1024);
op_stat_write(total_CAC_hndl,total_CAC);
switch ( req_type ) {
```

```
                        case 0:
                        hold_size = op_prg_list_size (hold_rec0);
                        for (n = 0; n < hold_size; n++)
                        {hold_reply = (FlowData*) op_prg_mem_alloc (sizeof (FlowData));

                        hold_reply = ((char*) op_prg_list_access (hold_rec0, n));
                        if ((user_id == hold_reply->UsrID) && (proc_id == hold_reply->ProcID)
&& (time_est == hold_reply->RecallTime))
                        {
                        block_req_num++;
                        req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
                        op_pk_total_size_set (req_adm_pkptr, 1024);
                        op_pk_nfd_set (req_adm_pkptr, "Msg_type", USR_REQ_ADMITTED);
                        op_pk_nfd_set (req_adm_pkptr, "Usr_id", hold_reply->UsrID);
                        op_pk_nfd_set (req_adm_pkptr, "Proc_id", hold_reply->ProcID);
                        op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                        op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", hold_reply->DestGtw);
                        op_pk_nfd_set (req_adm_pkptr, "Decision", BLOCKED);
                        printf("USER BLOCKED! at router %i gtw %i dest :%i req-type: %i at
time %lf\n",Router_ID,Gtw_ID,hold_reply->DestRouter,hold_reply->ReqType,op_sim_time());
                        op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
                        op_stat_write(blocking_rate_hndl,
(double)((double)block_req_num/(double)all_req_num));
                        op_prg_list_remove(hold_rec0, n);
                        break;
                        }}
                        break;
                        case 1:
                        hold_size = op_prg_list_size (hold_rec1);
                        for (n = 0; n < hold_size; n++)
                        {hold_reply = (FlowData*) op_prg_mem_alloc (sizeof (FlowData));
                        hold_reply = ((char*) op_prg_list_access (hold_rec1, n));
                        if ((user_id == hold_reply->UsrID) && (proc_id == hold_reply->ProcID)
&& (time_est == hold_reply->RecallTime))
                        {
                        block_req_num++;
                        req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
                        op_pk_total_size_set (req_adm_pkptr, 1024);
                        op_pk_nfd_set (req_adm_pkptr, "Msg_type", USR_REQ_ADMITTED);
                        op_pk_nfd_set (req_adm_pkptr, "Usr_id", hold_reply->UsrID);
                        op_pk_nfd_set (req_adm_pkptr, "Proc_id", hold_reply->ProcID);
                        op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                        op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", hold_reply->DestGtw);
                        op_pk_nfd_set (req_adm_pkptr, "Decision", BLOCKED);
                        printf("USER BLOCKED! at router %i gtw %i dest :%i req-type: %i at
time %lf\n",Router_ID,Gtw_ID,hold_reply->DestRouter,hold_reply->ReqType,op_sim_time());
                        op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
                        op_stat_write(blocking_rate_hndl,
(double)((double)block_req_num/(double)all_req_num));
                        op_prg_list_remove(hold_rec1, n);
                        break;
                        }}
                        break;
                        }
                        break;
                        }
                        op_pk_destroy(pkptr);
                        break;
                        /*case USR_DATA :
```

```
                        op_pk_nfd_get (pkptr, "Dest_addr", &dest_addr);
                        if(dest_addr==gtw_id)                    //only collect e2e between G1 G4
                        {
                        received_packets++;
                        packet_loss_rate= (send_out_packets-received_packets)/send_out_packets;
                        op_stat_write (packet_loss_rate_hndl, (double)packet_loss_rate);
                        op_pk_nfd_get (pkptr, "Src_addr", &src_addr);
                        op_pk_nfd_get (pkptr, "Sendout_time", &sendout_time);
                        if((src_addr==1)&&(dest_addr==4))
                        {// printf("received usrdata src_addr %i dest_addr %i sendout_time :%lf\n",
src_addr,dest_addr, sendout_time);
                        op_pk_nfd_get (pkptr, "Sendout_time", &sendout_time);
                        e2edelay=op_sim_time()-sendout_time;
                        op_pk_nfd_get (pkptr, "Qos_type", &qos_type);
                        if (qos_type==AUDIO_TRAFFIC)
                        op_stat_write (audio_e2edelay_hndl, e2edelay);
                        if (qos_type==VIDEO_TRAFFIC)
                        op_stat_write (video_e2edelay_hndl, e2edelay);
                        }
                        op_pk_destroy(pkptr);
                        }
                        else op_pk_send (pkptr, XMT_OUT_TO_ROUTER);
                        break;*/
                        }}
                        /** state (pktproc) exit executives **/
                        FSM_STATE_EXIT_FORCED (2, "pktproc", "QoSpGtw_proc () [pktproc
exit execs]")
                        {
                        }
                        /** state (pktproc) transition processing **/
                        FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "pktproc",
"dispatch")
                        /*---------------------------------------------------------*/
                        /** state (rpt) enter executives **/
                        FSM_STATE_ENTER_FORCED (3, state3_enter_exec, "rpt",
"QoSpGtw_proc () [rpt enter execs]")
                        {
                        /* --- check the resource usage information of each trunk to see whether
there is excess resourc to release ---- */
                        //printf(" Time out");
                        //scanf("%i", &rkey);
                        count = 0;
                        for ( i=0; i< MAX_TRUNK_NUM; i++ )
                        {  if (i<=19) lowest_threshhold=3000;
                            else    lowest_threshhold=64;
                          if (( EXCESS_THRESHOLD*trunk_table[i].UsdBdw <
trunk_table[i].RsvBdw )&& (trunk_table[i].RsvBdw-(int) trunk_table[i].RsvBdw /
RELEASE_PERCENT>=lowest_threshhold))
                           {
                        /* -- is this the beginning of a new trk_rsc_rls packet -- */
                        if ( count == 0 ) {
                        trk_rsc_pkptr = op_pk_create_fmt (QOS_TRK_RSC_RLS);
                        op_pk_total_size_set (trk_rsc_pkptr, 1024);
                        op_pk_nfd_set (trk_rsc_pkptr, "Msg_type", TRK_RSC_RLS);
                        op_pk_nfd_set (trk_rsc_pkptr, "Src_router", Router_ID);
                        op_pk_nfd_set (trk_rsc_pkptr, "Src_gtw", Gtw_ID);
                        op_pk_nfd_set (trk_rsc_pkptr, "Dest_router",
POLICY_SERVER_ROUTER);
                        op_pk_nfd_set (trk_rsc_pkptr, "Dest_gtw", POLICY_SERVER);
```

```
                        op_pk_nfd_set (trk_rsc_pkptr, "Route_id", rsv_route_id);
                        }
                        op_pk_fd_set (trk_rsc_pkptr, count+4, OPC_FIELD_TYPE_INTEGER,
trunk_table[i].TrunkID, 8);
                        op_pk_fd_set (trk_rsc_pkptr, count+5, OPC_FIELD_TYPE_INTEGER,
(int) trunk_table[i].RsvBdw / RELEASE_PERCENT, 16);
                        trunk_table[i].RsvBdw -= (int) trunk_table[i].RsvBdw /
RELEASE_PERCENT;
                        if(i>=20)
                        op_stat_write(trunk_rsvbdw_hndl[i],trunk_table[i].RsvBdw);
                        count=count+2;
                        if ( count == 20 ) {
                        op_pk_nfd_set (trk_rsc_pkptr, "Length", (int)count/2);
                        op_pk_send (trk_rsc_pkptr, XMT_OUT_TO_ROUTER);
                        count = 0;
                        local_overhead_num++;
                        }}}
                        if ( count > 0 ) {
                                        op_pk_nfd_set (trk_rsc_pkptr, "Length", (int)count/2);
                                        op_pk_send (trk_rsc_pkptr, XMT_OUT_TO_ROUTER);
                                        local_overhead_num++;
                                        }
                        /* --- schedule the next interruption for resource check and release --- */
                        op_intrpt_schedule_self (op_sim_time () + CHECK_PERIOD,
RESOURCE_REPORT);
                        }
                        /** state (rpt) exit executives **/
                        FSM_STATE_EXIT_FORCED (3, "rpt", "QoSpGtw_proc () [rpt exit
execs]")
                        {
                        }
                        /** state (rpt) transition processing **/
                        FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "rpt",
"dispatch")
                        /*---------------------------------------------------------*/
                        /** state (HOLD) enter executives **/
                        FSM_STATE_ENTER_FORCED (4, state4_enter_exec, "HOLD",
"QoSpGtw_proc () [HOLD enter execs]")
                                {
                        switch ( flowrec.ReqType ) {
                        case 0:
                        {//printf("HOLD CLASS 0\n");
                        temp_hold0 = (FlowData*) op_prg_mem_alloc (sizeof (FlowData));
                        temp_hold0->UsrID               = flowrec.UsrID;
                        temp_hold0->VipID               = flowrec.VipID;
                        temp_hold0->ReqType     = flowrec.ReqType;
                        temp_hold0->ProcID              = flowrec.ProcID;
                        temp_hold0->SrcRouter   = flowrec.SrcRouter;
                        temp_hold0->SrcGtw              = flowrec.SrcGtw;
                        temp_hold0->DestRouter  = flowrec.DestRouter;
                        temp_hold0->DestGtw     = flowrec.DestGtw;
                        temp_hold0->BdwReq              = flowrec.BdwReq;
                        temp_hold0->CallDuration = flowrec.CallDuration;
                        temp_hold0->Reqtime     = flowrec.Reqtime;
                        temp_hold0->QueueID             = queueC_id;
                        op_prg_list_insert_sorted (hold_rec0, temp_hold0, queue_comp);
                        info_timeout0    = (TimeOut*) op_prg_mem_alloc (sizeof (TimeOut));
                        info_timeout0->ProcID    = flowrec.ProcID;
```

```
                        info_timeout0->TimeOut  = op_sim_time() + LIFETIME;
                        next_timeout0 = op_sim_time() + LIFETIME;
                        op_prg_list_insert (LifeTime0, info_timeout0, OPC_LISTPOS_TAIL);
                        sent_code0 = info_timeout0->ProcID;
                        set_timeout0();
                        queueC_id++;
                        break;}
                        case 1:
                        {//printf("HOLD CLASS 1\n");
                        temp_hold1 = (FlowData*) op_prg_mem_alloc (sizeof (FlowData));
                        temp_hold1->UsrID                = flowrec.UsrID;
                        temp_hold1->VipID                = flowrec.VipID;
                        temp_hold1->ReqType      = flowrec.ReqType;
                        temp_hold1->ProcID               = flowrec.ProcID;
                        temp_hold1->SrcRouter    = flowrec.SrcRouter;
                        temp_hold1->SrcGtw               = flowrec.SrcGtw;
                        temp_hold1->DestRouter   = flowrec.DestRouter;
                        temp_hold1->DestGtw      = flowrec.DestGtw;
                        temp_hold1->BdwReq               = flowrec.BdwReq;
                        temp_hold1->CallDuration = flowrec.CallDuration;
                        temp_hold1->Reqtime      = flowrec.Reqtime;
                        temp_hold1->QueueID              = queueC_id1;
                        op_prg_list_insert_sorted (hold_rec1, temp_hold1, queue_comp);
                        info_timeout1     = (TimeOut*) op_prg_mem_alloc (sizeof (TimeOut));
                        info_timeout1->ProcID     = flowrec.ProcID;
                        info_timeout1->TimeOut  = op_sim_time() + LIFETIME;
                        next_timeout1 = op_sim_time() + LIFETIME;
                        op_prg_list_insert (LifeTime1, info_timeout1, OPC_LISTPOS_TAIL);
                        sent_code1 = info_timeout1->ProcID;
                        queueC_id1++;
                        break;}
                        break;}
                        }
                        /** state (HOLD) exit executives **/
                        FSM_STATE_EXIT_FORCED (4, "HOLD", "QoSpGtw_proc () [HOLD
exit execs]")
                        {
                        }
                        /** state (HOLD) transition processing **/
                        FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "HOLD",
"dispatch")
                        /*---------------------------------------------------------*/
                        /** state (RECALL) enter executives **/
                        FSM_STATE_ENTER_FORCED (5, state5_enter_exec, "RECALL",
"QoSpGtw_proc () [RECALL enter execs]")
                                {
                        if (frag == 1)
                        {//printf("RECALL STATE\n");
                        recall0_size = op_prg_list_size (hold_rec0);
                        //printf("SIZE %i\n",recall0_size);
                        recall1_size = op_prg_list_size (hold_rec1);
                        //printf("SIZEI %i\n",recall1_size);
                        if (recall0_size != 0)
                        {
                        //list_size = op_prg_list_size (hold_rec0);
                        for (n = 0; n < recall0_size; n++)
                        {//printf("NUMBER0 %i\n",recall0_size);
                        x = n - 1;
```

```
tmp_hold0 = (FlowData*) op_prg_mem_alloc (sizeof (FlowData));
tmp_hold0 = ((char *) op_prg_list_access (hold_rec0, n));
recall_table.UsrID                = tmp_hold0->UsrID;
recall_table.ReqType      = tmp_hold0->ReqType;
recall_table.ProcID             = tmp_hold0->ProcID;
recall_table.SrcRouter    = tmp_hold0->SrcRouter;
recall_table.SrcGtw       = tmp_hold0->SrcGtw;
recall_table.DestRouter   = tmp_hold0->DestRouter;
recall_table.DestGtw          = tmp_hold0->DestGtw;
recall_table.BdwReq           = tmp_hold0->BdwReq;
recall_table.CallDuration = tmp_hold0->CallDuration;
recall_table.Reqtime          = tmp_hold0->Reqtime;
recall_table.QueueID          = tmp_hold0->QueueID;
if(recall_table.DestRouter == Router_ID)
{
req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
op_pk_total_size_set (req_adm_pkptr, 1024);
op_pk_nfd_set (req_adm_pkptr, "Msg_type", USR_REQ_ADMITTED);
op_pk_nfd_set (req_adm_pkptr, "Usr_id", recall_table.UsrID);
op_pk_nfd_set (req_adm_pkptr, "Proc_id", recall_table.ProcID);
op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
op_pk_nfd_set (req_adm_pkptr, "Dest_gtw", recall_table.DestGtw);
op_pk_nfd_set (req_adm_pkptr, "Decision", BLOCKED_i);
op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
}
else {//printf("*******************CHECK POINT
I**********************\n");
route_ptr = route_table [recall_table.DestRouter].ptr;
max_avl_bdw = 0;
widest_route_ptr = NULL;
narrowest_route_ptr=NULL;
min_avl_bdw = 10000;
narrowest_route_id = -1;
need_reconfig_route_ptr=NULL;
if ( route_ptr != NULL ) {
count = 0;
route_avl_bdw = 10000;
        while ( count < route_ptr->Length )
        {
        if(recall_table.ReqType == VIDEO_TRAFFIC)
        tmp_trunk_ptr = trunk_table[route_ptr->TrunkList[count]];
        else
        tmp_trunk_ptr = trunk_table[route_ptr-
>TrunkList[count]+MAX_LINK_NUM];
        tmp_avl_bdw = tmp_trunk_ptr.RsvBdw - tmp_trunk_ptr.UsdBdw;
        if(tmp_trunk_ptr.RsvBdw < recall_table.BdwReq)
        need_reconfig_route_ptr=route_ptr;
        if ( route_avl_bdw > tmp_avl_bdw )
        route_avl_bdw = tmp_avl_bdw;
        count++;
        }
        if ( route_avl_bdw >= recall_table.BdwReq )
        {
        widest_route_ptr = route_ptr;
        if (route_avl_bdw < 2*recall_table.BdwReq)
        need_reconfig_route_ptr=route_ptr;
        }
        else {
        narrowest_route_ptr = route_ptr;
```

```
                                need_reconfig_route_ptr=route_ptr;
                                }}
                                if ( widest_route_ptr != NULL )
                                {
                                widest_route_id = widest_route_ptr->RouteID;
                                count = 0;
                                runtime = (QoSRuntime *) op_prg_mem_alloc (sizeof
(QoSRuntime));
                                time = ceil(op_sim_time() + recall_table.CallDuration + 1);
                                runtime->GatewayID = edge_name;
                                runtime->Vip_id = vip_no;
                                runtime->Class = recall_table.ReqType;
                                runtime->DestRouter = recall_table.DestRouter;
                                runtime->UsdBdw = recall_table.BdwReq;
                                runtime->Finish_time = time;
                                op_prg_list_insert_sorted (data_ptr, runtime, stoptime_comp);
                                while ( count < widest_route_ptr->Length )
                                {
                                /* --- generate a new VIP record --- */
                                tmp_vip_ptr = ( struct VipDef * ) malloc ( sizeof (struct VipDef) );
                                tmp_vip_ptr->VipID = vip_no;
                                tmp_vip_ptr->UsdBdw = recall_table.BdwReq;
                                /* --- update the trunk resource usage information --- */
                                if(recall_table.ReqType == VIDEO_TRAFFIC)
                                {trunk_table[widest_route_ptr->TrunkList[count]].UsdBdw +=
recall_table.BdwReq;
                                }
                                else
                                {trunk_table[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM].UsdBdw += recall_table.BdwReq;
                                if (widest_route_ptr->TrunkList[count]+MAX_LINK_NUM>=20)
                                op_stat_write(trunk_usdbdw_hndl[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM],(double)trunk_table[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM].UsdBdw);
                                }
                                count++;
                                }
                                req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
                                op_pk_total_size_set (req_adm_pkptr, 1024);
                                op_pk_nfd_set (req_adm_pkptr, "Msg_type",
USR_REQ_ADMITTED);
                                op_pk_nfd_set (req_adm_pkptr, "Usr_id", recall_table.UsrID);
                                op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Dest_gtw",
recall_table.DestGtw);
                                op_pk_nfd_set (req_adm_pkptr, "Route_id", widest_route_id);
                                op_pk_nfd_set (req_adm_pkptr, "Vip_id", vip_no);
                                op_pk_nfd_set (req_adm_pkptr, "Decision", ADMITTED_i);
                                op_pk_nfd_set (req_adm_pkptr, "Proc_id", recall_table.ProcID);
                                op_pk_nfd_set (req_adm_pkptr, "Req_time",
recall_table.Reqtime);
                                op_pk_nfd_set (req_adm_pkptr, "Src_router", Router_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Req_type",
recall_table.ReqType);
                                op_pk_nfd_set (req_adm_pkptr, "Destination",
recall_table.DestRouter);
                                op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
                                total_CAC = total_CAC + (1*1024);
                                op_stat_write(total_CAC_hndl,total_CAC);
                                /* calc the admit_req_num and blocking rate*/
```

```
                                /*send out the new vip information along the chosen QoS route*/
                                req_adm_pkptr = op_pk_create_fmt (QOS_VIP_EST);
                                op_pk_total_size_set (req_adm_pkptr, 1024);
                                op_pk_nfd_set (req_adm_pkptr, "Msg_type", VIP_EST);
                                op_pk_nfd_set (req_adm_pkptr, "Src_router", Router_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Dest_router",
recall_table.DestRouter);
                                op_pk_nfd_set (req_adm_pkptr, "Dest_gtw",
recall_table.DestGtw);
                                op_pk_nfd_set (req_adm_pkptr, "Route_id", widest_route_id);
                                op_pk_nfd_set (req_adm_pkptr, "Vip_id", vip_no);
                                op_pk_nfd_set (req_adm_pkptr, "Bdw_used",
recall_table.BdwReq);
                                /* --- increase the vip_no for future use --- */
                                vip_no++;
                                temp_record0 = (ReleasedRecord*) op_prg_mem_alloc (sizeof
(ReleasedRecord));
                                temp_record0->QueueID = recall_table.QueueID;
                                op_prg_list_insert (delete_info0, temp_record0,
OPC_LISTPOS_TAIL);
                                }
                                else {
                                /* --- send out the blocked user request to the user --- */
                                }
                                if ( need_reconfig_route_ptr != NULL )
                                {
                                trk_cfg_pkptr = op_pk_create_fmt (QOS_TRK_CFG_REQ);
                                op_pk_total_size_set (trk_cfg_pkptr, 1024);
                                op_pk_nfd_set (trk_cfg_pkptr, "Msg_type", TRK_CFG_REQ);
                                op_pk_nfd_set (trk_cfg_pkptr, "Src_router", Router_ID);
                                op_pk_nfd_set (trk_cfg_pkptr, "Src_gtw",    Gtw_ID);
                                op_pk_nfd_set (trk_cfg_pkptr, "Dest_router",
POLICY_SERVER_ROUTER);
                                op_pk_nfd_set (trk_cfg_pkptr, "Dest_gtw", POLICY_SERVER);
                                op_pk_nfd_set (trk_cfg_pkptr, "Bdw_req",
recall_table.BdwReq*INCREASING_TIMES);
                                op_pk_nfd_set (trk_cfg_pkptr, "Route_id", rsv_route_id);
                                count = 0;
                                i = 0;
                                notfeasible=0;
                                while ( count < need_reconfig_route_ptr->Length )
                                {
                                if(recall_table.ReqType == VIDEO_TRAFFIC)
                                tmp_trunk_ptr = trunk_table[need_reconfig_route_ptr-
>TrunkList[count]];
                                else
                                tmp_trunk_ptr = trunk_table[need_reconfig_route_ptr-
>TrunkList[count]+MAX_LINK_NUM];
                                tmp_avl_bdw = tmp_trunk_ptr.RsvBdw - tmp_trunk_ptr.UsdBdw;
                                if ( tmp_avl_bdw < recall_table.BdwReq )
                                {
                                op_pk_fd_set (trk_cfg_pkptr, i+5 ,
OPC_FIELD_TYPE_INTEGER, tmp_trunk_ptr.TrunkID, 8);
                                i++;
                                }
                                count++;
                                }
                                if(notfeasible==0)
```

```
                                {
                                op_pk_nfd_set (trk_cfg_pkptr, "Length", i);
                                op_pk_send (trk_cfg_pkptr, XMT_OUT_TO_ROUTER);
                                }
                                else op_pk_destroy (trk_cfg_pkptr);
                                }}}}
                                if (recall1_size != 0)
                                {
                                //list_size = op_prg_list_size (hold_rec1);
                                for (m = 0; m < recall1_size; m++)
                                {//printf("NUMBER1 %i\n",recall1_size);
                                x = m - 1;
                                tmp_hold1 = (FlowData*) op_prg_mem_alloc (sizeof
(FlowData));
                                tmp_hold1 = ((char *) op_prg_list_access (hold_rec1, m));
                                recall_table.UsrID          = tmp_hold1->UsrID;
                                recall_table.ReqType        = tmp_hold1->ReqType;
                                recall_table.ProcID         = tmp_hold1->ProcID;
                                recall_table.SrcRouter      = tmp_hold1->SrcRouter;
                                recall_table.SrcGtw         = tmp_hold1->SrcGtw;
                                recall_table.DestRouter     = tmp_hold1->DestRouter;
                                recall_table.DestGtw        = tmp_hold1->DestGtw;
                                recall_table.BdwReq         = tmp_hold1->BdwReq;
                                recall_table.CallDuration   = tmp_hold1->CallDuration;
                                recall_table.Reqtime        = tmp_hold1->Reqtime;
                                recall_table.QueueID        = tmp_hold1->QueueID;
                                if(recall_table.DestRouter == Router_ID)
                                {
                                req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
                                op_pk_total_size_set (req_adm_pkptr, 1024);
                                op_pk_nfd_set (req_adm_pkptr, "Msg_type",
USR_REQ_ADMITTED);
                                op_pk_nfd_set (req_adm_pkptr, "Usr_id", recall_table.UsrID);
                                op_pk_nfd_set (req_adm_pkptr, "Proc_id", recall_table.ProcID);
                                op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Dest_gtw",
recall_table.DestGtw);
                                op_pk_nfd_set (req_adm_pkptr, "Decision", BLOCKED_i);
                                op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
                                }
                                else {//printf("********************CHECK POINT
I**********************\n");
                                /*  calc the all user req*/
                                /* --- look up the QoS route table and trunk table to find the
widest and narrowest one for the user request --- */
                                route_ptr = route_table [recall_table.DestRouter].ptr;
                                max_avl_bdw = 0;
                                widest_route_ptr = NULL;
                                narrowest_route_ptr=NULL;
                                min_avl_bdw = 10000;
                                narrowest_route_id = -1;
                                need_reconfig_route_ptr=NULL;
                                if ( route_ptr != NULL ) {
                                count = 0;
                                /*check whether the QoS route is qualified for the user request*/
                                route_avl_bdw = 10000;
                                while ( count < route_ptr->Length )
                                {
                                if(recall_table.ReqType == VIDEO_TRAFFIC)
```

```
                                    tmp_trunk_ptr = trunk_table[route_ptr->TrunkList[count]];
                                    else
                                    tmp_trunk_ptr = trunk_table[route_ptr-
>TrunkList[count]+MAX_LINK_NUM];
                                    tmp_avl_bdw = tmp_trunk_ptr.RsvBdw - tmp_trunk_ptr.UsdBdw;
                                    if(tmp_trunk_ptr.RsvBdw < recall_table.BdwReq)
                                    need_reconfig_route_ptr=route_ptr;
                                    if ( route_avl_bdw > tmp_avl_bdw )
                                    route_avl_bdw = tmp_avl_bdw;
                                    count++;
                                    }
                                    if ( route_avl_bdw >= recall_table.BdwReq )
                                    {
                                    widest_route_ptr = route_ptr;
                                    if (route_avl_bdw < 2*recall_table.BdwReq)
                                    need_reconfig_route_ptr=route_ptr;
                                    }
                                    else {
                                    narrowest_route_ptr = route_ptr;
                                    need_reconfig_route_ptr=route_ptr;
                                    }}
                                    if ( widest_route_ptr != NULL )
                                    {
                                    widest_route_id = widest_route_ptr->RouteID;
                                    count = 0;
                                    runtime = (QoSRuntime *) op_prg_mem_alloc (sizeof
(QoSRuntime));
                                    time = ceil(op_sim_time() + recall_table.CallDuration + 1);
                                    runtime->GatewayID = edge_name;
                                    runtime->Vip_id = vip_no;
                                    runtime->Class = recall_table.ReqType;
                                    runtime->DestRouter = recall_table.DestRouter;
                                    runtime->UsdBdw = recall_table.BdwReq;
                                    runtime->Finish_time = time;
                                    op_prg_list_insert_sorted (data_ptr, runtime, stoptime_comp);
                                    while ( count < widest_route_ptr->Length )
                                    {
                                    /* --- generate a new VIP record --- */
                                    tmp_vip_ptr = ( struct VipDef * ) malloc ( sizeof (struct VipDef) );
                                    tmp_vip_ptr->VipID = vip_no;
                                    tmp_vip_ptr->UsdBdw = recall_table.BdwReq;
                                    /* --- update the trunk resource usage information --- */
                                    if(recall_table.ReqType == VIDEO_TRAFFIC)
                                {trunk_table[widest_route_ptr->TrunkList[count]].UsdBdw +=
recall_table.BdwReq;
                                    }
                                    else
                                    {trunk_table[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM].UsdBdw += recall_table.BdwReq;
                                    if (widest_route_ptr->TrunkList[count]+MAX_LINK_NUM>=20)
                                    op_stat_write(trunk_usdbdw_hndl[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM],(double)trunk_table[widest_route_ptr-
>TrunkList[count]+MAX_LINK_NUM].UsdBdw);
                                    }
                                    count++;
                                    }
                                    req_adm_pkptr = op_pk_create_fmt (QOS_USR_REQ_ADM);
                                    op_pk_total_size_set (req_adm_pkptr, 1024);
                                    op_pk_nfd_set (req_adm_pkptr, "Msg_type",
USR_REQ_ADMITTED);
```

```
                                op_pk_nfd_set (req_adm_pkptr, "Usr_id", recall_table.UsrID);
                                op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Dest_gtw",
recall_table.DestGtw);
                                op_pk_nfd_set (req_adm_pkptr, "Route_id", widest_route_id);
                                op_pk_nfd_set (req_adm_pkptr, "Vip_id", vip_no);
                                op_pk_nfd_set (req_adm_pkptr, "Decision", ADMITTED_i);
                                op_pk_nfd_set (req_adm_pkptr, "Proc_id", recall_table.ProcID);
                                op_pk_nfd_set (req_adm_pkptr, "Req_time",
recall_table.Reqtime);
                                op_pk_nfd_set (req_adm_pkptr, "Src_router", Router_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Req_type",
recall_table.ReqType);
                                op_pk_nfd_set (req_adm_pkptr, "Destination",
recall_table.DestRouter);
                                op_pk_send (req_adm_pkptr, XMT_OUT_TO_USER);
                                total_CAC = total_CAC + (1*1024);
                                op_stat_write(total_CAC_hndl,total_CAC);
                                req_adm_pkptr = op_pk_create_fmt (QOS_VIP_EST);
                                op_pk_total_size_set (req_adm_pkptr, 1024);
                                op_pk_nfd_set (req_adm_pkptr, "Msg_type", VIP_EST);
                                op_pk_nfd_set (req_adm_pkptr, "Src_router", Router_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Src_gtw", Gtw_ID);
                                op_pk_nfd_set (req_adm_pkptr, "Dest_router",
recall_table.DestRouter);
                                op_pk_nfd_set (req_adm_pkptr, "Dest_gtw",
recall_table.DestGtw);
                                op_pk_nfd_set (req_adm_pkptr, "Route_id", widest_route_id);
                                op_pk_nfd_set (req_adm_pkptr, "Vip_id", vip_no);
                                op_pk_nfd_set (req_adm_pkptr, "Bdw_used",
recall_table.BdwReq);
                                vip_no++;
                                temp_record1 = (ReleasedRecord*) op_prg_mem_alloc (sizeof
(ReleasedRecord));
                                temp_record1->QueueID = recall_table.QueueID;
                                op_prg_list_insert (delete_info1, temp_record1,
OPC_LISTPOS_TAIL);
                                }
                                else {
                                }
                                // reconfig route if necessary
                                if ( need_reconfig_route_ptr != NULL )
                                {
                                trk_cfg_pkptr = op_pk_create_fmt (QOS_TRK_CFG_REQ);
                                op_pk_total_size_set (trk_cfg_pkptr, 1024);
                                op_pk_nfd_set (trk_cfg_pkptr, "Msg_type", TRK_CFG_REQ);
                                op_pk_nfd_set (trk_cfg_pkptr, "Src_router", Router_ID);
                                op_pk_nfd_set (trk_cfg_pkptr, "Src_gtw",    Gtw_ID);
                                op_pk_nfd_set (trk_cfg_pkptr, "Dest_router",
POLICY_SERVER_ROUTER);
                                op_pk_nfd_set (trk_cfg_pkptr, "Dest_gtw", POLICY_SERVER);
                                op_pk_nfd_set (trk_cfg_pkptr, "Bdw_req",
recall_table.BdwReq*INCREASING_TIMES);
                                op_pk_nfd_set (trk_cfg_pkptr, "Route_id", rsv_route_id);
                                count = 0;
                                i = 0;
                                notfeasible=0;
                                while ( count < need_reconfig_route_ptr->Length )
                                {
                                if(recall_table.ReqType == VIDEO_TRAFFIC)
```

```
                                    tmp_trunk_ptr = trunk_table[need_reconfig_route_ptr-
>TrunkList[count]];
                                    else
                                    tmp_trunk_ptr = trunk_table[need_reconfig_route_ptr-
>TrunkList[count]+MAX_LINK_NUM];
                                    tmp_avl_bdw = tmp_trunk_ptr.RsvBdw - tmp_trunk_ptr.UsdBdw;
                                    if ( tmp_avl_bdw < recall_table.BdwReq )
                                    {
                                    op_pk_fd_set (trk_cfg_pkptr, i+5 ,
OPC_FIELD_TYPE_INTEGER, tmp_trunk_ptr.TrunkID, 8);
                                    i++;
                                    }
                                    count++;
                                    }
                                    if(notfeasible==0)
                                    {
                                    op_pk_nfd_set (trk_cfg_pkptr, "Length", i);
                                    op_pk_send (trk_cfg_pkptr, XMT_OUT_TO_ROUTER);
                                    }
                                    else op_pk_destroy (trk_cfg_pkptr);
                                    }}}}
                                    delete0_size = op_prg_list_size(delete_info0);
                                    //printf("delete SIZE %i\n",delete0_size);
                                    delete1_size = op_prg_list_size(delete_info1);
                                    //printf("delete SIZEI %i\n",delete1_size);
                                    if (delete0_size != 0)
                                    {
                                    for (num = 0; num < delete0_size; num++)
                                    {
                                    temp_record0 = (ReleasedRecord*) op_prg_mem_alloc (sizeof
(ReleasedRecord));
                                    temp_record0 = ((char *) op_prg_list_access (delete_info0,
OPC_LISTPOS_HEAD));
                                    list_size = op_prg_list_size (hold_rec0);
                                    for (n = 0; n < list_size; n++)
                                    {
                                    list_buff = op_prg_list_size (hold_rec0);
                                    diff = list_size - list_buff;
                                    position0 = n - diff;
                                    temps_hold0 = (FlowData*) op_prg_mem_alloc (sizeof
(FlowData));
                                    temps_hold0 = ((char *) op_prg_list_access (hold_rec0,
position0));
                                    if (temp_record0->QueueID == temps_hold0->QueueID)
                                    {op_prg_list_remove(delete_info0, OPC_LISTPOS_HEAD);
                                    op_prg_list_remove(hold_rec0, position0);
                                    }}}
                                    op_prg_list_free(delete_info0);
                                    }
                                    if (delete1_size != 0)
                                    {
                                    for (num = 0; num < delete1_size; num++)
                                    {
                                    temp_record1 = (ReleasedRecord*) op_prg_mem_alloc (sizeof
(ReleasedRecord));
                                    temp_record1 = ((char *) op_prg_list_access (delete_info1,
OPC_LISTPOS_HEAD));
                                    list_size = op_prg_list_size (hold_rec1);
                                    for (n = 0; n < list_size; n++)
```

```
                                        {
                                        list_buff = op_prg_list_size (hold_rec1);
                                        diff = list_size - list_buff;
                                        position1 = n - diff;
                                        temps_hold1 = (FlowData*) op_prg_mem_alloc (sizeof
(FlowData));
                                        temps_hold1 = ((char *) op_prg_list_access (hold_rec1,
position1));
                                        if (temp_record1->QueueID == temps_hold1->QueueID)
                                        {op_prg_list_remove(delete_info1, OPC_LISTPOS_HEAD);
                                        op_prg_list_remove(hold_rec1, position1);
                                        }}}
                                        op_prg_list_free(delete_info1);
                                        }
                                        queue_check0 = op_prg_list_size (hold_rec0);
                                        queue_check1 = op_prg_list_size (hold_rec1);
                                        if ((queue_check0 != 0) || (queue_check1 != 0))
                                        {frag = 1;
                                        op_intrpt_schedule_self (op_sim_time () + Released_Period,
RECALL_FLOW);
                                        }
                                        else if ((queue_check0 == 0) && (queue_check1 == 0))
                                        {frag = 0;
                                        stat_count = 0;
                                        }}}
                                /** state (RECALL) exit executives **/
                                FSM_STATE_EXIT_FORCED (5, "RECALL", "QoSpGtw_proc ()
[RECALL exit execs]")
                                        {
                                        }
                                /** state (RECALL) transition processing **/
                                FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "",
"RECALL", "dispatch")
                                /*---------------------------------------------------------*/
                                }
                        FSM_EXIT (0,QoSpGtw_proc)
                        }
        }
#if defined (__cplusplus)
        extern "C" {
#endif
        extern VosT_Fun_Status Vos_Catmem_Register (const char * , int , VosT_Void_Null_Proc,
VosT_Address *);
        extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
        extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
#if defined (__cplusplus)
        }
#endif
Compcode
QoSpGtw_proc_init (void ** gen_state_pptr)
        {
        int _block_origin = 0;
        static VosT_Address        obtype = OPC_NIL;
        FIN (QoSpGtw_proc_init (gen_state_pptr))
        if (obtype == OPC_NIL)
                {
                /* Initialize memory management */
                if (Vos_Catmem_Register ("proc state vars (QoSpGtw_proc)",
                        sizeof (QoSpGtw_proc_state), Vos_Vnop, &obtype) == VOSC_FAILURE)
```

```
                        {
                        FRET (OPC_COMPCODE_FAILURE)
                        }
                }
        *gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
        if (*gen_state_pptr == OPC_NIL)
                {
                FRET (OPC_COMPCODE_FAILURE)
                }
        else
                {
                /* Initialize FSM handling */
                ((QoSpGtw_proc_state *)(*gen_state_pptr))->current_block = 0;

                FRET (OPC_COMPCODE_SUCCESS)
                }
        }
void
QoSpGtw_proc_diag (void)
        {
        /* No Diagnostic Block */
        }
void
QoSpGtw_proc_terminate (void)
        {
        int _block_origin = __LINE__;
        FIN (QoSpGtw_proc_terminate (void))
        if (1)
                {
                int                     intrpt_code;
                int                     qos_type;
                struct TrunkDef         trunk_pnt;
                struct RouteDef         route_pnt;
                struct RouteDef *       tmp_route_pnt;
                FwdDef                  fwd_pnt;
                QoSRoute                q_route;
                QoSTrunk                q_trunk;
                Packet*                 pkptr;
                Packet*                 req_adm_pkptr;
                int                     msg_type;
                int                     req_type;
                int                     user_id;
                int                     dest_gtw;
                int                     dest_router;
                int                     bdw_req;
                int                     usd_bdw;
                int                     call_duration;
                int                     req_time;
                int                     proc_id;
                int                     route_id;
                int                     trunk_id;
                int                     vip_id;
                int                     out_index;
                Packet*                 trk_cfg_pkptr;
                struct RouteDef *       route_ptr;
                struct RouteDef *       need_reconfig_route_ptr;
                struct RouteDef *       widest_route_ptr;
                struct RouteDef *       narrowest_route_ptr;
                struct TrunkDef         tmp_trunk_ptr;
                int                     widest_route_id;
```

```
int                 narrowest_route_id;
int                 tmp_avl_bdw;
int                 route_avl_bdw;
int                 max_avl_bdw;
int                 min_avl_bdw;
struct VipDef *     vip_ptr;
struct VipDef *     tmp_vip_ptr;
double              e2edelay;
double              sendout_time;
int                 rkey;
int                 count;
int                 i;
int                 j;
Packet*             trk_rsc_pkptr;
int                 packet_sent;
FILE*               profile_hndl;
int                 outgoing;
int                 notfeasible;
int                 lowest_threshhold;
int                 plink_id;
int                 length;
int                 src_addr;
int                 dest_addr;
long                packet_loss_rate;
struct NMS_KEEP *   tmp_keep_table;
QoSRuntime          q_runtime;
int                 calltime;
int                 requesttime;
FILE*               profile;
char                name[128];
FILE*               hold_profile;
char                hold_name[128];
int                 time_req0;
int                 time_req1;
QoSRuntime*         runtime;
QoSRuntime*         tmp_ptr;
QoSRuntime*         temp_ptr;
char                msg0[128],msg1[128];
int                 time;
int                 list_size;
int                 n;
int                 m;
FlowData*           temp_hold0;
FlowData*           temp_hold1;
FlowData*           tmp_hold0;
FlowData*           tmp_hold1;
FlowData*           hold_reply;
int                 recall0_size;
int                 recall1_size;
int                 re_time;
Packet*             hold_admit_pkptr;
int                 decision;
int                 time_est;
int                 hold_size;
QoSRuntime*         del_ptr;
int                 del_size;
int                 del;
struct RouteDef *   flowrec_route_ptr;
struct RouteDef *   tmp_route_ptr;
int                 pass;
```

```
                int                     check_count;
                int                     Length_count;
                int                     Length_count1;
                int                     tmp_Length;
                int                     ii;
                int                     step;
                int                     x;
                ReleasedRecord*         temp_record0;
                ReleasedRecord*         temp_record1;
                ReleasedRecord*         temps_record0;
                ReleasedRecord*         temps_record1;
                int                     delete0_size;
                int                     delete1_size;
                int                     position0;
                int                     position1;
                int                     num;
                FlowData*               temps_hold0;
                FlowData*               temps_hold1;
                int                     queue_check0;
                int                     queue_check1;
                int                     list_buff;
                int                     diff;
                Packet*                 hold_ptr;
                TimeOut*                info_timeout0;
                TimeOut*                info_timeout1;
                int                     sent;
                /* No Termination Block */
                }
        Vos_Catmem_Dealloc (pr_state_ptr);
        FOUT;
        }
/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in QoSpGtw_proc_svar function. */
#undef ete_gsh
#undef own_id
#undef pkt_count
#undef vip_no
#undef route_table
#undef trunk_table
#undef forward_table
#undef parent_id
#undef rsv_route_id
#undef rsv_intf
#undef fwd_tbl_len
#undef overhead_hndl
#undef blocking_rate_hndl
#undef local_all_req_num
#undef local_overhead_hndl
#undef local_overhead_num
#undef local_admit_req_num
#undef node_name
#undef audio_e2edelay_hndl
#undef video_e2edelay_hndl
#undef Gtw_ID
#undef Router_ID
#undef trunk_rsvbdw_hndl
#undef trunk_usdbdw_hndl
#undef keep_table
```

```
#undef file_name
#undef buffer
#undef xx
#undef flowrec
#undef hold_queue_id
#undef hold_table
#undef hold_table1
#undef hold_queue_id1
#undef data_ptr
#undef edge_name
#undef tmp
#undef counter
#undef recall_table
#undef hold_rec0
#undef hold_rec1
#undef tmp1
#undef recall_overhead_hndl
#undef total_CAC_hndl
#undef delete_info0
#undef delete_info1
#undef frag
#undef LifeTime0
#undef LifeTime1
#undef queueC_id
#undef queueC_id1
#undef next_timeout0
#undef next_timeout1
#undef stat_count
#undef sent_code0
#undef sent_code1
void
QoSpGtw_proc_svar (void * gen_ptr, const char * var_name, char ** var_p_ptr)
	{
	QoSpGtw_proc_state		*prs_ptr;
	FIN (QoSpGtw_proc_svar (gen_ptr, var_name, var_p_ptr))
	if (var_name == OPC_NIL)
		{
		*var_p_ptr = (char *)OPC_NIL;
		FOUT;
		}
	prs_ptr = (QoSpGtw_proc_state *)gen_ptr;
	if (strcmp ("ete_gsh" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->ete_gsh);
		FOUT;
		}
	if (strcmp ("own_id" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->own_id);
		FOUT;
		}
	if (strcmp ("pkt_count" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->pkt_count);
		FOUT;
		}
	if (strcmp ("vip_no" , var_name) == 0)
		{
		*var_p_ptr = (char *) (&prs_ptr->vip_no);
```

```
        FOUT;
        }
if (strcmp ("route_table" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->route_table);
        FOUT;
        }
if (strcmp ("trunk_table" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->trunk_table);
        FOUT;
        }
if (strcmp ("forward_table" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->forward_table);
        FOUT;
        }
if (strcmp ("parent_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->parent_id);
        FOUT;
        }
if (strcmp ("rsv_route_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->rsv_route_id);
        FOUT;
        }
if (strcmp ("rsv_intf" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->rsv_intf);
        FOUT;
        }
if (strcmp ("fwd_tbl_len" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->fwd_tbl_len);
        FOUT;
        }
if (strcmp ("overhead_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->overhead_hndl);
        FOUT;
        }
if (strcmp ("blocking_rate_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->blocking_rate_hndl);
        FOUT;
        }
if (strcmp ("local_all_req_num" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->local_all_req_num);
        FOUT;
        }
if (strcmp ("local_overhead_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->local_overhead_hndl);
        FOUT;
        }
if (strcmp ("local_overhead_num" , var_name) == 0)
        {
```

```
            *var_p_ptr = (char *) (&prs_ptr->local_overhead_num);
            FOUT;
            }
if (strcmp ("local_admit_req_num" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->local_admit_req_num);
            FOUT;
            }
if (strcmp ("node_name" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->node_name);
            FOUT;
            }
if (strcmp ("audio_e2edelay_hndl" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->audio_e2edelay_hndl);
            FOUT;
            }
if (strcmp ("video_e2edelay_hndl" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->video_e2edelay_hndl);
            FOUT;
            }
if (strcmp ("Gtw_ID" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->Gtw_ID);
            FOUT;
            }
if (strcmp ("Router_ID" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->Router_ID);
            FOUT;
            }
if (strcmp ("trunk_rsvbdw_hndl" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->trunk_rsvbdw_hndl);
            FOUT;
            }
if (strcmp ("trunk_usdbdw_hndl" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->trunk_usdbdw_hndl);
            FOUT;
            }
if (strcmp ("keep_table" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->keep_table);
            FOUT;
            }
if (strcmp ("file_name" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->file_name);
            FOUT;
            }
if (strcmp ("buffer" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->buffer);
            FOUT;
            }
if (strcmp ("xx" , var_name) == 0)
```

```
        {
        *var_p_ptr = (char *) (&prs_ptr->xx);
        FOUT;
        }
if (strcmp ("flowrec" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->flowrec);
        FOUT;
        }
if (strcmp ("hold_queue_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->hold_queue_id);
        FOUT;
        }
if (strcmp ("hold_table" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->hold_table);
        FOUT;
        }
if (strcmp ("hold_table1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->hold_table1);
        FOUT;
        }
if (strcmp ("hold_queue_id1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->hold_queue_id1);
        FOUT;
        }
if (strcmp ("data_ptr" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->data_ptr);
        FOUT;
        }
if (strcmp ("edge_name" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->edge_name);
        FOUT;
        }
if (strcmp ("tmp" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->tmp);
        FOUT;
        }
if (strcmp ("counter" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->counter);
        FOUT;
        }
if (strcmp ("recall_table" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->recall_table);
        FOUT;
        }
if (strcmp ("hold_rec0" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->hold_rec0);
        FOUT;
        }
```

```
if (strcmp ("hold_rec1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->hold_rec1);
        FOUT;
        }
if (strcmp ("tmp1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->tmp1);
        FOUT;
        }
if (strcmp ("recall_overhead_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->recall_overhead_hndl);
        FOUT;
        }
if (strcmp ("total_CAC_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->total_CAC_hndl);
        FOUT;
        }
if (strcmp ("delete_info0" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->delete_info0);
        FOUT;
        }
if (strcmp ("delete_info1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->delete_info1);
        FOUT;
        }
if (strcmp ("frag" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->frag);
        FOUT;
        }
if (strcmp ("LifeTime0" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->LifeTime0);
        FOUT;
        }
if (strcmp ("LifeTime1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->LifeTime1);
        FOUT;
        }
if (strcmp ("queueC_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->queueC_id);
        FOUT;
        }
if (strcmp ("queueC_id1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->queueC_id1);
        FOUT;
        }
if (strcmp ("next_timeout0" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->next_timeout0);
        FOUT;
```

```
        }
if (strcmp ("next_timeout1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->next_timeout1);
        FOUT;
        }
if (strcmp ("stat_count" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->stat_count);
        FOUT;
        }
if (strcmp ("sent_code0" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->sent_code0);
        FOUT;
        }
if (strcmp ("sent_code1" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->sent_code1);
        FOUT;
        }
*var_p_ptr = (char *)OPC_NIL;

FOUT;
}
```

```
/**************************************************************************/
/*CORE ROUTER NODE*/
/**************************************************************************/
    /*proc module*/
/**************************************************************************/
/* This variable carries the header into the object file */
static const char QoSp_router_proc_pr_c [] = "MIL_3_Tfile_Hdr_ 80C 30A op_runsim 7 441CF08A
441CF08A 1 KiT KiT 0 0 none none 0 0 none 0 0 0 0 0 0 ";
#include <string.h>
/* OPNET system definitions */
#include <opnet.h>
#if defined (__cplusplus)
extern "C" {
#endif
FSM_EXT_DECS
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Header Block */
/* --- include files and type definition of my own ---*/
#include "..\headfile\McrDef.h"
#include "..\headfile\McrDS.h"
#include "..\headfile\PolicySvr_router.h"
/* --- transition macro --- */
#define         PKT_ARRVL     ( op_intrpt_type () == OPC_INTRPT_STRM )
void            LookUpFwd(int, int);
extern    overhead_msg_num;
/* End of Header Block */
#if !defined (VOSD_NO_FIN)
#undef   BIN
#undef   BOUT
#define  BIN            FIN_LOCAL_FIELD(last_line_passed) = __LINE__ - _block_origin;
#define  BOUT   BIN
#define  BINIT   FIN_LOCAL_FIELD(last_line_passed) = 0; _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 */
        RouteItem                       Route_tbl [MAX_ROUTE_NUM*2];
        VipItem*                        Vip_tbl_head;
        VipItem*                        Vip_tbl_tail;
        int                             Route_tbl_len;
        int                             out_going;
        Objid                           own_id;
        Objid                           parent_id;
        Stathandle                      overhead_hndl;
        Stathandle                      bdw_eff_hndl [MAX_ROUTER_LINK];
        int                             used_bdw[MAX_ROUTER_LINK];
        int                             link_cap[MAX_ROUTER_LINK];
        char                            node_name [15];
        int                             sequence;
        int                             Router_ID;
        Stathandle                      blocking_rate_hndl;
        } QoSp_router_proc_state;
```

```
#define pr_state_ptr                    ((QoSp_router_proc_state*) SimI_Mod_State_Ptr)
#define Route_tbl                       pr_state_ptr->Route_tbl
#define Vip_tbl_head                    pr_state_ptr->Vip_tbl_head
#define Vip_tbl_tail                    pr_state_ptr->Vip_tbl_tail
#define Route_tbl_len                   pr_state_ptr->Route_tbl_len
#define out_going                       pr_state_ptr->out_going
#define own_id                          pr_state_ptr->own_id
#define parent_id                       pr_state_ptr->parent_id
#define overhead_hndl                   pr_state_ptr->overhead_hndl
#define bdw_eff_hndl                    pr_state_ptr->bdw_eff_hndl
#define used_bdw                        pr_state_ptr->used_bdw
#define link_cap                        pr_state_ptr->link_cap
#define node_name                       pr_state_ptr->node_name
#define sequence                        pr_state_ptr->sequence
#define Router_ID                       pr_state_ptr->Router_ID
#define blocking_rate_hndl              pr_state_ptr->blocking_rate_hndl
/* This macro definition 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
#define FIN_PREAMBLE QoSp_router_proc_state *op_sv_ptr = pr_state_ptr;
/* Function Block */
enum { _block_origin = __LINE__ };
void    LookUpFwd (int id, int dest)
        {
        int                             i;
        Boolean         route_found;
/* This function looks up the forwarding table to find the outgoing interface for the packet */
        FIN (LookUpFwd (int, int));
        i = 0;
        route_found = FALSE;
        while ( (route_found == FALSE) && (i<Route_tbl_len) ) {
        if  (Route_tbl[i].Dest == dest)
        route_found = TRUE;
                else i++;
        }
        if ( route_found==FALSE )
                out_going = -1;
        else out_going = Route_tbl[i].Outgoing_intf;
        FOUT;
        }
int GetRouterIdFrmName ( char name[15] )
{
        char *          sptr;
        int             digit;
        /* --- the format of node name is node_11 ---*/
        /* --- we need to get 11 out of the name attribute --- */
        /* --- use the number as the id of the gateway  --- */
        sptr = strchr(name, '_');
        sptr ++;
        digit = atoi(sptr);
        return digit;
        }
/* 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 QoSp_router_proc (void);
	Compcode QoSp_router_proc_init (void **);
	void QoSp_router_proc_diag (void);
	void QoSp_router_proc_terminate (void);
	void QoSp_router_proc_svar (void *, const char *, char **);
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Process model interrupt handling procedure */
void
QoSp_router_proc (void)
	{
	int _block_origin = 0;
	FIN (QoSp_router_proc ());
	if (1)
		{
		char			file_name[128];
		FILE *			profile_hndl;
		RouteItem		fwd_rec;
		int			msg_type;
		Packet*			pkptr;
		Packet*			pkptr0;
		Packet*			pkptr1;
		Packet*			pkptr2;
		Packet*			pkptr3;
		Packet*			pkptr4;
		Packet*			pkptr5;
		Packet*			pkptr6;
		Packet*			pkptr7;
		Packet*			pkptr8;
		Packet*			pkptr9;
		Packet*			pkptr11;
		Packet*			pkptr12;
		Packet*			pkptr13;
		Packet*			pkptr14;
		/* --- variables used by VIP record --- */
		int			src_gtw;
		int			dest_gtw;
		int			dest_router;
		int			src_router;
		int			route_id;
		int			bdw_used;
		int			vip_id;
		VipItem *		Vip_tmp_pnt;
		VipItem *		Vip_tmp_pre;
		Boolean			Vip_found;
		int			count;
		int			i;
		int			rkey;
		float			tmp;
		int			seq_tmp;
		FSM_ENTER (QoSp_router_proc)
		FSM_BLOCK_SWITCH
```

```
{
/*---------------------------------------------------------*/
/** state (init) enter executives **/
FSM_STATE_ENTER_FORCED_NOLABEL (0, "init",
"QoSp_router_proc () [init enter execs]")
{
printf("I'm QoSP_router init\n");
own_id = op_id_self ();
parent_id = op_topo_parent (own_id);
op_ima_obj_attr_get (parent_id, "name", node_name);
Router_ID=GetRouterIdFrmName(node_name);
overhead_hndl = op_stat_reg ("Overhead",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
/* --- read in the forwarding table --- */
strcpy (file_name, FORWARD_FILE);
strcat (file_name, "_");
strcat (file_name, node_name);
count = 0;
if( (profile_hndl = fopen( file_name, "r" )) != NULL )
{
i = fread ((char *)&fwd_rec, sizeof( RouteItem ), 1, profile_hndl);
while ( !feof (profile_hndl) ) {
/* -- fill in the trunk table --- */
memcpy ((char *)&Route_tbl[count], (char *)&fwd_rec, sizeof
( RouteItem ));
count ++;
i = fread ((char *)&fwd_rec, sizeof( RouteItem ), 1, profile_hndl);
}
fclose(profile_hndl);
}
Route_tbl_len = count;
/* --- initialize the vip recording table --- */
Vip_tbl_head = NULL;
Vip_tbl_tail = NULL;
sequence=0;
}
/** state (init) exit executives **/
FSM_STATE_EXIT_FORCED (0, "init", "QoSp_router_proc () [init exit
execs]")
{
}
/** state (init) transition processing **/
FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "init",
"proc")
/*---------------------------------------------------------*/
/** state (proc) enter executives **/
FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "proc",
"QoSp_router_proc () [proc enter execs]")
{
}
/** blocking after enter executives of unforced state. **/
FSM_EXIT (3,QoSp_router_proc)
/** state (proc) exit executives **/
FSM_STATE_EXIT_UNFORCED (1, "proc", "QoSp_router_proc () [proc
exit execs]")
{
pkptr = op_pk_get (op_intrpt_strm ());
op_pk_nfd_get (pkptr, "Msg_type", &msg_type);
switch ( msg_type ) {
```

```
case TRK_CFG_REQ :
case TRK_CFG_CNF :
case TRK_RSC_RLS :
/* calc the overhead metric   */
overhead_msg_num=overhead_msg_num+1;
op_stat_write(overhead_hndl,overhead_msg_num);
 /* --- get route_id, destination from the packet --- */
op_pk_nfd_get (pkptr, "Route_id", &route_id);
op_pk_nfd_get (pkptr, "Dest_router", &dest_router);
if (dest_router==Router_ID)
  {op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
   op_pk_send (pkptr, dest_gtw);
}
else
/* --- look up the forwarding table to send out the packet --- */
            {LookUpFwd (route_id, dest_router);
                        if (out_going == -1)
                        printf("error route_id and destination\n");
                        else {//  if ((dest_router==6)&&(src_router==2))
                        op_pk_send (pkptr, out_going);
                        }
                        }
              break;
case VIP_EST :
/* --- keep record of the vip for bandwidth efficiency --- */
            op_pk_nfd_get (pkptr, "Dest_router", &dest_router);
            op_pk_nfd_get (pkptr, "Route_id", &route_id);
/*- look up the forwarding table to send out the packet again -*/
if (dest_router==Router_ID)
{op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
op_pk_send (pkptr, dest_gtw);
}
else
/* --- look up the forwarding table to send out the packet --- */
{LookUpFwd (route_id, dest_router);
if (out_going == -1)
printf("error route_id and destination\n");
else op_pk_send (pkptr, out_going);
}
break;
case VIP_RLS :
op_pk_nfd_get (pkptr, "Route_id", &route_id);
op_pk_nfd_get (pkptr, "Dest_router", &dest_router);
if (dest_router==Router_ID)
{op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
op_pk_send (pkptr, dest_gtw);
}
else
/* --- look up the forwarding table to send out the packet --- */
{LookUpFwd (route_id, dest_router);
if (out_going == -1)
 printf("error route_id and destination\n");
else op_pk_send (pkptr, out_going);
}
break;
}
 //scanf("%i",&rkey);
}
/** state (proc) transition processing **/
```

```
                        FSM_TRANSIT_ONLY ((PKT_ARRVL), 1, state1_enter_exec, ;, "proc",
"PKT_ARRVL", "", "proc", "proc")
                        /*---------------------------------------------------------*/
                        }
                FSM_EXIT (0,QoSp_router_proc)
                }
        }
#if defined (__cplusplus)
        extern "C" {
#endif
        extern VosT_Fun_Status Vos_Catmem_Register (const char * , int , VosT_Void_Null_Proc,
VosT_Address *);
        extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
        extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
#if defined (__cplusplus)
        }
#endif
Compcode
QoSp_router_proc_init (void ** gen_state_pptr)
        {
        int _block_origin = 0;
        static VosT_Address         obtype = OPC_NIL;
        FIN (QoSp_router_proc_init (gen_state_pptr))
        if (obtype == OPC_NIL)
                {
                /* Initialize memory management */
                if (Vos_Catmem_Register ("proc state vars (QoSp_router_proc)",
                        sizeof (QoSp_router_proc_state), Vos_Vnop, &obtype) ==
VOSC_FAILURE)
                        {
                        FRET (OPC_COMPCODE_FAILURE)
                        }
                }
        *gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
        if (*gen_state_pptr == OPC_NIL)
                {
                FRET (OPC_COMPCODE_FAILURE)
                }
        else
                {
                /* Initialize FSM handling */
                ((QoSp_router_proc_state *)(*gen_state_pptr))->current_block = 0;
                FRET (OPC_COMPCODE_SUCCESS)
                }
        }
void
QoSp_router_proc_diag (void)
        {
        /* No Diagnostic Block */
        }
void
QoSp_router_proc_terminate (void)
        {
        int _block_origin = __LINE__;
        FIN (QoSp_router_proc_terminate (void))
        if (1)
                {
                char                    file_name[128];
                FILE *                  profile_hndl;
                RouteItem               fwd_rec;
```

```
                int                     msg_type;
                Packet*                 pkptr;
                Packet*                 pkptr0;
                Packet*                 pkptr1;
                Packet*                 pkptr2;
                Packet*                 pkptr3;
                Packet*                 pkptr4;
                Packet*                 pkptr5;
                Packet*                 pkptr6;
                Packet*                 pkptr7;
                Packet*                 pkptr8;
                Packet*                 pkptr9;
                Packet*                 pkptr11;
                Packet*                 pkptr12;
                Packet*                 pkptr13;
                Packet*                 pkptr14;
                /* --- variables used by VIP record --- */
                int                     src_gtw;
                int                     dest_gtw;
                int                     dest_router;
                int                     src_router;
                int                     route_id;
                int                     bdw_used;
                int                     vip_id;
                VipItem *               Vip_tmp_pnt;
                VipItem *               Vip_tmp_pre;
                Boolean                 Vip_found;
                int                     count;
                int                     i;
                int                     rkey;
                float                   tmp;
                int                     seq_tmp;
                /* No Termination Block */
                }
        Vos_Catmem_Dealloc (pr_state_ptr);
        FOUT;
        }
/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in QoSp_router_proc_svar function. */
#undef Route_tbl
#undef Vip_tbl_head
#undef Vip_tbl_tail
#undef Route_tbl_len
#undef out_going
#undef own_id
#undef parent_id
#undef overhead_hndl
#undef bdw_eff_hndl
#undef used_bdw
#undef link_cap
#undef node_name
#undef sequence
#undef Router_ID
#undef blocking_rate_hndl

void
QoSp_router_proc_svar (void * gen_ptr, const char * var_name, char ** var_p_ptr)
        {
```

```
QoSp_router_proc_state		*prs_ptr;
FIN (QoSp_router_proc_svar (gen_ptr, var_name, var_p_ptr))
if (var_name == OPC_NIL)
	{
	*var_p_ptr = (char *)OPC_NIL;
	FOUT;
	}
prs_ptr = (QoSp_router_proc_state *)gen_ptr;
if (strcmp ("Route_tbl" , var_name) == 0)
	{
	*var_p_ptr = (char *) (prs_ptr->Route_tbl);
	FOUT;
	}
if (strcmp ("Vip_tbl_head" , var_name) == 0)
	{
	*var_p_ptr = (char *) (&prs_ptr->Vip_tbl_head);
	FOUT;
	}
if (strcmp ("Vip_tbl_tail" , var_name) == 0)
	{
	*var_p_ptr = (char *) (&prs_ptr->Vip_tbl_tail);
	FOUT;
	}
if (strcmp ("Route_tbl_len" , var_name) == 0)
	{
	*var_p_ptr = (char *) (&prs_ptr->Route_tbl_len);
	FOUT;
	}
if (strcmp ("out_going" , var_name) == 0)
	{
	*var_p_ptr = (char *) (&prs_ptr->out_going);
	FOUT;
	}
if (strcmp ("own_id" , var_name) == 0)
	{
	*var_p_ptr = (char *) (&prs_ptr->own_id);
	FOUT;
	}
if (strcmp ("parent_id" , var_name) == 0)
	{
	*var_p_ptr = (char *) (&prs_ptr->parent_id);
	FOUT;
	}
if (strcmp ("overhead_hndl" , var_name) == 0)
	{
	*var_p_ptr = (char *) (&prs_ptr->overhead_hndl);
	FOUT;
	}
if (strcmp ("bdw_eff_hndl" , var_name) == 0)
	{
	*var_p_ptr = (char *) (prs_ptr->bdw_eff_hndl);
	FOUT;
	}
if (strcmp ("used_bdw" , var_name) == 0)
	{
	*var_p_ptr = (char *) (prs_ptr->used_bdw);
	FOUT;
	}
if (strcmp ("link_cap" , var_name) == 0)
```

```
        {
        *var_p_ptr = (char *) (prs_ptr->link_cap);
        FOUT;
        }
if (strcmp ("node_name" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->node_name);
        FOUT;
        }
if (strcmp ("sequence" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->sequence);
        FOUT;
        }
if (strcmp ("Router_ID" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->Router_ID);
        FOUT;
        }
if (strcmp ("blocking_rate_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->blocking_rate_hndl);
        FOUT;
        }
*var_p_ptr = (char *)OPC_NIL;

FOUT;
}
```

```
/*****************************************************************************/
/*SERVER NODE*/
/*****************************************************************************/
    /*proc module*/
/*****************************************************************************/
/* This variable carries the header into the object file */
static const char QoSpPS_pr_c [] = "MIL_3_Tfile_Hdr_ 80C 30A op_runsim 7 441CF08B
441CF08B 1 KiT KiT 0 0 none none 0 0 none 0 0 0 0 0 0";
#include <string.h>
/* OPNET system definitions */
#include <opnet.h>
#if defined (__cplusplus)
extern "C" {
#endif
FSM_EXT_DECS
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Header Block */
/* --- include files and type definition of my own ---*/
#include "..\headfile\McrDef.h"
#include "..\headfile\McrDS.h"
#include "..\headfile\PolicySvr_server.h"
#include "..\headfile\PolicySvr_Runtime.h"
#define PKT_ARRVL (op_intrpt_type () == OPC_INTRPT_STRM)
#define XMT_OUT_TO_ROUTER            0
/* End of Header Block */
#if !defined (VOSD_NO_FIN)
#undef   BIN
#undef   BOUT
#define  BIN      FIN_LOCAL_FIELD(last_line_passed) = __LINE__ - _block_origin;
#define  BOUT   BIN
#define  BINIT   FIN_LOCAL_FIELD(last_line_passed) = 0; _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 */
        QoSRoute                        route_tbl [MAX_ROUTE_NUM];
        TrunkItem                       trunk_tbl [MAX_TRUNK_NUM];
        LinkItem                        link_tbl [MAX_LINK_NUM];
        PLinkItem                       p_link_tbl [MAX_PLINK_NUM];
        int                             fwd_tbl_len;
        Objid                           own_id;
        Objid                           parent_id;
        QoSFwd                          forward_table [MAX_ROUTE_NUM*2];
        Stathandle                      overhead_hndl;
        int                             out_going;
        int                             sequence;
        Stathandle                      reconfig_num_hndl;
        int                             reconfig_number;
        Stathandle                      utilization_hndl;
        double                          Sum_RsvBdw;
        Stathandle                      utilization_Link_hndl[19];
        double                          sum_link;
```

```
	Stathandle			utilization_PLink_hndl[39];
	} QoSpPS_state;

#define pr_state_ptr			((QoSpPS_state*) SimI_Mod_State_Ptr)
#define route_tbl			pr_state_ptr->route_tbl
#define trunk_tbl			pr_state_ptr->trunk_tbl
#define link_tbl			pr_state_ptr->link_tbl
#define p_link_tbl			pr_state_ptr->p_link_tbl
#define fwd_tbl_len			pr_state_ptr->fwd_tbl_len
#define own_id				pr_state_ptr->own_id
#define parent_id			pr_state_ptr->parent_id
#define forward_table			pr_state_ptr->forward_table
#define overhead_hndl			pr_state_ptr->overhead_hndl
#define out_going			pr_state_ptr->out_going
#define sequence			pr_state_ptr->sequence
#define reconfig_num_hndl		pr_state_ptr->reconfig_num_hndl
#define reconfig_number			pr_state_ptr->reconfig_number
#define utilization_hndl		pr_state_ptr->utilization_hndl
#define Sum_RsvBdw			pr_state_ptr->Sum_RsvBdw
#define utilization_Link_hndl		pr_state_ptr->utilization_Link_hndl
#define sum_link			pr_state_ptr->sum_link
#define utilization_PLink_hndl		pr_state_ptr->utilization_PLink_hndl
/* This macro definition 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
#define FIN_PREAMBLE QoSpPS_state *op_sv_ptr = pr_state_ptr;
/* No Function Block */
enum { _block_origin = __LINE__ };
/* 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 QoSpPS (void);
	Compcode QoSpPS_init (void **);
	void QoSpPS_diag (void);
	void QoSpPS_terminate (void);
	void QoSpPS_svar (void *, const char *, char **);
#if defined (__cplusplus)
} /* end of 'extern "C"' */
#endif
/* Process model interrupt handling procedure */
void
QoSpPS (void)
	{
	int _block_origin = 0;
	FIN (QoSpPS ());
	if (1)
		{
		Packet*				pkptr;
		Packet*				plink_busy_pkptr;
		char				file_name[128];
```

```
		char		node_name[15];
		int		msg_type;
		int		src_gtw;
		int		src_router;
		int		dest_gtw;
		int		dest_router;
		int		bdw_req;
		int		length;
		int		route_id;
		int		trunk_id;
		int		p_link_id;
		int		link_id;
		int		capacity;
		int		rsv_bdw;
		Packet*		cfg_cnf_pkptr;
		int		i;
		int		rkey;
		int		count;
		FILE*		f_hndl;
		QoSRoute	route_rec;
		TrunkItem	trunk_rec;
		PLinkItem	p_link_rec;
		LinkItem	link_rec;
		QoSFwd		fwd_pnt;
		int		k;
		double		t;
		int		j;
		int		notfeasible;
		int		number;
		double		Sum_UsdBdw;
		double		value_u;
		int		check;
		int		xx;
		int		c;
		FSM_ENTER (QoSpPS)
		FSM_BLOCK_SWITCH
			{
			/*-------------------------------------------------------*/
			/** state (init) enter executives **/
			FSM_STATE_ENTER_FORCED_NOLABEL (0, "init", "QoSpPS () [init
enter execs]")
				{
				printf("I'm QoSpPs_proc init\n");
				//scanf("%i", &rkey);
				own_id = op_id_self();
				parent_id = op_topo_parent (own_id);
				op_ima_obj_attr_get (parent_id, "name", node_name);
				overhead_hndl =op_stat_reg ("Overhead",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
				reconfig_num_hndl =op_stat_reg ("Reconfiguration Number",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
				reconfig_number=0;
				utilization_hndl=op_stat_reg ("Utilization %",
OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL );
				/* --- read in the QoS routes table --- */
				/* --- read in the provisioned link table --- */
				if( (f_hndl = fopen( PROVISIONED_LINK_FILE, "r" )) !=
NULL )
				{
				i = fread ((char *)&p_link_rec, sizeof( PLinkItem ), 1, f_hndl);
```

```
                                    while ( !feof (f_hndl) ) {
                                    memcpy (&(p_link_tbl[p_link_rec.PLinkID]), &p_link_rec, sizeof
(PLinkItem));
                                    if(p_link_rec.PLinkID<=19)
                                    p_link_tbl[p_link_rec.PLinkID].UsdBdw=450000;
                                    else p_link_tbl[p_link_rec.PLinkID].UsdBdw=9600;
                                    //printf("plink: %i Rsvbdw %i usdbdw %i link %i busy :%i",
p_link_rec.PLinkID,p_link_tbl[p_link_rec.PLinkID].RsvBdw,p_link_tbl[p_link_rec.PLinkID].UsdBd
w,p_link_tbl[p_link_rec.PLinkID].LinkID,p_link_tbl[p_link_rec.PLinkID].Busy);
                                    i = fread ((char *)&p_link_rec, sizeof( PLinkItem ), 1, f_hndl);
                                    }
                                    fclose(f_hndl);
                                    }
                                    else printf("can not open provisioned link file\n");
                                    sequence=0;       //broadcast msg series num initial
                                    Sum_UsdBdw = 0;
                                    Sum_RsvBdw = 0;
                                    check = 0;
                                    xx=1;
                                    for (xx=1;xx<40;xx++)
                                    {check++;
                                    Sum_UsdBdw = Sum_UsdBdw + p_link_tbl[xx].UsdBdw;
                                    Sum_RsvBdw = Sum_RsvBdw + p_link_tbl[xx].RsvBdw;
                                    }
                                    value_u = (Sum_UsdBdw*100 / Sum_RsvBdw);
                                    op_stat_write(utilization_hndl,(double)value_u);
                          /*----------------------------------KiT----------------------------------*/
                                    utilization_Link_hndl[1]=op_stat_reg ("Link 1 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[2]=op_stat_reg ("Link 2 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[3]=op_stat_reg ("Link 3 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[4]=op_stat_reg ("Link 4 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[5]=op_stat_reg ("Link 5 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[6]=op_stat_reg ("Link 6 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[7]=op_stat_reg ("Link 7 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[8]=op_stat_reg ("Link 8 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[9]=op_stat_reg ("Link 9 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[10]=op_stat_reg ("Link 10 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[11]=op_stat_reg ("Link 11 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[12]=op_stat_reg ("Link 12 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[13]=op_stat_reg ("Link 13 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[14]=op_stat_reg ("Link 14 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[15]=op_stat_reg ("Link 15 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[16]=op_stat_reg ("Link 16 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
```

```
                                    utilization_Link_hndl[17]=op_stat_reg ("Link 17 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[18]=op_stat_reg ("Link 18 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_Link_hndl[19]=op_stat_reg ("Link 19 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    sum_link = Sum_RsvBdw/19;
                                    utilization_PLink_hndl[1]=op_stat_reg ("ProLink 1 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[2]=op_stat_reg ("ProLink 2 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[3]=op_stat_reg ("ProLink 3 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[4]=op_stat_reg ("ProLink 4 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[5]=op_stat_reg ("ProLink 5 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[6]=op_stat_reg ("ProLink 6 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[7]=op_stat_reg ("ProLink 7 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[8]=op_stat_reg ("ProLink 8 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[9]=op_stat_reg ("ProLink 9 Utilization",
OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[10]=op_stat_reg ("ProLink 10
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[11]=op_stat_reg ("ProLink 11
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[12]=op_stat_reg ("ProLink 12
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[13]=op_stat_reg ("ProLink 13
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[14]=op_stat_reg ("ProLink 14
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[15]=op_stat_reg ("ProLink 15
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[16]=op_stat_reg ("ProLink 16
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[17]=op_stat_reg ("ProLink 17
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[18]=op_stat_reg ("ProLink 18
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[19]=op_stat_reg ("ProLink 19
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[20]=op_stat_reg ("ProLink 20
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[21]=op_stat_reg ("ProLink 21
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[22]=op_stat_reg ("ProLink 22
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[23]=op_stat_reg ("ProLink 23
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[24]=op_stat_reg ("ProLink 24
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[25]=op_stat_reg ("ProLink 25
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
```

```
                                    utilization_PLink_hndl[26]=op_stat_reg ("ProLink 26
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[27]=op_stat_reg ("ProLink 27
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[28]=op_stat_reg ("ProLink 28
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[29]=op_stat_reg ("ProLink 29
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[30]=op_stat_reg ("ProLink 30
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[31]=op_stat_reg ("ProLink 31
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[32]=op_stat_reg ("ProLink 32
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[33]=op_stat_reg ("ProLink 33
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[34]=op_stat_reg ("ProLink 34
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[35]=op_stat_reg ("ProLink 35
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[36]=op_stat_reg ("ProLink 36
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[37]=op_stat_reg ("ProLink 37
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    utilization_PLink_hndl[38]=op_stat_reg ("ProLink 38
Utilization", OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL );
                                    /*---------------------LOCAL----------------------------------------*/
                                    c=1;
                                    for (c=1;c<20;c++)
                                    {
        op_stat_write(utilization_Link_hndl[c],(double)((p_link_tbl[c].UsdBdw +
p_link_tbl[19+c].UsdBdw)/sum_link));
        op_stat_write(utilization_PLink_hndl[c],(double)(p_link_tbl[c].UsdBdw/(double)2420000));
        op_stat_write(utilization_PLink_hndl[c+19],(double)(p_link_tbl[c+19].UsdBdw/(double)640
00));
                                    }
                                    /*------------------------------KiT-------------------------*/
                                    //printf("link %f\n",sum_link);
                                    //printf("PS init over\n");
                                    //scanf("%i", &rkey);
                                    /* --- read in the forwarding table --- */
                                    /*strcpy (file_name, FORWARD_FILE);
                                    strcat (file_name, "_");
                                    strcat (file_name, node_name);
                                    count = 0;
                                    if( (f_hndl = fopen( file_name, "r" )) != NULL )
                                    {
                                    i = fread ((char *)&fwd_pnt, sizeof( QoSFwd ), 1, f_hndl);
                                    while ( !feof (f_hndl) ) {
                                    printf("open file %s succeed\n", file_name);
                                    memcpy ((char *)&forward_table[count], (char *)&fwd_pnt, sizeof
( QoSFwd ));
                                    count ++;
                                    printf("RouteID %i\n", fwd_pnt.Route_id);
                                    printf("Dest %i\n", fwd_pnt.Dest);
                                    printf("Out interface %i\n", fwd_pnt.Outgoing_intf);
                                    //scanf("%i", &rkey);
                                    i = fread ((char *)&fwd_pnt, sizeof( QoSFwd ), 1, f_hndl);
```

```
                                        }
                                        fclose(f_hndl);
                                        }
                                        fwd_tbl_len = count;*/
                                        }
                                /** state (init) exit executives **/
                                FSM_STATE_EXIT_FORCED (0, "init", "QoSpPS () [init exit execs]")
                                        {
                                        }
                                /** state (init) transition processing **/
                                FSM_TRANSIT_FORCE (1, state1_enter_exec, ;, "default", "", "init",
"proc")
                                        /*---------------------------------------------------------*/
                                /** state (proc) enter executives **/
                                FSM_STATE_ENTER_UNFORCED (1, state1_enter_exec, "proc",
"QoSpPS () [proc enter execs]")
                                        {
                                        }
                                /** blocking after enter executives of unforced state. **/
                                FSM_EXIT (3,QoSpPS)
                                /** state (proc) exit executives **/
                                FSM_STATE_EXIT_UNFORCED (1, "proc", "QoSpPS () [proc exit
execs]")
                                        {
                                        /* --- get the packet from the input stream --- */
                                        //printf("I'm QoSpPs_proc, processing the packet received\n");
                                        pkptr = op_pk_get (op_intrpt_strm ());
                                        reconfig_number++;
                                        op_stat_write(reconfig_num_hndl,reconfig_number);
                                        op_pk_nfd_get (pkptr, "Msg_type", &msg_type);
                                        //printf("the packet type is %i\n", msg_type);
                                        switch ( msg_type ) {
                                        case TRK_CFG_REQ:
                                                op_pk_nfd_get (pkptr, "Src_router", &src_router);
                                                op_pk_nfd_get (pkptr, "Src_gtw", &src_gtw);
                                                overhead_msg_num=overhead_msg_num+1;
                                                op_stat_write(overhead_hndl,overhead_msg_num);
        /* --- get the source gateway ID, # of expanded trunks and bandwidth requirement --- */
                                                op_pk_nfd_get (pkptr, "Src_router", &src_router);
                                                op_pk_nfd_get (pkptr, "Src_gtw", &src_gtw);
                                                op_pk_nfd_get (pkptr, "Dest_router", &dest_router);
                                                op_pk_nfd_get (pkptr, "Dest_gtw", &dest_gtw);
                                                op_pk_nfd_get (pkptr, "Bdw_req", &bdw_req);
                                                op_pk_nfd_get (pkptr, "Length", &length);
                                                op_pk_nfd_get (pkptr, "Route_id", &route_id);
                                           /* --- generate the trk_cfg_cnf packet --- */
                                                cfg_cnf_pkptr =
op_pk_create_fmt(QOS_TRK_CFG_CNF);
                                                op_pk_nfd_set (cfg_cnf_pkptr, "Msg_type",
TRK_CFG_CNF);
                                                op_pk_nfd_set (cfg_cnf_pkptr, "Src_router", dest_router);
                                                op_pk_nfd_set (cfg_cnf_pkptr, "Src_gtw", dest_gtw);
                                                op_pk_nfd_set (cfg_cnf_pkptr, "Dest_router", src_router);
                                                op_pk_nfd_set (cfg_cnf_pkptr, "Dest_gtw", src_gtw);
                                                op_pk_nfd_set (cfg_cnf_pkptr, "Route_id", route_id);
                                                notfeasible=0;
                                                count = 0;
/* --- for each trunk ID, look up the provisioned link table to see whether there is enough spare
resource --- */
```

```
                                        for (i=0; i<length; i++)
                                                {
                                                op_pk_fd_get (pkptr, i+5, &trunk_id);
                                                p_link_id = trunk_id;
                                                if ( p_link_tbl[p_link_id].RsvBdw >=
(p_link_tbl[p_link_id].UsdBdw + bdw_req) )
                                                {
                                                op_pk_fd_set (cfg_cnf_pkptr, count+4,
OPC_FIELD_TYPE_INTEGER, trunk_id, 8);
                                                op_pk_fd_set (cfg_cnf_pkptr, count+5,
OPC_FIELD_TYPE_INTEGER, bdw_req, 16);
          p_link_tbl[p_link_id].UsdBdw=p_link_tbl[p_link_id].UsdBdw + bdw_req;
                                                 count=count+2;
                                                 }
                                                else {
                                                if ( p_link_tbl[p_link_id].RsvBdw >=
(p_link_tbl[p_link_id].UsdBdw + (int)bdw_req/INCREASING_TIMES) )
                                                {
                                                op_pk_fd_set (cfg_cnf_pkptr, count+4,
OPC_FIELD_TYPE_INTEGER, trunk_id, 8);
                                                op_pk_fd_set (cfg_cnf_pkptr, count+5,
OPC_FIELD_TYPE_INTEGER,(int)bdw_req/INCREASING_TIMES , 16);
                                        p_link_tbl[p_link_id].UsdBdw=p_link_tbl[p_link_id].UsdBdw +
(int)bdw_req/INCREASING_TIMES;
                                                count=count+2;
                                                }
                                                else  { printf("plink %i rsv %i
used %i\n",p_link_id,p_link_tbl[p_link_id].RsvBdw,p_link_tbl[p_link_id].UsdBdw);
                                                notfeasible=1;
                                                }
                                                }}
                                                op_pk_nfd_set (cfg_cnf_pkptr, "Length",
(int)count/2);
                                                if(notfeasible==1)
                                                  {
                                                op_pk_destroy(cfg_cnf_pkptr);
                                                plink_busy_pkptr =
op_pk_create_fmt(QOS_PROVLINK_BUSY);
                                                op_pk_nfd_set (plink_busy_pkptr, "Msg_type",
PROVLINK_BUSY);
                                                op_pk_nfd_set (plink_busy_pkptr, "Route_id",
BROADCAST);
                                                count = 0;
                                                /* --- for each trunk ID, look up the provisioned
link table to see whether there is enough spare resource --- */
                                                for (i=0; i<length; i++)
                                                {
                                                op_pk_fd_get (pkptr, i+5, &trunk_id);
                                                p_link_id = trunk_id;


                                                if ( p_link_tbl[p_link_id].RsvBdw <
(p_link_tbl[p_link_id].UsdBdw + bdw_req/INCREASING_TIMES) )
                                                {
          op_pk_fd_set (plink_busy_pkptr, count+3, OPC_FIELD_TYPE_INTEGER, p_link_id, 8);
                                                p_link_tbl[p_link_id].Busy=1;
                                                count=count+1;
                                                }
                                                }
```

```
                                        if(count>0)
                                        { op_pk_nfd_set (plink_busy_pkptr, "Length",
count);
                                        op_pk_nfd_set (plink_busy_pkptr,
"Sequence",sequence);
                                        sequence++;
                                        op_pk_destroy (plink_busy_pkptr);
                                        }
                                        else op_pk_destroy(plink_busy_pkptr);
                                        }
                                        else    {
                                        op_pk_send (cfg_cnf_pkptr,
XMT_OUT_TO_ROUTER);
                                        }
                                        op_pk_destroy (pkptr);

                                /*----------------------KiT---------------------------------*/
                                Sum_UsdBdw = 0;
                                xx=1;
                                for (xx=1;xx<40;xx++)
                                        {Sum_UsdBdw = Sum_UsdBdw +
p_link_tbl[xx].UsdBdw;
                                        }
                                value_u = (Sum_UsdBdw*100 / Sum_RsvBdw);
                                op_stat_write(utilization_hndl,(double)value_u);
                                /*--------------------LOCAL-----------------------------*/
                                c=1;
                                for (c=1;c<20;c++)
                                        {
        op_stat_write(utilization_Link_hndl[c],(double)((p_link_tbl[c].UsdBdw +
p_link_tbl[19+c].UsdBdw)/sum_link));
        op_stat_write(utilization_PLink_hndl[c],(double)(p_link_tbl[c].UsdBdw/(double)2420000));
        op_stat_write(utilization_PLink_hndl[c+19],(double)(p_link_tbl[c+19].UsdBdw/(double)640
00));
                                        }
                                /*-----------------------------KiT----------------------------*/
                                break;
                                case TRK_RSC_RLS:
                                        overhead_msg_num=overhead_msg_num+1;
                                        op_stat_write(overhead_hndl,overhead_msg_num);
                                /* --- get the number of released trunks --- */
                                        op_pk_nfd_get (pkptr, "Length", &length);
                                        plink_busy_pkptr =
op_pk_create_fmt(QOS_PROVLINK_NOTBUSY);
                                        op_pk_nfd_set (plink_busy_pkptr, "Msg_type",
PROVLINK_NOTBUSY);
                                        op_pk_nfd_set (plink_busy_pkptr, "Route_id",
BROADCAST);
                                        count = 0;
                                        number=0;
                                        for(i=0;i<length;i++)
                                                {
                                                op_pk_fd_get (pkptr, count+4, &trunk_id);
                                                op_pk_fd_get (pkptr, count+5, &bdw_req);
                                                trunk_tbl[trunk_id].RsvBdw -= bdw_req;
                                                p_link_id = trunk_id;
                                                p_link_tbl[p_link_id].UsdBdw -= bdw_req;
                                                if(p_link_tbl[p_link_id].Busy==1)
                                                        {
```

```
                                                op_pk_fd_set (plink_busy_pkptr, number+3,
OPC_FIELD_TYPE_INTEGER, p_link_id, 8);
                                                p_link_tbl[p_link_id].Busy=0;
                                                number=number+1;
                                                }
                                                count=count+2;
                                                }
                                                if(count>0)
                                                {
                                                op_pk_nfd_set (plink_busy_pkptr, "Length",
number);
                                                op_pk_nfd_set (plink_busy_pkptr,
"Sequence",sequence);
                                                sequence++;
                                                //op_pk_send (plink_busy_pkptr,
XMT_OUT_TO_ROUTER);
                                                }
                                                else
                                                op_pk_destroy(plink_busy_pkptr);
                                                op_pk_destroy (pkptr);
                                                /*-------------------------KiT-------------*/
                                        Sum_UsdBdw = 0;
                                        xx=1;
                                        for (xx=1;xx<40;xx++)
                                                {Sum_UsdBdw = Sum_UsdBdw +
p_link_tbl[xx].UsdBdw;
                                                }
                                        value_u = (Sum_UsdBdw*100 / Sum_RsvBdw);
                                        op_stat_write(utilization_hndl,(double)value_u);
                                        /*--------------------LOCAL---------------------------------*/
                                        c=1;
                                        for (c=1;c<20;c++)
                                                {
        op_stat_write(utilization_Link_hndl[c],(double)((p_link_tbl[c].UsdBdw +
p_link_tbl[19+c].UsdBdw)/sum_link));

        op_stat_write(utilization_PLink_hndl[c],(double)(p_link_tbl[c].UsdBdw/(double)2420000));

        op_stat_write(utilization_PLink_hndl[c+19],(double)(p_link_tbl[c+19].UsdBdw/(double)640
00));
                                                }
                                        /*---------------------------------KiT-----------------------------*/
                                        break;
                                        }}
                                /** state (proc) transition processing **/
                                FSM_TRANSIT_ONLY ((PKT_ARRVL), 1, state1_enter_exec, ;, "proc",
"PKT_ARRVL", "", "proc", "proc")
                                        /*---------------------------------------------------------*/
                                        }
                        FSM_EXIT (0,QoSpPS)
                        }
        }
#if defined (__cplusplus)
        extern "C" {
#endif
        extern VosT_Fun_Status Vos_Catmem_Register (const char * , int , VosT_Void_Null_Proc,
VosT_Address *);
        extern VosT_Address Vos_Catmem_Alloc (VosT_Address, size_t);
        extern VosT_Fun_Status Vos_Catmem_Dealloc (VosT_Address);
```

```
#if defined (__cplusplus)
	}
#endif
Compcode
QoSpPS_init (void ** gen_state_pptr)
	{
	int _block_origin = 0;
	static VosT_Address	obtype = OPC_NIL;
	FIN (QoSpPS_init (gen_state_pptr))
	if (obtype == OPC_NIL)
		{
		/* Initialize memory management */
		if (Vos_Catmem_Register ("proc state vars (QoSpPS)",
			sizeof (QoSpPS_state), Vos_Vnop, &obtype) == VOSC_FAILURE)
			{
			FRET (OPC_COMPCODE_FAILURE)
			}
		}
	*gen_state_pptr = Vos_Catmem_Alloc (obtype, 1);
	if (*gen_state_pptr == OPC_NIL)
		{
		FRET (OPC_COMPCODE_FAILURE)
		}
	else
		{
		/* Initialize FSM handling */
		((QoSpPS_state *)(*gen_state_pptr))->current_block = 0;
		FRET (OPC_COMPCODE_SUCCESS)
		}
	}
void
QoSpPS_diag (void)
	{
	/* No Diagnostic Block */
	}
void
QoSpPS_terminate (void)
	{
	int _block_origin = __LINE__;
	FIN (QoSpPS_terminate (void))
	if (1)
		{
		Packet*		pkptr;
		Packet*		plink_busy_pkptr;
		char		file_name[128];
		char		node_name[15];
		int		msg_type;
		int		src_gtw;
		int		src_router;
		int		dest_gtw;
		int		dest_router;
		int		bdw_req;
		int		length;
		int		route_id;
		int		trunk_id;
		int		p_link_id;
		int		link_id;
		int		capacity;
		int		rsv_bdw;
		Packet*		cfg_cnf_pkptr;
```

```
		int		i;
		int		rkey;
		int		count;
		FILE*		f_hndl;
		QoSRoute	route_rec;
		TrunkItem	trunk_rec;
		PLinkItem	p_link_rec;
		LinkItem	link_rec;
		QoSFwd		fwd_pnt;
		Int		k;
		double		t;
		int		j;
		int		notfeasible;
		int		number;
		double		Sum_UsdBdw;
		double		value_u;
		int		check;
		int		xx;
		int		c;
		/* No Termination Block */
		}
	Vos_Catmem_Dealloc (pr_state_ptr);
	FOUT;
	}
/* Undefine shortcuts to state variables to avoid */
/* syntax error in direct access to fields of */
/* local variable prs_ptr in QoSpPS_svar function. */
#undef route_tbl
#undef trunk_tbl
#undef link_tbl
#undef p_link_tbl
#undef fwd_tbl_len
#undef own_id
#undef parent_id
#undef forward_table
#undef overhead_hndl
#undef out_going
#undef sequence
#undef reconfig_num_hndl
#undef reconfig_number
#undef utilization_hndl
#undef Sum_RsvBdw
#undef utilization_Link_hndl
#undef sum_link
#undef utilization_PLink_hndl
void
QoSpPS_svar (void * gen_ptr, const char * var_name, char ** var_p_ptr)
	{
	QoSpPS_state		*prs_ptr;
	FIN (QoSpPS_svar (gen_ptr, var_name, var_p_ptr))
	if (var_name == OPC_NIL)
		{
		*var_p_ptr = (char *)OPC_NIL;
		FOUT;
		}
	prs_ptr = (QoSpPS_state *)gen_ptr;
	if (strcmp ("route_tbl" , var_name) == 0)
		{
		*var_p_ptr = (char *) (prs_ptr->route_tbl);
		FOUT;
```

```
        }
if (strcmp ("trunk_tbl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->trunk_tbl);
        FOUT;
        }
if (strcmp ("link_tbl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->link_tbl);
        FOUT;
        }
if (strcmp ("p_link_tbl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->p_link_tbl);
        FOUT;
        }
if (strcmp ("fwd_tbl_len" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->fwd_tbl_len);
        FOUT;
        }
if (strcmp ("own_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->own_id);
        FOUT;
        }
if (strcmp ("parent_id" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->parent_id);
        FOUT;
        }
if (strcmp ("forward_table" , var_name) == 0)
        {
        *var_p_ptr = (char *) (prs_ptr->forward_table);
        FOUT;
        }
if (strcmp ("overhead_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->overhead_hndl);
        FOUT;
        }
if (strcmp ("out_going" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->out_going);
        FOUT;
        }
if (strcmp ("sequence" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->sequence);
        FOUT;
        }
if (strcmp ("reconfig_num_hndl" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->reconfig_num_hndl);
        FOUT;
        }
if (strcmp ("reconfig_number" , var_name) == 0)
        {
        *var_p_ptr = (char *) (&prs_ptr->reconfig_number);
```

```
            FOUT;
            }
if (strcmp ("utilization_hndl" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->utilization_hndl);
            FOUT;
            }
if (strcmp ("Sum_RsvBdw" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->Sum_RsvBdw);
            FOUT;
            }
if (strcmp ("utilization_Link_hndl" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->utilization_Link_hndl);
            FOUT;
            }
if (strcmp ("sum_link" , var_name) == 0)
            {
            *var_p_ptr = (char *) (&prs_ptr->sum_link);
            FOUT;
            }
if (strcmp ("utilization_PLink_hndl" , var_name) == 0)
            {
            *var_p_ptr = (char *) (prs_ptr->utilization_PLink_hndl);
            FOUT;
            }
*var_p_ptr = (char *)OPC_NIL;

FOUT;
}
```