Header Block

```
#include<stdio.h>
#define ARRIVAL op_intrpt_type()==OPC_INTRPT_STRM
#define IPSTREAM            0
#define QOSPF_STREAM    1

#define Path_Msg_Refresh_timeout 4000

#define InLIST_SIZE     10
#define TABLE_SIZE   InLIST_SIZE*5
#define MAX_GROUPS      100
#define TCP_PROTOCOL_ID             12
#define UDP_PROTOCOL_ID             20
#define RSVP_PROTOCOL_ID            46
#define REFRESH_PERIOD     300

#define RSVP_CURRENT_VER            0x01

#define PATH_MESSAGE                01
#define RESV_MESSAGE                02
#define PATH_ERROR                  03
#define RESV_ERROR                  04
#define PATH_TEAR                   05
#define RESV_TEAR                   06
#define RESV_CONF                   07

#define OFF           0
#define ON            1
#define UNDEFINED           -1
#define MCAST_ADDRESS        500
#define REFRESH_PERIOD 300

#define VL                  if(ON)
#define PMVL                if(OFF)
#define RMVL                if(ON)

#define MAX_NO_LINKS    10
/*
** IP Address structure
*/
typedef struct {
        int NetNum;
        int NodeNum;
} IPAddress;

struct mc_table {
        IPAddress outIn[InLIST_SIZE];
};

typedef struct {
        int interface[MAX_NO_LINKS];
        int mtu[MAX_NO_LINKS];
}mc_id;


/*
** Session Information
*/
typedef struct {
```

```
        IPAddress DestAddress;
        int ProtocolId;
        int DestPort;
} SessInfo;

/*
** Common Header
*/
typedef struct {
        int Version;
        int Msg_Type;
        int Send_TTL;
        double RSVP_Checksum;
        double RSVP_Length;
} CommonHdr;

/*
** Session object
*/
typedef struct {
        int Class_Num;
        double Length;
        IPAddress DestAddress;
        int Protocol_Id;
        int Flags;
        int DestPort;
} SessionObj;

/*
** RSVP Hop Object
*/
typedef struct {
        int Class_Num;
        double Length;
        IPAddress Next_Hop_Address;
        IPAddress Logical_Interface_Handle;
} RsvpHopObj;

/*
** Time values object
*/
typedef struct {
        int Class_Num;
        double Length;
        double Refresh_Period;
} TimeValObj;

/*
** Sender template
*/
typedef struct {
        int Class_Num;
        double Length;
        IPAddress SrcAddress;
        int SrcPort;
} SenderTemplateObj;

/*
** Sender T-Spec object
```

```
*/
typedef struct {
	int Class_Num;
	double Length;
	double req_bandwidth;
	double link_delay;
} SenderTspecObj;

/*
** FlowSpec Object
*/
typedef SenderTspecObj FlowSpecObj;

/*
** Filter Spec
*/
typedef SenderTemplateObj FilterSpecObj;

/*
** Style
*/
typedef struct {
	int Class_Num;
	double Length;
	int Resv_Style;
} StyleObj;

/*
** Scope object
*/
typedef struct {
	int Class_Num;
	double Length;
	IPAddress SrcAddress;
} ScopeObj;

/*
** Reservation Confirm object
*/
typedef struct {
	int Class_Num;
	double Length;
	IPAddress Receiver_Address;
} ResvConfirmObj;

/*
** Error Spec object
*/
typedef struct {
	int Class_Num;
	double Length;
	IPAddress Error_Node_Address;
	int Flags;
	int Error_code;
} ErrorSpecObj;



/*
```

```
** Sender Descriptor structure
*/
typedef struct {

        /* SENDER_TEMPLATE structure */
  SenderTemplateObj Sender_Template_Object;

        /* SENDER_TSPEC structure */
  SenderTspecObj Sender_Tspec_Object;

}SenderDescriptor;

/*
** Flow descriptor list structure
*/
typedef struct {

        /* FLOWSPEC structure */
  FlowSpecObj Flowspec_Object;

  /* this was added when it was known that FF style needs to be considered */
        /* FILTER_SPEC structure */
  FilterSpecObj Filter_Spec_Object;

}FlowDescriptorList;

/*
** PATH message structure
*/
typedef struct {

        /* Common Header structure */
  CommonHdr common_header;

        /* SESSION object structure */
  SessionObj Session_Object;

        /* RSVP_HOP Object structure */
  RsvpHopObj Rsvp_Hop_Object;

        /* TIME_VALUES Object structure */
  TimeValObj Time_Values_Object;

        /* Sender Descriptor structure */
  SenderDescriptor Sender_Descriptor;

}PathMessage;

/*
** RESV message structure
*/
typedef struct {

        /* Common Header structure */
  CommonHdr common_header           ;

        /* SESSION object structure */
  SessionObj Session_Object          ;

```

```
    /* RSVP_HOP Object structure */
 RsvpHopObj Rsvp_Hop_Object        ;

    /* TIME_VALUES Object structure */
 TimeValObj Time_Values_Object      ;

    /* RESV_CONFIRM Object structure */
 ResvConfirmObj Resv_Confirm_Object     ;

    /* SCOPE Object structure */
 ScopeObj Scope_Object           ;

    /* STYLE Object structure */
 StyleObj Style_Object           ;

    /* Flow descriptor list structure */
 FlowDescriptorList flow_descriptor_list[InLIST_SIZE];

}ResvMessage;

/*
** PathTear message structure
*/
typedef struct {

    /* Common Header structure */
 CommonHdr common_header          ;

    /* SESSION object structure */
 SessionObj Session_Object         ;

    /* RSVP_HOP Object structure */
 RsvpHopObj Rsvp_Hop_Object        ;

    /* Sender Descriptor structure */
 SenderDescriptor Sender_Descriptor       ;

}PathTearMessage;

/*
** ResvTear message structure
*/
typedef struct {
    /* Common Header structure */
 CommonHdr common_header          ;

    /* SESSION object structure */
 SessionObj Session_Object         ;

    /* RSVP_HOP Object structure */
 RsvpHopObj Rsvp_Hop_Object        ;

    /* SCOPE Object structure */
 ScopeObj Scope_Object           ;

    /* STYLE Object structure */
 StyleObj Style_Object           ;

    /* Flow descriptor list structure */
```

```
  FlowDescriptorList flow_descriptor_list  ;

}ResvTearMessage;

/*
** PathErr message structure
*/
typedef struct {

	/* Common Header structure */
  CommonHdr common_header           ;

	/* SESSION object structure */
  SessionObj Session_Object           ;

	/* ERROR_SPEC Object structure */
  ErrorSpecObj Error_Spec_Object        ;

	/* Sender Descriptor structure */
  SenderDescriptor Sender_Descriptor        ;

}PathErrMessage;

/*
** ResvErr message structure
*/
typedef struct {

	/* Common Header structure */
  CommonHdr common_header           ;

	/* SESSION object structure */
  SessionObj Session_Object           ;

	/* ERROR_SPEC Object structure */
  ErrorSpecObj Error_Spec_Object        ;

	/* SCOPE Object structure */
  ScopeObj Scope_Object;

	/* STYLE Object structure */
  StyleObj Style_Object;

	/* Flow descriptor list structure */
  /*  ErrorFlowDescriptor error_flow_descriptor;  */

} ResvErrMessage;

/*
** ResvConf message structure
*/
typedef struct {

	/* Common Header structure */
  CommonHdr common_header           ;

	/* SESSION object structure */
  SessionObj Session_Object           ;
```

```
      /* ERROR_SPEC Object structure */
  ErrorSpecObj Error_Spec_Object       ;


      /* RESV_CONFIRM Object structure */
  ResvConfirmObj Resv_Confirm_Object      ;

      /* STYLE Object structure */
  StyleObj Style_Object           ;

      /* Flow descriptor list structure */
  FlowDescriptorList flow_descriptor_list  ;

}ResvConfMessage;

/*
** Path State Block (PSB) structure
*/
typedef struct {

      /* SESSION object structure */
  SessionObj Session_Object;

      /* SENDER_TEMPLATE structure */
  SenderTemplateObj Sender_Template_Object;

      /* SENDER_TSPEC structure */
  SenderTspecObj Sender_Tspec_Object;

      /* Previous Hop IP address from PHOP object */
  IPAddress Previous_Hop_Address;

      /* Logical Interface Handle from PHOP object*/
  IPAddress LIH;

      /* Remaining IP TTL */
  int rem_IP_TTL;

      /* To determine EDGE of the RSVP network */
  int E_Police_Flag;

      /* To determine if the App. is local or not */
  int Local_Only_Flag;

      /* List of outgoing Interfaces for this
         (sender, destination) */
  IPAddress OutInterface_List[InLIST_SIZE];

      /* Expected incoming interface */
  IPAddress IncInterface               ;

      /* Blockade FLOWSPEC */
  FlowSpecObj Qb;

}PathStateBlock;

/*
** Reservation State Block (RSB) structure
*/
```

```
typedef struct {

      /* SESSION object structure */
  SessionObj Session_Object;

      /* Next Hop IP address from PHOP object */
  IPAddress Next_Hop_Address;

      /* Outgoing Interface on which reservation is
         to be made or has been made*/
  IPAddress OI;

      /* Newly added when it was known that FF style need to be considered */
      /* This would be a single FILTER_SPEC for FF style */
  FilterSpecObj Filter_Spec_Object;

      /* STYLE Object structure */
  StyleObj Style_Object            ;

      /* FLOWSPEC structure */
  FlowSpecObj Flowspec_Object;

      /* SCOPE Object structure */
  ScopeObj Scope_Object            ;

      /* RESV_CONFIRM Object structure */
  ResvConfirmObj Resv_Confirm_Object      ;

}ResvStateBlock;

/*
** Traffic Control State Block
*/
typedef struct {

      /* SESSION object structure */
  SessionObj Session_Object;

      /* Outgoing Interface on which reservation is
         to be made or has been made*/
  IPAddress OI;

      /* Newly added when it was known that FF style need to be considered */
  FilterSpecObj Filter_Spec_Object[InLIST_SIZE];

      /* STYLE Object structure */
  StyleObj Style_Object            ;

      /* Effective flowspec (LUB of FLOWSPEC's from
      matching RSB's) structure */
  FlowSpecObj TC_Flowspec;

      /* Updated object structure to be forwarded after merging */
  FlowSpecObj Fwd_Flowspec;

      /* Effective sender Tspec structure */
  SenderTspecObj TC_Tspec;

      /* Entry Police flags */
```

```
  int E_Police_Flag;

      /* Merge Police flags: Need not use this one*/
  int M_Police_Flag;

      /* Branch Police flags */
  int B_Police_Flag;

      /* Reservation Handle */
  int Rhandle;

      /* Filter handle list */
  int Fhandle;

      /* RESV_CONFIRM Object structure */
  ResvConfirmObj Resv_Confirm_Object;

}TrafficControlStateBlock;


struct GenList {
      struct GenList * next;
      struct GenList * curr;
      char *ele;
};

typedef struct {
      int isfull;
      IPAddress OI;
      FlowSpecObj TC_Flowspec;
      SenderTspecObj Path_Te;
} RHandleType;

typedef struct {
      int isfull;
      IPAddress OI;
      int Rhandle;
      SessionObj session;
      FilterSpecObj FilterSpecObject;
} FHandleType;

typedef struct GenList GenList;

#ifndef _DEBUG_FUNCS_H
#define _DEBUG_FUNCS_H

#include<stdio.h>
#include<errno.h>

#define PrintErr(p)  \
            printf p ; printf(" : FileName = %s : LineNo = %d :\n", __FILE__, __LINE__); printf("ErrNo = %d : E

#define MALLOC(sz)          calloc(1,sz)
#define Open(x,y,z)      \
        open(x,y,z); \
        if( errno ) \
        { PrintErr(("Cannot open file %s ",x)); }

#define OpenE(x,y,z) \
```

```
            open(x,y,z); \
            if( errno ) \
            { PrintErr( ("Cannot open file %s ",x) ); exit(errno); }

#define Malloc(x) \
            malloc(x); \
            if( errno ) \
            { PrintErr(("Cannot Allocate memory")); }

#define MallocE(x) \
            malloc(x); \
            if( errno ) \
            { PrintErr(("Cannot Allocate memory")); exit(errno); }

#define Calloc(x,y) \
            calloc(x,y); \
            if( errno ) \
            { PrintErr(("Cannot Allocate memory")); }

#define CallocE(x,y) \
            calloc(x,y); \
            if( errno ) \
            { PrintErr(("Cannot Allocate memory")); exit(errno); }

#define Realloc(x,y) \
            realloc(x,y); \
            if( errno ) \
            { PrintErr(("Cannot Allocate memory")); }

#define ReallocE(x,y) \
            realloc(x,y); \
            if( errno ) \
            { PrintErr(("Cannot Allocate memory")); exit(errno); }

#define Read(x,y,z) \
            read(x,y,z); \
            if( errno ) \
            { PrintErr(("Read failed")); }

#define ReadE(x,y,z) \
            read(x,y,z); \
            if( errno ) \
            { PrintErr(("Read failed")); exit(errno); }

#define Write(x,y,z) \
            write(x,y,z); \
            if( errno ) \
            { PrintErr(("Read failed")); }

#define WriteE(x,y,z) \
            write(x,y,z); \
            if( errno ) \
            { PrintErr(("Read failed")); exit(errno); }

#define Fopen(x,y) \
            fopen(x,y); \
            if( errno ) \
            { PrintErr(("Cannot open file %s", x)); }
```

```
#define FopenE(x,y) \
        fopen(x,y); \
        if( errno ) \
        { PrintErr(("Cannot open file %s", x)); exit(errno); }

#define Fread(x,y,z,s) \
        fread(x,y,z,s); \
        if( errno ) \
        { PrintErr(("Cannot read file")); }

#define FreadE(x,y,z,s) \
        fread(x,y,z,s); \
        if( errno ) \
        { PrintErr(("Cannot read file")); exit(errno); }

#define Fwrite(x,y,z,s) \
        fwrite(x,y,z,s); \
        if( errno ) \
        { PrintErr(("Cannot read file")); }

#define FwriteE(x,y,z,s) \
        fwrite(x,y,z,s); \
        if( errno ) \
        { PrintErr(("Cannot read file")); exit(errno); }

#define Memcpy(x,y,z) op_prg_mem_copy( y, x, z );

#endif

/*
** Function Prototypes
*/
int CmpSession(SessionObj * so1, SessionObj *so2);
int CmpSenderTspec(SenderTspecObj * sts1, SenderTspecObj *sts2);
int CmpSenderTemplate(SenderTemplateObj * st1, SenderTemplateObj *st2);
int CmpFlowSpec(FlowSpecObj * sts1, FlowSpecObj *sts2);
int CmpFilterSpec(FilterSpecObj * fs1, FilterSpecObj *fs2);
int CmpScope(ScopeObj * sc1, ScopeObj *sc2);
int CmpAddress(IPAddress * ipa1, IPAddress * ipa2);
int UndefAddress( IPAddress * ipa );
int IsMcast( IPAddress * ipa );
int CmpAddressL(IPAddress * ipa1, IPAddress * ipa2);
int CmpLocalAddress(IPAddress * ipa);
void MakeAddress( IPAddress * ipa, int netno, int nodeno);
void CpyAddress(IPAddress * ipa1, IPAddress * ipa2);
void InitAddressL( IPAddress * ipa );
void CpyAddressL(IPAddress * ipa1, IPAddress * ipa2);
void RouterInitialize();
/** Added By Lavu **/
void Mcast_Route_Query(IPAddress sa, IPAddress, IPAddress da, IPAddress *outl);

void PathMsgPro(PathMessage *pmess, IPAddress InIf);
void ResvMsgPro(ResvMessage *rmess);
void UpdateTrafficControl(ResvStateBlock *activeRSB);
void ResvRefresh( ResvStateBlock *rsbEle, IPAddress );
void PathTearPro(PathTearMessage *pmess);
int PathRefresh( PathStateBlock *psbEle , IPAddress OI);
void DeleteListEle( GenList *lst, char *Ele, int size );
void ListInit(GenList* lst);
```

```
void * AddListEle( GenList *lst, char *Ele, int size );
char * GetNextListEle( GenList *lst, int flag );
int IsListEmpty(GenList* lst);
void SendPktToIP( char *, int, int );
void SendResvPktToIP( ResvMessage * resvMsg, IPAddress PH);
int GetStream(IPAddress);
FILE * GetFilePtr( void );
void PrintCommHdr( CommonHdr ch );
void PrintSess( SessionObj sh );
void PrintRsvpHop( RsvpHopObj ch );
void PrintTime( TimeValObj ch );
void PrintSendTemp( SenderTemplateObj ch );
void PrintSendTSpec( SenderTspecObj ch );
void PrintFilter( FilterSpecObj ch );
void PrintFlow( FlowSpecObj ch );
void PrintScope( ScopeObj ch );
void PrintStyle( StyleObj ch );
void PrintConfirm( ResvConfirmObj ch );
void PrintLine( char * ch );
void PrintMsg( char * MsgPk );
void PrintPSB( PathStateBlock * PathStateBlk );
int GetFwdFS(IPAddress oi, FlowSpecObj *fwdFS);
int TC_AddFlowspec( IPAddress OI, FlowSpecObj fs, SenderTspecObj ts,
                                FlowSpecObj *fwdFS);
int TC_ModFlowspec( IPAddress OI, int Rhandle, FlowSpecObj fs,
        SenderTspecObj ts, FlowSpecObj *fwdFS);
int TC_DelFlowspec( IPAddress OI, int Rhandle );
int TC_AddFilter( IPAddress OI, int Rhandle, SessionObj so,
        FilterSpecObj filter);
int TC_DelFilter( IPAddress OI, int Fhandle );
```

State Variable Block

```
Objid \my_id;
Objid \ip_objid, \qospf_objid;
int \NoOfLinks;

struct mc_table \mcast_route_table[MAX_GROUPS];

int \in_strm;
int \ip_strm;
struct GenList \PSBList;
struct GenList \RSBList;
struct GenList \TCSBList;
IPAddress \LocalAddress[InLIST_SIZE];
int \obt_pk_srcnet;
int \obt_pk_srcnode;
int \send_pathmsg;
int \send_resvmsg;
int \send_pathtearmsg;
int \send_resvtearmsg;
int \send_resvconf;
int \protocolid;
int \Indep_Rou;
RHandleType \FlowTable[TABLE_SIZE];
FHandleType \FilterTable[TABLE_SIZE];
```

Temporary Variable Block

```
/*  mc_id         *mc_table;  */
```

Function Block

```
char *     LogFileName = "logfile";

void
InitVariables(IPAddress *LA,GenList psblist, GenList rsblist, GenList tcsblist)
{
        CpyAddressL( LocalAddress, LA );
        PSBList = psblist;
        RSBList = rsblist;
        TCSBList = tcsblist;
}
int
CmpSession(SessionObj * so1, SessionObj *so2)
{
        if( so1->DestAddress.NetNum == so2->DestAddress.NetNum &&
            so1->DestAddress.NodeNum == so2->DestAddress.NodeNum &&
                so1->Protocol_Id == so2->Protocol_Id &&
                so1->DestPort == so2->DestPort )
        return 1;
        return 0;
}

int
CmpSenderTspec(SenderTspecObj * sts1, SenderTspecObj *sts2)
{
        if( sts1->req_bandwidth == sts2->req_bandwidth &&
                sts1->link_delay == sts2->link_delay )
        return 1;
        return 0;
}

int
CmpSenderTemplate(SenderTemplateObj * st1, SenderTemplateObj *st2)
{
        if( st1->SrcAddress.NetNum == st2->SrcAddress.NetNum &&
            st1->SrcAddress.NodeNum == st2->SrcAddress.NodeNum &&
                st1->SrcPort == st2->SrcPort )
        return 1;
        return 0;
}

int
CmpFlowSpec(FlowSpecObj * sts1, FlowSpecObj *sts2)
{
        if( sts1->req_bandwidth == sts2->req_bandwidth &&
                sts1->link_delay == sts2->link_delay )
        return 1;
        return 0;
}

int
CmpFilterSpec(FilterSpecObj * fs1, FilterSpecObj *fs2)
```

```
{
	if( fs1->SrcAddress.NetNum == fs2->SrcAddress.NetNum &&
	   fs1->SrcAddress.NodeNum == fs2->SrcAddress.NodeNum &&
	      fs1->SrcPort == fs2->SrcPort )
	return 1;
	return 0;
}

int
CmpScope(ScopeObj * sc1, ScopeObj *sc2)
{
	if( sc1->SrcAddress.NetNum == sc2->SrcAddress.NetNum &&
	   sc1->SrcAddress.NodeNum == sc2->SrcAddress.NodeNum )
	return 1;
	return 0;
}

int
CmpAddress(IPAddress * ipa1, IPAddress * ipa2)
{
	if( ipa1->NetNum == ipa2->NetNum &&
	   ipa1->NodeNum == ipa2->NodeNum )
	return 1;
	return 0;

}

void
MakeAddress( IPAddress * ipa, int netno, int nodeno)
{
	ipa->NetNum = netno;
	ipa->NodeNum = nodeno;
}

void
CpyAddress(IPAddress * ipa1, IPAddress * ipa2)
{
	ipa1->NetNum = ipa2->NetNum;
	ipa1->NodeNum = ipa2->NodeNum;
}

void
RouterInitialize()
{
	char mess[100];
	int i;
	Objid sub_netid;
	ListInit(&PSBList);
	ListInit(&RSBList);
	ListInit(&TCSBList);
	/* Initialize LocalAddress */
	my_id = op_id_self();

	ip_objid = op_topo_assoc( my_id, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 0 );
	qospf_objid = op_topo_assoc( my_id, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 1);


	op_ima_obj_attr_get( ip_objid, "Indep_Rou", &Indep_Rou );

```

```
	if( !Indep_Rou )
		sub_netid=op_topo_parent( op_topo_parent(my_id));
	else
		sub_netid=op_topo_parent( my_id );

	NoOfLinks = op_topo_assoc_count( sub_netid,OPC_TOPO_ASSOC_IN,OPC_OBJTYPE_LKSIMP );

	for( i=0; i<NoOfLinks; ++i )
	{
		sprintf( mess,"net_num%1d",i );
		op_ima_obj_attr_get( ip_objid, mess, &LocalAddress[i].NetNum );
		sprintf( mess,"node_num%1d",i );
		op_ima_obj_attr_get( ip_objid, mess, &LocalAddress[i].NodeNum );
	}
}

int
UndefAddress( IPAddress * ipa )
{
	if(ipa->NetNum == 0 || ipa->NetNum == UNDEFINED)
		return 1;
	if(ipa->NodeNum == UNDEFINED)
		return 1;
	return 0;
}

int
IsMcast( IPAddress * ipa )
{
	if( ipa->NetNum == MCAST_ADDRESS )
		return 1;
	return 0;
}

void
InitAddressL( IPAddress * ipa )
{
	int i;
	for( i=0; i<InLIST_SIZE; ++i )
	{
		ipa[i].NetNum = 0;
		ipa[i].NodeNum = 0;
	}
}

void
CpyAddressL(IPAddress * ipa1, IPAddress * ipa2)
{
	int i;
	for( i=0; i<InLIST_SIZE; ++i )
	{
		ipa1[i].NetNum = ipa2[i].NetNum;
		ipa1[i].NodeNum = ipa2[i].NodeNum;
	}
}

int
GetStream( IPAddress strm )
{
```

```
	int i;
	for( i=0; i<=NoOfLinks; ++i )
	{
		if( strm.NetNum == LocalAddress[i].NetNum &&
			strm.NodeNum == LocalAddress[i].NodeNum )
			return i;
	}
	return -1;
}

int
CmpAddressL(IPAddress * ipa1, IPAddress * ipa2)
{
	int i;
	for( i=0; i<InLIST_SIZE; ++i )
	{
		if(OFF)   printf("CmpAddressL:%d: %d.%d  %d.%d\n", i,
					ipa1[i].NetNum, ipa1[i].NodeNum,
					ipa2[i].NetNum, ipa2[i].NodeNum);
		if( ipa1[i].NetNum != ipa2[i].NetNum ||
			ipa1[i].NodeNum != ipa2[i].NodeNum )
			return 0;
	}
	return 1;
}

int
CmpLocalAddress(IPAddress * ipa)
{
	int i;
	for( i=0; i<InLIST_SIZE; ++i )
	{
		if( LocalAddress[i].NetNum == ipa->NetNum &&
			LocalAddress[i].NodeNum == ipa->NodeNum )
			return 1;
	}
	return 0;
}

void
PathMsgPro(PathMessage *pmess, IPAddress InIf)
{
	PathStateBlock * fPSB = NULL;
	PathStateBlock * cPSB = NULL;

	PathStateBlock * psbEle = NULL;
	ResvStateBlock * rsbEle = NULL;
	ResvStateBlock * activeRSB = NULL;
	int Path_Refresh_Needed_Flag = OFF;
	int Resv_Refresh_Needed_Flag = OFF;
	int foundFlag = OFF;
	int routeChangeNoti;
	int LocalInFlag = OFF;
	int i;
	IPAddress Rt_InIf, Rt_OutL[InLIST_SIZE];

	/*
	VL printf("Entered PathMsgPro from %d.%d\n", InIf.NetNum, InIf.NodeNum);
	VL PMVL  printf("Received Path Message Structure \n"
```

```
                "Sender Template Object's Src Address = %d.%d",
                    pmess->Sender_Descriptor.Sender_Template_Object.SrcAddress.NetNum,
                    pmess->Sender_Descriptor.Sender_Template_Object.SrcAddress.NodeNum );
    */
    InitAddressL( Rt_OutL );
    foundFlag = OFF;
    psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 1 );
    while( psbEle )
    {
        if( CmpAddress( &psbEle->Session_Object.DestAddress,
                                &pmess->Session_Object.DestAddress) &&
          psbEle->Session_Object.Protocol_Id ==
            pmess->Session_Object.Protocol_Id  &&
          psbEle->Session_Object.DestPort !=
            pmess->Session_Object.DestPort && (
          psbEle->Session_Object.DestPort == 0 ||
            pmess->Session_Object.DestPort == 0 ))
        {
            /**
            *** Send Path Error Message : Conflicting Destination Port
            **/
            VL printf("Send Path Error Message : Conflicting Destination Port\n");
            return;
        }
        if( CmpAddress( &psbEle->Sender_Template_Object.SrcAddress,
                    &pmess->Sender_Descriptor.Sender_Template_Object.SrcAddress) &&
            psbEle->Sender_Template_Object.SrcPort !=
            pmess->Sender_Descriptor.Sender_Template_Object.SrcPort && (
            psbEle->Sender_Template_Object.SrcPort == 0 ||
            pmess->Sender_Descriptor.Sender_Template_Object.SrcPort==0 ))
        {
            /**
            *** Send Path Error Message : Ambiguous Path
            **/
            VL  printf("Send Path Error Message : Ambiguous Path\n");
            return;
        }
        if( CmpSession( &psbEle->Session_Object, &pmess->Session_Object) &&
            CmpSenderTemplate( &psbEle->Sender_Template_Object,
                &pmess->Sender_Descriptor.Sender_Template_Object) &&
            psbEle->Local_Only_Flag == OFF )
        {
            fPSB = psbEle;
        }
        else
        {
            fPSB = NULL;
        }

        /**
        *** Search for the PSB whose session and sender template
        *** matches the corresponding objects in the path message.
        *** and whose incomming interface matches the IncInterface.
        **/
        if( CmpSession( &psbEle->Session_Object, &pmess->Session_Object) &&
            CmpSenderTemplate( &psbEle->Sender_Template_Object,
                &pmess->Sender_Descriptor.Sender_Template_Object) &&
            CmpAddress( &psbEle->IncInterface, &InIf) )
        {
```

```
            /**
            *** If matching PSB is found
            **/
                VL PMVL     printf("Matching PSB found \n");
                /*  PrintPSB(psbEle);  */
                if( !CmpAddress( &psbEle->Previous_Hop_Address,
                                &pmess->Rsvp_Hop_Object.Next_Hop_Address) ||
                     !CmpAddress( &psbEle->LIH,
                                &pmess->Rsvp_Hop_Object.Logical_Interface_Handle) ||
                     !CmpSenderTspec( &psbEle->Sender_Tspec_Object,
                            &pmess->Sender_Descriptor.Sender_Tspec_Object) )
                {
                    CpyAddress( &psbEle->Previous_Hop_Address,
                                &pmess->Rsvp_Hop_Object.Next_Hop_Address );
                    CpyAddress( &psbEle->LIH,
                                &pmess->Rsvp_Hop_Object.Logical_Interface_Handle);
                     Memcpy( &psbEle->Sender_Tspec_Object,
                            &pmess->Sender_Descriptor.Sender_Tspec_Object,
                            sizeof(SenderTspecObj) );
                    Path_Refresh_Needed_Flag = ON;
                }
                if( !CmpAddress( &psbEle->Previous_Hop_Address,
                                &pmess->Rsvp_Hop_Object.Next_Hop_Address) ||
                     !CmpAddress( &psbEle->LIH,
                                &pmess->Rsvp_Hop_Object.Logical_Interface_Handle) )
                {
                    Resv_Refresh_Needed_Flag = ON;
                }
                cPSB = psbEle;
                foundFlag = ON;
                break;
            }
            psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 0 );
        }

        if( foundFlag == OFF )
        {
        /**
        *** If no matching PSB is found
        **/
            VL PMVL     printf("No matching PSB \n");

            psbEle = (PathStateBlock *) MALLOC( sizeof(PathStateBlock) );
            if( psbEle == NULL )
            {
                PrintErr( ("No Memory") );
            }
            Memcpy( &psbEle->Session_Object,
                    &pmess->Session_Object,
                    sizeof(SessionObj));
            Memcpy( &psbEle->Sender_Template_Object,
                    &pmess->Sender_Descriptor.Sender_Template_Object,
                    sizeof(SenderTemplateObj) );
            Memcpy( &psbEle->Sender_Tspec_Object,
                    &pmess->Sender_Descriptor.Sender_Tspec_Object,
                    sizeof(SenderTspecObj) );
            CpyAddress( &psbEle->Previous_Hop_Address,
                    &pmess->Rsvp_Hop_Object.Next_Hop_Address );
            CpyAddress( &psbEle->LIH,
```

...
...

```
                &pmess->Rsvp_Hop_Object.Logical_Interface_Handle );
        InitAddressL( psbEle->OutInterface_List );
        VL PMVL printf("LocalAddress = %d.%d Sender = %d.%d\n",
            LocalAddress[0].NetNum, LocalAddress[0].NodeNum,
            pmess->Sender_Descriptor.Sender_Template_Object.SrcAddress.NetNum,
            pmess->Sender_Descriptor.Sender_Template_Object.SrcAddress.NodeNum);
        if( CmpLocalAddress(
            &pmess->Sender_Descriptor.Sender_Template_Object.SrcAddress ) )
        {
            VL PMVL printf("Local Application\n");
            CpyAddress( &psbEle->OutInterface_List[0],
                &pmess->Sender_Descriptor.Sender_Template_Object.SrcAddress);
            psbEle->IncInterface.NetNum = UNDEFINED;
            psbEle->IncInterface.NodeNum = UNDEFINED;
            LocalInFlag = ON;
            psbEle->Local_Only_Flag = OFF;
        }
        else
        {
            VL PMVL printf("Not Local Application\n");
            CpyAddress( &psbEle->IncInterface, &InIf );
            psbEle->Local_Only_Flag = ON;
        }
        cPSB = (PathStateBlock *)AddListEle( &PSBList, (char*)psbEle, sizeof(PathStateBlock) );
        Path_Refresh_Needed_Flag = ON;
        free(psbEle);
        psbEle = cPSB;
    }

    if( cPSB == NULL )
    {
        /**
        *** Nothing in cPSB so return
        **/
            VL  printf("Nothing in cPSB so return\n");
        return;
    }

    /**
    *** Start or Restart the Cleanup timer
    **/

    /**
    *** Update cPSB
    **/
    cPSB->rem_IP_TTL = pmess->common_header.Send_TTL;

    /**
    *** If the PSB is new or there is no route change notification
    *** but not if the cPSB is from the local API
    **/

    /**
    *** routeChangeNoti =
    **/
    if( (foundFlag == OFF ) && LocalInFlag == OFF )
    {
        if( IsMcast( &cPSB->Session_Object.DestAddress ) )
        {
```

```
                Mcast_Route_Query(
                    cPSB->Sender_Template_Object.SrcAddress, InIf,
                    cPSB->Session_Object.DestAddress, Rt_OutL );
                VL PMVL printf("Mcast_Route_Query\n");

                /*** Commenting Temporarily ****************

                if( fPSB != cPSB )
                {
                    if( fPSB != NULL )
                    {
                        CpyAddressL( cPSB->OutInterface_List,
                                    fPSB->OutInterface_List );
                        InitAddressL(fPSB->OutInterface_List);
                        fPSB->Local_Only_Flag = ON;
                    }
                    cPSB->Local_Only_Flag = OFF;
                    fPSB = cPSB;
                }
                */
                fPSB = cPSB;
                if( !CmpAddressL( Rt_OutL, fPSB->OutInterface_List ))
                {
                    CpyAddressL( fPSB->OutInterface_List, Rt_OutL );

                    for(i=0; i<InLIST_SIZE; ++i)
                    {
                        if( !UndefAddress( &fPSB->OutInterface_List[i] ) )
                        {
                            PathRefresh( fPSB , fPSB->OutInterface_List[i] );
                        }
                    }
                    psbEle->rem_IP_TTL--;
                    /**
                    *** Execute PATH LOCAL REPAIR
                    **/
                    VL PMVL     printf("Execute PATH LOCAL REPAIR\n");
                }
            }
        }
        if( Path_Refresh_Needed_Flag == OFF )
        {
            VL RMVL printf("No Path Refresh Needed : Leaving PathMsgPro\n");
            return;
        }
        else
        {
            IPAddress tOutList[InLIST_SIZE];
            int olFlag = OFF;
            InitAddressL( tOutList );
            fPSB=cPSB;
            {
                /**
                *** Make an path event upcall for each local application
                *** for this session
                **/
            }
            if( fPSB == NULL )
            {
```

```
            PrintErr( ("fPSB is NULL") );
        }
        /*
        ** if outInterfacelist is not empty
        */
        if( fPSB && !CmpAddressL( fPSB->OutInterface_List, tOutList ))
        {
            /**
            *** Execute PATH REFRESH SEQUENCE
            **/
            VL PMVL     printf("Execute PATH REFRESH SEQUENCE\n");
            for(i=0; i<InLIST_SIZE; ++i)  {
                if( !UndefAddress( &fPSB->OutInterface_List[i] ) )
                {
                    PathRefresh( fPSB , fPSB->OutInterface_List[i] );
                }
            }
            psbEle->rem_IP_TTL--;
        }
        rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 1 );
        olFlag = OFF;
        activeRSB = NULL;
        while( rsbEle )
        {
            if( CmpFilterSpec( &rsbEle->Filter_Spec_Object,
                    (FilterSpecObj *)&fPSB->Sender_Template_Object ) )
            {
                for(i=0; i<InLIST_SIZE; ++i)  {
                    if( CmpAddress( &rsbEle->OI, &fPSB->OutInterface_List[i]) )
                    {
                        activeRSB = rsbEle;
                        olFlag = ON;
                        break;
                    }
                }
            }
            rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 0 );
        }
        if( olFlag == OFF )
        {
            return;
        }
        /**
        *** Execute RESV REFRESH SEQUENCE for the PHOP in the fPSB
        **/
        /**
        *** Execute the UPDATE TRAFFIC CONTROL
        **/
        if( activeRSB )
        {
            UpdateTrafficControl(activeRSB);
        }
        else
        {
            VL RMVL printf("No active RSB found\n");
        }
        VL RMVL printf("Leaving PathMsgPro\n");
        return;
    }
```

```
}

int
PathRefresh( PathStateBlock *psbEle , IPAddress OI )
{
        PathMessage* pm;
        PathMessage* tmppm;
        int data_len;
        VL  PMVL printf("PathRefresh from %d.%d to %d.%d\n",
                        psbEle->IncInterface.NetNum,psbEle->IncInterface.NodeNum,
                        OI.NetNum, OI.NodeNum);

        pm = (PathMessage*) op_prg_mem_alloc( sizeof(PathMessage) );
        tmppm = (PathMessage*) op_prg_mem_alloc( sizeof(PathMessage) );
        memset( tmppm, 0, sizeof(PathMessage) );
        memset( pm, 0, sizeof(PathMessage) );

        if( psbEle->rem_IP_TTL == 1 )
        {
                /* Life ended */
                return;
        }
        /* TO BE DONE */
        /* E_Police Flag */
        /*  FillCommonHdr(&pm->common_header, PATH_MESSAGE, psbEle->rem_IP_TTL);  */
        pm->common_header.Msg_Type = PATH_MESSAGE;
        pm->common_header.Send_TTL = psbEle->rem_IP_TTL;
        pm->Session_Object = psbEle->Session_Object;
        pm->Time_Values_Object.Refresh_Period = REFRESH_PERIOD;
        Memcpy( &pm->Sender_Descriptor.Sender_Template_Object,
                  &psbEle->Sender_Template_Object, sizeof(SenderTemplateObj) );
        Memcpy( &pm->Sender_Descriptor.Sender_Tspec_Object,
                    &psbEle->Sender_Tspec_Object, sizeof(SenderTspecObj) );

        pm->Rsvp_Hop_Object.Next_Hop_Address = OI;
        pm->Rsvp_Hop_Object.Logical_Interface_Handle = psbEle->IncInterface;

        SendPktToIP( (char *)pm, GetStream(OI), PATH_MESSAGE );
        /*  op_prg_mem_free(pm);  */

        if(Indep_Rou)
                op_intrpt_schedule_remote( op_sim_time, 8, qosof_objid );
}



void
ResvMsgPro(ResvMessage *rmess)
{
        /**
        *** These variables are used to control immediate reservation refreshes.
        **/
        IPAddress Refresh_PHOP_list[InLIST_SIZE];
        int Resv_Refresh_Needed_Flag = OFF,
                NeworMod = OFF;
        IPAddress OI;
        PathStateBlock * psbEle;
        ResvStateBlock * rsbEle;
        ResvStateBlock * activeRSB;
```

```
FlowDescriptorList *fdlist;
int inx;
int i;
int fdinx;
int newFlag = OFF;
int psbFoundFlag = OFF;
GenList locList;

VL  RMVL printf("Entered ResvMsgPro Local Address = %d.%d\n",
            LocalAddress[0].NetNum, LocalAddress[0].NodeNum);
ListInit(&locList);
InitAddressL( Refresh_PHOP_list );

/**
*** Determine the outgoing interface
**/
OI = rmess->Rsvp_Hop_Object.Logical_Interface_Handle;
/*  VL PrintMsg(rmess);  */

/**
*** Check the path state
**/
psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 1 );
psbFoundFlag = OFF;
while( psbEle )
{
    VL RMVL printf("PSB: sender = %d.%d\n",
        psbEle->Sender_Template_Object.SrcAddress.NetNum,
        psbEle->Sender_Template_Object.SrcAddress.NodeNum );
    for(inx=0;inx<InLIST_SIZE;++inx)
    {
            fdlist = &rmess->flow_descriptor_list[inx];
        if( CmpAddress( &psbEle->Sender_Template_Object.SrcAddress,
                    &fdlist->Filter_Spec_Object.SrcAddress) &&
            psbEle->Sender_Template_Object.SrcPort !=
            fdlist->Filter_Spec_Object.SrcPort && (
            fdlist->Filter_Spec_Object.SrcPort == 0 ||
            psbEle->Sender_Template_Object.SrcPort == 0 ) )
        {
            /**
            *** Build and send an "Ambiguous Path" PathErr message
            **/
            VL  RMVL printf("Ambiguous Path \n");
            return;
        }
    }
    if( CmpSession( &rmess->Session_Object, &psbEle->Session_Object ))
        psbFoundFlag = ON;
    psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 0 );
}
if( psbFoundFlag == OFF )
{
    /**
    *** Build and send a ResvErr message specifying "No Path Infomation"
    *** drop the Resv message.
    **/
    VL  RMVL printf("No Path Infomation\n");
    return;
}
```

```
        /**
        *** Check for incompatible styles.
        **/
        /**
        *** Process the flow descriptor list to make reservations.
        ***
        *** Check the path state
        **/
for(fdinx=0;fdinx<InLIST_SIZE;++fdinx)
{
        fdlist = &rmess->flow_descriptor_list[fdinx];
        psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 1 );
        while( psbEle )
        {
                /*  VL PrintPSB(psbEle);  */
                if( CmpAddress( &psbEle->Sender_Template_Object.SrcAddress,
                                        &fdlist->Filter_Spec_Object.SrcAddress) &&
                        psbEle->Sender_Template_Object.SrcPort ==
                        fdlist->Filter_Spec_Object.SrcPort )
                {
                        for(i=0; i<InLIST_SIZE; ++i)  {
                                if( CmpAddress( &psbEle->OutInterface_List[i],
                                                        &OI ) )
                                {
                                        AddListEle( &locList, (char*)psbEle, sizeof(PathStateBlock) );
                                }
                        }
                }
                psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 0 );
        }
        if( IsListEmpty(&locList) )
        {
                /**
                *** Build and send an error message specifying "No sender information
                **/
                        VL  RMVL printf("No sender information\n");
                return;
        }
        inx=0;
        psbEle = (PathStateBlock *)GetNextListEle( &locList, 1 );
        while( psbEle )
        {
        int flag;
                flag=OFF;
                for(i=0; i<InLIST_SIZE; ++i)  {
                        if( CmpAddress( &Refresh_PHOP_list[i],
                                                &psbEle->Previous_Hop_Address) )
                        {
                                flag=ON;
                        }
                }
                if( flag == OFF )
                        Refresh_PHOP_list[inx++] = psbEle->Previous_Hop_Address;
                psbEle = (PathStateBlock *)GetNextListEle( &locList, 0 );
        }
        rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 1 );
        activeRSB = NULL;
        while( rsbEle )
        {
```

```
        if( CmpSession( &rsbEle->Session_Object, &rmess->Session_Object) &&
            CmpAddress( &rsbEle->Next_Hop_Address,
                        &rmess->Rsvp_Hop_Object.Next_Hop_Address) &&
            CmpFilterSpec( &rsbEle->Filter_Spec_Object,
                           &fdlist->Filter_Spec_Object) )
        {
            activeRSB = rsbEle;
            break;
        }
        rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 0 );
    }
    if( activeRSB == NULL )
    {
        ResvStateBlock *trsb;

        Resv_Refresh_Needed_Flag = ON;
        /** Create a new RSB **/
        VL  RMVL printf("Create a new RSB\n");
        activeRSB = (ResvStateBlock *) MALLOC( sizeof(ResvStateBlock) );
        Memcpy( &activeRSB->Session_Object, &rmess->Session_Object,
                sizeof(SessionObj));
        activeRSB->Next_Hop_Address = rmess->Rsvp_Hop_Object.Next_Hop_Address;
        activeRSB->OI = OI;
        Memcpy( &activeRSB->Style_Object, &rmess->Style_Object,
                sizeof(StyleObj));
        Memcpy( &activeRSB->Filter_Spec_Object,
                &fdlist->Filter_Spec_Object,
                sizeof(FilterSpecObj));
        Memcpy( &activeRSB->Flowspec_Object,
                &fdlist->Flowspec_Object,
                sizeof(FlowSpecObj));
        NeworMod = ON;
        trsb = (ResvStateBlock *)AddListEle( &RSBList, (char *)activeRSB, sizeof(ResvStateBlock) );
        free(activeRSB);
        activeRSB = trsb;
    }
    else
    {
        /*  VL      RMVL printf("Found an RSB\n");  */
        if( CmpSenderTspec( &activeRSB->Flowspec_Object,
            &fdlist->Flowspec_Object ) )
        {
            Memcpy( &activeRSB->Flowspec_Object,
                    &fdlist->Flowspec_Object,
                    sizeof(FlowSpecObj));
            NeworMod = ON;
        }
        if( CmpScope( &activeRSB->Scope_Object, &rmess->Scope_Object ) )
        {
            Memcpy( &activeRSB->Scope_Object, &rmess->Scope_Object,
                    sizeof(ScopeObj) );
            NeworMod = ON;
        }
    }
    /**
    *** Start or restart the cleanup timer on the active RSB.
    **/
    Memcpy( &activeRSB->Resv_Confirm_Object, &rmess->Resv_Confirm_Object,
            sizeof(ResvConfirmObj) );
```

```
        if( NeworMod == OFF )
            continue;
        /**
        *** Call the UPDATE TRAFFIC CONTROL
        **/
        UpdateTrafficControl(activeRSB);
}
        if( Resv_Refresh_Needed_Flag == ON )
        {
            /**
            *** Send RESV REFRESH for each PHOP in Refresh_PHOP_list.
            **/
            VL  RMVL printf("Sending RESV REFRESH for each PHOP in Refresh_PHOP_list\n");
            for(i=0; i<InLIST_SIZE; ++i)  {
                if( !UndefAddress( &Refresh_PHOP_list[i] ) )
                {
                    ResvRefresh( activeRSB, Refresh_PHOP_list[i] );
                }
            }
        }
        return;
}

void
UpdateTrafficControl(ResvStateBlock *activeRSB)
{
        int Is_Biggest=OFF;
        int TC_E_Police_flag = OFF;
        int Resv_Refresh_Needed_Flag=OFF;
        int inx, i;
        int found;
        int handle;
        FlowSpecObj Tc_Flowspec;
        FlowSpecObj fwdFS;
        SenderTspecObj Path_Te;

        ResvStateBlock *rsbEle;
        PathStateBlock *psbEle;
        TrafficControlStateBlock * tcsbEle;
        TrafficControlStateBlock * tEle;
        FilterSpecObj TC_Filter_Spec[InLIST_SIZE];

        VL printf("Entered UpdateTrafficControl\n");
        Tc_Flowspec.req_bandwidth = 0;
        Tc_Flowspec.link_delay = 0;
        memset( TC_Filter_Spec, 0, InLIST_SIZE*sizeof(FilterSpecObj) );

        rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 1 );
        inx=0;
        while( rsbEle )
        {
            if( CmpSession( &rsbEle->Session_Object, &activeRSB->Session_Object) &&
                CmpFilterSpec( &rsbEle->Filter_Spec_Object,
                                &activeRSB->Filter_Spec_Object) &&
                CmpAddress(&rsbEle->OI, &activeRSB->OI) )
            {
                /* Compute the effective kernel flowspec */
                if( rsbEle->Flowspec_Object.req_bandwidth >
                    Tc_Flowspec.req_bandwidth )
```

...
...

```
        {
            Tc_Flowspec.req_bandwidth =
                rsbEle->Flowspec_Object.req_bandwidth;
        }
        if( rsbEle->Flowspec_Object.link_delay > Tc_Flowspec.link_delay )
        {
            Tc_Flowspec.link_delay = rsbEle->Flowspec_Object.link_delay;
        }
        found=OFF;
        for( i=0; i<inx; ++i )
        {
            if( CmpFilterSpec( &rsbEle->Filter_Spec_Object,
                                  &TC_Filter_Spec[i] ) )
            {
                found=ON;
                break;
            }
        }
        if( found==OFF )
        {
            TC_Filter_Spec[inx++] = rsbEle->Filter_Spec_Object;
        }
    }
    rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 0 );
}
if( activeRSB->Flowspec_Object.req_bandwidth >
    Tc_Flowspec.req_bandwidth && activeRSB->Flowspec_Object.link_delay >
    Tc_Flowspec.link_delay )
{
    Is_Biggest = ON;
}
/* Set TC_B_Police_flag */

Path_Te.req_bandwidth = 0;
Path_Te.link_delay = 0;
psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 1 );
while( psbEle )
{
    if( CmpFilterSpec( &psbEle->Sender_Template_Object,
                          &activeRSB->Filter_Spec_Object ) )
    {
        for(i=0; i<InLIST_SIZE; ++i)  {
            if( CmpAddress( &psbEle->OutInterface_List[i],
                               &activeRSB->OI ) )
            {
                if( psbEle->E_Police_Flag == ON )
                {
                    TC_E_Police_flag = ON;
                }
                Path_Te.req_bandwidth +=
                    psbEle->Sender_Tspec_Object.req_bandwidth;
                Path_Te.link_delay +=
                    psbEle->Sender_Tspec_Object.link_delay;
            }
        }
    }
    psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 0 );
}
```

...
...

```
        found=OFF;
        tcsbEle = (TrafficControlStateBlock*) GetNextListEle( &TCSBList, 1 );
        while( tcsbEle )
        {
            if( CmpSession( &tcsbEle->Session_Object,
                                &activeRSB->Session_Object ) &&
                CmpAddress( &tcsbEle->OI, &activeRSB->OI ) )
            {
                for(i=0; i<InLIST_SIZE; ++i)  {
                    if( CmpFilterSpec( &tcsbEle->Filter_Spec_Object[i],
                                        &activeRSB->Filter_Spec_Object ) )
                    {
                        found=ON;
                        break;
                    }
                }
            }

            tcsbEle = (TrafficControlStateBlock*) GetNextListEle( &TCSBList, 0 );
        }
        if( found==OFF )
        {
            /* TCSB is new */
            TrafficControlStateBlock * newTcsb;
            VL printf("TCSB is new\n");
            tcsbEle = (TrafficControlStateBlock *)
                         MALLOC(sizeof( TrafficControlStateBlock ) );
            Memcpy( &tcsbEle->Session_Object,
                        &activeRSB->Session_Object,
                        sizeof( SessionObj ) );
            CpyAddress( &tcsbEle->OI, &activeRSB->OI );
            Memcpy( &tcsbEle->TC_Flowspec,
                        &Tc_Flowspec,
                        sizeof( FlowSpecObj ) );
            Memcpy( tcsbEle->Filter_Spec_Object,
                        TC_Filter_Spec,
                        InLIST_SIZE*sizeof(FilterSpecObj) );
            Memcpy( &tcsbEle->TC_Tspec,
                        &Path_Te,
                        sizeof( SenderTspecObj ) );
            tcsbEle->E_Police_Flag = TC_E_Police_flag;
          Resv_Refresh_Needed_Flag=ON;
            /* Make call TC_AddFlowSpec */
            handle = TC_AddFlowspec( tcsbEle->OI, tcsbEle->TC_Flowspec,
                tcsbEle->TC_Tspec, &fwdFS );
            /* If this call fails send ResvErr message specifying
              "Admission Control Failed" */
            /* Otherwise */
            /* Record Rhandle and Fwd_Flowspec in the TCSB. For each
              filter_spec F in TC_Filter_Spec * call TC_AddFilter
              and record the returned Fhandle in TCSB */
            tcsbEle->Rhandle = handle;
            tcsbEle->Fwd_Flowspec = fwdFS;
            /*  handle = TC_AddFilter( tcsbEle->OI, tcsbEle->Rhandle,   */
                            /*  tcsbEle->Session_Object, tcsbEle->Filter_Spec_Object );  */
            tcsbEle->Fhandle = handle;
            tEle = AddListEle( &TCSBList, (char *)tcsbEle, sizeof(TrafficControlStateBlock) );
            free( tcsbEle );
            tcsbEle = tEle;
```

```
        }
        else
        {
            /* If TCSB is not new */
            VL printf("Found a matching TCSB\n");
            if( CmpFlowSpec( &tcsbEle->TC_Flowspec, &Tc_Flowspec ) ||
                CmpSenderTspec( &tcsbEle->TC_Tspec, &Path_Te ) ||
                tcsbEle->E_Police_Flag != TC_E_Police_flag )
            {
                if( CmpFlowSpec( &tcsbEle->TC_Flowspec, &Tc_Flowspec ) ||
                    CmpSenderTspec( &tcsbEle->TC_Tspec, &Path_Te ) )
                {
                    Resv_Refresh_Needed_Flag=ON;
                }
                /* Modify the reservations */
                /* TC_ModFlowspec */
                handle = TC_ModFlowspec( tcsbEle->OI, tcsbEle->Rhandle,
                            tcsbEle->TC_Flowspec, tcsbEle->TC_Tspec, &tcsbEle->Fwd_Flowspec );
            }
            if( memcmp( tcsbEle->Filter_Spec_Object,
                    TC_Filter_Spec,
                    InLIST_SIZE*sizeof(FilterSpecObj) ) )
            {
                /* Make an appropriate set of TC_DelFilter and TC_AddFilter
                  calls to transform the Filter_spec_list in the TCSB
                  into the new TC_Filter_Spec* */
            }
        }
        if ( Is_Biggest == ON )
        {
            tcsbEle->Resv_Confirm_Object = activeRSB->Resv_Confirm_Object;
            Resv_Refresh_Needed_Flag=ON;
        }
        else
        {
            /* Create and send a ResvConf message to the address */
            op_intrpt_schedule_remote( op_sim_time(), 9, qospf_objid );
        }
        return;
}

void
ResvRefresh( ResvStateBlock *rsbEle, IPAddress PH )
{
        int Need_Scope=OFF;
        int B_Merge=OFF;
        int i,j;
        int found;
        int data_len;
        IPAddress PHOP;

        ResvMessage outRM;
        FlowSpecObj Tc_Flowspec;
        PathStateBlock *psbEle;
        TrafficControlStateBlock * tcsbEle;
        printf("ResvRefresh: PHOP = %d.%d\n", PH.NetNum, PH.NodeNum );

        outRM.Session_Object = rsbEle->Session_Object;
        outRM.Time_Values_Object.Refresh_Period = REFRESH_PERIOD;
```

...
...

```
outRM.Rsvp_Hop_Object.Next_Hop_Address = rsbEle->Next_Hop_Address;
outRM.Rsvp_Hop_Object.Logical_Interface_Handle = rsbEle->OI;
psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 1 );
while( psbEle )
{
    PHOP = psbEle->Previous_Hop_Address;
    Tc_Flowspec.req_bandwidth = 0;
    Tc_Flowspec.link_delay = 0;
    tcsbEle = (TrafficControlStateBlock *)GetNextListEle( &TCSBList, 1 );
    while( tcsbEle )
    {
        found=OFF;
        for( i=0; i<InLIST_SIZE; ++i )
        {
            if( CmpSenderTemplate( &psbEle->Sender_Template_Object,
                        &tcsbEle->Filter_Spec_Object[i] ) )
            {
                found=ON;
                break;
            }
        }
        if( found==ON )
        {
            found=OFF;
            for( i=0; i<InLIST_SIZE; ++i )
            {
                if( CmpAddress( &psbEle->OutInterface_List[i],
                                &tcsbEle->OI ) )
                {
                    found=ON;
                    break;
                }
            }
        }
        if( found==ON )
        {
            if( CmpLocalAddress(
                &psbEle->Sender_Template_Object.SrcAddress ) )
            {
                outRM.Resv_Confirm_Object = tcsbEle->Resv_Confirm_Object;
                /* Send the message */
                break;
            }
            if( B_Merge==OFF )
            {
                /* Ignore a blockaded TCSB */
                if ( tcsbEle->Fwd_Flowspec.req_bandwidth >
                     Tc_Flowspec.req_bandwidth )
                {
                    Tc_Flowspec.req_bandwidth =
                        tcsbEle->Fwd_Flowspec.req_bandwidth;
                }
                if ( tcsbEle->Fwd_Flowspec.link_delay >
                     Tc_Flowspec.link_delay )
                {
                    Tc_Flowspec.link_delay =
                        tcsbEle->Fwd_Flowspec.link_delay;
                }
            }
```

```
                        }
                        tcsbEle = (TrafficControlStateBlock*)GetNextListEle( &TCSBList, 0 );
                }
                psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 0 );
        }
        SendResvPktToIP( &outRM, PH);
}

void
PathTearPro(PathTearMessage *pmess)
{
        int found;
        int i;
        PathStateBlock * psbEle = NULL;
        ResvStateBlock * rsbEle = NULL;

        found = OFF;
        psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 1 );
        while( psbEle )
        {
                if( CmpSession( &psbEle->Session_Object,
                                        &pmess->Session_Object ) &&
                        CmpSenderTemplate( &psbEle->Sender_Template_Object,
                                    &pmess->Sender_Descriptor.Sender_Template_Object ) )
                {
                        found = ON;
                        break;
                }
                psbEle = (PathStateBlock *)GetNextListEle( &PSBList, 0 );
        }
        if( psbEle == NULL )
        {
                return;
        }
        if( found == ON )
        {
                rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 1 );
                while( rsbEle )
                {
                        if( CmpFilterSpec( &rsbEle->Filter_Spec_Object,
                                        (FilterSpecObj *)&psbEle->Sender_Template_Object ))
                        {
                                for(i=0; i<InLIST_SIZE; ++i)
                                {
                                        if( CmpAddress( &rsbEle->OI,&psbEle->OutInterface_List[i]) )
                                        {
                                                DeleteListEle(&RSBList, (char*)rsbEle, sizeof(ResvStateBlock));
                                                /* Execute the event sequence UPDATE TRAFFIC CONTROL */
                                        }
                                }
                        }
                        rsbEle = (ResvStateBlock *)GetNextListEle( &RSBList, 0 );
                }
        }
        DeleteListEle(&PSBList, (char*)psbEle, sizeof(PathStateBlock));
        return;
}
```

```
void *
AddListEle( GenList *lst, char *Ele, int size )
{
        GenList * new=NULL;
        GenList * next=NULL;
        GenList * prev=NULL;
        char * tele=NULL;
        if(lst == NULL)
                return NULL;
        if(Ele == NULL)
                return NULL;
        new = (GenList *)MALLOC(sizeof(GenList));
        if(new == NULL)
        {
                PrintErr( ("No Memory") );
        }
        tele = (char *)MALLOC(size);
        if(tele == NULL)
        {
                PrintErr( ("No Memory") );
        }
    Memcpy( tele, Ele, size );
        next = lst;
        prev = lst;
        while(next)
        {
                prev=next;
                next=next->next;
        }
        prev->next = new;
        new->next = NULL;
        new->ele = tele;
        return new->ele;
}

void
DeleteListEle( GenList *lst, char *Ele, int size )
{
        GenList *next;
        GenList *prev;
        if(lst == NULL)
                return ;
        if(Ele == NULL)
                return ;
        prev = lst;
        next = lst;
        while(next)
        {
                if( next->ele != NULL )
                {
                        if( !memcmp( Ele, next->ele, size ) )
                        {
                                prev->next = next->next;
                                free(next->ele);
                                free(next);
                                return;
                        }
                }
```

```
                prev=next;
                next=next->next;
        }
}

char *
GetNextListEle( GenList *lst, int flag )
{
        GenList * tmp;
        char *ele;
        if(lst == NULL)
                return NULL;
        if( flag )
                lst->curr = lst->next;
        if( lst->curr == NULL )
        {
                ele = NULL;
        }
        else
        {
                tmp = lst->curr;
                lst->curr = lst->curr->next;
                ele = tmp->ele;
        }

        return ele;
}

void
ListInit(GenList* lst)
{
        lst->next=NULL;
        lst->curr=NULL;
        lst->ele=NULL;
}

int
IsListEmpty(GenList* lst)
{
        if(lst->next == NULL)
                return 1;
        return 0;
}

/*
void
Mcast_Route_Query(IPAddress sa, IPAddress iad, IPAddress da, IPAddress *outl)
{
        int i;
        int j;
        j=0;

        for(i=0;i<=NoOfLinks;++i)
        {
                if( !CmpAddress( &LocalAddress[i], &iad ) )
                {
                        CpyAddress( &outl[j++], &LocalAddress[i] );
                }
        }
```

```
	return;
}
*/

void
Mcast_Route_Query(IPAddress sa, IPAddress iad, IPAddress da, IPAddress *outl)
{
	int i;
	int j;
	mc_id *mc_table;
	j=0;

	mc_table = (mc_id *) op_ima_obj_svar_get(ip_objid,"mc_table");

	for(i=0;i<=NoOfLinks;++i)
	{
		if( mc_table[da.NodeNum].interface[i] == 1 )
		{
			CpyAddress( &outl[j++], &LocalAddress[i] );
		}
	}
	return;
}

void
SendResvPktToIP( ResvMessage * resvMsg, IPAddress PH)
{
	Packet* ResvMsgPkptr;
	Packet* ipdgramptr;
	int data_len;

	/*  VL PrintMsg(resvMsg);    */

	ResvMsgPkptr = op_pk_create_fmt("Resv_pkfmt");
	op_pk_nfd_set( ResvMsgPkptr, "ResvMsg", resvMsg,
		op_prg_mem_alloc, op_prg_mem_free, sizeof(ResvMessage) );
	op_pk_nfd_set( ResvMsgPkptr, "Msg_Type", RESV_MESSAGE );

	data_len = op_pk_total_size_get(ResvMsgPkptr);
	ipdgramptr = op_pk_create_fmt("ip_dgram");
	op_pk_nfd_set( ipdgramptr, "data", ResvMsgPkptr );
	op_pk_nfd_set( ipdgramptr, "outstrm", -1 );
	op_pk_nfd_set( ipdgramptr, "prot_id", RSVP_PROTOCOL_ID );
	op_pk_bulk_size_set( ipdgramptr, data_len*8 );
	op_pk_nfd_set( ipdgramptr, "dest_net", PH.NetNum );
	op_pk_nfd_set( ipdgramptr, "dest_node", PH.NodeNum );
	op_pk_nfd_set( ipdgramptr, "ttl", 0 );
	op_pk_nfd_set( ipdgramptr, "src_net",
			resvMsg->Rsvp_Hop_Object.Logical_Interface_Handle.NetNum );
	op_pk_nfd_set( ipdgramptr, "src_node",
			resvMsg->Rsvp_Hop_Object.Logical_Interface_Handle.NodeNum );
	op_pk_nfd_set( ipdgramptr, "orig_len", data_len );
	op_pk_nfd_set( ipdgramptr, "frag_len", data_len );
	op_pk_nfd_set( ipdgramptr, "frag", 0 );
	op_pk_send( ipdgramptr, IPSTREAM );
}

void
SendPktToIP( char * mess, int strm, int type )
```

```
{
    PathMessage *pathMsg;
    ResvMessage *resvMsg;
    PathTearMessage *pathTearMsg;
    ResvTearMessage *resvTearMsg;
    PathErrMessage *pathErrMsg;
    ResvErrMessage *resvErrMsg;
    ResvConfMessage *resvConfMsg;
    Packet* ipdgramptr;
    Packet* MsgPkt;
    int data_len;

    if( strm == UNDEFINED )
    {
        VL printf("SendPktToIP: UNDEFINED stream\n");
        return;
    }

    switch( type )
    {
        case PATH_MESSAGE :
        {
            pathMsg = (PathMessage *)mess;
            /*  pathMsg = (PathMessage *)op_prg_mem_alloc(sizeof(PathMessage));  */
            /*  pathMsg = (PathMessage *)malloc(sizeof(PathMessage));  */
            /*  memcpy( pathMsg, mess, sizeof(PathMessage) );   */
            VL PMVL PrintMsg((char*)pathMsg);

            MsgPkt = op_pk_create_fmt("Path_pkfmt");
            op_pk_nfd_set( MsgPkt, "PathMsg", pathMsg,
                          op_prg_mem_alloc, op_prg_mem_free,
                          sizeof(PathMessage) );
            op_pk_nfd_set( MsgPkt, "Msg_Type", PATH_MESSAGE );

            data_len = op_pk_total_size_get(MsgPkt);
            ipdgramptr = op_pk_create_fmt("ip_dgram");
            op_pk_nfd_set( ipdgramptr, "data", MsgPkt );
            op_pk_nfd_set( ipdgramptr, "outstrm", strm );
            op_pk_nfd_set( ipdgramptr, "prot_id", RSVP_PROTOCOL_ID );
            op_pk_bulk_size_set( ipdgramptr, data_len*8 );
            op_pk_nfd_set( ipdgramptr, "dest_net", pathMsg->Session_Object.DestAddress.NetNum );
            op_pk_nfd_set( ipdgramptr, "dest_node", pathMsg->Session_Object.DestAddress.NodeNum );
            op_pk_nfd_set( ipdgramptr, "ttl", pathMsg->common_header.Send_TTL );
            op_pk_nfd_set( ipdgramptr, "src_net", pathMsg->Rsvp_Hop_Object.Next_Hop_Address.NetNum );
            op_pk_nfd_set( ipdgramptr, "src_node", pathMsg->Rsvp_Hop_Object.Next_Hop_Address.NodeNum );
            op_pk_nfd_set( ipdgramptr, "orig_len", data_len );
            op_pk_nfd_set( ipdgramptr, "frag_len", data_len );
            op_pk_nfd_set( ipdgramptr, "frag", 0 );
            op_pk_send( ipdgramptr, IPSTREAM );

            VL PMVL  printf("Sent the PATH message from %d.%d to %d.%d\n",
                    pathMsg->Rsvp_Hop_Object.Next_Hop_Address.NetNum,
                    pathMsg->Rsvp_Hop_Object.Next_Hop_Address.NodeNum,
                    pathMsg->Session_Object.DestAddress.NetNum,
                    pathMsg->Session_Object.DestAddress.NodeNum );
            /*  op_prg_mem_free(pathMsg);  */
        }
        break;
        case RESV_MESSAGE :
```

```
				break;
			case PATH_ERROR :
				break;
			case RESV_ERROR :
				break;
			case PATH_TEAR :
				break;
			case RESV_TEAR :
				break;
			case RESV_CONF :
				break;
		}
	}

/***
****  Utility Functions
***/


FILE *
GetFilePtr( void )
{
	static FILE *fp;

	if( strcmp( LogFileName, "stdout" ) )
	{
		if( (fp=fopen( LogFileName, "a" )) == NULL )
		{
			/*  VL	PrintErr(("Could not open file :%s", LogFileName));  */
			return NULL;
		}
	}
	else
	{
		fp=stdout;
	}
	return fp;
}

void
PrintCommHdr( CommonHdr ch )
{
	FILE *fp;
	fp = GetFilePtr();
	fprintf(fp, "CommonHdr: Msg_Type = %d\n", ch.Msg_Type);
	fprintf(fp, "CommonHdr: Send_TTL = %d\n", ch.Send_TTL);
	if( fp != stdout )
		fclose(fp);
}


void
PrintSess( SessionObj sh )
{
	FILE *fp;
	fp = GetFilePtr();
	fprintf(fp, "SessionObj: DestAddress = %d.%d\n", sh.DestAddress.NetNum,
			sh.DestAddress.NodeNum);
	fprintf(fp, "SessionObj: DestPort = %d\n", sh.DestPort );
```

```
		if( fp != stdout )
			fclose(fp);
	}

	void
	PrintRsvpHop( RsvpHopObj ch )
	{
		FILE *fp;
		fp = GetFilePtr();
		fprintf(fp, "RsvpHopObj: Next_Hop_Address = %d.%d\n",
				ch.Next_Hop_Address.NetNum, ch.Next_Hop_Address.NodeNum);
		fprintf(fp, "RsvpHopObj: Logical_Interface_Handle = %d.%d\n",
				ch.Logical_Interface_Handle.NetNum,
				ch.Logical_Interface_Handle.NodeNum);
		if( fp != stdout )
			fclose(fp);
	}

	void
	PrintTime( TimeValObj ch )
	{
		FILE *fp;
		fp = GetFilePtr();
		fprintf(fp, "TimeValObj: Refresh_Period = %d\n", ch.Refresh_Period);
		if( fp != stdout )
			fclose(fp);
	}

	void
	PrintSendTemp( SenderTemplateObj ch )
	{
		FILE *fp;
		fp = GetFilePtr();
		fprintf(fp, "SenderTemplateObj: SrcAddress = %d.%d\n",
				ch.SrcAddress.NetNum, ch.SrcAddress.NodeNum);
		fprintf(fp, "SenderTemplateObj: SrcPort = %d\n", ch.SrcPort );
		if( fp != stdout )
			fclose(fp);
	}

	void
	PrintSendTSpec( SenderTspecObj ch )
	{
		FILE *fp;
		fp = GetFilePtr();
		fprintf(fp, "SenderTspecObj: req_bandwidth = %d\n", ch.req_bandwidth);
		fprintf(fp, "SenderTspecObj: link_delay = %d\n", ch.link_delay);
		if( fp != stdout )
			fclose(fp);
	}

	void
	PrintFilter( FilterSpecObj ch )
	{
		FILE *fp;
		fp = GetFilePtr();
		fprintf(fp, "FilterSpecObj: SrcAddress = %d.%d\n",
				ch.SrcAddress.NetNum, ch.SrcAddress.NodeNum);
		fprintf(fp, "FilterSpecObj: SrcPort = %d\n", ch.SrcPort );
```

```
	if( fp != stdout )
		fclose(fp);
}

void
PrintFlow( FlowSpecObj ch )
{
	FILE *fp;
	fp = GetFilePtr();
	fprintf(fp, "FlowSpecObj: req_bandwidth = %d\n", ch.req_bandwidth);
	fprintf(fp, "FlowSpecObj: link_delay = %d\n", ch.link_delay);
	if( fp != stdout )
		fclose(fp);
}

void
PrintScope( ScopeObj ch )
{
	FILE *fp;
	fp = GetFilePtr();
	fprintf(fp, "ScopeObj: SrcAddress = %d.%d\n",
				ch.SrcAddress.NetNum, ch.SrcAddress.NodeNum);
	if( fp != stdout )
		fclose(fp);
}

void
PrintStyle( StyleObj ch )
{
	FILE *fp;
	fp = GetFilePtr();
	fprintf(fp, "StyleObj: Resv_Style = %d\n", ch.Resv_Style);
	if( fp != stdout )
		fclose(fp);
}

void
PrintConfirm( ResvConfirmObj ch )
{
	FILE *fp;
	fp = GetFilePtr();
	fprintf(fp, "ResvConfirmObj: Receiver_Address = %d.%d\n",
				ch.Receiver_Address.NetNum, ch.Receiver_Address.NodeNum);
	if( fp != stdout )
		fclose(fp);
}

void
PrintLine( char * ch )
{
	FILE *fp;
	fp = GetFilePtr();
	fprintf(fp, "%s\n", ch);
	if( fp != stdout )
		fclose(fp);
}

void
PrintMsg( char * MsgPk )
```

```
{
	int i;
	CommonHdr *cHdr;
	SessionObj sessObj;
	IPAddress ipa;
	SessInfo sessInfo;
	RsvpHopObj rsvpHObj;
	TimeValObj timeObj;
	SenderTemplateObj sendTempObj;
	SenderTspecObj sendTspecObj;
	FlowSpecObj flowSpecObj;
	FilterSpecObj filterSpecObj;
	StyleObj styleObj;
	ScopeObj scopeObj;
	ErrorSpecObj errorSpecObj;
	PathMessage *pathMsg;
	ResvMessage *resvMsg;
	PathTearMessage *pathTearMsg;
	ResvTearMessage *resvTearMsg;
	PathErrMessage *pathErrMsg;
	ResvErrMessage *resvErrMsg;
	ResvConfMessage *resvConfMsg;

	cHdr = (CommonHdr *)MsgPk;

	switch( cHdr->Msg_Type )
	{
		case PATH_MESSAGE :
			PrintLine("\n----------------Path Message---------------------------");
			pathMsg = (PathMessage *)MsgPk;
			PrintCommHdr( pathMsg->common_header );
			PrintSess( pathMsg->Session_Object );
			PrintRsvpHop( pathMsg->Rsvp_Hop_Object );
			PrintTime( pathMsg->Time_Values_Object );
			PrintSendTemp(pathMsg->Sender_Descriptor.Sender_Template_Object);
			PrintSendTSpec(pathMsg->Sender_Descriptor.Sender_Tspec_Object);
		break;
		case RESV_MESSAGE :
			PrintLine("\n----------------Resv Message---------------------------");
			resvMsg = (ResvMessage *)MsgPk;
			PrintCommHdr( resvMsg->common_header );
			PrintSess( resvMsg->Session_Object );
			PrintRsvpHop( resvMsg->Rsvp_Hop_Object );
			PrintTime( resvMsg->Time_Values_Object );
			PrintConfirm( resvMsg->Resv_Confirm_Object );
			PrintScope( resvMsg->Scope_Object );
			PrintStyle( resvMsg->Style_Object );
			for( i=0; i<InLIST_SIZE; ++i )
			{
			PrintFlow( resvMsg->flow_descriptor_list[i].Flowspec_Object );
			PrintFilter(resvMsg->flow_descriptor_list[i].Filter_Spec_Object);
			}
		break;
		case PATH_TEAR :
			PrintLine("\n----------------Path Tear Message---------------------------");
			pathTearMsg = (PathTearMessage *)MsgPk;
			PrintCommHdr( pathTearMsg->common_header );
			PrintSess( pathTearMsg->Session_Object );
			PrintRsvpHop( pathTearMsg->Rsvp_Hop_Object );
```

```
                    PrintSendTemp(pathTearMsg->Sender_Descriptor.Sender_Template_Object);
                    PrintSendTSpec(pathTearMsg->Sender_Descriptor.Sender_Tspec_Object);
                break;
                case RESV_TEAR :
                    PrintLine("\n----------------Resv Tear Message---------------------------");
                    resvTearMsg = (ResvTearMessage *)MsgPk;
                    PrintCommHdr( resvTearMsg->common_header );
                    PrintSess( resvTearMsg->Session_Object );
                    PrintRsvpHop( resvTearMsg->Rsvp_Hop_Object );
                    PrintScope( resvTearMsg->Scope_Object );
                    PrintStyle( resvTearMsg->Style_Object );
                    PrintFlow( resvTearMsg->flow_descriptor_list.Flowspec_Object );
                    PrintFilter(resvTearMsg->flow_descriptor_list.Filter_Spec_Object);
                break;
                case PATH_ERROR :
                    PrintLine("\n----------------Path Error Message---------------------------");
                    pathErrMsg = (PathErrMessage *)MsgPk;
                    PrintCommHdr( pathErrMsg->common_header );
                    PrintSess( pathErrMsg->Session_Object );
                    PrintSendTemp(pathErrMsg->Sender_Descriptor.Sender_Template_Object);
                    PrintSendTSpec(pathErrMsg->Sender_Descriptor.Sender_Tspec_Object);
                break;
                case RESV_ERROR :
                    PrintLine("\n----------------Resv Error Message---------------------------");
                    resvErrMsg = (ResvErrMessage *)MsgPk;
                    PrintCommHdr( resvErrMsg->common_header );
                    PrintSess( resvErrMsg->Session_Object );
                    PrintScope( resvErrMsg->Scope_Object );
                    PrintStyle( resvErrMsg->Style_Object );
                break;
                case RESV_CONF :
                    PrintLine("\n----------------Resv Conf Message---------------------------");
                    resvConfMsg = (ResvConfMessage *)MsgPk;
                    PrintCommHdr( resvConfMsg->common_header );
                    PrintSess( resvConfMsg->Session_Object );
                    PrintStyle( resvConfMsg->Style_Object );
                    PrintFlow( resvConfMsg->flow_descriptor_list.Flowspec_Object );
                    PrintFilter(resvConfMsg->flow_descriptor_list.Filter_Spec_Object);
                break;
                default :
                    PrintLine("\n----------------Unrecognized Message---------------------------");
                    VL  printf("Could not Recognize the Message\n");
                break;
            }
        PrintLine("----------------------------------------------------");
}

void
PrintPSB( PathStateBlock * PathStateBlk )
{
        int i;
        char mess[500];

        PrintLine("----------------Path State Block---------------------");
        PrintSess( PathStateBlk->Session_Object);
        PrintSendTemp(PathStateBlk->Sender_Template_Object);
        PrintSendTSpec(PathStateBlk->Sender_Tspec_Object);
        sprintf(mess, "Previous_Hop_Address : %d.%d\n",
                    PathStateBlk->Previous_Hop_Address.NetNum,
```

```
                PathStateBlk->Previous_Hop_Address.NodeNum );
        PrintLine( mess );
        sprintf(mess, "Logical Interface Handle : %d.%d\n",
                PathStateBlk->LIH.NetNum, PathStateBlk->LIH.NodeNum );
        PrintLine( mess );
        sprintf(mess, "Remaining IP TTL : %d\n", PathStateBlk->rem_IP_TTL );
        PrintLine( mess );
        sprintf(mess, "Local_Only_Flag : %d\n", PathStateBlk->Local_Only_Flag );
        PrintLine( mess );
        for(i = 0; i<InLIST_SIZE; i++ )
        {
            sprintf(mess, "OutInterface_List[%d] : %d.%d\n", i,
                    PathStateBlk->OutInterface_List[i].NetNum,
                    PathStateBlk->OutInterface_List[i].NodeNum );
            PrintLine( mess );

        }
        sprintf(mess, "IncInterface : %d.%d\n",
                PathStateBlk->IncInterface.NetNum,
                PathStateBlk->IncInterface.NodeNum );
        PrintLine( mess );
        PrintFlow(PathStateBlk->Qb);
        PrintLine("-------------------------------------------------------");
}

int
GetFwdFS(IPAddress oi, FlowSpecObj *fwdFS)
{
        int i;
        fwdFS->req_bandwidth = 0;
        fwdFS->link_delay = 0;
        for( i=0; i<TABLE_SIZE; ++i )
        {
            if( FlowTable[i].isfull )
            {
                if( CmpAddress( &FlowTable[i].OI, &oi ) )
                {
                    fwdFS->req_bandwidth += FlowTable[i].TC_Flowspec.req_bandwidth;
                    fwdFS->link_delay += FlowTable[i].TC_Flowspec.link_delay;
                }
            }
        }
        return 0;
}

int
TC_AddFlowspec( IPAddress OI, FlowSpecObj fs, SenderTspecObj ts,
                    FlowSpecObj *fwdFS)
{
        int i;
        int Rhandle;
        static int initFlag=0;
        if(initFlag==0)
        {
            initFlag++;
            memset( FlowTable, 0, sizeof(RHandleType)*TABLE_SIZE );
        }
        for( i=0; i<TABLE_SIZE; ++i )
        {
```

```
			if( FlowTable[i].isfull == 0 )
			{
				FlowTable[i].isfull = 1;
				Rhandle = i;
				break;
			}
	}
	FlowTable[Rhandle].OI = OI;
	FlowTable[Rhandle].TC_Flowspec = fs;
	FlowTable[Rhandle].Path_Te = ts;
	if( GetFwdFS(OI, fwdFS) == -1 )
	{
		memset( &FlowTable[Rhandle], 0, sizeof(RHandleType) );
		fwdFS=NULL;
		return -1;
	}
	return(Rhandle);
}

int
TC_ModFlowspec( IPAddress OI, int Rhandle, FlowSpecObj fs, SenderTspecObj ts,
				FlowSpecObj *fwdFS)
{
	fwdFS=NULL;
	if( Rhandle < 0 )
		return -1;
	if( FlowTable[Rhandle].isfull == 0 )
		return -1;
	FlowTable[Rhandle].OI = OI;
	FlowTable[Rhandle].TC_Flowspec = fs;
	FlowTable[Rhandle].Path_Te = ts;
	if( GetFwdFS(OI, fwdFS) == -1 )
	{
		memset( &FlowTable[Rhandle], 0, sizeof(RHandleType) );
		fwdFS=NULL;
		return -1;
	}
	return(Rhandle);
}

int
TC_DelFlowspec( IPAddress OI, int Rhandle )
{
	if( Rhandle < 0 )
		return -1;
	if( FlowTable[Rhandle].isfull == 0 )
		return -1;
	memset( &FlowTable[Rhandle], 0, sizeof(RHandleType) );
}


int
TC_AddFilter( IPAddress OI, int Rhandle, SessionObj so, FilterSpecObj filter)
{
	int i;
	int Fhandle;
	static int initFlag=0;
	if(initFlag==0)
	{
```

...
...

```
        initFlag++;
        memset( FilterTable, 0, sizeof(FHandleType)*TABLE_SIZE );
    }
    for( i=0; i<TABLE_SIZE; ++i )
    {
        if( FilterTable[i].isfull == 0 )
        {
            FilterTable[i].isfull = 1;
            Fhandle = i;
            break;
        }
    }
    FilterTable[Fhandle].OI = OI;
    FilterTable[Fhandle].Rhandle = Rhandle;
    FilterTable[Fhandle].session = so;
    FilterTable[Fhandle].FilterSpecObject = filter;
    return(Fhandle);
}

int
TC_DelFilter( IPAddress OI, int Fhandle )
{
    if( Fhandle < 0 )
        return -1;
    if( FilterTable[Fhandle].isfull == 0 )
        return -1;
    memset( &FilterTable[Fhandle], 0, sizeof(FHandleType) );
}
```

forced state **init**			
attribute	*value*	*type*	*default value*
name	init	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **init**

```
RouterInitialize();
```

transition **init -> idle**			
attribute	*value*	*type*	*default value*
name	tr_3	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **idle**			
attribute	*value*	*type*	*default value*
name	idle	string	st

enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	unforced	toggle	unforced

transition **idle -> arr**

attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition	ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **arr**

attribute	*value*	*type*	*default value*
name	arr	string	st
enter execs	(See below.)	textlist	(See below.)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

enter execs **arr**

```
Packet *ReceivedPkptr;
Packet *ReceivedRsvpPkptr;
Packet *tmp_pkptr;
PathMessage * pm;
ResvMessage * rm;
int ReceivedMsgType;

in_strm = op_intrpt_strm();

ReceivedPkptr = op_pk_get( in_strm );

op_pk_nfd_get( ReceivedPkptr, "strm", &ip_strm );
op_pk_nfd_get( ReceivedPkptr, "prot_id", &protocolid );
op_pk_nfd_get( ReceivedPkptr, "src_net", &obt_pk_srcnet );
op_pk_nfd_get( ReceivedPkptr, "src_node", &obt_pk_srcnode );

printf(                                                    stream %d with
        obt_pk_srcnet, obt_pk_srcnode, ip_strm, protocolid );

switch(protocolid)
{
    case RSVP_PROTOCOL_ID  :
        tmp_pkptr = op_pk_copy( ReceivedPkptr );
        op_pk_nfd_get(ReceivedPkptr, "data", &ReceivedRsvpPkptr);
        op_pk_nfd_get(ReceivedRsvpPkptr, "Msg_Type", &ReceivedMsgType);

        switch( ReceivedMsgType )
        {
            case PATH_MESSAGE:
                VL  printf("-------------------- Processing Path Message --------------------\n");
                VL  printf("(%d.%d)Received a Path Message from %d.%d \n", LocalAddress[0].NetNum,
                            LocalAddress[0].NodeNum, obt_pk_srcnet, obt_pk_srcnode );
                pm = (PathMessage *)op_prg_mem_alloc( sizeof(PathMessage) );
                memset( pm, 0, sizeof(PathMessage) );
```

```
                    op_pk_nfd_get(ReceivedRsvpPkptr, "PathMsg", &pm );

                    if( Indep_Rou )
                        ip_strm--;
                    PathMsgPro( pm, LocalAddress[ip_strm] );
                    /*  op_pk_send( tmp_pkptr, QOSPF_STREAM );      */
                    /*  VL      printf("Sent the received Path message to the QOSPF Daemon\n");  */
                    VL  printf("--------------------- End Processing Path Message ---------------------\n")
                break;
                case RESV_MESSAGE:
                    VL  printf("--------------------- Processing Resv Message ---------------------\n");
                    VL  printf("(%d.%d)Received a Resv Message from %d.%d \n", LocalAddress[0].NetNum,
                                LocalAddress[0].NodeNum, obt_pk_srcnet, obt_pk_srcnode );
                    rm = (ResvMessage *)op_prg_mem_alloc( sizeof(ResvMessage) );
                    memset( rm, 0, sizeof(ResvMessage) );

                    op_pk_nfd_get(ReceivedRsvpPkptr, "ResvMsg", &rm );
                    ResvMsgPro( rm );
                    /*  op_pk_send( tmp_pkptr, QOSPF_STREAM );      */
                    /*  VL      printf("Sent the received Resv message to the QOSPF Daemon\n");  */
                    VL  printf("--------------------- End Processing Resv Message ---------------------\n")
                break;
                case PATH_ERROR:
                break;
                case RESV_ERROR:
                break;
                case PATH_TEAR:
                    PathTearPro( (PathTearMessage *)ReceivedRsvpPkptr );
                break;
                case RESV_TEAR:
                break;
                case RESV_CONF:
                break;
            }
        break;
}

/*  op_pk_destroy( ReceivedPkptr );    */
```

transition **arr -> pathmsg**			
attribute	*value*	*type*	*default value*
name	tr_5	string	tr
condition	send_pathmsg	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **arr -> resvmsg**			
attribute	*value*	*type*	*default value*
name	tr_6	string	tr
condition	send_resvmsg	string	
executive		string	

color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **arr -> ptearmsg**			
attribute	*value*	*type*	*default value*
name	tr_7	string	tr
condition	send_pathtearmsg	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **arr -> rtearmsg**			
attribute	*value*	*type*	*default value*
name	tr_9	string	tr
condition	send_resvtearmsg	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **arr -> rconfmsg**			
attribute	*value*	*type*	*default value*
name	tr_10	string	tr
condition	send_resvconf	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **arr -> idle**			
attribute	*value*	*type*	*default value*
name	tr_20	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

forced state **pathmsg**			
attribute	*value*	*type*	*default value*
name	pathmsg	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

transition **pathmsg -> idle**			
attribute	*value*	*type*	*default value*
name	tr_11	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

forced state **resvmsg**			
attribute	*value*	*type*	*default value*
name	resvmsg	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

transition **resvmsg -> idle**			
attribute	*value*	*type*	*default value*
name	tr_13	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

forced state **ptearmsg**			
attribute	*value*	*type*	*default value*
name	ptearmsg	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

transition **ptearmsg -> idle**			
attribute	*value*	*type*	*default value*
name	tr_14	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

forced state **rtearmsg**			
attribute	*value*	*type*	*default value*
name	rtearmsg	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

transition **rtearmsg -> idle**			
attribute	*value*	*type*	*default value*
name	tr_15	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

...
...

forced state **rconfmsg**			
attribute	*value*	*type*	*default value*
name	rconfmsg	string	st
enter execs	(empty)	textlist	(empty)
exit execs	(empty)	textlist	(empty)
status	forced	toggle	unforced

transition **rconfmsg -> idle**			
attribute	*value*	*type*	*default value*
name	tr_16	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline