...
...

Process Model Attributes

attribute	*value*	*type*	*default value*
AreaId	promoted	integer	0
Link1Cost	promoted	integer	0
Link2Cost	promoted	integer	0
Link3Cost	promoted	integer	0
Link4Cost	promoted	integer	0
AreaBorderRouter	promoted	integer	0
DesignatedRouter	promoted	integer	0
IsRouter	promoted	integer	0
Link5Cost	promoted	integer	0
Link6Cost	promoted	integer	0
Link7Cost	promoted	integer	0
Link8Cost	promoted	integer	0
Link9Cost	promoted	integer	0

Header Block

```
#include <string.h>
#include <sys/time.h>

#define ARRIVAL        op_intrpt_type() == OPC_INTRPT_STRM
#define END_SIM            op_intrpt_type () ==  OPC_INTRPT_ENDSIM
#define send_ospf_lsa  op_intrpt_type() == (OPC_INTRPT_SELF)&&(op_intrpt_code()==0)
#define send_qospf_lsa op_intrpt_type() == (OPC_INTRPT_SELF)&&(op_intrpt_code()==1)||send_reslsa||send_rralsa
#define send_mospf_lsa (op_intrpt_type() == OPC_INTRPT_SELF && op_intrpt_code()==3)||(op_intrpt_type() == OPC_INTR
#define do_mospf_calc   (op_intrpt_type() == OPC_INTRPT_REMOTE && op_intrpt_code()== 7)
#define do_qospf_calc   (op_intrpt_type() == OPC_INTRPT_REMOTE && op_intrpt_code()== 8)||(op_intrpt_type() == OPC_I
#define hellopk_send   op_intrpt_type() == (OPC_INTRPT_SELF)&&(op_intrpt_code()==2)||send_hellopk
#define send_lsreq     op_intrpt_type() == (OPC_INTRPT_SELF)&&(op_intrpt_code()==4)||send_lsreq_pk

#define LSA_OUT_STAT            0
#define TOTAL_LSA_OUT_STAT  1

/**
*** OSPF Architectural Constants
**/
int MinLSInterval         = 5;
int MaxAge               = 3600;
int MaxAgeDiff           = 900;

double OspfLsaRefreshTime  = 20;
double MospfLsaRefreshTime = 20;
double HelloPkIntvl        = 20;
double QospfLsaRefreshTime = 20;
double LsaReqTime          = 20;

/**
*** Link Attributes
**/
#define LINKBANDWIDTH               256000
#define LINKDELAY                   (1/256000)
#define LINKBUFFERSIZE              100000

#define ER_IS_USED
#define ROUTE_PINNING_IS_USED   1
```

```
#define LSInfinity                  1000000


/**
*** LSA Types and Packet types defintions
**/

#define RSVP_PROTOCOL_ID            46
#define QOSPF_PROTOCOL_ID           89

#define OspfPk                      89
#define RsvpPk                      46
#define VERSION                     1
#define HELLOPK                     1
#define DDPK                        2
#define LSREQPK                     3
#define OSPFLSUPDATEPK              4
#define QOSPFLSUPDATEPK             7
#define MOSPFLSUPDATEPK             8
#define LSACKPK                     5

#define ROUTERLSA                   1
#define NETWORKLSA                  2
#define SUMMARYLSA                  3
#define RRALSA                      15
#define RESLSA                      16
#define GRPMBRSHIPLSA               6
#define ILNone                      21

#define MAX_NO_NEIGHBORS            10
#define MAX_NO_SPROUTERS            100
#define MAX_NO_LSAS                 50
#define MAX_NO_RRA_LSAS             50
#define MAX_NO_AREAS                5
#define MAX_NO_GRPS             10
#define MAX_NO_NETWORKS             110
#define MaxNoNetsInArea         12
#define MaxNoRoutersInArea      8
#define MAX_INIT_TIME               10
#define MAX_REFRESH_TIME            5
#define MAX_NO_LINKS                10
#define Router_Start_Index      60

#define OFF                         0
#define ON                          1
#define NONE                        0
#define TRUE                        1
#define FALSE                       0
#define UNDEFINED                   -1

#define VL                          if(ON)
#define DL                          if(OFF)
#define ILDL                        if(OFF)
#define OPDL                        if(OFF)
#define QPDL                        if(OFF)
#define MPDL                        if(OFF)
#define HPDL                        if(OFF)
#define SPFDL                       if(OFF)
```

```
#define MCSPFDL                 if(OFF)
#define QSPFDL                  if(OFF)

#define Tracking_Interval       0.1

#define BROADCASTLOCAL          1
#define ABRONLY                 2
#define RSVPSTREAM              2
#define IPSTREAM                0
#define IGMPSTREAM              1

#define DEF_STREAM              110

#define AllSPFRoutersNetNum     400
#define AllSPFRoutersNodeNum    5
#define AllDRoutersNetNum       400
#define AllDRoutersNodeNum      6

#define ROUTER                  1
#define TRANSITNET              2
#define STUBNET                 3

enum Flodding_Scope {AreaLocal = 0, ABR_Only = 1};

/**
*** IP Address Data Structure
**/
struct ipAddress{
        int NetNum;
        int NodeNum;
};
typedef struct ipAddress IPAddress;

typedef struct{
    IPAddress SenderAddr;
    IPAddress DestAddr;
    int SenderPort;
    double Bandwidth;
    double Delay;
}sd;


/**
*** Structures for the list manipulation
**/
#ifndef  _LIST_H_
#define  _LIST_H_

#define PrintErr(x)   printf("%s : Filename %s : Line NO %d:\n",\
                        x, __FILE__, __LINE__);


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

typedef struct List ListL;
```

```
#endif

typedef struct {
	int Version;
	int Type;
	int PacketLength;
	IPAddress ThisRouterId;
	int ThisAreaId;
	int CheckSum;
	int AuType;
	double Authentication;
}ospfpackethdr;

typedef struct {
	int LsAge;
	int Options;
	int LSA_type;
	IPAddress LinkStateId;
	IPAddress AdvRouter;
	int ThisAreaId;
	double LSSeqNo;
	double LSChecksum;
	double Length;
	int rtype;
	int No_of_Links;
}reslsaheader;

typedef struct {
	int LsAge;
	int Options;
	int LSA_type;
	int OpaqueType;
	int OpaqueId;
	IPAddress AdvRouter;
	int ThisAreaId;
	double LSSeqNo;
	double LSChecksum;
	double Length;
	int FloddingScope;
}opaquelsaheader ;

typedef struct {
	int LsAge;
	int LsType;
	IPAddress LinkStateId;
	IPAddress AdvRouter;
	int ThisAreaId;
	double LSSeqNo;
	double LSChecksum;
	double Length;
}linkstateheader;

typedef struct {
	IPAddress destination_Addr;
	IPAddress Source_Addr;
	int src_prefix_length;
	int adjust_offset;
}EraHeader ;
```

```
/**
***  QOSPF Info. Structures
**/
typedef struct{
	int Types;
	double Delay;
	double Bandwidth;
	double BufferSize;
	double ReservedBandwidth;
	double ReservedBufferSize;
	double ReservedDelay;
	int IsNghborRou;
	int NghborNetNum;
	IPAddress IntfIPAddr;
	IPAddress NeighborRtrId;
	int CostMetric;
}Linkinfo;

typedef struct {
	int TOS;
	double metric;
}tosinfo;

typedef struct {
	IPAddress LinkId;
	IPAddress LinkData;
	int Type;
	int NoOfTOS;
	int TOSMetric;
	tosinfo Tos_Info[5];
}linkinfo;

typedef struct {
	IPAddress LinkId;
	IPAddress LinkData;
	int Type;
	int TOSMetric;
	double LinkDelay;
	double Bandwidth;
	double BufferSize;
}reslinkinfo ;

typedef struct {
	IPAddress LinkId;
	IPAddress LinkData;
	int Type;
	int PinFlag;
	double ReservedBandwidth;
	double ReservedBufferSize;
}linkresvinfo ;

typedef struct {
	IPAddress router_id;
	double delay;
}AbrRouterInfo ;

typedef struct {
	int incoming_intf_type;
	int MOSPF_IL_type;
```

```
        int MOSPF_Init_case;
        int No_Of_outgoing_Intf;
        IPAddress incoming_intf_addr;
        int outgoing_intf_type;
        int child_offset;
        IPAddress outgoing_intf_addr;
}EraBody ;

typedef struct {
        struct Opaque_LSA_Header;
        struct ERA_Header;
        EraBody ERA_Body[100];
}EraSubTree ;

typedef struct{
    int VertexType;
    IPAddress VertexId;
}vertexinfo;

typedef struct{
        int interface[MAX_NO_LINKS];
}mctable;

/**
*** QOSPF Link State Advertisements
**/
typedef struct {
        linkstateheader LinkStateHeader;
        int bit_b;
        int bit_e;
        int No_of_Links;
        linkinfo Link_Info[MAX_NO_LINKS];
}RouterLinkAdv;

typedef struct {
        linkstateheader LinkStateHeader;
        int ThisNetworkMask;
        IPAddress AttachedRouter[MAX_NO_LINKS];
}NetworkLinkAdv;

typedef struct {
        linkstateheader LinkStateHeader;
        int ThisNetworkMask;
        tosinfo Tos_Info[5];
}SummaryLinkAdv;

typedef struct {
    linkstateheader LinkStateHeader;
    vertexinfo VertexInfo[MAX_NO_LINKS];
}grpmbrlsa;

typedef struct {
        reslsaheader ResLsaHeader;
        reslinkinfo  ResLinkInfo[MAX_NO_LINKS];
}reslsa;

typedef struct {
        opaquelsaheader OpaqueLsaHeader;
        IPAddress DestAddr;
```

```
	IPAddress SourceAddr;
	int src_prefix_len;
	int dst_prefix_len;
	int No_of_Links;
	linkresvinfo LinkResvInfo[MAX_NO_LINKS];
}rralsa;

typedef struct {
	long LsType;
	long LinkStateId;
	IPAddress AdvRouter;
	long LsSeqNo;
	short LsCheckSum;
	short LsAge;
}ddlsadv;

typedef struct {
	long LsType;
	long LinkStateId;
	IPAddress AdvRouter;
}lsreqpklsadv;

/**
*** QOSPF Packet Structures
**/

typedef struct {
	ospfpackethdr OspfPkHeader;
	int ThisNetworkMask;
	int Deadint;
	int Helloint;
	int ThisRtrPri;
	int Indep_Router;
	IPAddress DesgRouter;
	IPAddress BackupDesgRouter;
	IPAddress RtrsFrmWhichHelloRec[MAX_NO_NETWORKS];
}hellopacket;

typedef struct {
	ospfpackethdr OspfPkHeader;
	lsreqpklsadv LSReqPkLsAdv[MAX_NO_LINKS];
}LSReqPk;

typedef struct {
	ospfpackethdr OspfPkHeader;
	int IBit;
	int Mbit;
	int MSbit;
	int DDSeqNo;
	ddlsadv DDLsAdv[MAX_NO_LINKS];
}ddpk;

typedef struct {
	ospfpackethdr OspfPkHeader;
	int NoOfResAdvs;
	int NoOfRraAdvs;
	reslsa		ResLinkStateAdv[MAX_NO_LSAS];
	rralsa		RraLinkStateAdv[MAX_NO_LSAS];
}QospfLSUpdatePk;
```

```
typedef struct {
	ospfpackethdr OspfPkHeader;
	int NoOfRouterAdvs;
	int NoOfNetworkAdvs;
	int NoOfSummaryAdvs;
	RouterLinkAdv   RouterLinkStateAdv[MAX_NO_LSAS];
	NetworkLinkAdv  NetworkLinkStateAdv[MAX_NO_LSAS];
	SummaryLinkAdv  SummaryLinkStateAdv[MAX_NO_LSAS];
}OspfLSUpdatePk;

typedef struct {
	ospfpackethdr OspfPkHeader;
	int NoOfGrpMbrAdvs;
	grpmbrlsa       GrpmbrLinkStateAdv[MAX_NO_LSAS];
	/*  grpmbrlsa       GrpmbrLinkStateAdv[MAX_NO_GRPS][MAX_NO_LSAS];   */
}MospfLSUpdatePk;

/**
*** STRUCTURES THAT ARE NEEDED TO BE MAINTAINED ON EACH ROUTER
**/
/**
*** Routing Table structure
**/
typedef struct {
	int DestinationType;
	IPAddress DestinationId;
	int AddressMask;
	int TypeOfService;
	int AreaId;
	int PathType;
	double cost;
	int ExternalMetricType;
	int Type2Cost;
	IPAddress NextHop;
	IPAddress AdvRouter;
}RoutingTable_Struct;

/**
*** Vertex Structure
**/
typedef struct{
   IPAddress VertexId;
   int DistanceToRoot;
   IPAddress NextHopsList[MAX_NO_NEIGHBORS];
}Vertex_Struct;

/**
*** Shortest Path tree stucture
**/
typedef struct {
	Vertex_Struct SPVertex[MAX_NO_SPROUTERS];
}ShortestPathTree_Struct;

/**
*** Address Ranges structure
**/
typedef struct {
	int address;
```

...
...

```
        int mask;
}Address_Range_Struct;

/**
*** Area Data Structure
**/
typedef struct {
        int AreaId;
        Address_Range_Struct AddressRanges[MAX_NO_LSAS];
        int RouterInterfaces;
        rralsa          RRALsaList[MAX_NO_LSAS];
        grpmbrlsa       GrpMbrLsaList[MAX_NO_GRPS][MAX_NO_LSAS];
        RouterLinkAdv   RouterLsaList[MAX_NO_LSAS];
        NetworkLinkAdv  NetworkLsaList[MAX_NO_LSAS];
        SummaryLinkAdv  SummaryLsaList[MAX_NO_LSAS];
        reslsa          ResourceLsaList[MAX_NO_LSAS];
        ShortestPathTree_Struct ShortestPathTree;
        ShortestPathTree_Struct MospfShortestPathTree[MAX_NO_GRPS];
    ShortestPathTree_Struct QospfShortestPathTree[MAX_NO_GRPS];
}Area_Struct;

/**
*** Interface Data Structure
**/
typedef struct {
        int type;
        int state;
        IPAddress IPIntfAddr;
        int IPIntfMask;
        int AreaId;
        int HelloInterval;
        int RouterDeadInterval;
        int InfTransDelay;
        int RouterPriority;
        IPAddress NeighboringRouters[MAX_NO_NEIGHBORS];
        IPAddress DesignatedRouter;
        IPAddress BackupDesgRouter;
        int IntfOutputCosts[MAX_NO_LINKS];
        int RxmtInterval;
} Interface_Data_Str;

/**
*** Neighbor Data Structure
**/
typedef struct{
        int state;
        int InactivityTimer;
        int Master;
        int Slave;
        int SequenceNo;
        IPAddress NeighborRouterId;
        int NeighborPriority;
                int IsIndepRou;
        IPAddress NeighborIPAddr;
        IPAddress NghbrDesgRouter;
        IPAddress NghbrBackupDesgRouter;
        ListL LinkStateRetransmissionList;
        ListL DatabaseSummaryList;
        ListL LinkStateRequestList;
```

```
} Neighbor_Data_Str;


typedef struct{
        IPAddress VertexIPAddr;
        int           DistanceToVertex;
        IPAddress NextHops[MAX_NO_NEIGHBORS];
} CandidateStruct;

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

typedef struct{
    int RegNetNum[MAX_NO_NETWORKS];
}regmcnet;

/*  #ifdef IS_AREA_BORDER_ROUTER  */
/*     int NoOfVirtualLinks; */                /* The virtual links configured having this*/
                                                    /* Router as an endpoint                  */
/*     int VirtualLinks[MAX_NO_LINKS]; */   /* This is Router Id of the other endpoint */
/*  #endif  */

/**
*** Structures that are needed to process the RSVP protocol messages
**/

typedef struct commonheader {
          int Version;
          int Msg_Type;
          int SendTTL;
          double RSVPChecksum;
          double RSVPLength;
}Common_Header;

typedef struct sessionobject {
          int Class_Num;
          IPAddress DestAddr;
          int ProtocolId;
      int Flags;
      int DstPort;
      double Length;
}Session_Object;

typedef struct rsvphopobject {
          int Class_Num;
          IPAddress NextHopAddr;
          double LogicalIntfHandle;
          double Length;
}RsvpHop_Object;

typedef struct sendertmplobject {
          int Class_Num;
          IPAddress SrcAddr;
          double SrcPort;
          double Length;
}SenderTemplate_Object;
```

```
typedef struct sendertspecobject {
         int Class_Num;
         double RequiredBandwidth;
         double RequiredLinkDelay;
         double Length;
}SenderTspec_Object;

typedef struct resvconfirm {
        int Class_Num;
        IPAddress RecieverAddr;
        double Length;
}ResvConfirm_Object;

typedef struct flowspecobject {
        int Class_Num;
        double ReqBandwidth;
        double LinkDelay;
        double Length;
}FlowSpec_Object;

typedef struct qospfpathstateblock {
         Session_Object SessionObject;
         SenderTemplate_Object SenderTemplateObject;
         SenderTspec_Object SenderTspecObject;
         RsvpHop_Object RsvpHopObject;
}Qospf_Path_State_Block;

typedef struct resvconfinfo {
        Session_Object SessionObject;
        ResvConfirm_Object ResvConfirmObject;
        FlowSpec_Object FlowSpecObject;
}Resv_Conf_Info;

/**
*** Functions prototypes that are being used
**/
void RsvpPkPro( Packet *x);
void QospfPkPro( Packet *x);

void Path_Msg_pro(Packet* x);
void Resv_Msg_pro(Packet* x);
void ResvConf_Msg_pro(Packet* x);

void HelloPk_pro(Packet *x);
void DataDescPk_pro(Packet *x);
void LsReqPk_pro(Packet *x);
void QospfLsUpdatePk_pro(Packet *x);
void MospfLsUpdatePk_pro(Packet *x);
void OspfLsUpdatePk_pro(Packet *x);
void LsUpdatePk_pro(Packet *x);
void LsAckPk_pro(Packet *x);

void Router_Lsa_pro(Packet* x);
void Resource_Lsa_pro(Packet* x);
void RRA_Lsa_pro(Packet* x);
void Network_Lsa_pro(Packet* x);
void Summary_Lsa_pro(Packet* x);
void GrpMbrship_Lsa_pro(Packet* x);
void IgmpJoin_pro(Packet* x);
```

```
void trigger_the_route_calc();


void * AddListEle( ListL *lst, char *Ele, int size );
void  DeleteListEle( ListL *lst, char *Ele, int size );
char * GetNextListEle( ListL *lst, int flag );
void ListInit( ListL *lst );

void CalculateNextHops( Vertex_Struct, Vertex_Struct, RouterLinkAdv );
int IsNetwork( IPAddress );

/** Print Lsa Functions **/
void PrintLinkStateHdr( linkstateheader );
void PrintLinkInfo( linkinfo*, int );
void PrintAttRtrs( IPAddress* );
void PrintTosInfo( tosinfo* );
void PrintResLsaHdr( reslsaheader );
void PrintResLinkInfo( reslinkinfo* );
void PrintOpaqueLsaHdr( opaquelsaheader );
void PrintLinkResvInfo( linkresvinfo*, int );
void PrintVertexInfo( vertexinfo* );
void PrintLsa( char*, int );
void OspfCandidateAddPro( Vertex_Struct, ListL* );
void OspfSPTreeBuild( ListL* );
void QospfRouterInitialize( int );
void RoutingTableCalc( Area_Struct*, int* );
void CandListInt(int, ListL* );
void MospfCandAddPro(int, Vertex_Struct*, ListL* );
void MCRoutetableCalc( int );
void MospfSPFTreeCalc( int, ListL* );
void QospfCandListInt(int, ListL* );
void QospfCandAddPro(int, Vertex_Struct*, ListL* );
void QospfSPFTreeCalc( int, ListL* );

/**
*** Global variables for all the routers
**/
IPAddress AreaBorderRouters[MAX_NO_AREAS];
```

State Variable Block

```
Objid \my_id;
Objid \ip_objid;
Objid \rsvp_objid;
Objid \igmp_objid;
Objid \sub_netid;
Objid \ip_id;
Objid \rx_id;
Objid \tx_id;
Objid \dup_lk_id;
Objid \qospf_id;

int \IS_DESIGNATED_ROUTER;
int \IS_AREA_BORDER_ROUTER;

int                   \net_num0, \node_num0;
int                   \net_num1, \node_num1;
int                   \net_num2, \node_num2;
```

```
 int            \net_num3, \node_num3;
 int            \net_num4, \node_num4;
 int            \net_num5, \node_num5;
 int            \net_num6, \node_num6;
 int            \net_num7, \node_num7;
 int            \net_num8, \node_num8;
 int            \net_num9, \node_num9;

 int \res_ls_seq_no;
 int \rra_ls_seq_no;
 int \router_ls_seq_no;
 int \summary_ls_seq_no;
 int \network_ls_seq_no;
 int \grpmbr_ls_seq_no;

 int \NoOfResLsa;
 int \NoOfRraLsa;
 int \NoOfGrpMbrLsa;
 int \NoOfRouterLsa;
 int \NoOfNetworkLsa;
 int \NoOfSummaryLsa;


/**
*** Definitions of the conditions that are used to transit from one state to other
**/
 int \send_grpmbrlsa;
 int \send_grp_mbr_lsa;
 int \send_reslsa;
 int \send_rralsa;
 int \send_ospflsa;
 int \send_lsreq_pk;
 int \spfcalc;
 int \detupstrnode;
 int \send_dabra;
 int \send_datadesc;
 int \qospfcalc;
 int \ospfspfcalc;
 int \mospfspfcalc;
 int \send_ls_update_pk;
 int \send_hellopk;

 int \Ip_Network_No;
 int \Network_Mask;
/**
*** Information that should be with a router
**/
 int \NoOfLinks;
 int \Indep_Rou;
 int \AreaId;
 int \DeadInt;
 int \HelloInt;
 int \RtrPri;
 int \NetworkMask;
 int \No_of_Networks;
 int \NoOfVertices;
 int \No_of_TOS;
 int \Network_Routers[MAX_NO_LINKS][1];
 double    \TOS_Metric[5];
```

...
...

```
 Linkinfo  \LinksInfo[MAX_NO_LINKS];
 IPAddress \DsgRouterIntfAddr;
 IPAddress \BackupDsgRouterIntfAddr;
 IPAddress \DsgRtr_IpAddr;
 IPAddress \RouterId;
 IPAddress \DefaultDestination;

 Neighbor_Data_Str \NeighborsInfo[MAX_NO_NEIGHBORS];

/**
*** Lists that are to be maintained on the router
**/
 ListL \QospfPSBList;
 ListL \ResvConfInfoList;
 ListL \NeighborList;

/**
*** Structures that are being maintained on the router
**/
 int \NoOfAreasRouterIsConnected;
 Area_Struct \AreaStructure;
 Area_Struct \BackboneStructure;
 RoutingTable_Struct \RoutingTable;
 regmcnet  \RegMcNets[MAX_NO_GRPS];
 mctable   \mc_rte_table[MAX_NO_GRPS];

 int \obt_pk_srcnet;
 int \obt_pk_srcnode;

/**
*** Temporary variables for debugging purposes
**/

 int \NoOfNeighborElements;
 int \InstNoLsa;

 double \NoOfOspfLsaRecv;
 double \NoOfMospfLsaRecv;
 double \NoOfQospfLsaRecv;
 double \TotalNoOfLsaRecv;

 long \TotalQospfCalcTime;
 int \TotalNoOfTimesQospfCalcDone;
 long \TotalMospfCalcTime;
 int \TotalNoOfTimesMospfCalcDone;
 long \TotalOspfCalcTime;
 int \TotalNoOfTimesOspfCalcDone;
 long \TotalDijCalcTime;
 int \TotalNoDijCalcDone;

 double \print_time;
```

Temporary Variable Block

```
	int in_strm;
	int ip_strm;
	int data_len;
```

```
    int tmp_msg_type;
    int protocolid;

    int NeighborFoundFlag;

    Packet *ReceivedPkptr;
    Packet *ReceivedRsvpPkptr;
    Packet *ReceivedQospfPkptr;

    mc_id        *mc_table;

    double tmp_sim_time;
```

Function Block

```
int NeighborFoundFlag;
Packet* ReceivedRsvpPkptr;
Packet* ReceivedOspfPkptr;

void
Path_Msg_pro(Packet* ReceivedRsvpPkptr)
{
 /**
 *** temporary variables that are required for the Path message processing
 **/
      /*  PathMessage * pm  */
      /*
   Session_Object* tmp_session_object;
   RsvpHop_Object* tmp_rsvphop_object;
   SenderTemplate_Object* tmp_sendertemp_object;
   SenderTspec_Object* tmp_sendertspec_object;
      Qospf_Path_State_Block* QospfPathStateBlock;
      memset( QospfPathStateBlock, 0, sizeof(Qospf_Path_State_Block) );

   op_pk_nfd_get( ReceivedRsvpPkptr, "SessionObject", &tmp_session_object );
   op_pk_nfd_get( ReceivedRsvpPkptr, "RsvpHopObject", &tmp_rsvphop_object );
   op_pk_nfd_get( ReceivedRsvpPkptr, "SenderTemplateObject",
                                  &tmp_sendertemp_object );
   op_pk_nfd_get( ReceivedRsvpPkptr, "SenderTspecObject",
                                  &tmp_sendertspec_object );
     */
 /**
 *** Create the Path State Block for this session from the inforamtion that is
 *** obtained from the Path message coming from the RSVP daemon
 **/
      /*
   op_prg_mem_copy( tmp_session_object, &(QospfPathStateBlock->SessionObject),
                                  sizeof(Session_Object) );
   op_prg_mem_copy( tmp_sendertemp_object,
                                          &(QospfPathStateBlock->SenderTemplateObject),
                                  sizeof(SenderTemplate_Object) );
   op_prg_mem_copy( tmp_rsvphop_object, &(QospfPathStateBlock->RsvpHopObject),
                                  sizeof(RsvpHop_Object) );
   op_prg_mem_copy( tmp_sendertspec_object,
                                          &(QospfPathStateBlock->SenderTspecObject),
                                  sizeof(SenderTspec_Object) );
      AddListEle( &QospfPSBList, (char *)QospfPathStateBlock,
                                                    sizeof(Qospf_Path_State_Block) );
     */
```

```

 /**
 *** Deallocate the memory allocated to the structure’s element
 **/
        /*
     op_prg_mem_free( tmp_session_object );
     op_prg_mem_free( tmp_rsvphop_object );
     op_prg_mem_free( tmp_sendertemp_object );
     op_prg_mem_free( tmp_sendertspec_object );
        */

        VL printf("QOSPF(%d.%d), Created PSB for corresponding PathMessage and Calling the
                Qospf Shortest Path Calculation Function \n", RouterId.NetNum, RouterId.NodeNum );

        qospfcalc = 1;
     return;
}

void Resv_Msg_pro(Packet* ReceivedRsvpPkptr)
{
       return;
}
void
ResvConf_Msg_pro( Packet* ReceivedRsvpPkptr )
{
       int transmiting_link_id;
       Session_Object* tmp_session_object;
       ResvConfirm_Object* tmp_resvconfirm_object;
       FlowSpec_Object* tmp_flowspec_object;
       Resv_Conf_Info* ResvConfInfo;
       memset( ResvConfInfo, 0, sizeof(Resv_Conf_Info) );

       op_pk_nfd_get( ReceivedRsvpPkptr, "SessionObject", &tmp_session_object );
    op_pk_nfd_get( ReceivedRsvpPkptr, "ResvConfirmObject",
                                     &tmp_resvconfirm_object );
    op_pk_nfd_get( ReceivedRsvpPkptr, "FlowSpecObject", &tmp_flowspec_object );


       op_prg_mem_copy( tmp_session_object, &(ResvConfInfo->SessionObject),
                                     sizeof(Session_Object) );
    op_prg_mem_copy( tmp_resvconfirm_object, &(ResvConfInfo->ResvConfirmObject),
                                     sizeof(ResvConfirm_Object) );
    op_prg_mem_copy( tmp_flowspec_object, &(ResvConfInfo->FlowSpecObject),
                                     sizeof(FlowSpec_Object) );

       AddListEle( &ResvConfInfoList, (char *)ResvConfInfo,
                                                          sizeof(Resv_Conf_Info) );
    op_prg_mem_free( tmp_session_object );
    op_prg_mem_free( tmp_resvconfirm_object );
    op_prg_mem_free( tmp_flowspec_object );

       /**
       *** Get the link through which the destination address would be reached
       *** and on which the reservation of the resources is made
       **/
       transmiting_link_id =
                     get_transmiting_link_id(ResvConfInfo->SessionObject.DestAddr);
       LinksInfo[transmiting_link_id].ReservedBandwidth =
                                          ResvConfInfo->FlowSpecObject.ReqBandwidth;
```

```
        LinksInfo[transmiting_link_id].ReservedDelay =
                                        ResvConfInfo->FlowSpecObject.LinkDelay;
        send_rralsa = 1;

        return;
}

int
get_transmiting_link_id(double destaddr)
{
        int tmp_link_id;

        return tmp_link_id;
}

void
trigger_the_route_calc()
{
        spfcalc = 1;
        return;
}


/**
***  Hello Packet Processing
**/
/*
void
HelloPk_pro(Packet * ReceivedOspfPkptr)
{
        int NeighborFoundFlag = 0;
        Neighbor_Data_Str* neighborEle;
        hellopacket* tmp_hellopk;

        tmp_hellopk = (hellopacket*) op_prg_mem_alloc( sizeof(hellopacket) );
        op_pk_nfd_get(ReceivedOspfPkptr, "hellopk", &tmp_hellopk);
        if(tmp_hellopk->Deadint != DeadInt ||
          tmp_hellopk->Helloint != HelloInt ||
          tmp_hellopk->ThisNetworkMask != Network_Mask)
                return;
        if( !IsListEmpty(&NeighborList) )
        {
                neighborEle = (Neighbor_Data_Str*) GetNextListEle( &NeighborList, 1 );
                while(neighborEle)
                {
                        if( tmp_hellopk->OspfPkHeader.ThisRouterId.NetNum ==
                                neighborEle->NeighborRouterId.NetNum &&
                                tmp_hellopk->OspfPkHeader.ThisRouterId.NodeNum ==
                                neighborEle->NeighborRouterId.NodeNum )
                        {
                                NeighborFoundFlag = 1;
                                break;
                        }
                        neighborEle = (Neighbor_Data_Str*) GetNextListEle( &NeighborList, 0 );
                }
          if(NeighborFoundFlag == 0)
          {
                neighborEle = (Neighbor_Data_Str*) op_prg_mem_alloc(sizeof(Neighbor_Data_Str) );
                neighborEle->NeighborRouterId.NetNum =
```

```
                                            tmp_hellopk->OspfPkHeader.ThisRouterId.NetNum;
                neighborEle->NeighborRouterId.NodeNum =
                                            tmp_hellopk->OspfPkHeader.ThisRouterId.NodeNum;
                neighborEle->IsIndepRou = tmp_hellopk->Indep_Router;
                AddListEle( &NeighborList, (char *)neighborEle, sizeof(Neighbor_Data_Str) );
                NoOfNeighborElements++;
            }
        }
        else
        {
            neighborEle = (Neighbor_Data_Str*) op_prg_mem_alloc(sizeof(Neighbor_Data_Str) );
            neighborEle->NeighborRouterId.NetNum = tmp_hellopk->OspfPkHeader.ThisRouterId.NetNum;
            neighborEle->NeighborRouterId.NodeNum = tmp_hellopk->OspfPkHeader.ThisRouterId.NodeNum;
            AddListEle( &NeighborList, (char *)neighborEle, sizeof(Neighbor_Data_Str) );
            NoOfNeighborElements++;

        }
        VL   HPDL
        {
            printf("--------------------- Hello Packet Processed ------------------------\n");
            printf("\n Router(%d.%d), has the following Neighbors ", RouterId.NetNum, RouterId.NodeNum );
            if( !IsListEmpty(&NeighborList) )
            {
                neighborEle = (Neighbor_Data_Str*) GetNextListEle( &NeighborList, 1 );
                while(neighborEle)
                {
                    if( !(neighborEle->NeighborRouterId.NetNum == 0 &&
                         neighborEle->NeighborRouterId.NodeNum == 0) )
                        printf("%d.%d, ", neighborEle->NeighborRouterId.NetNum,
                                neighborEle->NeighborRouterId.NodeNum );
                    neighborEle = (Neighbor_Data_Str*) GetNextListEle( &NeighborList, 0 );
                }
            }
            printf("\n---------------------- End Hello Packet Processed ------------------------\n");
        }
        op_prg_mem_free( tmp_hellopk );
}
*/
/**
*** Hello Packet Processing
**/
void
HelloPk_pro(Packet * ReceivedOspfPkptr)
{
        int NeighborFoundFlag = 0;
        int i;
    hellopacket* tmp_hellopk;

        tmp_hellopk = (hellopacket*) op_prg_mem_alloc( sizeof(hellopacket) );
    op_pk_nfd_get(ReceivedOspfPkptr, "hellopk", &tmp_hellopk);
    if( tmp_hellopk->Deadint != DeadInt ||
            tmp_hellopk->Helloint != HelloInt ||
        tmp_hellopk->ThisNetworkMask != Network_Mask)
            return;
        for(i = 0; i<= MAX_NO_NEIGHBORS; i++ )
        {
            if( tmp_hellopk->OspfPkHeader.ThisRouterId.NetNum ==
                NeighborsInfo[i].NeighborRouterId.NetNum &&
                tmp_hellopk->OspfPkHeader.ThisRouterId.NodeNum ==
```

```
                        NeighborsInfo[i].NeighborRouterId.NodeNum )
                {
                        NeighborFoundFlag = 1;
                        break;
                }
        }
  if(NeighborFoundFlag == 0)
  {
                for(i = 0; i<= MAX_NO_NEIGHBORS; i++ )
                {
                        if( NeighborsInfo[i].NeighborRouterId.NodeNum == 0 &&
                                NeighborsInfo[i].NeighborRouterId.NetNum == 0 )
                        {
                                NeighborsInfo[i].NeighborRouterId.NetNum =
                                        tmp_hellopk->OspfPkHeader.ThisRouterId.NetNum;
                                NeighborsInfo[i].NeighborRouterId.NodeNum =
                                        tmp_hellopk->OspfPkHeader.ThisRouterId.NodeNum;
                                NeighborsInfo[i].IsIndepRou = tmp_hellopk->Indep_Router;
                                break;
                        }
                }
        }

        VL  HPDL
        {
                printf("\n Router(%d.%d), has the following Neighbors ", RouterId.NetNum, RouterId.NodeNum );
                for(i = 0; i<= MAX_NO_NEIGHBORS; i++ )
                {
                        if( !(NeighborsInfo[i].NeighborRouterId.NodeNum == 0 &&
                                 NeighborsInfo[i].NeighborRouterId.NetNum == 0 ) )
                                 printf( "%d.%d ,", NeighborsInfo[i].NeighborRouterId.NetNum,
                                                NeighborsInfo[i].NeighborRouterId.NodeNum );
                }
        }
        op_prg_mem_free( tmp_hellopk );
        return;
}
/**
*** Function : QospfLsUpdatePk_pro
*** Input    : A pointer to the recived Qospf packet pointer is passed.
*** Output   : This function will check for the Lsa's that are in the Qospf Ls update packet and update the
***            local databases
**/
void
QospfLsUpdatePk_pro( Packet *ReceivedQospfPkptr )
{
        int i = 0;
        int j = 0;
        int FoundFlag = OFF;

        QospfLSUpdatePk* qupdate;
        qupdate = (QospfLSUpdatePk *) op_prg_mem_alloc( sizeof(QospfLSUpdatePk) );
        memset( qupdate, 0, sizeof(QospfLSUpdatePk) );
        op_pk_nfd_get( ReceivedQospfPkptr, "Lsupdate" , &qupdate );
        if( qupdate->OspfPkHeader.Type != QOSPFLSUPDATEPK )
        {
                VL  QPDL    printf(" Not Valid OspfPacket Header Type %d \n", qupdate->OspfPkHeader.Type );
                return;
        }
```

```
else
{
        for( i = 0; i < qupdate->NoOfResAdvs; i++ )
        {
            FoundFlag = OFF;
            for(j = 0; j < MAX_NO_LSAS; j++ )
            {
                if( (AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NetNum ==
                     qupdate->ResLinkStateAdv[i].ResLsaHeader.AdvRouter.NetNum &&
                     AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NodeNum ==
                     qupdate->ResLinkStateAdv[i].ResLsaHeader.AdvRouter.NodeNum) )

                {
                    if( IS_AREA_BORDER_ROUTER == 1 ||
                        qupdate->ResLinkStateAdv[i].ResLsaHeader.ThisAreaId == AreaId )
                    {
                        op_prg_mem_copy( &(qupdate->ResLinkStateAdv[i]),
                                        &AreaStructure.ResourceLsaList[j], sizeof(reslsa) );
                        FoundFlag = ON;
                    }
                    break;
                }

            }
            if( FoundFlag == OFF )
            {
                for(j = 0; j < MAX_NO_LSAS; j++ )
                {
                    if( (AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NetNum == 0 &&
                         AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NodeNum == 0 &&
                         AreaStructure.ResourceLsaList[j].ResLsaHeader.LSSeqNo == 0) )
                    {
                        if( IS_AREA_BORDER_ROUTER == 1 ||
                            qupdate->ResLinkStateAdv[i].ResLsaHeader.ThisAreaId == AreaId )
                            op_prg_mem_copy( &(qupdate->ResLinkStateAdv[i]),
                                            &AreaStructure.ResourceLsaList[j], sizeof(reslsa) );
                        break;
                    }
                }
            }
        }
        for( i = 0; i < qupdate->NoOfRraAdvs; i++ )
        {
            FoundFlag = OFF;
            for(j = 0; j < MAX_NO_LSAS; j++ )
            {
                if( (AreaStructure.RRALsaList[j].OpaqueLsaHeader.AdvRouter.NetNum ==
                     qupdate->RraLinkStateAdv[i].OpaqueLsaHeader.AdvRouter.NetNum &&
                     AreaStructure.RRALsaList[j].OpaqueLsaHeader.AdvRouter.NodeNum ==
                     qupdate->RraLinkStateAdv[i].OpaqueLsaHeader.AdvRouter.NodeNum) &&
                    (qupdate->RraLinkStateAdv[i].OpaqueLsaHeader.LSSeqNo >=
                     AreaStructure.RRALsaList[j].OpaqueLsaHeader.LSSeqNo) )
                {
                    if( IS_AREA_BORDER_ROUTER == 1 ||
                        qupdate->RraLinkStateAdv[i].OpaqueLsaHeader.ThisAreaId == AreaId )
                    {
                        op_prg_mem_copy( &(qupdate->RraLinkStateAdv[i]),
                                        &AreaStructure.RRALsaList[j], sizeof(reslsa) );
                        FoundFlag = ON;
```

```
                        }
                        break;
                    }

                }
                if( FoundFlag == OFF )
                {
                    for(j = 0; j < MAX_NO_LSAS; j++ )
                    {
                        if( (AreaStructure.RRALsaList[j].OpaqueLsaHeader.AdvRouter.NetNum == 0 &&
                             AreaStructure.RRALsaList[j].OpaqueLsaHeader.AdvRouter.NodeNum == 0 &&
                             AreaStructure.RRALsaList[j].OpaqueLsaHeader.LSSeqNo == 0) )
                        {
                            if( IS_AREA_BORDER_ROUTER == 1 ||
                                qupdate->RraLinkStateAdv[i].OpaqueLsaHeader.ThisAreaId == AreaId )
                                op_prg_mem_copy( &(qupdate->RraLinkStateAdv[i]),
                                                 &AreaStructure.RRALsaList[j], sizeof(reslsa) );
                            break;
                        }

                    }
                }
            }
    }

    NoOfQospfLsaRecv = NoOfQospfLsaRecv + qupdate->NoOfRraAdvs + qupdate->NoOfResAdvs;
    TotalNoOfLsaRecv = TotalNoOfLsaRecv + NoOfQospfLsaRecv;
    op_stat_local_write( TOTAL_LSA_OUT_STAT, TotalNoOfLsaRecv );

    op_prg_mem_free( qupdate );
    qospfcalc = TRUE;

    VL  QPDL
    {
        printf("---------------------- Qospf LsUpdate Packet ----------------------\n");
        printf( "(%d.%d)Reservation Lsa's : \n", RouterId.NetNum, RouterId.NodeNum );
        for( i = 0; i < MAX_NO_LSAS; i++ )
        {
            if( !(AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NetNum == 0 &&
                  AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NetNum == 0 ) )
                printf(" Adv. Router is %d.%d ",
                AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NetNum,
                AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NodeNum );
        }
        printf("\n");
        printf( "(%d.%d)Resource Lsa's : \n", RouterId.NetNum, RouterId.NodeNum );
        for( i = 0; i < MAX_NO_LSAS; i++ )
        {
            if( !(AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NetNum == 0 &&
                  AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NetNum == 0 ) )
            printf(" Adv. Router is %d.%d ",
            AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NetNum,
            AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NodeNum );
        }
        printf("\n");
        printf("-------------------- End Qospf LsUpdate Packet --------------------\n");
    }
    return;
}
```

```
void
MospfLsUpdatePk_pro( Packet *ReceivedQospfPkptr )
{
	int i = 0;
	int j = 0;
	int FoundFlag = OFF;
	int Dest_Grp;

	MospfLSUpdatePk *qupdate;

	VL  MPDL   printf("---------- Mospf LS Update Packet Processing ---------- \n");
	qupdate = (MospfLSUpdatePk *) op_prg_mem_alloc( sizeof(MospfLSUpdatePk) );
	memset( qupdate, 0, sizeof(MospfLSUpdatePk) );

	op_pk_nfd_get( ReceivedQospfPkptr, "Lsupdate" , &qupdate );
	if( qupdate->OspfPkHeader.Type != MOSPFLSUPDATEPK )
	{
		return;
	}
	else
	{
		if(qupdate->OspfPkHeader.ThisAreaId = AreaId || IS_AREA_BORDER_ROUTER )
		{
			NoOfMospfLsaRecv = NoOfMospfLsaRecv + qupdate->NoOfGrpMbrAdvs;
			TotalNoOfLsaRecv = TotalNoOfLsaRecv + NoOfMospfLsaRecv;
			op_stat_local_write( TOTAL_LSA_OUT_STAT, TotalNoOfLsaRecv );

			for( i = 0; i < qupdate->NoOfGrpMbrAdvs; i++ )
			{
				FoundFlag = OFF;
				Dest_Grp = qupdate->GrpmbrLinkStateAdv[0].LinkStateHeader.LinkStateId.NetNum;
				VL  MPDL   printf("Recevied a MospfLsa for the group %d from %d.%d\n",
								Dest_Grp, obt_pk_srcnet, obt_pk_srcnode );
				while( !(AreaStructure.GrpMbrLsaList[Dest_Grp][j].LinkStateHeader.AdvRouter.NetNum == 0 &&
						AreaStructure.GrpMbrLsaList[Dest_Grp][j].LinkStateHeader.AdvRouter.NodeNum == 0) )
				{
					if( AreaStructure.GrpMbrLsaList[Dest_Grp][j].LinkStateHeader.AdvRouter.NetNum ==
						qupdate->GrpmbrLinkStateAdv[i].LinkStateHeader.AdvRouter.NetNum &&
						AreaStructure.GrpMbrLsaList[Dest_Grp][j].LinkStateHeader.AdvRouter.NodeNum ==
						qupdate->GrpmbrLinkStateAdv[i].LinkStateHeader.AdvRouter.NodeNum )
					{

						if( qupdate->GrpmbrLinkStateAdv[i].LinkStateHeader.LSSeqNo >=
							AreaStructure.GrpMbrLsaList[Dest_Grp][j].LinkStateHeader.LSSeqNo )
						{
							op_prg_mem_copy( &(qupdate->GrpmbrLinkStateAdv[i]),
										&AreaStructure.GrpMbrLsaList[Dest_Grp][j],
										sizeof(grpmbrlsa) );
						 }
						FoundFlag = ON;
						break;
					}
					j++;
				}
				if(FoundFlag == OFF)
				{
					while( !(qupdate->GrpmbrLinkStateAdv[j].LinkStateHeader.AdvRouter.NetNum == 0 &&
							qupdate->GrpmbrLinkStateAdv[j].LinkStateHeader.AdvRouter.NodeNum == 0) )
```

```
                        {
                                op_prg_mem_copy( &(qupdate->GrpmbrLinkStateAdv[i]),
                                                &AreaStructure.GrpMbrLsaList[Dest_Grp][j],
                                                sizeof(grpmbrlsa) );
                                j++;
                        }
                    }
                }
            }
        }
        op_prg_mem_free( qupdate );
        mospfspfcalc = 1;

        VL  MPDL
        {
            int i = 0;
            int j = 0;
            int k = 0;

            printf("Adv. Router is %d.%d and its Links Information is \n",
                    RouterId.NetNum, RouterId.NodeNum );
            printf("- - - - - - - - - - - - - - - - - - - - - - - - - \n");

            for( i = 0; i < MAX_NO_GRPS; i++ )
            {
                if( !(AreaStructure.GrpMbrLsaList[i][0].LinkStateHeader.AdvRouter.NetNum == 0 &&
                     AreaStructure.GrpMbrLsaList[i][0].LinkStateHeader.AdvRouter.NodeNum == 0) )
                {
                    for( j = 0; j < MAX_NO_LSAS; j++ )
                    {
                        if( !(AreaStructure.GrpMbrLsaList[i][j].LinkStateHeader.AdvRouter.NetNum == 0 &&
                             AreaStructure.GrpMbrLsaList[i][j].LinkStateHeader.AdvRouter.NodeNum == 0) )
                        {
                            for( k = 0; k < NoOfLinks; k++ )
                            {
                                if( !(AreaStructure.GrpMbrLsaList[i][j].VertexInfo[k].VertexId.NetNum <= 0 &&
                                     AreaStructure.GrpMbrLsaList[i][j].VertexInfo[k].VertexId.NodeNum <= 0) )
                                {
                                    printf( "AreaStructure.GrpMbrLsaList[%d][%d].VertexInfo[%d].VertexId = "
                                            "%d.%d\n", i, j, k,
                                            AreaStructure.GrpMbrLsaList[i][j].VertexInfo[k].VertexId.NetNum,
                                            AreaStructure.GrpMbrLsaList[i][j].VertexInfo[k].VertexId.NodeNum );
                                }
                            }
                        }
                    }
                }
            }
        }

        VL  MPDL    printf("--------- End Mospf Link State Adverisments On the Router -------- \n" );
        return;
}

void
OspfLsUpdatePk_pro( Packet *ReceivedQospfPkptr )
{
        int i = 0;
        int j = 0;
```

```
int FoundFlag = OFF;
OspfLSUpdatePk *qupdate;
qupdate = (OspfLSUpdatePk *) op_prg_mem_alloc( sizeof(OspfLSUpdatePk) );
memset( qupdate, 0, sizeof(QospfLSUpdatePk) );
op_pk_nfd_get( ReceivedQospfPkptr, "Lsupdate" , &qupdate );
if( qupdate->OspfPkHeader.Type != OSPFLSUPDATEPK )
{
	return;
}
else
{
	NoOfOspfLsaRecv = NoOfOspfLsaRecv + qupdate->NoOfRouterAdvs +
						qupdate->NoOfNetworkAdvs + qupdate->NoOfSummaryAdvs;
	TotalNoOfLsaRecv = TotalNoOfLsaRecv + NoOfOspfLsaRecv;
	op_stat_local_write( TOTAL_LSA_OUT_STAT, TotalNoOfLsaRecv );

	for( i = 0; i < qupdate->NoOfRouterAdvs; i++ )
	{
		FoundFlag = OFF;
		for(j = 0; j < MAX_NO_LSAS; j++ )
		{
			if( (AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NetNum ==
				qupdate->RouterLinkStateAdv[i].LinkStateHeader.AdvRouter.NetNum &&
				AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NodeNum ==
				qupdate->RouterLinkStateAdv[i].LinkStateHeader.AdvRouter.NodeNum) )

			{
				if( IS_AREA_BORDER_ROUTER == 1 ||
					qupdate->RouterLinkStateAdv[i].LinkStateHeader.ThisAreaId == AreaId )
				{
					op_prg_mem_copy( &(qupdate->RouterLinkStateAdv[i]),
							&AreaStructure.RouterLsaList[j], sizeof(RouterLinkAdv) );
					FoundFlag = ON;
				}
				break;
			}

		}
		if( FoundFlag == OFF )
		{
			for(j = 0; j < MAX_NO_LSAS; j++ )
			{
				if( (AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NetNum == 0 &&
					AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NodeNum == 0 &&
					AreaStructure.RouterLsaList[j].LinkStateHeader.LSSeqNo ==  0) )
				{
					if( IS_AREA_BORDER_ROUTER == 1 ||
						qupdate->RouterLinkStateAdv[i].LinkStateHeader.ThisAreaId == AreaId )
						op_prg_mem_copy( &(qupdate->RouterLinkStateAdv[i]),
								&AreaStructure.RouterLsaList[j], sizeof(RouterLinkAdv) );
					break;
				}
			}
		}
	}

	if( IS_DESIGNATED_ROUTER == 1 )
	{
		for( i = 0; i < qupdate->NoOfNetworkAdvs; i++ )
```

...
...

```
{
    FoundFlag = OFF;
    for(j = 0; j < MAX_NO_LSAS; j++ )
    {
        if( (AreaStructure.NetworkLsaList[j].LinkStateHeader.AdvRouter.NetNum ==
             qupdate->NetworkLinkStateAdv[i].LinkStateHeader.AdvRouter.NetNum &&
             AreaStructure.NetworkLsaList[j].LinkStateHeader.AdvRouter.NodeNum ==
             qupdate->NetworkLinkStateAdv[i].LinkStateHeader.AdvRouter.NodeNum) )
        {
            if( IS_AREA_BORDER_ROUTER == 1 ||
                qupdate->NetworkLinkStateAdv[i].LinkStateHeader.ThisAreaId == AreaId )
            {
                op_prg_mem_copy( &(qupdate->NetworkLinkStateAdv[i]),
                                 &AreaStructure.NetworkLsaList[j], sizeof(NetworkLinkAdv) );
                FoundFlag = ON;
            }
            break;
        }

    }
    if( FoundFlag == OFF )
    {
        for(j = 0; j < MAX_NO_LSAS; j++ )
        {
            if( (AreaStructure.NetworkLsaList[j].LinkStateHeader.AdvRouter.NetNum == 0 &&
                 AreaStructure.NetworkLsaList[j].LinkStateHeader.AdvRouter.NodeNum == 0 &&
                 AreaStructure.NetworkLsaList[j].LinkStateHeader.LSSeqNo == 0) )
            {
                if( IS_AREA_BORDER_ROUTER == 1 ||
                    qupdate->NetworkLinkStateAdv[i].LinkStateHeader.ThisAreaId == AreaId )
                    op_prg_mem_copy( &(qupdate->NetworkLinkStateAdv[i]),
                                     &AreaStructure.NetworkLsaList[j], sizeof(NetworkLinkAdv) );
                break;
            }

        }
    }
}
if( IS_AREA_BORDER_ROUTER == 1 )
{
    for( i = 0; i < qupdate->NoOfSummaryAdvs; i++ )
    {
        FoundFlag = OFF;
        for(j = 0; j < MAX_NO_LSAS; j++ )
        {
            if( (AreaStructure.SummaryLsaList[j].LinkStateHeader.AdvRouter.NetNum ==
                 qupdate->SummaryLinkStateAdv[i].LinkStateHeader.AdvRouter.NetNum &&
                 AreaStructure.SummaryLsaList[j].LinkStateHeader.AdvRouter.NodeNum ==
                 qupdate->SummaryLinkStateAdv[i].LinkStateHeader.AdvRouter.NodeNum) )
            {
                if( IS_AREA_BORDER_ROUTER == 1 ||
                    qupdate->SummaryLinkStateAdv[i].LinkStateHeader.ThisAreaId == AreaId )
                {
                    op_prg_mem_copy( &(qupdate->SummaryLinkStateAdv[i]),
                                     &AreaStructure.SummaryLsaList[j], sizeof(SummaryLinkAdv) );
                    FoundFlag = ON;
                }
                break;
            }
```

```
                    }
                    if( FoundFlag == OFF )
                    {
                        for(j = 0; j < MAX_NO_LSAS; j++ )
                        {
                            if( (AreaStructure.SummaryLsaList[j].LinkStateHeader.AdvRouter.NetNum == 0 &&
                                AreaStructure.SummaryLsaList[j].LinkStateHeader.AdvRouter.NodeNum == 0 &&
                                AreaStructure.SummaryLsaList[j].LinkStateHeader.LSSeqNo == 0) )
                            {
                                if( IS_AREA_BORDER_ROUTER == 1 ||
                                    qupdate->SummaryLinkStateAdv[i].LinkStateHeader.ThisAreaId == AreaId )
                                    op_prg_mem_copy( &(qupdate->SummaryLinkStateAdv[i]),
                                                     &AreaStructure.SummaryLsaList[j],
                                                     sizeof(SummaryLinkAdv) );
                                break;
                            }

                        }
                    }
                }
            }
        }
    }
    op_prg_mem_free( qupdate );
    ospfspfcalc = 1;
    VL  OPDL
    {
        printf("\n--------------------- (%d.%d) OSPF LSUPDATE PACKET ---------------------\n",
                RouterId.NetNum, RouterId.NodeNum );
        printf("ROUTER LSA'S : \n");
        for( i = 0; i < MAX_NO_LSAS; i++ )
        {
            if( !(AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 &&
                 AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 ) )
            {
                printf("Adv. Router is %d.%d and its Links Information is \n",
                AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NetNum,
                AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NodeNum );
                printf("- - - - - - - - - - - - - - - - - - - - - - - - - - - - - \n");
                printf("No Of Links are %d\n", AreaStructure.RouterLsaList[i].No_of_Links);

                for( j = 0; j < AreaStructure.RouterLsaList[i].No_of_Links; j++ )
                {
                    printf( "Type of the Link is %d, ",
                            AreaStructure.RouterLsaList[i].Link_Info[j].Type );
                    printf( "Link Id is %d.%d, ",
                            AreaStructure.RouterLsaList[i].Link_Info[j].LinkId.NetNum,
                            AreaStructure.RouterLsaList[i].Link_Info[j].LinkId.NodeNum );
                    printf( "Link Data is %d.%d, ",
                            AreaStructure.RouterLsaList[i].Link_Info[j].LinkData.NetNum,
                            AreaStructure.RouterLsaList[i].Link_Info[j].LinkData.NodeNum );
                    printf( "TOS Metric is %d\n",
                            AreaStructure.RouterLsaList[i].Link_Info[j].TOSMetric );
                }
                printf("\n");
            }
        }
        printf("\n");
```

```
            printf( "NETWORK LSA'S : \n" );
            for( i = 0; i < MAX_NO_LSAS; i++ )
            {
                if( !(AreaStructure.NetworkLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 &&
                      AreaStructure.NetworkLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 ) )
                printf(" Adv. Router is %d.%d ",
                AreaStructure.NetworkLsaList[i].LinkStateHeader.AdvRouter.NetNum,
                AreaStructure.NetworkLsaList[i].LinkStateHeader.AdvRouter.NodeNum );
            }
            printf("\n");
            printf( "SUMMARY LSA'S : \n" );
            for( i = 0; i < MAX_NO_LSAS; i++ )
            {
                if( !(AreaStructure.SummaryLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 &&
                      AreaStructure.SummaryLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 ) )
                printf(" Adv. Router is %d.%d ",
                AreaStructure.SummaryLsaList[i].LinkStateHeader.AdvRouter.NetNum,
                AreaStructure.SummaryLsaList[i].LinkStateHeader.AdvRouter.NodeNum );
            }
            printf("\n");
            printf("-------------------- (%d.%d)END OSPF LSUPDATE PACKET --------------------\n",
                    RouterId.NetNum, RouterId.NodeNum );
        }
        return;
}

/**
*** Mospf Routing Table calculating functions
**/
void
CandListInt(int s_net, ListL* CandidatesList)
{
  int SrcNet;
  Vertex_Struct CandVertex;

  int SrcNet_Found_Flag = OFF;

  memset( &CandVertex, 0, sizeof(Vertex_Struct) );

  SrcNet = s_net;
  if( SrcNet <= (AreaId*MaxNoNetsInArea) ||
    SrcNet <= (Router_Start_Index + AreaId*MaxNoRoutersInArea) )
    SrcNet_Found_Flag = ON;

  if(SrcNet_Found_Flag == ON)
  {
    CandVertex.VertexId.NetNum = SrcNet;
    CandVertex.DistanceToRoot = 0;
    AddListEle( CandidatesList, (char *)&CandVertex, sizeof(Vertex_Struct) );
  }
  else
  {
    CandVertex.VertexId = AreaBorderRouters[AreaId];
    AddListEle( CandidatesList, (char *)&CandVertex, sizeof(Vertex_Struct) );
  }
}

void
MospfCandAddPro( int GrpId, Vertex_Struct* VertexV, ListL* MospfCandidatesList )
```

```
{
}

void
MospfSPFTreeCalc(int GrpId,  ListL* CandidateList )
{
   int Inx, i;
   Vertex_Struct VertexW;
   if( !IsListEmpty( CandidateList ) )
   {
      int LeastDistance = 10000;
      int IsLessThanFlag = OFF;
      int FoundFlag = OFF;
      Vertex_Struct* tmpCandidateEle;
      tmpCandidateEle = (Vertex_Struct*)GetNextListEle( CandidateList, 1 );
      while( tmpCandidateEle )
      {
         LeastDistance = tmpCandidateEle->DistanceToRoot;
         Inx = 0;
         while( !(AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NetNum == 0 &&
                 AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NodeNum == 0) )
         {
            if( AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NetNum ==
               tmpCandidateEle->VertexId.NetNum &&
               AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NodeNum ==
               tmpCandidateEle->VertexId.NodeNum )
            {
               FoundFlag = ON;
               break;
            }

            if( AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].DistanceToRoot
               > LeastDistance )
            {
               IsLessThanFlag = ON;
               break;
            }
            Inx++;
         }
         if( FoundFlag == ON )
            if( AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].DistanceToRoot
               > LeastDistance )
               AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].DistanceToRoot
                                                                = LeastDistance;

         if( FoundFlag == OFF )
         {
            if(IsLessThanFlag == ON)
            {
               Vertex_Struct tmpSPVertex;
               Vertex_Struct sectmpSPVertex;
               tmpSPVertex = AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx];
               op_prg_mem_copy( tmpCandidateEle,
                           &(AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx]),
                           sizeof(Vertex_Struct) );
               while(!(AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NetNum == 0 &&
                     AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NodeNum == 0) )
              {
                 sectmpSPVertex = AreaStructure.MospfShortestPathTree[GrpId].SPVertex[i+1];
```

```
                        AreaStructure.MospfShortestPathTree[GrpId].SPVertex[i+1] = tmpSPVertex;
                        tmpSPVertex = sectmpSPVertex;
                        Inx++;
                    }
                }
                if( IsLessThanFlag == OFF )
                    op_prg_mem_copy( tmpCandidateEle,
                            &(AreaStructure.MospfShortestPathTree[GrpId].SPVertex[Inx]),
                            sizeof(Vertex_Struct) );

                op_prg_mem_copy( tmpCandidateEle, &VertexW, sizeof(Vertex_Struct) );
                OspfCandidateAddPro( VertexW , CandidateList );
            }
            DeleteListEle(CandidateList, (char*)tmpCandidateEle, sizeof(Vertex_Struct) );
            tmpCandidateEle = (Vertex_Struct*)GetNextListEle( CandidateList, 1 );
        }
    }
}

void
MCRoutetableCalc( int GrpId )
{
    int i = 1;
    int* rte_table;

    rte_table = (int *)op_ima_obj_svar_get(ip_objid, "rte_tab" );

    while(!(AreaStructure.MospfShortestPathTree[GrpId].SPVertex[i].VertexId.NetNum == 0 &&
            AreaStructure.MospfShortestPathTree[GrpId].SPVertex[i].VertexId.NodeNum == 0) )
    {
        int j = 0;
        int k = 0;
        int RouteFoundFlag = OFF;

        for( j = 0; j < NoOfLinks; j++ )
        {

                mc_rte_table[GrpId].interface[rte_table[AreaStructure.MospfShortestPathTree[GrpId].SPVertex[i].VertexId.NetNum]
            RouteFoundFlag = ON;
        }
        if( RouteFoundFlag == OFF )
        {
            if( !IS_AREA_BORDER_ROUTER )
                mc_rte_table[GrpId].interface[rte_table[DEF_STREAM]] = 1;
        }
        i++;
    }
}

/**
*** Qospf Routing Table Calculating Functions
**/

void
QospfCandidateListInit(int SrcNet, ListL* QospfCandidatesList)
{
    Vertex_Struct CandVertex;

    int SrcNet_Found_Flag = OFF;
```

...
...

```
    memset( &CandVertex, 0, sizeof(Vertex_Struct) );

    if( SrcNet <= (AreaId*MaxNoNetsInArea) ||
        SrcNet <= (Router_Start_Index + AreaId*MaxNoRoutersInArea) )
        SrcNet_Found_Flag = ON;

    if(SrcNet_Found_Flag == ON)
    {
        CandVertex.VertexId.NetNum = SrcNet;
        CandVertex.DistanceToRoot = 0;
        AddListEle( QospfCandidatesList, (char *)&CandVertex, sizeof(Vertex_Struct) );
    }
    else
    {
        CandVertex.VertexId = AreaBorderRouters[AreaId];
        AddListEle( QospfCandidatesList, (char *)&CandVertex, sizeof(Vertex_Struct) );
    }
}

void
QospfCandAddPro( int GrpId, Vertex_Struct* VertexV, ListL* QospfCandidatesList )
{
    int i=0;
    int k=0;

    Vertex_Struct VertexW;
    Vertex_Struct tmpVertexW;

    reslsa  WResLsa;
    rralsa  WRraLsa;
    reslsa  VResLsa;
    rralsa  VRraLsa;

    Vertex_Struct* CandidateEle;
    Vertex_Struct* tmpCandEle;
    int CandidateFoundFlag = OFF;
    int Rra_Found_Flag = OFF;
    sd SessionDirectory[MAX_NO_GRPS];

    memset( &VertexW, 0, sizeof(Vertex_Struct) );
    memset( &tmpVertexW, 0, sizeof(Vertex_Struct) );
    memset( &WResLsa, 0, sizeof(reslsa) );
    memset( &VResLsa, 0, sizeof(reslsa) );
    memset( &WRraLsa, 0, sizeof(rralsa) );
    memset( &VRraLsa, 0, sizeof(rralsa) );

    for(i = 0; i<=MAX_NO_LSAS; i++ )
    {
        if( AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NetNum == VertexV->VertexId.NetNum &&
            AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NodeNum == VertexV->VertexId.NodeNum  )
        {
            op_prg_mem_copy( &AreaStructure.ResourceLsaList[i], &VResLsa, sizeof(reslsa) );
            break;
        }
    }

    for(i = 0; i<=MAX_NO_RRA_LSAS; i++ )
    {
```

...
...

```
        if( AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NetNum == VertexV->VertexId.NetNum &&
            AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NodeNum == VertexV->VertexId.NodeNum  &&
            AreaStructure.RRALsaList[i].DestAddr.NodeNum == GrpId )
        {
            op_prg_mem_copy( &AreaStructure.RRALsaList[i], &VRraLsa, sizeof(reslsa) );
            Rra_Found_Flag = ON;
            break;
        }
    }

    for(i = 0; i <= VResLsa.ResLsaHeader.No_of_Links; i++ )
    {
        int Link_Eligible_Flag = OFF;


        if( Rra_Found_Flag == ON )
        {
            if( (VRraLsa.LinkResvInfo[i].ReservedBandwidth >= SessionDirectory[GrpId].Bandwidth) ||
                (VResLsa.ResLinkInfo[i].Bandwidth >= (SessionDirectory[GrpId].Bandwidth -
                VRraLsa.LinkResvInfo[i].ReservedBandwidth)) )

                Link_Eligible_Flag = ON;
        }
        else
        {
            if( VResLsa.ResLinkInfo[i].Bandwidth >= SessionDirectory[GrpId].Bandwidth )
            {
                Link_Eligible_Flag = ON;
            }
        }
        if( Link_Eligible_Flag == ON)
        {
            int FoundFlag = OFF;
            int j = 0;
            op_prg_mem_copy( &VResLsa.ResLinkInfo[i].LinkId, &VertexW.VertexId, sizeof(IPAddress) );
            j = 0;
            while( !(AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NodeNum == 0 &&
                     AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NetNum == 0 ) )
            {
                if( AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NetNum
                    == VertexW.VertexId.NetNum &&
                    AreaStructure.ResourceLsaList[j].ResLsaHeader.AdvRouter.NodeNum
                    == VertexW.VertexId.NodeNum )
                {
                    FoundFlag = ON;
                    op_prg_mem_copy( &AreaStructure.ResourceLsaList[j], &WResLsa, sizeof(reslsa) );
                    break;
                }
                j++;
            }
            if( FoundFlag = ON )
            {
                int WonSPF_Flag = OFF;
                j = 0;

                for( k = 0; k < WResLsa.ResLsaHeader.No_of_Links; k++ )
                {
                    while( !(AreaStructure.QospfShortestPathTree[GrpId].SPVertex[j].VertexId.NetNum == 0 &&
                            AreaStructure.QospfShortestPathTree[GrpId].SPVertex[j].VertexId.NodeNum == 0 ) )
```

```
                {
                    if( AreaStructure.QospfShortestPathTree[GrpId].SPVertex[j].VertexId.NetNum
                        == WResLsa.ResLinkInfo[i].LinkData.NetNum &&

                        AreaStructure.QospfShortestPathTree[GrpId].SPVertex[j].VertexId.NodeNum
                        == WResLsa.ResLinkInfo[i].LinkData.NodeNum )
                    {
                        WonSPF_Flag = ON;
                        break;
                    }
                    j++;
                }
                if( WonSPF_Flag == OFF && !(WResLsa.ResLinkInfo[i].LinkData.NetNum == 0 &&
                    WResLsa.ResLinkInfo[i].LinkData.NodeNum == 0 ) )
                {
                    op_prg_mem_copy(&WResLsa.ResLinkInfo[i].LinkData, &tmpVertexW.VertexId,
                                sizeof(IPAddress) );
                    if( VResLsa.ResLinkInfo[i].TOSMetric != LSInfinity )
                    {
                        tmpVertexW.DistanceToRoot =
                                VertexV->DistanceToRoot + VResLsa.ResLinkInfo[i].TOSMetric;
                    }

                    if( !IsListEmpty(QospfCandidatesList) )
                    {
                        CandidateEle = (Vertex_Struct*)GetNextListEle( QospfCandidatesList, 1 );
                        while( CandidateEle )
                        {
                            if( CandidateEle->VertexId.NetNum == tmpVertexW.VertexId.NetNum &&
                                CandidateEle->VertexId.NodeNum == tmpVertexW.VertexId.NodeNum )
                            {
                                CandidateFoundFlag = ON;
                                break;
                            }
                            CandidateEle = (Vertex_Struct*)GetNextListEle( QospfCandidatesList, 0 );
                        }
                    }
                    if( CandidateFoundFlag == ON )
                    {
                        if( CandidateEle->DistanceToRoot > tmpVertexW.DistanceToRoot )
                        {
                            CandidateEle->DistanceToRoot = tmpVertexW.DistanceToRoot;
                        }
                    }
                    if( CandidateFoundFlag == OFF )
                    {
                        AddListEle(QospfCandidatesList, (char *)&tmpVertexW, sizeof(Vertex_Struct) );
                    }
                }
            }
        }

    }
  }
}

void
QospfSPFTreeCalc(int GrpId,  ListL* CandidateList )
{
```

```
int Inx, i;
Vertex_Struct VertexW;
if( !IsListEmpty( CandidateList ) )
{
   int LeastDistance = 10000;
   int IsLessThanFlag = OFF;
   int FoundFlag = OFF;
   Vertex_Struct* tmpCandidateEle;
   tmpCandidateEle = (Vertex_Struct*)GetNextListEle( CandidateList, 1 );
   while( tmpCandidateEle )
   {
      LeastDistance = tmpCandidateEle->DistanceToRoot;
      Inx = 0;
      while( !(AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NetNum == 0 &&
             AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NodeNum == 0) )
      {
         if( AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NetNum ==
            tmpCandidateEle->VertexId.NetNum &&
            AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NodeNum ==
            tmpCandidateEle->VertexId.NodeNum )
         {
            FoundFlag = ON;
            break;
         }

         if( AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].DistanceToRoot
            > LeastDistance )
         {
            IsLessThanFlag = ON;
            break;
         }
         Inx++;
      }
      if( FoundFlag == ON )
         if( AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].DistanceToRoot
            > LeastDistance )
            AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].DistanceToRoot
                                                   = LeastDistance;
      if( FoundFlag == OFF )
      {
         if(IsLessThanFlag == ON)
         {
            Vertex_Struct tmpSPVertex;
            Vertex_Struct sectmpSPVertex;
            tmpSPVertex = AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx];
            op_prg_mem_copy( tmpCandidateEle,
                     &(AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx]),
                     sizeof(Vertex_Struct) );
            while(!(AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NetNum == 0 &&
                 AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx].VertexId.NodeNum == 0) )
            {
               sectmpSPVertex = AreaStructure.QospfShortestPathTree[GrpId].SPVertex[i+1];
               AreaStructure.QospfShortestPathTree[GrpId].SPVertex[i+1] = tmpSPVertex;
               tmpSPVertex = sectmpSPVertex;
               Inx++;
            }
         }
         if( IsLessThanFlag == OFF )
            op_prg_mem_copy( tmpCandidateEle,
```

```
                                &(AreaStructure.QospfShortestPathTree[GrpId].SPVertex[Inx]),
                                sizeof(Vertex_Struct) );

                op_prg_mem_copy( tmpCandidateEle, &VertexW, sizeof(Vertex_Struct) );
                OspfCandidateAddPro( VertexW , CandidateList );
            }
            DeleteListEle(CandidateList, (char*)tmpCandidateEle, sizeof(Vertex_Struct) );
            tmpCandidateEle = (Vertex_Struct*)GetNextListEle( CandidateList, 1 );
        }
    }
}

/**
*** Function : QospfPkPro.
*** Input    : A pointer to Qospf packet that is obtained from the IP stream
*** Output   : This function will ckeck for the Qospf/Ospf packet type that it has recived and call the
***            appropriate functions accordingly.
**/
void
QospfPkPro( Packet *ReceivedQospfPkptr )
{
    int tmp_msg_type;
  op_pk_nfd_get(ReceivedQospfPkptr, "Msg_type", &tmp_msg_type);

  switch(tmp_msg_type)
    {
    case 1  :
            HelloPk_pro(ReceivedQospfPkptr);
        break;
    case 4  :
            VL  OPDL    printf("(%d.%d) Received OspfLsUpdate Packet \n",
                                RouterId.NetNum, RouterId.NodeNum );
            InstNoLsa++;
            op_stat_local_write(LSA_OUT_STAT, InstNoLsa);
            OspfLsUpdatePk_pro(ReceivedQospfPkptr);
        break;
    case 7  :
            InstNoLsa++;
            op_stat_local_write(LSA_OUT_STAT, InstNoLsa);
            QospfLsUpdatePk_pro(ReceivedQospfPkptr);
        break;
    case 8  :
            VL MPDL printf("ROUTER ID(%d.%d) : Mospf LS Update packet is obtained \n",
                        RouterId.NetNum, RouterId.NodeNum );
            InstNoLsa++;
            op_stat_local_write(LSA_OUT_STAT, InstNoLsa);
            MospfLsUpdatePk_pro(ReceivedQospfPkptr);
        break;
        /*
    case 2  : DataDescPk_pro(ReceivedQospfPkptr);
          break;
    case 3  : LsReqPk_pro(ReceivedQospfPkptr);
          break;
    case 5  : LsAckPk_pro(ReceivedQospfPkptr);
          break;
    default : break;
    */
  }
    op_pk_destroy( ReceivedQospfPkptr );
```

...
...

```
}

/**
*** Function : RsvpPkPro.
*** Input    : A pointer Rsvp packet that is obtained from the Rsvp stream
*** Output   : This function will ckeck for the Rsvp packet type that it has recived and call the
***            appropriate functions accordingly.
**/
void
RsvpPkPro( Packet * ReceivedRsvpPkptr )
{
      int tmpMsgType;
   op_pk_nfd_get(ReceivedRsvpPkptr, "Msg_Type", &tmpMsgType);

   switch(tmpMsgType) {
       case 1  :
                  printf("%d.%d Received a Path Message from Local Rsvp Deamon \n",
                              RouterId.NetNum, RouterId.NodeNum );
                  Path_Msg_pro(ReceivedRsvpPkptr);
             break;
       case 2  : Resv_Msg_pro(ReceivedRsvpPkptr);
              break;
       case 7  : ResvConf_Msg_pro(ReceivedRsvpPkptr);
              break;
       default : break;
   }
}
/*
void
CalculateNextHops( Vertex_Struct VertexW, Vertex_Struct VertexV , RouterLinkAdv WRouterLsa )
{

   int Hop_Count = 0;
      int j;
   if( !(VertexW.VertexId.NetNum == RouterId.NetNum &&
        VertexW.VertexId.NodeNum == RouterId.NodeNum) )
      if( IsNetwork(VertexW.VertexId) )
         for( j = 0; j < WRouterLsa.No_of_Links; j++ )
            if( WRouterLsa.Link_Info[j].LinkId.NetNum ==
                                    VertexV.VertexId.NetNum &&
               WRouterLsa.Link_Info[j].LinkId.NodeNum ==
                                    VertexV.VertexId.NodeNum )
            {
               VertexW.NextHopsList[Hop_Count] =
                           WRouterLsa.Link_Info[j].LinkData;
               Hop_Count++;
            }
      VL   SPFDL
      {
            printf("My Vertes Id is : %d.%d and my parent Vertex Id is : %d.%d\n",
                        VertexW.VertexId.NetNum, VertexW.VertexId.NodeNum,
                        VertexV.VertexId.NetNum, VertexV.VertexId.NodeNum );
            printf("NextHopsList is : \n");
            for( j = 0; j < Hop_Count; j++ )
            {
                  printf( "Hop Count is %d and the Next Hop is %d.%d \n", j,
                              VertexW.NextHopsList[j].NetNum, VertexW.NextHopsList[j].NodeNum );
            }
      }
```

```
        printf( "!!!!!!!!!!!!!!!   End CalculateNextHops   !!!!!!!!!!!!\n" );
}
*/

void
CalculateNextHops( Vertex_Struct VertexW, Vertex_Struct VertexV , RouterLinkAdv VRouterLsa )
{

    int Hop_Count = 0;
        int j;
        int FoundFlag = OFF;
    /*  if( !(VertexW.VertexId.NetNum == RouterId.NetNum &&  */
         /*  VertexW.VertexId.NodeNum == RouterId.NodeNum) )  */
        /*  if( IsNetwork(VertexW.VertexId) )  */
            for( j = 0; j < VRouterLsa.No_of_Links; j++ )
                    {
                if( VRouterLsa.Link_Info[j].LinkId.NetNum ==
                                VertexW.VertexId.NetNum &&
                    VRouterLsa.Link_Info[j].LinkId.NodeNum ==
                                VertexW.VertexId.NodeNum )
                {
                    VertexW.NextHopsList[Hop_Count] =
                            VRouterLsa.Link_Info[j].LinkData;
                    Hop_Count++;
                                FoundFlag = ON;
                }
                    }
                    if( FoundFlag = OFF )
                    {
                        VertexW.NextHopsList[Hop_Count] = VertexV.VertexId;
                        Hop_Count++;
                    }
        VL   SPFDL
        {
            printf("My Vertes Id is : %d.%d and my parent Vertex Id is : %d.%d\n",
                        VertexW.VertexId.NetNum, VertexW.VertexId.NodeNum,
                        VertexV.VertexId.NetNum, VertexV.VertexId.NodeNum );
            printf("NextHopsList is : \n");
            for( j = 0; j < Hop_Count; j++ )
            {
                printf( "Hop Count is %d and the Next Hop is %d.%d \n", j,
                            VertexW.NextHopsList[j].NetNum, VertexW.NextHopsList[j].NodeNum );
            }
        }
}
int
IsNetwork( IPAddress ipa )
{
        if(ipa.NetNum < 60)
            return 1;
        else
            return 0;
}

void
OspfCandidateAddPro( Vertex_Struct VertexV, ListL* CandidatesList )
{
        int i = 0;
        Vertex_Struct VertexW;
```

```
RouterLinkAdv WRouterLsa;
RouterLinkAdv VRouterLsa;
Vertex_Struct* CandidateEle;
Vertex_Struct* tmpCandEle;
int CandidateFoundFlag = OFF;

memset( &VRouterLsa, 0, sizeof(RouterLinkAdv) );

for(i = 0; i<= MAX_NO_LSAS; i++ )
{
    if( AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NetNum ==
    VertexV.VertexId.NetNum &&
    AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NodeNum ==
    VertexV.VertexId.NodeNum  )
    {
        op_prg_mem_copy( &AreaStructure.RouterLsaList[i], &VRouterLsa, sizeof(RouterLinkAdv) );
        break;
    }
}

for(i = 0; i < VRouterLsa.No_of_Links; i++ )
{
    if( VRouterLsa.Link_Info[i].Type == ROUTER ||
    VRouterLsa.Link_Info[i].Type == TRANSITNET )
    {
        int FoundFlag = OFF;
        int j = 0;
        op_prg_mem_copy( &VRouterLsa.Link_Info[i].LinkId, &VertexW.VertexId,
                         sizeof(IPAddress) );
        j = 0;
        while(!(AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NetNum == 0 &&
                AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NodeNum == 0) )
        {
            if( AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NetNum ==
                VertexW.VertexId.NetNum &&
                AreaStructure.RouterLsaList[j].LinkStateHeader.AdvRouter.NodeNum ==
                VertexW.VertexId.NodeNum  )
            {
                FoundFlag = ON;
                op_prg_mem_copy( &AreaStructure.RouterLsaList[j], &WRouterLsa,
                                 sizeof(RouterLinkAdv) );
                break;
            }
            j++;
        }
        if(FoundFlag == ON)
        {
            int WonSPF_Flag = OFF;
            j = 0;
            while( !(AreaStructure.ShortestPathTree.SPVertex[j].VertexId.NetNum == 0 &&
                     AreaStructure.ShortestPathTree.SPVertex[j].VertexId.NodeNum == 0 ) )
            {
                if( AreaStructure.ShortestPathTree.SPVertex[j].VertexId.NetNum ==
                    VertexW.VertexId.NetNum&&
                    AreaStructure.ShortestPathTree.SPVertex[j].VertexId.NodeNum ==
                    VertexW.VertexId.NodeNum )
                {
                    WonSPF_Flag = ON;
                    break;
```

```
    }
    j++;
}
if( WonSPF_Flag == OFF )
{
    if( VRouterLsa.Link_Info[i].TOSMetric != LSInfinity )
    {
        VertexW.DistanceToRoot = VertexV.DistanceToRoot +
                                    VRouterLsa.Link_Info[i].TOSMetric;
    }
    if( !IsListEmpty(CandidatesList) )
    {
        CandidateEle = (Vertex_Struct*)GetNextListEle( CandidatesList, 1 );
        while( CandidateEle )
        {
            if( CandidateEle->VertexId.NetNum == VertexW.VertexId.NetNum &&
                CandidateEle->VertexId.NodeNum == VertexW.VertexId.NodeNum )
            {
                CandidateFoundFlag = ON;
                break;
            }
            CandidateEle = (Vertex_Struct*)GetNextListEle( CandidatesList, 0 );
        }
    }
    if ( CandidateFoundFlag == ON )
    {
        if( CandidateEle->DistanceToRoot == VertexW.DistanceToRoot )
        {
            /*  CalculateNextHops( VertexW, VertexV, WRouterLsa );  */
            CalculateNextHops( VertexW, VertexV, VRouterLsa );
        }
        else if( CandidateEle->DistanceToRoot > VertexW.DistanceToRoot  )
        {
            CandidateEle->VertexId = VertexW.VertexId;
            CandidateEle->DistanceToRoot= VertexW.DistanceToRoot;
            /*  CalculateNextHops( VertexW, VertexV, WRouterLsa );  */
            CalculateNextHops( VertexW, VertexV, VRouterLsa );
        }
    }
    if( CandidateFoundFlag == OFF )
    {
        /*  CalculateNextHops( VertexW, VertexV, VRouterLsa );   */
        int Hop_Count = 0;
        int hopfound = OFF;
        int k;
        for( k = 0; k < VRouterLsa.No_of_Links; k++ )
        {
            if( VertexW.VertexId.NetNum == VRouterLsa.Link_Info[i].LinkId.NetNum &&
                VertexW.VertexId.NetNum == VRouterLsa.Link_Info[i].LinkId.NodeNum )
            {
                VertexW.NextHopsList[Hop_Count] = VRouterLsa.Link_Info[i].LinkData;
                Hop_Count++;
                hopfound = ON;
            }
        }
        if(hopfound == OFF )
        {
            VertexW.NextHopsList[Hop_Count] = VertexV.VertexId;
        }
```

```
                        VL  SPFDL  printf("VertexW.NextHopsList[Hop_Count] = %d.%d \n",
                                     VertexW.NextHopsList[Hop_Count].NetNum,
                                     VertexW.NextHopsList[Hop_Count].NodeNum );
                        AddListEle( CandidatesList, (char *)&VertexW, sizeof(Vertex_Struct) );
                        /*  CalculateNextHops( VertexW, VertexV, WRouterLsa );  */
                    }
                }
            }
        }
    }
}

void
OspfSPTreeBuild( ListL* CandidateList )
{
    int Inx, i;
    Vertex_Struct VertexW;
  if( !IsListEmpty( CandidateList ) )
  {
    int LeastDistance = 10000;
    int IsLessThanFlag = OFF;
    int FoundFlag = OFF;
    Vertex_Struct* tmpCandidateEle;
    tmpCandidateEle = (Vertex_Struct*)GetNextListEle( CandidateList, 1 );
    while( tmpCandidateEle )
    {
      LeastDistance = tmpCandidateEle->DistanceToRoot;
      Inx = 0;
      while( !(AreaStructure.ShortestPathTree.SPVertex[Inx].VertexId.NetNum == 0 &&
           AreaStructure.ShortestPathTree.SPVertex[Inx].VertexId.NodeNum == 0) )
      {
        if( AreaStructure.ShortestPathTree.SPVertex[Inx].VertexId.NetNum ==
          tmpCandidateEle->VertexId.NetNum &&
          AreaStructure.ShortestPathTree.SPVertex[Inx].VertexId.NodeNum ==
          tmpCandidateEle->VertexId.NodeNum )
        {
          FoundFlag = ON;
          break;
        }

        if( AreaStructure.ShortestPathTree.SPVertex[Inx].DistanceToRoot > LeastDistance )
        {
          IsLessThanFlag = ON;
          break;
        }
        Inx++;
      }
      if( FoundFlag == ON )
        if( AreaStructure.ShortestPathTree.SPVertex[Inx].DistanceToRoot > LeastDistance )
          AreaStructure.ShortestPathTree.SPVertex[Inx].DistanceToRoot = LeastDistance;
      if( FoundFlag == OFF )
      {
        if(IsLessThanFlag == ON)
        {
          Vertex_Struct tmpSPVertex;
          Vertex_Struct sectmpSPVertex;
          tmpSPVertex = AreaStructure.ShortestPathTree.SPVertex[Inx];
          op_prg_mem_copy( tmpCandidateEle, &(AreaStructure.ShortestPathTree.SPVertex[Inx]),
                  sizeof(Vertex_Struct) );
```

```
                        while(!(AreaStructure.ShortestPathTree.SPVertex[Inx].VertexId.NetNum == 0 &&
                                AreaStructure.ShortestPathTree.SPVertex[Inx].VertexId.NodeNum == 0) )
                /*  for(i = Inx; i <= MAX_NO_SPROUTERS; i++)  */
                {
                    sectmpSPVertex = AreaStructure.ShortestPathTree.SPVertex[i+1];
                    AreaStructure.ShortestPathTree.SPVertex[i+1] = tmpSPVertex;
                    tmpSPVertex = sectmpSPVertex;
                            Inx++;
                }
            }
            if( IsLessThanFlag == OFF )
                op_prg_mem_copy( tmpCandidateEle, &(AreaStructure.ShortestPathTree.SPVertex[Inx]),
                        sizeof(Vertex_Struct) );

            op_prg_mem_copy( tmpCandidateEle, &VertexW, sizeof(Vertex_Struct) );
            OspfCandidateAddPro( VertexW , CandidateList );
        }
        DeleteListEle(CandidateList, (char*)tmpCandidateEle, sizeof(Vertex_Struct) );
        tmpCandidateEle = (Vertex_Struct*)GetNextListEle( CandidateList, 1 );
    }
  }
}

void
RoutingTableCalc(Area_Struct* tmpAreaStructure, int* rte_table )
{
    int i = 1;
    while(!(tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum == 0 &&
                tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NodeNum == 0) )
    {
        int j = 0;
        if(!(tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NetNum == 0 &&
             tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NodeNum == 0) )
        {
            int k = 0;
            int RouteFoundFlag = OFF;
            for( k = 0; k < NoOfLinks; k++ )
            {
                /*
                if( tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NetNum ==
                    LinksInfo[k].IntfIPAddr.NetNum &&
                    tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NodeNum ==
                    LinksInfo[k].IntfIPAddr.NodeNum )
                */
                if( tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum ==
                    LinksInfo[k].NeighborRtrId.NetNum ||
                    tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum ==
                    LinksInfo[k].NghborNetNum )
                {
                    rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum] = k+1;
                    RouteFoundFlag = ON;
                    break;
                }
            }
            if( RouteFoundFlag == OFF )
            {
                if( rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NetNum] != 0 )
                {
                    rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum] =
```

```
                                                rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NetNum];
                        }
                    }
                }
                i++;
        }

        if( !IS_AREA_BORDER_ROUTER )
                rte_table[DEF_STREAM] = rte_table[AreaBorderRouters[AreaId].NetNum];
        else
                rte_table[DEF_STREAM] = UNDEFINED;

        rte_table[net_num0] = 0;

        VL  SPFDL
        {
                printf("---------- (%d.%d) Routing Table -------------- \n",
                        RouterId.NetNum, RouterId.NodeNum );
                for(i = 0; i < 200; i++ )
                {
                        if( rte_table[i] != 0 )
                        {
                                printf("Route to the net %d is through the stream %d \n", i, rte_table[i] );
                        }
                }
        }
}
void
QospfRouterInitialize(int ini)
{
        int i;

        VL  ILDL        printf("Router Id is : %d.%d\n", RouterId.NetNum, RouterId.NodeNum );

        for(i = ini; i < NoOfLinks; i++)
        {
                ip_id = op_topo_assoc(ip_objid, OPC_TOPO_ASSOC_OUT, OPC_OBJTYPE_QUEUE, i);
                tx_id = op_topo_assoc( ip_id, OPC_TOPO_ASSOC_OUT, OPC_OBJTYPE_PTTX, 0 );
                dup_lk_id = op_topo_assoc( tx_id, OPC_TOPO_ASSOC_OUT, OPC_OBJTYPE_LKSIMP, 0 );
                rx_id = op_topo_assoc( dup_lk_id, OPC_TOPO_ASSOC_OUT, OPC_OBJTYPE_PTRX, 0 );
                ip_id = op_topo_assoc( rx_id, OPC_TOPO_ASSOC_OUT, OPC_OBJTYPE_QUEUE, 0 );
                qospf_id = op_topo_assoc( ip_id, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 13 );

                /*  op_ima_obj_attr_get ( ip_id, "Indep_Rou", &LinksInfo[i].IsNghborRou );  */
                /*  op_ima_obj_attr_get ( qospf_id, "AreaBorderRouter", &LinksInfo[i].IsNghborRou );  */
                /*  op_ima_obj_attr_get ( qospf_id, "IsRouter", &LinksInfo[i].IsNghborRou );   */
                LinksInfo[i].IsNghborRou = 1;
                if( LinksInfo[i].IsNghborRou == 1 )
                {
                        LinksInfo[i].Types = ROUTER;
                        op_ima_obj_attr_get( ip_id, "net_num0", &LinksInfo[i].NeighborRtrId.NetNum );
                        op_ima_obj_attr_get( ip_id, "node_num0", &LinksInfo[i].NeighborRtrId.NodeNum );
                }
                else
                {
                        LinksInfo[i].Types = STUBNET;
                        op_ima_obj_attr_get( ip_id, "net_num0", &LinksInfo[i].NeighborRtrId.NetNum );
                        IS_DESIGNATED_ROUTER = 1;
                }
```

```
                LinksInfo[i].Bandwidth = LINKBANDWIDTH;
                LinksInfo[i].Delay = LINKDELAY;
                LinksInfo[i].BufferSize = LINKBUFFERSIZE;
                switch(i)
                {
                        case 0  : LinksInfo[i].IntfIPAddr.NetNum = net_num0;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num0;
                                  op_ima_obj_attr_get( my_id, "Link1Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 1  : LinksInfo[i].IntfIPAddr.NetNum = net_num1;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num1;
                                  op_ima_obj_attr_get( my_id, "Link2Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 2  : LinksInfo[i].IntfIPAddr.NetNum = net_num2;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num2;
                                  op_ima_obj_attr_get( my_id, "Link3Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 3  : LinksInfo[i].IntfIPAddr.NetNum = net_num3;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num3;
                                  op_ima_obj_attr_get( my_id, "Link4Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 4  : LinksInfo[i].IntfIPAddr.NetNum = net_num4;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num4;
                                  op_ima_obj_attr_get( my_id, "Link4Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 5  : LinksInfo[i].IntfIPAddr.NetNum = net_num5;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num5;
                                  op_ima_obj_attr_get( my_id, "Link5Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 6  : LinksInfo[i].IntfIPAddr.NetNum = net_num6;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num6;
                                  op_ima_obj_attr_get( my_id, "Link6Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 7  : LinksInfo[i].IntfIPAddr.NetNum = net_num7;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num7;
                                  op_ima_obj_attr_get( my_id, "Link7Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 8  : LinksInfo[i].IntfIPAddr.NetNum = net_num8;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num8;
                                  op_ima_obj_attr_get( my_id, "Link8Cost", &LinksInfo[i].CostMetric );
                                  break;
                        case 9  : LinksInfo[i].IntfIPAddr.NetNum = net_num9;
                                  LinksInfo[i].IntfIPAddr.NodeNum = node_num9;
                                  op_ima_obj_attr_get( my_id, "Link9Cost", &LinksInfo[i].CostMetric );
                                  break;
                }
        }
}

/**
***     Qospf Routing Table calculation functions
**/
/*
void
QOSPFCalcTree(void )
{
        if( IsListEmpty(&CandidateList) ) return;
```

```
    else
      {
      MoveClosestVertexToSPTList();

      AddVertexVNeighborCandList();

      if(VertexV->VertexType == ROUTER)
            {
        VertexVResLsa = VertexV->VertexResLsa;
        VertexVRraLsa = VertexV->VertexRraLsa;
        if(!VertexVResLsa)
                {
          MoveClosestVertexToSPTList();
        }
        else
                {
          for(i = 0; i < NoOfLinks; i++)
                    {
            AvailableBWOnLink =
                VertexVResLsa->ResLinkInfo.Bandwidth - ReservedLinkBandwidth[i];
            if(AvailableBWOnLink >= ReqiuredBWOnLink)
                        {
              LinkSPFPro( &VertexVResLsa, i );
                        }
                    }
                }
      QOSPFTreeCalc();
    }
      }
}
void
MoveClosestVertexToSPTList(void)
{
      VertexEle = (Vertex_Struct *) GetNextListEle( &CandidateList, 1)
      VertexV = VertexEle;
      while( VertexEle )
      {
            #ifdef ROUTE_PINNING_IS_USED
                  if( VertexEle-> Reserved == TRUE )
                  {
                        VertexV = VertexEle;
                        return;
                  }
            #endif
            if( VertexEle->Cost < VertexV->Cost )
                  VertexV = VertexEle;
            elseif( VertexEle->Cost = VertexV->Cost )
            {
                  if( (VertexEle->VertexType == TRANSITNETWORK &&
                         VertexV->VertexType != TRANSITNETWORK) )
                        VertexV = VertexEle;
                  elseif( VertexEle->VertexType == TRANSITNETWORK &&
                           VertexV->VertexType == TRANSITNETWORK) &&
                           VertexEle->VertexId > VertexV->VertexId) )
                           VertexV = VertexEle;
            }
            VertexEle = (Vertex_Struct *) GetNextListEle( &CandidateList, 1)
      }
```

```
        DeleteListEle( &CandidateList, &VertexV, sizeof(Vertex_Struct) );
        AddListEle( &ShortestPathTreeList, &VertexV, sizeof(Vertex_Struct) );
}
void
LinkSPFPro( &VertexVResLsa, i )
{
        reslsa* VertexVResLsa;
        Vertex_Struct* VertexW
        VertexW = getneighbor(LinkId);
        if( (VertexW->ResLsa.LinkId == VertexV->VertexId)||
            (VertexW->ResLsa.ResLsaHeader.LSAge == LSMaxAge)
                (VertexW->ResLsa.ResLsaHeader.Options.MCBit == 0)
                (VertexW->ResLsa.ResLsaHeader.Options.QospfBit == 0) )
                return;
        elseif( VertexV.VertexType == ROUTER )
                DelayAssocLink[i] = VertexV->ResLsa.ResLinkInfo[i].LinkDelay;
        elseif( VertexV.VertexType == TRANSITNETWORK )
                DelayAssocLink[i] = 0;

        DelayToVertexW = DelayToVertexV + DelayAssocLink[i];

        if(IsVertexWInCandList()){
                if(DelayToVertexW < VertexW->Delay)
                {
                         Do the processing of the specified in mospf rfc

                }
                elseif(DelayToVertexW == VertexW->Delay)
                {
                         Do the processing of the specified in mospf rfc
                }
                elseif(DelayToVertexW > VertexW->Delay){
                        return;
                }
        }
        else{
                AddListEle( &CandidateList, &VertexW, sizeof(Vertex_Struct) );
                Do the processing of the specified in mospf rfc
        }
}

void
MospfRouteCalc( &ShortestPathTreeList )
{
        int i = 1;
        while(!(tmpAreaStructure->ShortestPathTreeList.SPVertex[i].VertexId.NetNum == 0 &&
                        tmpAreaStructure->ShortestPathTreeList.SPVertex[i].VertexId.NodeNum == 0) )
        {
                int j = 0;
                if(!(tmpAreaStructure->ShortestPathTreeList.SPVertex[i].NextHopsList[j].NetNum == 0 &&
                         tmpAreaStructure->ShortestPathTreeList.SPVertex[i].NextHopsList[j].NodeNum == 0) )
                {
                        int k = 0;
                        int RouteFoundFlag = OFF;
                        for( k = 0; k < NoOfLinks; k++ )
                        {
                                if( tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NetNum ==
                                        LinksInfo[k].IntfIPAddr.NetNum &&
                                        tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NodeNum ==
```

```
                        LinksInfo[k].IntfIPAddr.NodeNum )
                    if( tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum ==
                        LinksInfo[k].NeighborRtrId.NetNum ||
                        tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum ==
                        LinksInfo[k].NghborNetNum )
                    {
                        rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum] = k+1;
                        RouteFoundFlag = ON;
                        break;
                    }
                }
                if( RouteFoundFlag == OFF )
                {
                    if( rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NetNum] != 0 )
                    {
                        rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].VertexId.NetNum] =
                                rte_table[tmpAreaStructure->ShortestPathTree.SPVertex[i].NextHopsList[j].NetNum];
                    }
                    else printf("No Route to this host\n");
                }
            }
            i++;
        }
}
*/

void
PrintLinkStateHdr( linkstateheader lshdr )
{
    printf("Link State Header is : \n");
    printf("LsAge         : %d\n", lshdr.LsAge );
    printf("LsType        : %d\n", lshdr.LsType );
    printf("LinkStateId   : %d.%d\n",
            lshdr.LinkStateId.NetNum, lshdr.LinkStateId.NodeNum );
    printf("AdvRouter     : %d.%d\n",
            lshdr.AdvRouter.NetNum, lshdr.AdvRouter.NodeNum );
    printf("ThisAreaId    : %d\n", lshdr.ThisAreaId );
    printf("LSSeqNo       : %f\n", lshdr.LSSeqNo );
    printf("LSChecksum    : %f\n", lshdr.LSChecksum );
    printf("Length        : %f\n", lshdr.Length );
}
void
PrintLinkInfo( linkinfo* LI, int Nlinks )
{
    int i;
    printf("Router Link Information \n");
    for(i = 0; i < Nlinks; i++ )
    {
        printf("LinkId        : %d.%d\n",
                LI[i].LinkId.NetNum, LI[i].LinkId.NodeNum );
        printf("LinkData      : %d.%d\n",
                LI[i].LinkData.NetNum, LI[i].LinkData.NodeNum );
        printf("Type          : %d\n", LI[i].Type );
        printf("NoOfTOS       : %d\n", LI[i].NoOfTOS );
        printf("TOSMetric     : %d\n", LI[i].TOSMetric );
    }
}
void
PrintAttRtrs( IPAddress* ip )
```

...
...

```
{
    int i = 0;
    printf("Attached Routers are : " );
    while( !(ip[i].NetNum == 0 && ip[i].NodeNum == 0) )
    {
        printf("%d.%d, ", ip[i].NetNum, ip[i].NodeNum );
        i++;
    }
}
void
PrintTosInfo( tosinfo* TI )
{
    int i = 0;
    printf("TOS Info is : \n");
    while( !(TI[i].TOS == 0 && TI[i].metric == 0) )
    {
        printf("TI[i].TOS     : %d\n", TI[i].TOS );
        printf("TI[i].Metric  : %f\n", TI[i].metric );
        i++;
    }
}
void
PrintResLsaHdr( reslsaheader rlshdr )
{
    printf(" Resource Link State Header is : \n");
    printf("LsAge         : %d\n", rlshdr.LsAge );
    printf("Options       : %d\n", rlshdr.Options );
    printf("LSA_type      : %d\n", rlshdr.LSA_type );
    printf("LinkStateId   : %d.%d\n",
                rlshdr.LinkStateId.NetNum, rlshdr.LinkStateId.NodeNum );
    printf("AdvRouter     : %d.%d\n",
                rlshdr.AdvRouter.NetNum, rlshdr.AdvRouter.NodeNum );
    printf("ThisAreaId    : %d\n", rlshdr.ThisAreaId );
    printf("LSSeqNo       : %f\n", rlshdr.LSSeqNo );
    printf("LSChecksum    : %f\n", rlshdr.LSChecksum );
    printf("Length        : %f\n", rlshdr.Length );
    printf("rtype         : %f\n", rlshdr.rtype );
    printf("No_of_Links   : %f\n", rlshdr.No_of_Links );
}
void
PrintResLinkInfo( reslinkinfo* RLI )
{
    int i = 0;
    printf("Resource Link Information \n");
    while( !(RLI[i].LinkId.NetNum == 0 && RLI[i].LinkId.NodeNum == 0) )
    {
        printf("LinkId         : %d.%d\n",
                RLI[i].LinkId.NetNum, RLI[i].LinkId.NodeNum );
        printf("LinkData       : %d.%d\n",
                RLI[i].LinkData.NetNum, RLI[i].LinkData.NodeNum );
        printf("Type           : %d\n", RLI[i].Type );
        printf("TOSMetric      : %d\n", RLI[i].TOSMetric );
        printf("LinkDelay      : %f\n", RLI[i].LinkDelay );
        printf("Bandwidth      : %f\n", RLI[i].Bandwidth );
        printf("BufferSize     : %f\n", RLI[i].BufferSize );
        i++;
    }
}
void
```

```
PrintOpaqueLsaHdr( opaquelsaheader olshdr )
{
	printf(" Reservation Link State Header is : \n");
	printf("LsAge          : %d\n", olshdr.LsAge );
	printf("Options        : %d\n", olshdr.Options );
	printf("LSA_type       : %d\n", olshdr.LSA_type );
	printf("OpaqueType     : %d\n", olshdr.OpaqueType );
	printf("OpaqueId       : %d\n", olshdr.OpaqueId );
	printf("AdvRouter      : %d.%d\n",
			olshdr.AdvRouter.NetNum, olshdr.AdvRouter.NodeNum );
	printf("ThisAreaId     : %d\n", olshdr.ThisAreaId );
	printf("LSSeqNo        : %f\n", olshdr.LSSeqNo );
	printf("LSChecksum     : %f\n", olshdr.LSChecksum );
	printf("Length         : %f\n", olshdr.Length );
	printf("FloddingScope: %d\n", olshdr.FloddingScope );
}
void
PrintLinkResvInfo( linkresvinfo* RLI, int Nlinks )
{
	int i;
	printf("Reserved Link Information \n");
	for(i = 0; i < Nlinks; i++ )
	{
		printf("LinkId              : %d.%d\n",
				RLI[i].LinkId.NetNum, RLI[i].LinkId.NodeNum );
		printf("LinkData            : %d.%d\n",
				RLI[i].LinkData.NetNum, RLI[i].LinkData.NodeNum );
		printf("Type                : %d\n", RLI[i].Type );
		printf("PinFlag             : %d\n", RLI[i].PinFlag );
		printf("ReservedBandwidth    : %f\n", RLI[i].ReservedBandwidth );
		printf("ReservedBufferSize   : %f\n", RLI[i].ReservedBufferSize );
	}
}
void
PrintVertexInfo( vertexinfo* VI )
{
	int i = 0;
	printf("Vertex Information\n");
	while( !(VI[i].VertexId.NetNum == 0 &&VI[i].VertexId.NodeNum == 0) )
	{
		printf("VertexType : %d\n", VI[i].VertexType );
		printf("VertexId   : %d.%d\n", VI[i].VertexId.NetNum,
				VI[i].VertexId.NodeNum);
		i++;
	}
}
void
PrintLsa( char * MsgPk, int Type )
{
	RouterLinkAdv * RouterLsa;
	NetworkLinkAdv * NetworkLsa;
	SummaryLinkAdv * SummaryLsa;
	reslsa * ResourceLsa;
	rralsa * RraLsa;
	grpmbrlsa * GrpMbrShipLsa;

  switch( Type )
  {
	case ROUTERLSA :
```

...
...

```
        printf("\n---------------- (%d.%d)Router Lsa-------------------------\n",
                        RouterId.NetNum, RouterId.NodeNum );
        RouterLsa = (RouterLinkAdv *)MsgPk;
            printf("RouterLsa->LinkStateHeader.AdvRouter = %d.%d\n",
                        RouterLsa->LinkStateHeader.AdvRouter.NetNum,
                        RouterLsa->LinkStateHeader.AdvRouter.NodeNum );
        PrintLinkStateHdr( RouterLsa->LinkStateHeader );
        printf( "No_of_Links : %d\n", RouterLsa->No_of_Links );
        PrintLinkInfo( RouterLsa->Link_Info, RouterLsa->No_of_Links );
      break;
      case NETWORKLSA :
        printf("\n----------------Network Lsa------------------------\n ");
        NetworkLsa = (NetworkLinkAdv *)MsgPk;
        PrintLinkStateHdr( NetworkLsa->LinkStateHeader );
        printf( "ThisNetworkMask : %d\n", NetworkLsa->ThisNetworkMask );
        PrintAttRtrs( NetworkLsa->AttachedRouter );
      break;
      case SUMMARYLSA :
        printf("\n----------------Summary Lsa------------------------\n");
        SummaryLsa = (SummaryLinkAdv *)MsgPk;
        PrintLinkStateHdr( SummaryLsa->LinkStateHeader );
        printf( "ThisNetworkMask : %d\n", SummaryLsa->ThisNetworkMask );
        PrintTosInfo( SummaryLsa->Tos_Info );
      break;
      case RESLSA :
        printf("\n----------------Resource Lsa-----------------------\n");
        ResourceLsa = (reslsa *)MsgPk;
        PrintResLsaHdr( ResourceLsa->ResLsaHeader );
        PrintResLinkInfo( ResourceLsa->ResLinkInfo );
      break;
      case RRALSA :
        printf("\n----------------Rra Lsa----------------------------\n");
        RraLsa = (rralsa *)MsgPk;
        PrintOpaqueLsaHdr( RraLsa->OpaqueLsaHeader );
            printf("DestAddr : %d.%d\n",
                  RraLsa->DestAddr.NetNum, RraLsa->DestAddr.NodeNum );
            printf("SourceAddr : %d.%d\n",
                  RraLsa->SourceAddr.NetNum, RraLsa->SourceAddr.NodeNum );
            printf("No_of_Links : %d\n", RraLsa->No_of_Links );
            PrintLinkResvInfo( RraLsa->LinkResvInfo, RraLsa->No_of_Links );
      break;
      case GRPMBRSHIPLSA :
        printf("\n----------------Group Membership Lsa----------------\n");
        GrpMbrShipLsa = (grpmbrlsa *)MsgPk;
        PrintLinkStateHdr( GrpMbrShipLsa->LinkStateHeader );
            PrintVertexInfo( GrpMbrShipLsa->VertexInfo );
      break;
      default :
        printf("\n----------------Unrecognized Message----------------\n");
        VL printf("Could not Recognize the Message\n");
      break;
    }
    printf("-----------------------------------------------------\n");
}
```

...
...

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**

```
int i;
my_id = op_id_self();

IS_DESIGNATED_ROUTER = 0;
IS_AREA_BORDER_ROUTER = 0;

op_ima_obj_attr_get( my_id, "AreaId", &AreaId );
op_ima_obj_attr_get( my_id, "AreaBorderRouter", &IS_AREA_BORDER_ROUTER );
op_ima_obj_attr_get( my_id, "DesignatedRouter", &IS_DESIGNATED_ROUTER );

ip_objid   = op_topo_assoc( my_id, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 0 );
rsvp_objid = op_topo_assoc( my_id, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 1 );
igmp_objid = op_topo_assoc( my_id, OPC_TOPO_ASSOC_OUT, OPC_OBJMTYPE_MODULE, 2 );

/**
***  Using this IP layer Object Id get all the required attributes from the IP layer
**/
op_ima_obj_attr_get( ip_objid, "net_num0", &net_num0 );
op_ima_obj_attr_get( ip_objid, "node_num0", &node_num0 );
op_ima_obj_attr_get( ip_objid, "net_num1", &net_num1 );
op_ima_obj_attr_get( ip_objid, "node_num1", &node_num1 );
op_ima_obj_attr_get( ip_objid, "net_num2", &net_num2 );
op_ima_obj_attr_get( ip_objid, "node_num2", &node_num2 );
op_ima_obj_attr_get( ip_objid, "net_num3", &net_num3 );
op_ima_obj_attr_get( ip_objid, "node_num3", &node_num3 );
op_ima_obj_attr_get( ip_objid, "net_num4", &net_num4 );
op_ima_obj_attr_get( ip_objid, "node_num4", &node_num4 );
op_ima_obj_attr_get( ip_objid, "net_num5", &net_num5 );
op_ima_obj_attr_get( ip_objid, "node_num5", &node_num5 );
op_ima_obj_attr_get( ip_objid, "net_num6", &net_num6 );
op_ima_obj_attr_get( ip_objid, "node_num6", &node_num6 );
op_ima_obj_attr_get( ip_objid, "net_num7", &net_num7 );
op_ima_obj_attr_get( ip_objid, "node_num7", &node_num7 );
op_ima_obj_attr_get( ip_objid, "net_num8", &net_num8 );
op_ima_obj_attr_get( ip_objid, "node_num8", &node_num8 );
op_ima_obj_attr_get( ip_objid, "net_num9", &net_num9 );
op_ima_obj_attr_get( ip_objid, "node_num9", &node_num9 );
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 );

/*
```

```
if( !Indep_Rou )
      NoOfLinks++;
*/

memset( LinksInfo, 0, sizeof(Linkinfo)*NoOfLinks );

RouterId.NetNum = net_num0;
RouterId.NodeNum = node_num0;

if(IS_AREA_BORDER_ROUTER)
{
      AreaBorderRouters[AreaId] = RouterId;
}

QospfRouterInitialize(0);

op_intrpt_schedule_self( op_sim_time()+2, 0 );
op_intrpt_schedule_self( op_sim_time()+2, 1 );
op_intrpt_schedule_self( op_sim_time()+1, 2 );
op_intrpt_schedule_self( op_sim_time()+20, 3 );
op_intrpt_schedule_self( op_sim_time()+1, 4 );

NoOfResLsa = 0;
NoOfRraLsa = 0;
NoOfGrpMbrLsa = 0;
NoOfRouterLsa = 0;
NoOfNetworkLsa = 0;
NoOfSummaryLsa = 0;
NoOfNeighborElements = 0;

ListInit( &NeighborList );

send_ls_update_pk = 0;
send_grp_mbr_lsa = 0;
print_time = 0;
TotalOspfCalcTime = 0;
TotalMospfCalcTime = 0;
TotalQospfCalcTime = 0;
TotalDijCalcTime = 0;
TotalNoOfTimesOspfCalcDone = 0;
TotalNoOfTimesMospfCalcDone = 0;
TotalNoOfTimesQospfCalcDone = 0;
TotalNoDijCalcDone = 0;
VL  ILDL
{
      printf("--------------  INIT BLOCK PROCESSING  ----------------\n");
      printf("Router(%d.%d) Link Information \n", RouterId.NetNum, RouterId.NodeNum );
      printf("Number of Links is %d, ", NoOfLinks);
      for( i = 0; i < NoOfLinks; i++ )
      {
            printf("Neighbors Address is %d.%d\n",
                        LinksInfo[i].NeighborRtrId.NetNum, LinksInfo[i].NeighborRtrId.NodeNum );
            printf(" Interface Address is %d.%d\n",
                        LinksInfo[i].IntfIPAddr.NetNum, LinksInfo[i].IntfIPAddr.NodeNum );
            printf("Link Cost is %d\n", LinksInfo[i].CostMetric );
      }
      printf("------------  END INIT BLOCK PROCESSING  --------------\n");
}
```

transition **init -> idle**			
attribute	*value*	*type*	*default value*
name	tr_0	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_18	string	tr
condition	ARRIVAL	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> BCOspfLsa**			
attribute	*value*	*type*	*default value*
name	tr_70	string	tr
condition	send_ospf_lsa	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> BCQospfLsa**			
attribute	*value*	*type*	*default value*
name	tr_76	string	tr
condition	send_qospf_lsa	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> hello_pks**			
attribute	*value*	*type*	*default value*
name	tr_89	string	tr
condition	hellopk_send	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> BCLsReqPk**			
attribute	*value*	*type*	*default value*
name	tr_96	string	tr
condition	send_lsreq	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> BCdatadescpk**			
attribute	*value*	*type*	*default value*
name	tr_105	string	tr
condition	send_datadesc	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> BCMospfLsa**			
attribute	*value*	*type*	*default value*
name	tr_113	string	tr
condition	send_mospf_lsa	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> mospfspfcalc**			
attribute	*value*	*type*	*default value*
name	tr_114	string	tr
condition	do_mospf_calc	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> qospfspfcalc**			
attribute	*value*	*type*	*default value*
name	tr_115	string	tr
condition	do_qospf_calc	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

...
...

enter execs **qospfspfcalc**

```
{
int s_net;
int GrpId;
Vertex_Struct* VertexV;
ListL* QospfCandidatesList;
double qospfcalc_time = 0;
long diff;
struct timeval enter, leave;
double tmp_sim_time;
FILE* fp;

gettimeofday(&enter, (struct timezone*)0);
tmp_sim_time = op_sim_time();

if(do_qospf_calc)
{
      s_net = *((int*) op_ima_obj_svar_get(ip_objid, "SourceNet"));
      GrpId = *((int*) op_ima_obj_svar_get(ip_objid, "GrpId"));
}
else
{
      s_net = 0;
      GrpId = 0;
}
ListInit( QospfCandidatesList );

CandListInt( s_net, QospfCandidatesList );

if( !IsListEmpty( QospfCandidatesList ) )
{
      VertexV = (Vertex_Struct*) GetNextListEle( QospfCandidatesList, 1 );
}

QospfCandAddPro( GrpId, VertexV, QospfCandidatesList );
QospfSPFTreeCalc( GrpId, QospfCandidatesList );
MCRoutetableCalc( GrpId );

VL  QSPFDL
{
      printf("Router(%d.%d) doing the Qospf Dijkstra Calculation\n", RouterId.NetNum, RouterId.NodeNum );
}

qospfcalc = 0;
detupstrnode = 1;

TotalNoOfTimesQospfCalcDone++;
TotalNoDijCalcDone++;

if(op_sim_time() > print_time + 50*Tracking_Interval)
{
   fp = fopen("/export/home/workesys/QospfPro/top_layer/DijCalc", "a" );
   if(!fp)
      op_sim_end("Unable to open the file DijCalc....  ","","","");
      fprintf(fp, "Average time spent by Router %d.%d for Dijkstra's Calculation is %ld. "
           "Current Simulation time is %f \n", RouterId.NetNum, RouterId.NodeNum,
           TotalDijCalcTime/TotalNoDijCalcDone, op_sim_time() );
   fclose(fp);
```

```
    print_time = op_sim_time();
}
}
```

transition **qospfspfcalc -> upstr_node**			
attribute	*value*	*type*	*default value*
name	tr_92	string	tr
condition	detupstrnode	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **dabra**

```
send_dabra = 0;
```

transition **dabra -> idle**			
attribute	*value*	*type*	*default value*
name	tr_104	string	tr
condition	default	string	
executive		string	
color	RGB203	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**

```
/**
***  The Arrival state will check for the packet protocol Id field that is in the incoming IP A
***  datagram. Depending on the protocol ID the packet is known as either RSVP or QOSPF packet
***  and the appropriate functions are called
**/
```

...
...

```
in_strm = op_intrpt_strm();

ReceivedPkptr = op_pk_get( in_strm );
if(in_strm == IPSTREAM )
{
	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 );
}

/*
VL	DL	printf( "Router(%d.%d). A Packet is obtained from the router %d.%d through the stream %d "
				"with the protocol Id of %d \n", RouterId.NetNum, RouterId.NodeNum,
				 obt_pk_srcnet, obt_pk_srcnode, ip_strm, protocolid );
*/
switch (in_strm)
{
	case IGMPSTREAM :
		printf("--------- (%d.%d) Igmp Packet Arrived -------- \n",
				RouterId.NetNum, RouterId.NodeNum );
		send_grp_mbr_lsa = TRUE;
	break;
	case RSVPSTREAM :
	break;
	default :
	break;
}


switch(protocolid)
{
	case 89  :
		op_pk_nfd_get(ReceivedPkptr, "data", &ReceivedQospfPkptr);
		QospfPkPro( ReceivedQospfPkptr );
	break;
	case 46  :
		printf("Recived an RSVP packet \n");
		op_pk_nfd_get(ReceivedPkptr, "data", &ReceivedRsvpPkptr);
		RsvpPkPro( ReceivedRsvpPkptr );
	break;
	default  :
	break;
}

op_pk_destroy( ReceivedPkptr );
```

transition **arr -> ospfspfcalc**			
attribute	*value*	*type*	*default value*
name	tr_39	string	tr
condition	ospfspfcalc	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **arr -> idle**			
attribute	*value*	*type*	*default value*
name	tr_50	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **mospfspfcalc**

```
{
int s_net;
int GrpId;
ListL* MospfCandidatesList;
Vertex_Struct* VertexV;
double mospfcalc_time = 0;
double tmp_sim_time;
long diff;
struct timeval enter, leave;
FILE* fp;

tmp_sim_time = op_sim_time();

gettimeofday(&enter, (struct timezone*)0);
if(do_mospf_calc)
{
	s_net = *((int*) op_ima_obj_svar_get(ip_objid, "SourceNet"));
	GrpId = *((int*) op_ima_obj_svar_get(ip_objid, "GrpId"));
}
else
{
	s_net = 0;
	GrpId = 0;
}
VL	MCSPFDL
{
	printf("Doing Mospf Dijikstra's Algorithm \n");
}

ListInit( MospfCandidatesList );

CandListInt(s_net, MospfCandidatesList );

if( !IsListEmpty( MospfCandidatesList ) )
{
	VertexV = (Vertex_Struct*) GetNextListEle( MospfCandidatesList, 1 );
}

MospfCandAddPro( GrpId, VertexV, MospfCandidatesList );
```

...
...

```
MospfSPFTreeCalc( GrpId, MospfCandidatesList );
MCRoutetableCalc( GrpId );

detupstrnode = 1;
mospfspfcalc = 0;

TotalNoOfTimesMospfCalcDone++;
TotalNoDijCalcDone++;

gettimeofday(&leave, (struct timezone*)0);

if( enter.tv_usec > leave.tv_usec)
      diff = (((leave.tv_sec - 1) - enter.tv_sec)*1000000) + (leave.tv_usec + 1000000) - enter.tv_usec;
else
      diff = ((leave.tv_sec - enter.tv_sec)*1000000) + leave.tv_usec - enter.tv_usec;

TotalMospfCalcTime = TotalMospfCalcTime + diff;
TotalDijCalcTime = TotalDijCalcTime + diff;
if(op_sim_time() > print_time + 50*Tracking_Interval)
{
      fp = fopen("/export/home/workesys/QospfPro/top_layer/DijCalc", "a" );
      if(!fp)
            op_sim_end("Unable to open the file DijCalc....  ","","","");
      fprintf(fp, "Average time spent by Router %d.%d for Dijkstra’s Calculation is %ld. "
                  "Current Simulation time is %f \n", RouterId.NetNum, RouterId.NodeNum,
                  TotalDijCalcTime/TotalNoDijCalcDone, op_sim_time() );
      fclose(fp);

      print_time = op_sim_time();
}

if(op_sim_time() > 60)
{
      fp = fopen("/export/home/workesys/QospfPro/top_layer/DijCalc", "a" );
      if(!fp)
            op_sim_end("Unable to open the file DijCalc....  ","","","");
      fprintf(fp, "Total Number of Dijkstra’s done by Router %d.%d is %d. Total time"
                  "spent for these calculation is %ld\n", RouterId.NetNum, RouterId.NodeNum,
                  TotalNoDijCalcDone, TotalDijCalcTime );
}
}
```

transition **mospfspfcalc -> upstr_node**			
attribute	*value*	*type*	*default value*
name	tr_100	string	tr
condition	detupstrnode	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

unforced state **st_12**			
attribute	*value*	*type*	*default value*
name	st_12	string	st
enter execs	(empty)	textlist	(empty)

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

forced state **ospfspfcalc**

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

enter execs **ospfspfcalc**

```
int backup_rte_table[501];
ListL CandidateList;
Vertex_Struct* CandidateEle;
Vertex_Struct VertexV;
Vertex_Struct VertexW;
int i;
int CandidateFoundFlag = OFF;
int Inx;
int *rte_table;
double ospfcalc_time = 0;
double tmp_sim_time;
struct timeval enter, leave;
long diff;
FILE* fp;

VL  SPFDL   printf("-------  Unicast Dijikstra's algorithm (%d.%d)    --------- \n",
                        RouterId.NetNum, RouterId.NodeNum);

gettimeofday(&enter, (struct timezone*)0);
tmp_sim_time = op_sim_time();

rte_table = (int *)op_ima_obj_svar_get(ip_objid, "rte_tab" );

op_prg_mem_copy( rte_table, backup_rte_table, sizeof(int)*501 );

memset( &(AreaStructure.ShortestPathTree), 0, sizeof( ShortestPathTree_Struct ) );

ListInit( &CandidateList );

AreaStructure.ShortestPathTree.SPVertex[0].VertexId = RouterId;
AreaStructure.ShortestPathTree.SPVertex[0].DistanceToRoot = 0;

VertexV.VertexId = RouterId;
VertexV.DistanceToRoot = 0;
memset( VertexV.NextHopsList, 0, sizeof(IPAddress)*MAX_NO_NEIGHBORS );
memset( &VertexW, 0, sizeof(Vertex_Struct) );

OspfCandidateAddPro( VertexV, &CandidateList );
OspfSPTreeBuild( &CandidateList );

VL SPFDL
{
     printf("~~~~~~~~~~~~~~~  Shortest Path Information ~~~~~~~~~~~~~~~~~~~~\n");
     i = 0;
     while( !(AreaStructure.ShortestPathTree.SPVertex[i].VertexId.NetNum == 0 &&
```

```
                AreaStructure.ShortestPathTree.SPVertex[i].VertexId.NodeNum == 0) )
    {
        printf("SP Router No %d is %d.%d and its distance to root is %d \n",
                i, AreaStructure.ShortestPathTree.SPVertex[i].VertexId.NetNum,
                AreaStructure.ShortestPathTree.SPVertex[i].VertexId.NodeNum,
                AreaStructure.ShortestPathTree.SPVertex[i].DistanceToRoot );
        printf("The Next Hops for this Router is %d.%d \n",
                AreaStructure.ShortestPathTree.SPVertex[i].NextHopsList[0].NetNum,
                AreaStructure.ShortestPathTree.SPVertex[i].NextHopsList[0].NodeNum );
        i++;
    }
    printf("~~~~~~~~~~~~~  End Shortest Path Information ~~~~~~~~~~~~~~~~~~\n");
}


RoutingTableCalc( &AreaStructure, rte_table );

VL  SPFDL  printf("-------  End Unicast Dijikstra's algorithm (%d.%d)   --------- \n",
                RouterId.NetNum, RouterId.NodeNum);

TotalNoOfTimesOspfCalcDone++;
TotalNoDijCalcDone++;

gettimeofday(&leave, (struct timezone*)0);

if( enter.tv_usec > leave.tv_usec)
    diff = (((leave.tv_sec - 1) - enter.tv_sec)*1000000) + (leave.tv_usec + 1000000) - enter.tv_usec;
else
    diff = ((leave.tv_sec - enter.tv_sec)*1000000) + leave.tv_usec - enter.tv_usec;

TotalOspfCalcTime = TotalOspfCalcTime + diff;
TotalDijCalcTime = TotalDijCalcTime + diff;

if(op_sim_time() > print_time + 20*Tracking_Interval)
{
    fp = fopen("/home/workesys/QospfPro/top_layer/DijCalc", "a" );
    if(!fp)
        op_sim_end("Unable to open the file DijCalc....  ","","","");
    fprintf(fp, "Average time spent by Router %d.%d for Dijkstra's Calculation is %ld. "
                "Current Simulation time is %f \n", RouterId.NetNum, RouterId.NodeNum,
                TotalDijCalcTime/TotalNoDijCalcDone, op_sim_time() );
    fclose(fp);
    print_time = op_sim_time();
}

ospfspfcalc = 0;
detupstrnode = 1;
```

transition **ospfspfcalc -> upstr_node**			
attribute	*value*	*type*	*default value*
name	tr_103	string	tr
condition	detupstrnode	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **BCQospfLsa**

```
int send_qospf_ls_update_pk = 0;
/**
*** Creating the RESOURCE LSA
**/
	int i;
	IPAddress Neighbors_Router_Id;
	reslsa* temp_res_lsa;

	res_ls_seq_no++;

	temp_res_lsa = (reslsa *) op_prg_mem_alloc(sizeof (reslsa));
	memset(temp_res_lsa, 0, sizeof(reslsa));

	temp_res_lsa->ResLsaHeader.LsAge = 0;
	temp_res_lsa->ResLsaHeader.Options = 0;
	temp_res_lsa->ResLsaHeader.LSA_type = RESLSA;
	temp_res_lsa->ResLsaHeader.LinkStateId.NetNum = RouterId.NetNum;
	temp_res_lsa->ResLsaHeader.LinkStateId.NodeNum = RouterId.NodeNum;
	temp_res_lsa->ResLsaHeader.AdvRouter.NetNum = RouterId.NetNum;
	temp_res_lsa->ResLsaHeader.AdvRouter.NodeNum = RouterId.NodeNum;
	temp_res_lsa->ResLsaHeader.ThisAreaId = AreaId;
	temp_res_lsa->ResLsaHeader.LSSeqNo = res_ls_seq_no;
	temp_res_lsa->ResLsaHeader.LSChecksum = 0;
	temp_res_lsa->ResLsaHeader.Length = sizeof(reslsa);
	temp_res_lsa->ResLsaHeader.rtype = ROUTE_PINNING_IS_USED;
	temp_res_lsa->ResLsaHeader.No_of_Links = NoOfLinks;
	for(i = 0; i < temp_res_lsa->ResLsaHeader.No_of_Links; i++)
	{
		temp_res_lsa->ResLinkInfo[i].Type = LinksInfo[i].Types;
		switch (temp_res_lsa->ResLinkInfo[i].Type)
		{
			case 1  : op_prg_mem_copy( &(temp_res_lsa->ResLinkInfo[i].LinkId), &Neighbors_Router_Id,
							sizeof(IPAddress) );
				  op_prg_mem_copy( &(temp_res_lsa->ResLinkInfo[i].LinkData), &(LinksInfo[i].IntfIPAddr),
							sizeof(IPAddress) );
				  break;
			case 2  : op_prg_mem_copy( &(temp_res_lsa->ResLinkInfo[i].LinkData), &(LinksInfo[i].IntfIPAddr),
							sizeof(IPAddress) );
				  op_prg_mem_copy( &RouterId, &(temp_res_lsa->ResLinkInfo[i].LinkData),
							sizeof(IPAddress) );
				  break;
			case 3  : temp_res_lsa->ResLinkInfo[i].LinkId.NetNum = LinksInfo[i].IntfIPAddr.NetNum;
				  temp_res_lsa->ResLinkInfo[i].LinkData.NetNum = NetworkMask;
				  break;
			default : break;
		}
		temp_res_lsa->ResLinkInfo[i].TOSMetric = LinksInfo[i].CostMetric;
```

```
        temp_res_lsa->ResLinkInfo[i].LinkDelay = LinksInfo[i].Delay;
        temp_res_lsa->ResLinkInfo[i].Bandwidth = LinksInfo[i].Bandwidth;
        temp_res_lsa->ResLinkInfo[i].BufferSize = LinksInfo[i].BufferSize;
    }

    op_prg_mem_copy( temp_res_lsa, &(AreaStructure.ResourceLsaList[0]), sizeof(reslsa) );
    op_prg_mem_free( temp_res_lsa );

    send_qospf_ls_update_pk = TRUE;
/**
*** Creating the RRA LSA
**/
{
    int i;
    IPAddress Neighbors_Router_Id;
    rralsa* temp_rra_lsa;
    rra_ls_seq_no++;

    temp_rra_lsa = (rralsa *) op_prg_mem_alloc (sizeof (rralsa));
    memset(temp_rra_lsa, 0, sizeof(rralsa));

    temp_rra_lsa->OpaqueLsaHeader.LsAge = 0;
    temp_rra_lsa->OpaqueLsaHeader.Options = 0;
    temp_rra_lsa->OpaqueLsaHeader.LSA_type = RRALSA;
    temp_rra_lsa->OpaqueLsaHeader.OpaqueType = 11;
    temp_rra_lsa->OpaqueLsaHeader.OpaqueId = 5;
    temp_rra_lsa->OpaqueLsaHeader.AdvRouter.NetNum = RouterId.NetNum;
    temp_rra_lsa->OpaqueLsaHeader.AdvRouter.NodeNum = RouterId.NodeNum;
    temp_rra_lsa->OpaqueLsaHeader.ThisAreaId = AreaId;
    temp_rra_lsa->OpaqueLsaHeader.LSSeqNo = rra_ls_seq_no;
    temp_rra_lsa->OpaqueLsaHeader.LSChecksum = 0;
    temp_rra_lsa->OpaqueLsaHeader.Length = sizeof(rralsa);
    #ifdef ER_IS_USED
        temp_rra_lsa->OpaqueLsaHeader.FloddingScope = 1;
    #endif;
    #ifndef ER_IS_USED
        temp_rra_lsa->OpaqueLsaHeader.FloddingScope = 0;
    #endif;
/*
    temp_rra_lsa->DestAddr = QospfPathStateBlock->SessionObject.DestAddr;
    temp_rra_lsa->SourceAddr = QospfPathStateBlock->SenderTemplateObject.SrcAddr;
*/
    temp_rra_lsa->src_prefix_len = 0;
    temp_rra_lsa->dst_prefix_len = 0;
    temp_rra_lsa->No_of_Links = NoOfLinks;
    for(i = 0; i < NoOfLinks; i++)
    {
        temp_rra_lsa->LinkResvInfo[i].Type = LinksInfo[i].Types;
        switch(temp_rra_lsa->LinkResvInfo[i].Type) {
                case 1  : op_prg_mem_copy( &(temp_rra_lsa->LinkResvInfo[i].LinkId), &Neighbors_Router_Id,
                                        sizeof(IPAddress) );
                          op_prg_mem_copy( &(temp_rra_lsa->LinkResvInfo[i].LinkData),
                                        &(LinksInfo[i].IntfIPAddr), sizeof(IPAddress) );
                          break;
                case 2  : op_prg_mem_copy( &(temp_rra_lsa->LinkResvInfo[i].LinkData),
                                        &(LinksInfo[i].IntfIPAddr), sizeof(IPAddress) );
                          op_prg_mem_copy( &RouterId, &(temp_rra_lsa->LinkResvInfo[i].LinkData),
                                        sizeof(IPAddress) );
                          break;
```

```
                    case 3  : temp_rra_lsa->LinkResvInfo[i].LinkId.NetNum = LinksInfo[i].IntfIPAddr.NetNum;
                              temp_rra_lsa->LinkResvInfo[i].LinkData.NetNum = NetworkMask;
                              break;
                    default : break;
            }
            temp_rra_lsa->LinkResvInfo[i].PinFlag = 1;
            temp_rra_lsa->LinkResvInfo[i].ReservedBandwidth = LinksInfo[i].ReservedBandwidth;
            temp_rra_lsa->LinkResvInfo[i].ReservedBufferSize = LinksInfo[i].ReservedBufferSize;
        }

        op_prg_mem_copy( temp_rra_lsa, &(AreaStructure.RRALsaList[0]), sizeof(rralsa) );
        op_prg_mem_free( temp_rra_lsa );
        send_qospf_ls_update_pk = TRUE;
}
/**
*** Sending the Link state Update packets
**/
if(send_qospf_ls_update_pk == TRUE)
{
        QospfLSUpdatePk* temp_ls_update_pk;
        Packet* QospfLsUpdatePkptr;
        Packet* ipdgramptr;
        int Prev_sended_ResLsaSeqNo, Prev_sended_RRALsaSeqNo;
        int LSUpdatePkCheckSum = 0;
        int i;

        temp_ls_update_pk = ( QospfLSUpdatePk *) op_prg_mem_alloc( sizeof(QospfLSUpdatePk) );
        memset( temp_ls_update_pk, 0, sizeof(QospfLSUpdatePk) );

        temp_ls_update_pk->OspfPkHeader.Version = VERSION;
        temp_ls_update_pk->OspfPkHeader.Type = QOSPFLSUPDATEPK;
        temp_ls_update_pk->OspfPkHeader.PacketLength = sizeof(QospfLSUpdatePk);
        op_prg_mem_copy( &(RouterId), &(temp_ls_update_pk->OspfPkHeader.ThisRouterId),
                                sizeof(IPAddress) );
        temp_ls_update_pk->OspfPkHeader.ThisAreaId = AreaId;
        temp_ls_update_pk->OspfPkHeader.CheckSum = LSUpdatePkCheckSum;
        temp_ls_update_pk->OspfPkHeader.AuType = NONE;
        temp_ls_update_pk->OspfPkHeader.Authentication = 0;
        op_prg_mem_copy( &(AreaStructure.ResourceLsaList[0]), &(temp_ls_update_pk->ResLinkStateAdv[0]),
                                 sizeof(reslsa) );
        temp_ls_update_pk->NoOfResAdvs++;
        for(i = 1; i <=MAX_NO_LSAS; i++ )
        {
            if( AreaStructure.ResourceLsaList[i].ResLsaHeader.LSA_type != RESLSA ||
                ( AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NetNum == 0 &&
                  AreaStructure.ResourceLsaList[i].ResLsaHeader.AdvRouter.NodeNum == 0 ) )
            {
                continue;
            }
            else
            {
                op_prg_mem_copy( &(AreaStructure.ResourceLsaList[i]), &(temp_ls_update_pk->ResLinkStateAdv[i]),
                                        sizeof(reslsa) );
              temp_ls_update_pk->NoOfResAdvs++;
            }

        }

        op_prg_mem_copy( &(AreaStructure.RRALsaList[0]), &(temp_ls_update_pk->RraLinkStateAdv[0]),
```

...
...

```
                                        sizeof(rralsa) );
        temp_ls_update_pk->NoOfRraAdvs++;
        for(i = 1; i <=MAX_NO_LSAS; i++ )
        {
            if( AreaStructure.RRALsaList[i].OpaqueLsaHeader.LSA_type != RESLSA ||
                ( AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NetNum == 0 &&
                 AreaStructure.RRALsaList[i].OpaqueLsaHeader.AdvRouter.NodeNum == 0 ) )
            {
                continue;
            }
            else
            {
                op_prg_mem_copy( &(AreaStructure.RRALsaList[i]), &(temp_ls_update_pk->RraLinkStateAdv[0]),
                                        sizeof(rralsa) );
                temp_ls_update_pk->NoOfRraAdvs++;
            }

        }

        QospfLsUpdatePkptr = op_pk_create_fmt("qospflsupdate_pkfmt");
        op_pk_nfd_set( QospfLsUpdatePkptr, "Lsupdate", temp_ls_update_pk, op_prg_mem_alloc,
                        op_prg_mem_free, sizeof(QospfLSUpdatePk) );
        op_pk_nfd_set( QospfLsUpdatePkptr, "Msg_type", QOSPFLSUPDATEPK );

        data_len = op_pk_total_size_get( QospfLsUpdatePkptr );
        ipdgramptr = op_pk_create_fmt("ip_dgram");
        op_pk_nfd_set( ipdgramptr, "data", QospfLsUpdatePkptr );
        op_pk_nfd_set( ipdgramptr, "prot_id", QOSPF_PROTOCOL_ID );
        op_pk_bulk_size_set( ipdgramptr, data_len*8 );

        op_pk_nfd_set( ipdgramptr, "dest_net", AllSPFRoutersNetNum );
        op_pk_nfd_set( ipdgramptr, "dest_node", AllSPFRoutersNodeNum );
        op_pk_nfd_set( ipdgramptr, "ttl", 2 );

        op_pk_nfd_set( ipdgramptr, "src_net", RouterId.NetNum );
        op_pk_nfd_set( ipdgramptr, "src_node", RouterId.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 );
    }

    if(op_sim_time() < MAX_INIT_TIME)
    {
        op_intrpt_schedule_self(op_sim_time()+1, 1);
        tmp_sim_time = op_sim_time();
    }
    else
    {
        if(op_sim_time() < tmp_sim_time + MAX_REFRESH_TIME)
            op_intrpt_schedule_self(op_sim_time()+1, 1);
        else
        {
            op_intrpt_schedule_self(op_sim_time()+QospfLsaRefreshTime, 1);
            tmp_sim_time = op_sim_time();
```

```
        }
}
```

transition **BCQospfLsa -> idle**			
attribute	*value*	*type*	*default value*
name	tr_80	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **BCOspfLsa**

```

/**
*** Creating the OSPF LSA's
**/
int i,j;
int Prev_sended_RouterLsaSeqNo;
int Prev_sended_NetworkLsaSeqNo;
int Prev_sended_SummaryLsaSeqNo;

/**
*** Create the Router Link State Advertisement for the Aera
**/
{
        RouterLinkAdv* tmp_router_lsa;

        router_ls_seq_no++;
        tmp_router_lsa = (RouterLinkAdv *) op_prg_mem_alloc (sizeof (RouterLinkAdv));
        memset(tmp_router_lsa, 0, sizeof(RouterLinkAdv));

        tmp_router_lsa->LinkStateHeader.LsAge = 0;
        tmp_router_lsa->LinkStateHeader.LsType = ROUTERLSA;
        op_prg_mem_copy( &(RouterId), &(tmp_router_lsa->LinkStateHeader.LinkStateId),
                                    sizeof(IPAddress) );
        op_prg_mem_copy( &(RouterId), &(tmp_router_lsa->LinkStateHeader.AdvRouter),
                                    sizeof(IPAddress) );
        tmp_router_lsa->LinkStateHeader.ThisAreaId = AreaId;
        tmp_router_lsa->LinkStateHeader.LSSeqNo = router_ls_seq_no;
        tmp_router_lsa->LinkStateHeader.LSChecksum = 0;
        tmp_router_lsa->LinkStateHeader.Length = sizeof(RouterLinkAdv);
        tmp_router_lsa->bit_b = 1;
        tmp_router_lsa->bit_e = 0;
        tmp_router_lsa->No_of_Links = NoOfLinks;
        for(i = 0; i < tmp_router_lsa->No_of_Links; i++)
        {
```

```
            tmp_router_lsa->Link_Info[i].Type = LinksInfo[i].Types;
            switch(tmp_router_lsa->Link_Info[i].Type)
            {
                case 1 : op_prg_mem_copy( &(LinksInfo[i].NeighborRtrId),
                                    &(tmp_router_lsa->Link_Info[i].LinkId), sizeof(IPAddress) );
                        op_prg_mem_copy( &(LinksInfo[i].IntfIPAddr),
                                    &(tmp_router_lsa->Link_Info[i].LinkData), sizeof(IPAddress) );
                        break;
                case 2 : op_prg_mem_copy( &(LinksInfo[i].IntfIPAddr),
                                    &(tmp_router_lsa->Link_Info[i].LinkId), sizeof(IPAddress) );
                        op_prg_mem_copy( &(RouterId), &(tmp_router_lsa->Link_Info[i].LinkData),
                                     sizeof(IPAddress) );
                        break;
                case 3 : tmp_router_lsa->Link_Info[i].LinkId.NetNum = LinksInfo[i].NeighborRtrId.NetNum;
                        tmp_router_lsa->Link_Info[i].LinkData.NetNum = NetworkMask;
                        break;
                default : break;
            }
            tmp_router_lsa->Link_Info[i].NoOfTOS = 0;
            tmp_router_lsa->Link_Info[i].TOSMetric = LinksInfo[i].CostMetric;
        }
        VL  OPDL
        {
            printf("------------------  BC OSPF LSA STATE  -----------------\n");
            printf("Advertising Router is %d.%d and its Link information is : \n",
                    RouterId.NetNum, RouterId.NodeNum );
            for( i = 0; i < NoOfLinks; i++ )
            {
                printf("Link Type is %d and its cost is %d \n",
                        tmp_router_lsa->Link_Info[i].Type,
                        tmp_router_lsa->Link_Info[i].TOSMetric );
                printf("Link_Info[i].LinkId = %d.%d, Link_Info[i].LinkData = %d.%d\n",
                        tmp_router_lsa->Link_Info[i].LinkId.NetNum,
                        tmp_router_lsa->Link_Info[i].LinkId.NodeNum,
                        tmp_router_lsa->Link_Info[i].LinkData.NetNum,
                        tmp_router_lsa->Link_Info[i].LinkData.NodeNum );
                /*  PrintLsa( (char*)tmp_router_lsa, ROUTERLSA );  */
            }
            printf("----------------  END BC OSPF LSA STATE  ---------------\n");
        }
        op_prg_mem_copy( tmp_router_lsa, &AreaStructure.RouterLsaList[0], sizeof(RouterLinkAdv) );
        op_prg_mem_free( tmp_router_lsa );
}
/**
*** Create the Network Link State Advertisement
**/
if( IS_DESIGNATED_ROUTER == 1)
{
        NetworkLinkAdv* tmp_network_lsa;

        tmp_network_lsa = (NetworkLinkAdv*) op_prg_mem_alloc(sizeof(NetworkLinkAdv));
        memset(tmp_network_lsa, 0, sizeof(NetworkLinkAdv));

        network_ls_seq_no++;
        tmp_network_lsa->LinkStateHeader.LsAge = 0;
        tmp_network_lsa->LinkStateHeader.LsType = NETWORKLSA;
        op_prg_mem_copy( &(RouterId), &(tmp_network_lsa->LinkStateHeader.LinkStateId), sizeof(IPAddress) );
        op_prg_mem_copy( &(RouterId), &(tmp_network_lsa->LinkStateHeader.AdvRouter), sizeof(IPAddress) );
        tmp_network_lsa->LinkStateHeader.ThisAreaId = AreaId;
```

```
        tmp_network_lsa->LinkStateHeader.LSSeqNo = network_ls_seq_no;
        tmp_network_lsa->LinkStateHeader.LSChecksum = 0;
        tmp_network_lsa->LinkStateHeader.Length = sizeof(NetworkLinkAdv);;
        tmp_network_lsa->ThisNetworkMask = NetworkMask;
        op_prg_mem_copy( &(RouterId), &(tmp_network_lsa->AttachedRouter[0]), sizeof(IPAddress) );
        j=1;
        for(i=0; i<No_of_Networks; i++)
        {
            op_prg_mem_copy( &(Network_Routers[i][0]),
                            &(tmp_network_lsa->AttachedRouter[j]), sizeof(IPAddress) );
            j++;
        }

        op_prg_mem_copy( tmp_network_lsa, &AreaStructure.NetworkLsaList[0], sizeof(NetworkLinkAdv) );

        op_prg_mem_free(tmp_network_lsa);
    }

    /**
    *** IF the router is an Area Border Router Create the Summary Link State Adv
    **/
    if( IS_AREA_BORDER_ROUTER == 1)
    {
        SummaryLinkAdv* tmp_summary_lsa;

        tmp_summary_lsa = (SummaryLinkAdv*) op_prg_mem_alloc( sizeof(SummaryLinkAdv) );
        memset(tmp_summary_lsa, 0, sizeof(SummaryLinkAdv));

        summary_ls_seq_no++;
        tmp_summary_lsa->LinkStateHeader.LsAge = 0;
        tmp_summary_lsa->LinkStateHeader.LsType = SUMMARYLSA;
        tmp_summary_lsa->LinkStateHeader.LinkStateId.NetNum = Ip_Network_No;
        op_prg_mem_copy( &(RouterId), &(tmp_summary_lsa->LinkStateHeader.AdvRouter), sizeof(IPAddress) );
        tmp_summary_lsa->LinkStateHeader.ThisAreaId = AreaId;
        tmp_summary_lsa->LinkStateHeader.LSSeqNo = summary_ls_seq_no;
        tmp_summary_lsa->LinkStateHeader.LSChecksum = 0;
        tmp_summary_lsa->LinkStateHeader.Length = sizeof(SummaryLinkAdv);
        tmp_summary_lsa->ThisNetworkMask = NetworkMask;
        for (i=0; i<No_of_TOS; i++)
        {
            tmp_summary_lsa->Tos_Info[i].TOS = 0;
            tmp_summary_lsa->Tos_Info[i].metric = TOS_Metric[tmp_summary_lsa->Tos_Info[i].TOS];
        }

        op_prg_mem_copy( tmp_summary_lsa, &AreaStructure.SummaryLsaList[0], sizeof(SummaryLinkAdv) );
        op_prg_mem_free( tmp_summary_lsa );
    }

        /**
        *** Send the Ospf Lsa Update packet
        **/
    {
        Packet* OspfLsUpdatePkptr;
        Packet* ipdgramptr;
        OspfLSUpdatePk* temp_ls_update_pk;
        int LSUpdatePkCheckSum = 0;

        temp_ls_update_pk = ( OspfLSUpdatePk *) op_prg_mem_alloc( sizeof(OspfLSUpdatePk) );
        memset( temp_ls_update_pk, 0, sizeof(OspfLSUpdatePk) );
```

```
temp_ls_update_pk->OspfPkHeader.Version = VERSION;
temp_ls_update_pk->OspfPkHeader.Type = OSPFLSUPDATEPK;
temp_ls_update_pk->OspfPkHeader.PacketLength = sizeof(OspfLSUpdatePk);
op_prg_mem_copy( &(RouterId), &(temp_ls_update_pk->OspfPkHeader.ThisRouterId),
                        sizeof(IPAddress) );
temp_ls_update_pk->OspfPkHeader.ThisAreaId = AreaId;
temp_ls_update_pk->OspfPkHeader.CheckSum = LSUpdatePkCheckSum;
temp_ls_update_pk->OspfPkHeader.AuType = NONE;
temp_ls_update_pk->OspfPkHeader.Authentication = 0;

op_prg_mem_copy( &AreaStructure.RouterLsaList[0], &(temp_ls_update_pk->RouterLinkStateAdv[0]),
                        sizeof(RouterLinkAdv) );
temp_ls_update_pk->NoOfRouterAdvs++;
for(i = 1; i <=MAX_NO_LSAS; i++ )
{
    if( AreaStructure.RouterLsaList[i].LinkStateHeader.LsType != ROUTERLSA ||
      ( AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 &&
        AreaStructure.RouterLsaList[i].LinkStateHeader.AdvRouter.NodeNum == 0 ) )
    {
    }
    else
    {
        op_prg_mem_copy( &AreaStructure.RouterLsaList[i], &(temp_ls_update_pk->RouterLinkStateAdv[i]),
                            sizeof(RouterLinkAdv) );
      temp_ls_update_pk->NoOfRouterAdvs++;
    }

}

if(IS_DESIGNATED_ROUTER == 1)
{
    op_prg_mem_copy( &AreaStructure.NetworkLsaList[0], &(temp_ls_update_pk->NetworkLinkStateAdv[0]),
                        sizeof(NetworkLinkAdv) );
    temp_ls_update_pk->NoOfNetworkAdvs++;
    for(i = 1; i <=MAX_NO_LSAS; i++ )
    {
        if( AreaStructure.NetworkLsaList[i].LinkStateHeader.LsType != NETWORKLSA ||
          ( AreaStructure.NetworkLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 &&
              AreaStructure.NetworkLsaList[i].LinkStateHeader.AdvRouter.NodeNum == 0 ) )
        {
        }
        else
        {
            op_prg_mem_copy( &AreaStructure.NetworkLsaList[i],
                                &(temp_ls_update_pk->NetworkLinkStateAdv[i]),
                                sizeof(NetworkLinkAdv) );
          temp_ls_update_pk->NoOfNetworkAdvs++;
        }

    }
}

if(IS_AREA_BORDER_ROUTER == 1)
{
    op_prg_mem_copy( &AreaStructure.SummaryLsaList[0], &(temp_ls_update_pk->SummaryLinkStateAdv[0]),
                        sizeof(SummaryLinkAdv) );
    temp_ls_update_pk->NoOfSummaryAdvs++;
    for(i = 1; i <=MAX_NO_LSAS; i++ )
```

...
...

```
                {
                        if( AreaStructure.SummaryLsaList[i].LinkStateHeader.LsType != SUMMARYLSA ||
                          ( AreaStructure.SummaryLsaList[i].LinkStateHeader.AdvRouter.NetNum == 0 &&
                            AreaStructure.SummaryLsaList[i].LinkStateHeader.AdvRouter.NodeNum == 0 ) )
                        {
                        }
                        else
                        {
                                op_prg_mem_copy( &AreaStructure.SummaryLsaList[i],
                                                        &(temp_ls_update_pk->SummaryLinkStateAdv[i]),
                                                        sizeof(SummaryLinkAdv) );
                          temp_ls_update_pk->NoOfSummaryAdvs++;
                        }
                }
        }

        OspfLsUpdatePkptr = op_pk_create_fmt("ospflsupdate_pkfmt");
        op_pk_nfd_set( OspfLsUpdatePkptr, "Lsupdate", temp_ls_update_pk, op_prg_mem_alloc,
                                op_prg_mem_free, sizeof(OspfLSUpdatePk) );
        op_pk_nfd_set( OspfLsUpdatePkptr, "Msg_type", OSPFLSUPDATEPK );

        data_len = op_pk_total_size_get( OspfLsUpdatePkptr );
        ipdgramptr = op_pk_create_fmt("ip_dgram");
        op_pk_nfd_set( ipdgramptr, "data", OspfLsUpdatePkptr );
        op_pk_nfd_set( ipdgramptr, "prot_id", QOSPF_PROTOCOL_ID );
        op_pk_bulk_size_set( ipdgramptr, data_len*8 );

        op_pk_nfd_set( ipdgramptr, "dest_net", AllSPFRoutersNetNum );
        op_pk_nfd_set( ipdgramptr, "dest_node", AllSPFRoutersNodeNum );
        op_pk_nfd_set( ipdgramptr, "ttl", 2 );

        op_pk_nfd_set( ipdgramptr, "src_net", RouterId.NetNum );
        op_pk_nfd_set( ipdgramptr, "src_node", RouterId.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 );

        if( op_sim_time() < MAX_INIT_TIME )
        {
                op_intrpt_schedule_self(op_sim_time()+1, 0);
                tmp_sim_time = op_sim_time();
        }
        else
        {
                if( op_sim_time() < tmp_sim_time + MAX_REFRESH_TIME )
                        op_intrpt_schedule_self(op_sim_time()+1, 0);
                else
                {
                        op_intrpt_schedule_self(op_sim_time()+OspfLsaRefreshTime, 0);
                        tmp_sim_time = op_sim_time();
                }
        }
}
```

...
...

transition **BCOspfLsa -> idle**			
attribute	*value*	*type*	*default value*
name	tr_79	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **BCMospfLsa**

```
/**
*** Creating the GRPMBRSHIP LSA for a all the groups and send the LSA as
*** a mospf ls update packet.
**/
int send_mospf_ls_update_pk = FALSE;

{
	int i, j;
	int DestGrp;
	int VertexNo;
	int RegGrp_Flag;
	grpmbrlsa* tmp_grpmbrship_lsa;
	int No_of_Reg_Grps = 0;
	int RegGrps[MAX_NO_GRPS];

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

	VL
	{
		/*  printf( "-------  (%d.%d) Registered Group Information --------- \n",   */
				/*  RouterId.NetNum, RouterId.NodeNum );  */
		for( i = 0; i < MAX_NO_GRPS; i++ )
		{
			int Reg_Grp_Flag = OFF;
			for( j = 0; j <= NoOfLinks; j++ )
				if( mc_table[i].interface[j] == 1)
				{
					/*  printf(" GRP ID : %d, REGISTERED INTERFACE : %d \n", i, j );  */
					RegGrps[No_of_Reg_Grps] = i;
					Reg_Grp_Flag = ON;
				}
				if (Reg_Grp_Flag == ON )
					No_of_Reg_Grps++;
		}
		/*  printf( "-------  End Registered Group Information --------- \n" );  */
	}

	for( i = 0; i < No_of_Reg_Grps; i++ )
```

```
	{
		RegGrp_Flag = OFF;
		tmp_grpmbrship_lsa = (grpmbrlsa *) op_prg_mem_alloc(sizeof(grpmbrlsa));
		memset(tmp_grpmbrship_lsa, 0, sizeof(grpmbrlsa));

		grpmbr_ls_seq_no++;
		tmp_grpmbrship_lsa->LinkStateHeader.LsAge = 0;
		tmp_grpmbrship_lsa->LinkStateHeader.LsType = GRPMBRSHIPLSA;
		tmp_grpmbrship_lsa->LinkStateHeader.AdvRouter.NetNum = RouterId.NetNum;
		tmp_grpmbrship_lsa->LinkStateHeader.AdvRouter.NodeNum = RouterId.NodeNum;
		tmp_grpmbrship_lsa->LinkStateHeader.ThisAreaId = AreaId;
		tmp_grpmbrship_lsa->LinkStateHeader.LSSeqNo = grpmbr_ls_seq_no;
		tmp_grpmbrship_lsa->LinkStateHeader.LSChecksum = 0;
		tmp_grpmbrship_lsa->LinkStateHeader.Length = sizeof(grpmbrlsa);
		VertexNo = 0;
		/*  DestGrp = RegGrps[No_of_Reg_Grps];  */
		DestGrp = RegGrps[i];

		tmp_grpmbrship_lsa->LinkStateHeader.LinkStateId.NetNum = DestGrp;

		for(j=0; j < NoOfLinks; j++)
		{
			if( mc_table[i].interface[j] == 1)
			{
				tmp_grpmbrship_lsa->VertexInfo[j].VertexType = LinksInfo[j].Types;
				tmp_grpmbrship_lsa->VertexInfo[j].VertexId.NetNum = LinksInfo[j].NeighborRtrId.NetNum;
				tmp_grpmbrship_lsa->VertexInfo[j].VertexId.NodeNum = LinksInfo[j].NeighborRtrId.NodeNum;
				RegGrp_Flag = ON;
			}
		}

		if( RegGrp_Flag == ON )
		{
			op_prg_mem_copy( tmp_grpmbrship_lsa, &AreaStructure.GrpMbrLsaList[DestGrp][0],
						sizeof(grpmbrlsa) );
			send_mospf_ls_update_pk = TRUE;
		}
		op_prg_mem_free(tmp_grpmbrship_lsa);
	}

/*******		SEND THE MOSPF LSA UPDATE PACKET		**********/
	if(send_mospf_ls_update_pk == TRUE)
	{
		int j=0;
		int LSUpdatePkCheckSum = 0;

		Packet* MospfLsUpdatePkptr;
		Packet* ipdgramptr;

		MospfLSUpdatePk* temp_ls_update_pk;

		temp_ls_update_pk = ( MospfLSUpdatePk *) op_prg_mem_alloc( sizeof(MospfLSUpdatePk) );
		memset( temp_ls_update_pk, 0, sizeof(MospfLSUpdatePk) );

		temp_ls_update_pk->OspfPkHeader.Version = VERSION;
		temp_ls_update_pk->OspfPkHeader.Type = MOSPFLSUPDATEPK;
		temp_ls_update_pk->OspfPkHeader.PacketLength = sizeof(MospfLSUpdatePk);
		op_prg_mem_copy( &(RouterId), &(temp_ls_update_pk->OspfPkHeader.ThisRouterId),
					sizeof(IPAddress) );
```

```
            temp_ls_update_pk->OspfPkHeader.ThisAreaId = AreaId;
            temp_ls_update_pk->OspfPkHeader.CheckSum = LSUpdatePkCheckSum;
            temp_ls_update_pk->OspfPkHeader.AuType = NONE;
            temp_ls_update_pk->OspfPkHeader.Authentication = 0;
            temp_ls_update_pk->NoOfGrpMbrAdvs = 0;

            while( !(AreaStructure.GrpMbrLsaList[i][j].LinkStateHeader.AdvRouter.NetNum == 0 &&
                    AreaStructure.GrpMbrLsaList[i][j].LinkStateHeader.AdvRouter.NodeNum == 0 ) )
            {
                if( AreaStructure.GrpMbrLsaList[i][j].LinkStateHeader.LsType == GRPMBRSHIPLSA )
                {
                    op_prg_mem_copy( &AreaStructure.GrpMbrLsaList[i][j],
                                     &(temp_ls_update_pk->GrpmbrLinkStateAdv[temp_ls_update_pk->NoOfGrpMbrAdvs] ),
                                     sizeof(grpmbrlsa) );
                    temp_ls_update_pk->NoOfGrpMbrAdvs++;
                }
                j++;
            }

            MospfLsUpdatePkptr = op_pk_create_fmt("mospflsupdate_pkfmt");
            op_pk_nfd_set( MospfLsUpdatePkptr, "Lsupdate", temp_ls_update_pk, op_prg_mem_alloc,
                        op_prg_mem_free, sizeof(MospfLSUpdatePk) );
            op_pk_nfd_set( MospfLsUpdatePkptr, "Msg_type", MOSPFLSUPDATEPK );

            data_len = op_pk_total_size_get( MospfLsUpdatePkptr );
            ipdgramptr = op_pk_create_fmt("ip_dgram");
            op_pk_nfd_set( ipdgramptr, "data", MospfLsUpdatePkptr );
            op_pk_nfd_set( ipdgramptr, "prot_id", QOSPF_PROTOCOL_ID );
            op_pk_bulk_size_set( ipdgramptr, data_len*8 );

            op_pk_nfd_set( ipdgramptr, "dest_net", AllSPFRoutersNetNum );
            op_pk_nfd_set( ipdgramptr, "dest_node", AllSPFRoutersNodeNum );
            op_pk_nfd_set( ipdgramptr, "ttl", 2 );

            op_pk_nfd_set( ipdgramptr, "src_net", RouterId.NetNum );
            op_pk_nfd_set( ipdgramptr, "src_node", RouterId.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  MPDL    printf( "Sent an MospfLSUpdate Packet to IP Layer for the Group %d \n", RegGrps[i] );
        }
}
op_intrpt_schedule_self(op_sim_time()+MospfLsaRefreshTime, 3);
send_grp_mbr_lsa = 0;
```

transition **BCMospfLsa -> idle**			
attribute	*value*	*type*	*default value*
name	tr_87	string	tr
condition	default	string	
executive		string	

color	RGB203	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **hello_pks**

```
/**
*** This state creates the hello packet that needs to be sent to all the neighbors. The hello
*** packet sets the destination as AllSPFRouters which indicates that this packet should be
*** sent to only the other end of the links and the packet shouldn't be forwarded from there.
*** The source address of the packet is the Router Id of the router originating the packet
**/

int NoOfRtrsFrmWhichHelloRec;
hellopacket* tmp_hello_pk;
Packet* helloPkptr;
Packet* ipdgramptr;

tmp_hello_pk = (hellopacket*) op_prg_mem_alloc( sizeof(hellopacket) );
memset( tmp_hello_pk, 0, sizeof(hellopacket) );

tmp_hello_pk->OspfPkHeader.Version = VERSION;
tmp_hello_pk->OspfPkHeader.Type = HELLOPK;
tmp_hello_pk->OspfPkHeader.PacketLength = sizeof(hellopacket);
tmp_hello_pk->OspfPkHeader.ThisRouterId.NetNum = RouterId.NetNum;
tmp_hello_pk->OspfPkHeader.ThisRouterId.NodeNum = RouterId.NodeNum;
tmp_hello_pk->OspfPkHeader.ThisAreaId = AreaId;
tmp_hello_pk->OspfPkHeader.AuType = NONE;
tmp_hello_pk->OspfPkHeader.Authentication = 0;
tmp_hello_pk->ThisNetworkMask = NetworkMask;
tmp_hello_pk->Deadint = DeadInt;
tmp_hello_pk->Hellopint = HelloInt;
tmp_hello_pk->ThisRtrPri = RtrPri;
tmp_hello_pk->Indep_Router = Indep_Rou;
op_prg_mem_copy( &(tmp_hello_pk->DesgRouter), &(DsgRouterIntfAddr), sizeof(IPAddress) );
op_prg_mem_copy( &(tmp_hello_pk->BackupDesgRouter), &(BackupDsgRouterIntfAddr), sizeof(IPAddress) );
NoOfRtrsFrmWhichHelloRec = 0;
if(!IsListEmpty(&NeighborList) )
{
	Neighbor_Data_Str* neighborEle;

	neighborEle = (Neighbor_Data_Str*) op_prg_mem_alloc( sizeof(Neighbor_Data_Str) );

	neighborEle = (Neighbor_Data_Str*) GetNextListEle( &(NeighborList), 1 );
	while(neighborEle)
	{
		op_prg_mem_copy( &(tmp_hello_pk->RtrsFrmWhichHelloRec[NoOfRtrsFrmWhichHelloRec]),
					&(neighborEle->NeighborRouterId), sizeof(IPAddress) );
		NoOfRtrsFrmWhichHelloRec++;
		neighborEle = (Neighbor_Data_Str*) GetNextListEle( &(NeighborList), 0 );
	}
```

```
        op_prg_mem_free( neighborEle );
}

helloPkptr = op_pk_create_fmt("hello_pkfmt");
op_pk_nfd_set( helloPkptr, "hellopk", tmp_hello_pk, op_prg_mem_alloc, op_prg_mem_free, sizeof(hellopacket) );

data_len = op_pk_total_size_get( helloPkptr )/8;
ipdgramptr = op_pk_create_fmt("ip_dgram");
op_pk_nfd_set( ipdgramptr, "data", helloPkptr );
op_pk_nfd_set( ipdgramptr, "prot_id", OspfPk );
op_pk_bulk_size_set( ipdgramptr, data_len*8 );
op_pk_nfd_set( ipdgramptr, "dest_net", AllSPFRoutersNetNum );
op_pk_nfd_set( ipdgramptr, "dest_node", AllSPFRoutersNodeNum );
op_pk_nfd_set( ipdgramptr, "ttl", 2 );

op_pk_nfd_set( ipdgramptr, "src_net", net_num0);
op_pk_nfd_set( ipdgramptr, "src_node", node_num0);

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 );

/*  printf("\nSending Hello Packet from the Router %d.%d \n", RouterId.NetNum, RouterId.NodeNum);  */
op_pk_send( ipdgramptr, IPSTREAM );

op_intrpt_schedule_self( op_sim_time()+HelloPkIntvl, 2 );
```

transition **hello_pks -> idle**			
attribute	*value*	*type*	*default value*
name	tr_90	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

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

transition **BCLsReqPk -> idle**			
attribute	*value*	*type*	*default value*
name	tr_97	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline

...
...

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

enter execs **upstr_node**

```
send_dabra = 1;
detupstrnode = 0;
```

transition **upstr_node -> dabra**			
attribute	*value*	*type*	*default value*
name	tr_93	string	tr
condition	send_dabra	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

transition **BCdatadescpk -> idle**			
attribute	*value*	*type*	*default value*
name	tr_108	string	tr
condition	default	string	
executive		string	
color	RGB203	color	RGB333
drawing style	spline	toggle	spline