Header Block

```
#define    ARR op_intrpt_type() == OPC_INTRPT_STRM
#define SELF_INT (op_intrpt_type () == OPC_INTRPT_SELF && op_intrpt_code() == 0)

#define UPDATE_MAC_OUTSTAT  0

#define ON                  1
#define OFF                 0

#define VL                  if(ON)
#define IGDL                if(OFF)

#define IP_STREAM           0
#define MULTICAST_NET       500

extern int IGMP;
extern int UDP;
```

State Variable Block

```
Objid   \my_id,\ip_objid;
Ici*        \encap_iciptr;
int         \in_strm,\src_net,\src_node;
int         \app_table[100][20],\UP,\grp_id;
double      \timer[100];
Distribution*       \time_dis;

int  \dgram_id;
```

Temporary Variable Block

```
Packet  *pkptr,*ip_pkptr;
Packet* pkptr1,*pkptr_tmp;
int i,j,join,term,exist;

int data_len,protocol,dest_node;
```

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

```
my_id = op_id_self ();

VL   IGDL
        encap_iciptr = op_ici_create("ip_encap_req");

for(i = 0; i < 100; ++i)
{
```

```
		for(j = 0; j < 20; ++j)
		{
			app_table[i][j] = -1;
		}
		timer[i] = 0;
	}

time_dis = op_dist_load("uniform",1,44);

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

/*
** Get the IP Address of the node from its IP layer
*/
op_ima_obj_attr_get (ip_objid, "net_num0", &src_net);
op_ima_obj_attr_get (ip_objid, "node_num0", &src_node);

dgram_id = 0;
```

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 -> spillter**

attribute	*value*	*type*	*default value*
name	tr_2	string	tr
condition	ARR	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

transition **idle -> RESPON**

attribute	*value*	*type*	*default value*
name	tr_7	string	tr
condition	SELF_INT	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

...
...

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

enter execs **data_decap**

```
/*
** the pk is either a query or a data packet or a report response
*/
ip_pkptr = op_pk_get(in_strm);
op_pk_nfd_get(ip_pkptr,"data",&pkptr);
op_pk_nfd_get(ip_pkptr,"prot_id",&protocol);
op_pk_nfd_get(ip_pkptr,"dest_node",&dest_node);
if(protocol == IGMP)
{
	op_pk_nfd_get(pkptr,"GRP_ID",&grp_id);
	op_pk_nfd_get(pkptr,"join",&join);
	op_pk_nfd_get(pkptr,"terminate",&term);
}

VL  IGDL
{
	printf("Packet %d, GRP_ID %d at IGMP %d.%d\n",
			op_pk_id(pkptr), grp_id, src_net, src_node);
}

if(protocol == UDP)
{
	/* it's a data pk */
	VL  IGDL
	{
		printf("Data packet  GRP_ID %d at Igmp %d.%d\n",grp_id,src_net,src_node);
	}

	op_pk_destroy(ip_pkptr);

	i=1;
	while(i < 20)
	{
		VL  IGDL
		{
			printf(" app_table[%d][%d]=%d\n",grp_id,i,app_table[grp_id][i]);
		}
		if(app_table[grp_id][i] == 1)
		{
			pkptr1 = op_pk_copy(pkptr);
			op_pk_send(pkptr1,i);
		}
		i++;
	}
}

else
{
```

...
...

```
/*
** If the received packet is query
*/
if(protocol==IGMP && dest_node == 500)
{
    VL  IGDL
        printf("Got a query for grp_id=%d  at %d.%d\n",grp_id,src_net,src_node);

    i=1;
    exist=0;
    while( !exist && i < 20 )
    {
        if(app_table[grp_id][i] == 1)
            exist = 1;
        i++;
    }
    if(exist)
    {
        /*send a report to the router*/
        timer[grp_id] = op_sim_time() + op_dist_outcome(time_dis);
        op_intrpt_schedule_self( timer[grp_id], 0 );

        VL  IGDL
            printf("Will sent a response at %f (%d.%d) for grp: %d\n",
                    timer[grp_id],src_net,src_node,grp_id);
    }
    op_pk_destroy(ip_pkptr);
}
else
{
    VL  IGDL
        printf("Got a response at host, destroying and setting the timer(%d.%d)\n",
                src_net, src_node );
    op_pk_destroy(ip_pkptr);
    op_pk_nfd_get(pkptr,"GRP_ID",&grp_id);
    timer[grp_id]=0;
}
}
op_pk_destroy(pkptr);
```

transition **data_decap -> idle**

attribute	*value*	*type*	*default value*
name	tr_6	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

forced state **APP_INF**

attribute	*value*	*type*	*default value*
name	APP_INF	string	st
enter execs	(See below.)	textlist	(See below.)

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

enter execs **APP_INF**

```
pkptr = op_pk_get(in_strm);
op_pk_nfd_get(pkptr,"GRP_ID",&grp_id);
op_pk_nfd_get(pkptr,"join",&join);
op_pk_nfd_get(pkptr,"terminate",&term);

if( join && (app_table[grp_id][in_strm] == 1) )
  op_sim_end("Already joined then why a join req","","","");

if(term && (app_table[grp_id][in_strm] == -1) )
{
	printf("(%d.%d)in_strm = %d term %d app_table[%d][%d]=%d\n",
			src_net,src_node,in_strm,term,grp_id,in_strm,app_table[grp_id][in_strm]);
	op_sim_end("No existing session then why terminateq","","","");
	getchar();
}

if(!join && !term)
{
	/*it's a data packet*/
	VL  IGDL
		printf("Sending a data to IP: %d.%d GRP: %d\n",src_net,src_node,grp_id);

	data_len = op_pk_total_size_get (pkptr) / 8;
	ip_pkptr = op_pk_create_fmt ("ip_dgram");
	op_pk_nfd_set (ip_pkptr, "data", pkptr);
	op_pk_bulk_size_set (ip_pkptr, data_len * 8);
	op_pk_nfd_set (ip_pkptr, "dest_net", 500);
	op_pk_nfd_set (ip_pkptr, "dest_node", grp_id);

	op_pk_nfd_set (ip_pkptr, "src_net", src_net);
	op_pk_nfd_set (ip_pkptr, "src_node", src_node);

	op_pk_nfd_set (ip_pkptr, "orig_len", data_len);
	op_pk_nfd_set (ip_pkptr, "frag_len", data_len);

	op_pk_nfd_set (ip_pkptr, "frag", 0);
	op_pk_nfd_set (ip_pkptr, "ident", dgram_id++);
	op_pk_nfd_set (ip_pkptr, "prot_id", UDP);

	op_pk_send( ip_pkptr, IP_STREAM );
}
else
{
	if( term==1 && (app_table[grp_id][in_strm] == 1))
	{
		app_table[grp_id][in_strm] = -1;
		op_stat_local_write( UPDATE_MAC_OUTSTAT,grp_id );
		op_pk_destroy(pkptr);
		VL  IGDL
			printf("I am ending my session %d.%d for grp %d\n",
					src_net, src_node, grp_id );
	}
	if( join==1 && (app_table[grp_id][in_strm] == -1) && term==0 )
	{
```

...
...

```
			op_pk_destroy(pkptr);
			i=1;
			exist=0;
			while(!exist && i < 20)
			{
				if(app_table[grp_id][i]==1)
					exist = 1;
				i++;
			}
			app_table[grp_id][in_strm]=1;

			if(!exist)
			{
				pkptr1 = op_pk_create_fmt("MC_APP_CNTRL");
				op_pk_nfd_set(pkptr1,"join",1);
				op_pk_nfd_set(pkptr1,"terminate",0);
				op_pk_nfd_set(pkptr1,"GRP_ID",grp_id);

				data_len = op_pk_total_size_get (pkptr1) / 8;
				ip_pkptr = op_pk_create_fmt ("ip_dgram");
				op_pk_nfd_set (ip_pkptr, "data", pkptr1);
				op_pk_bulk_size_set (ip_pkptr, data_len * 8);
				op_pk_nfd_set (ip_pkptr, "dest_net", 500);
				op_pk_nfd_set (ip_pkptr, "dest_node", grp_id);

				op_pk_nfd_set (ip_pkptr, "src_net", src_net);
				op_pk_nfd_set (ip_pkptr, "src_node", src_node);

				op_pk_nfd_set (ip_pkptr, "orig_len", data_len);
				op_pk_nfd_set (ip_pkptr, "frag_len", data_len);

				op_pk_nfd_set (ip_pkptr, "frag", 0);
				op_pk_nfd_set (ip_pkptr, "ident", dgram_id++);
				op_pk_nfd_set (ip_pkptr, "prot_id", IGMP);

				VL  IGDL
					printf("%d.%d  Igmp sending a join grp: %d\n",
							src_net, src_node, grp_id );

				op_pk_send( ip_pkptr, IP_STREAM );

				op_stat_local_write( UPDATE_MAC_OUTSTAT,grp_id );
			}
		}
	}
```

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

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

enter execs **spillter**

```
in_strm = op_intrpt_strm();

if(in_strm > 0)
	UP=0;
else
	UP=1;
```

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

transition **spillter -> APP_INF**			
attribute	*value*	*type*	*default value*
name	tr_4	string	tr
condition	!UP	string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline

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

enter execs **RESPON**

```
i = 0;
while(i < 21)
{
	for(j = 0; j < 3; ++j)
	{
		if(timer[i] == op_sim_time() && app_table[i][j] == 1)
		{
			VL  IGDL
			{
				printf("Sending back a response for grp_id %d %d.%d\n",
```

```
                              i, src_net, src_node );
                }

                pkptr_tmp = op_pk_create_fmt("MC_APP_CNTRL");
                ip_pkptr = op_pk_create_fmt("ip_dgram");

                data_len = op_pk_total_size_get( pkptr_tmp ) / 8;

                op_pk_nfd_set(pkptr_tmp,"GRP_ID",i);
                op_pk_nfd_set(pkptr_tmp,"join",1);
                op_pk_nfd_set(pkptr_tmp,"terminate",0);
                op_pk_nfd_set(ip_pkptr,"data",pkptr_tmp);

                op_pk_nfd_set (ip_pkptr, "src_net", src_net);
                op_pk_nfd_set (ip_pkptr, "src_node", src_node);

                op_pk_nfd_set (ip_pkptr, "orig_len", data_len);
                op_pk_nfd_set (ip_pkptr, "frag_len", data_len);

                op_pk_nfd_set (ip_pkptr, "frag", 0);

                /* Assign a unique identity to this packet (unique among */
                /* packets injected into the net by this host. */
                op_pk_nfd_set (ip_pkptr, "ident", dgram_id++);

                op_pk_nfd_set (ip_pkptr, "prot_id", IGMP);

                op_pk_nfd_set(ip_pkptr,"dest_node",i);
                op_pk_nfd_set(ip_pkptr,"dest_net",500);
                op_pk_nfd_set(ip_pkptr,"ttl",2);
                op_pk_send(ip_pkptr, IP_STREAM);

                VL  IGDL
                {
                    printf("Sending a response for grp : %d from %d.%d\n",i,src_net,src_node);
                }
            }
        }
        timer[i] = 0;
        i++;
}
```

transition **RESPON -> idle**			
attribute	*value*	*type*	*default value*
name	tr_8	string	tr
condition		string	
executive		string	
color	RGB333	color	RGB333
drawing style	spline	toggle	spline