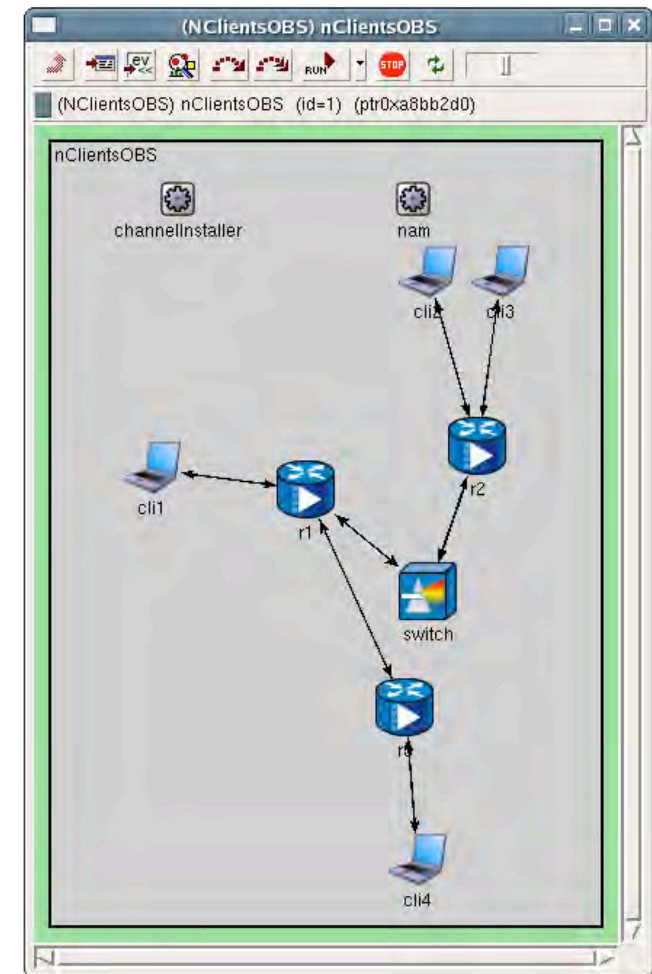
OBSInterface

Área de Ingeniería Telemática
http://www.tlm.unavarra.es

Grupo de Redes, Sistemas y Servicios Telamáticos

OBSInterface

- OBS: nodo frontera y nodos del núcleo => Router y Switch
- OBSInterface: routerOBS y switchOBS
- Son módulos complejos
- Extra: SimuladorPerdidas



Nuevos Mensajes: ColoresFrames.msg

- Array de enteros que representa los colores de las longitudes de onda de un interfaz OBS.

```
message ColoresFrame {
   fields:  int colores[];
};
```

- La intención era que los interfaces comunicaran a los switch sus colores, para que los switch pudieran realizar un encaminamiento más preciso.
- No se ha implementado este uso.

Nuevos Mensajes: OBSControlFrame.msg

- Mensaje de control OBS

```
message OBSControlFrame{
  fields:
    simtime_t tiempoLlegada;
    int lambda;
    int identificador;
};
```

- Sólo se usa *lambda*, que indica lambda de la ráfaga
- **Problema**: tiene que indicar también el tamaño de la ráfaga:
  - *getSize()* y *setSize(int sizeRafaga_var)*
  - Modificando los _m.h y _m.cc que genera opp_makemake

Nuevos Mensajes: OBSFrame.msg

- Ráfaga OBS => almacena todos los paquetes de la ráfaga

```
message OBSFrame{
  fields:
    int numPaquetes;
};
```

- **Problema**: con esto no sirve:
  - Modificando los _m.h y _m.cc que genera opp_makemake:

```
list<cMessage*> messages;
virtual int getNumPaquetes() const;
virtual void setNumPaquetes(int numPaquetes_var);
virtual void insertMessage(cMessage *msg);
virtual void deleteMessage(cMessage *msg);
virtual cMessage* nextMessage();
virtual int numMessages();
```

Nuevos mensajes

- Compilación típica de desarrollo Omnet/INET => Makefile.gen

```
INET = /home/felix/omnetpp-3.3/INET-20061020
ALL_INET_INCLUDES = -I$(INET)/Base …….. -I$(INET)/NetworkInterfaces/PPP
all:
    opp_makemake -f -N -x -w $(ALL_INET_INCLUDES)
# DO NOT DELETE THIS LINE -- make depend depends on it.
```

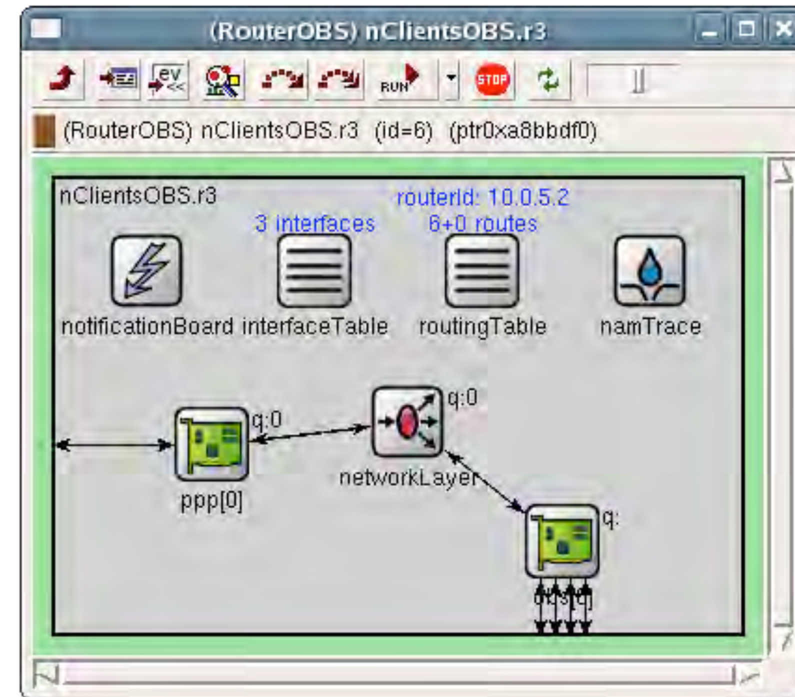
- Modificación de los _m.h y _m.cc provoca que se tenga que tener mucho cuidado al compilar con Makefile.gen:

  Makefile.gen => Makefile => Mod. Makefile => make

- Solución: clase base + subclase

RouterOBS

- Igual que Router de INET
  - Interfaces PPP/Ethernet
  - Tabla de rutas
- Pero también interfaces OBS



RouterOBS.ned

```
parameters:
  routingFile: string,
  numInterfaces: numeric const,
  lambda: numeric const;

gates:
   in: in[];
   out: out[];
   in: ethIn[];
   out: ethOut[];
   in: OBSin[];
   out: OBSout[];

submodules:
  routingTable: RoutingTable;
     parameters:
        IPForward = true,
        routerId = "auto",
        routingFile = routingFile;
     display: "p=240,60;i=block/table";

networkLayer: NetworkLayer;
     gatesizes:
          ifIn[sizeof(out)+sizeof(ethOut)+numInterfaces],
          ifOut[sizeof(out)+sizeof(ethOut)+numInterfaces];
     display: "p=200,141;i=block/fork;q=queue";

ppp: PPPInterface[sizeof(out)];
     display: "q=l2queue;i=block/ifcard";

obs: OBSInterface[numInterfaces];
     parameters:
          lambdaInterface = lambda;
     display: "q=l2queue;i=block/ifcard";

eth: EthernetInterface[sizeof(ethOut)];
     display: "q=l2queue;i=block/ifcard";
```

RouterOBS.ned

```
connections nocheck:

    for i=0..sizeof(out)-1 do
        in[i] --> ppp[i].physIn;
        out[i] <-- ppp[i].physOut;
        ppp[i].netwOut --> networkLayer.ifIn[i];
        ppp[i].netwIn <-- networkLayer.ifOut[i];
    endfor;
    for i=0..numInterfaces-1 do
        obs[i].netwOut --> networkLayer.ifIn[sizeof(out)+i];
        obs[i].netwIn <-- networkLayer.ifOut[sizeof(out)+i];
    endfor;
    for i=0..numInterfaces-1,j=0..lambda-1 do
      OBSin[i*lambda+j] --> obs[i].physIn++;
      OBSout[i*lambda+j] <-- obs[i].physOut++;
    endfor;
```

```
for i=0..sizeof(ethOut)-1 do
    ethIn[i] --> eth[i].physIn;
    ethOut[i] <-- eth[i].physOut;
    eth[i].netwOut -->
      networkLayer.ifIn[sizeof(out)+sizeof(OBSout)+i];
    eth[i].netwIn <--
      networkLayer.ifOut[sizeof(out)+sizeof(OBSout)+i];
endfor;
```

OBSInterface.ned

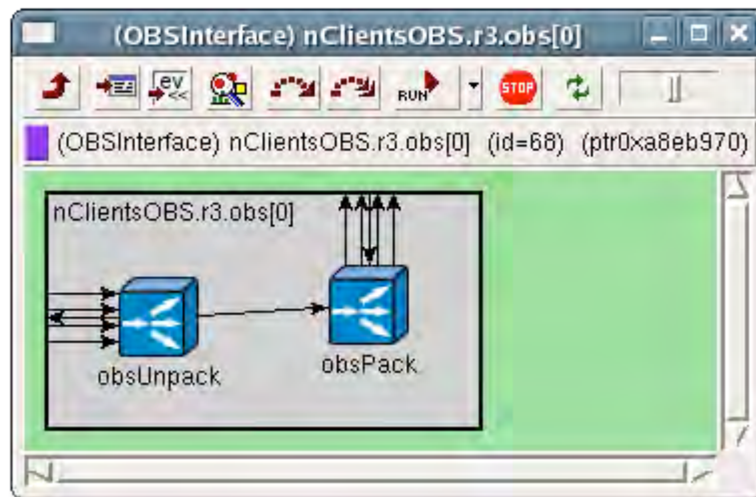
```
module OBSInterface
   parameters:
      lambdaInterface: numeric;
   gates:
      in: physIn[];
      out: physOut[];
      in: netwIn;
      out: netwOut;
   submodules:
      obsPack: OBSPack;
            parameters:
                  lambda=lambdaInterface;
                  //nColas=2;
            display: "i=abstract/dispatcher";
      obsUnpack: OBSUnpack;
            parameters:
                  lambda=lambdaInterface;
            display: "i=abstract/dispatcher";
   connections:
      for i=0..lambdaInterface-1 do
            physIn++ --> obsUnpack.in++;
            physOut++ <-- obsPack.out++;
      endfor;

      netwOut <-- obsUnpack.out;
      netwIn --> obsPack.in;
      obsPack.fromUnpack <-- obsUnpack.toPack;

endmodule
```

OBSInterface.ned

- obsPack: módulo complejo
- obsUnPack: módulo simple



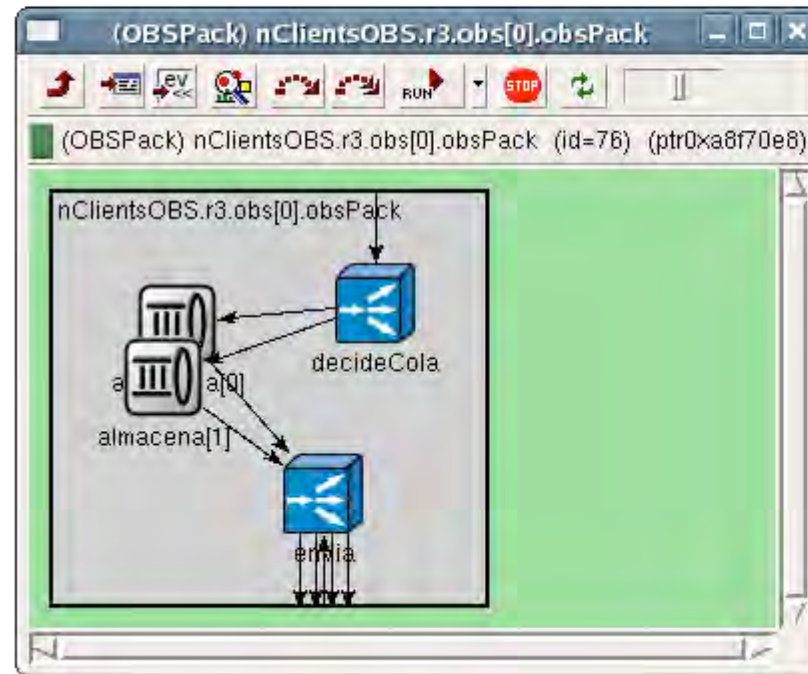
OBSInterface.ned

```
module OBSPack
  parameters:
    nColas: numeric,
    lambda: numeric,
    queueCapacity: numeric;
  gates:
    in:in;
    out:out[];
    in:fromUnpack;
  submodules:
    decideCola: DecideCola;
      parameters:
        nColas=nColas;
      display: "i=abstract/dispatcher";
    almacena: ColaRafaga[nColas];
      display: "i=block/queue;";
    envia: EnviaRafagas;
      parameters:
        lambda=lambda,
        queueCapacity=queueCapacity;
      display: "i=abstract/dispatcher";
  connections:
    in --> decideCola.in;
    for i=0..nColas-1 do
        decideCola.out++ --> almacena[i].in;
        almacena[i].out -->  envia.in++;
    endfor;
    for i=0..lambda-1 do
        envia.out++ --> out++;
    endfor;
    fromUnpack --> envia.fromUnpack;

endmodule
```

OBSInterface.ned

- obsPack: módulo complejo



OBSInterface.ned

```
simple OBSUnpack
    parameters:
      lambda:numeric;
    gates:
      in:in[];
      out:out;
      out:toPack;
endsimple

simple DecideCola
    parameters:
      nColas:numeric;
    gates:
      in:in;
      out:out[];
endsimple

simple ColaRafaga
    parameters:
      paqMax:numeric,
      tiempoMax:numeric;
    gates:
      in:in;
      out:out;
endsimple

simple EnviaRafagas
    parameters:
      lambda:numeric,
      idInterface:numeric,
      colores:string,
      queueCapacity:numeric;
    gates:
      in:in[];
      out:out[];
      in:fromUnpack;
endsimple
```

DecideCola

- Le llega paquete de red, lo envía a una cola (ráfaga)
- Selecciona la cola (ráfaga) de forma aleatorio: ***intuniform***

```
void DecideCola::handleMessage(cMessage *msg)
{
    //Sólo pueden llegar de un sitio, de network

    //Decide a que cola mandar el paquete
    int cola=intuniform(0, nColas-1);

    //Manda el paquete a la cola correspondiente
    send(msg, "out", cola);
}
```

ColaRafaga

- Representa al formador de ráfaga de OBS
- Decide cuando se ha formador una ráfaga
  - Temporizador: tiempoMax
  - # paquetes: paqMax

ColaRafaga::handleMessage

```
void ColaRafaga::handleMessage(cMessage *msg)
{
   if (msg->arrivedOn("in")) {
       // Si se ha recibido un paquete de network
       //Mete el paquete en la cola
       encapsulate(msg);
       numLlegados++;
   } else if (msg==mandarRafaga) {
       //Si hay que mandar una rafaga
       int j=0;
       //Crea la rafaga y la manda a EnviaRafaga
       OBSFrame *rafaga= new OBSFrame("rafaga");
       int i;
       int nPaquetes=mensajesRafaga.size();
       rafaga->setNumPaquetes(nPaquetes);
       //Mete cada uno de los paquetes de la cola
       for(i=0;i<nPaquetes;i++) {
         cMessage* m = mensajesRafaga.back();
         mensajesRafaga.pop_back();
         rafaga->insertMessage(m);
       }
       rafaga->setLength(tamRafaga);
       send(rafaga, "out");
       tamRafaga=0;
   }
}
```

ColaRafaga::encapsulate

```
void ColaRafaga::encapsulate(cMessage *msg)
{
   tamRafaga+=msg->length();
   if (mensajesRafaga.size()==0) {
      //Activo el contador
     //Esto tambien habra que parametrizar
      scheduleAt(simTime()+tiempoMax, mandarRafaga);
   }
   mensajesRafaga.push_back(msg);
   if (mensajesRafaga.size() >= paqMax) {
     cancelEvent(mandarRafaga);
     scheduleAt(simTime(),mandarRafaga);
   }
}
```

EnviaRafaga

- Se encarga del envío de las ráfagas:
  - Cuando le llega ráfaga:
    - Si tiene todas las lambdas ocupadas almacena en cola propia
    - Si tiene lambdas libres pero lambda de control ocupada (transmisión de mensaje de control) almacena en cola propia
    - Si puede enviar:
      - selecciona lambda
      - envía mensaje de control
      - envía ráfaga
      - envía mensaje de fin de transmisión
  - Cuando acaba transmisión comprueba cola interna

EnviaRafaga::registerInterface

```
InterfaceEntry *EnviaRafagas::registerInterface(double datarate)
{
    InterfaceEntry *e = new InterfaceEntry();

    // interface name: our module name without special characters ([])
    char *interfaceName = new char[strlen(parentModule()->parentModule()->fullName())+1];
    char *d=interfaceName;
    for (const char *s=parentModule()->parentModule()->fullName(); *s; s++)
        if (isalnum(*s))
            *d++ = *s;
    *d = '\0';
    e->setName(interfaceName);
    delete [] interfaceName;

    // data rate
    e->setDatarate(datarate);
```

EnviaRafaga::registerInterface

```
    // generate a link-layer address to be used as interface token for IPv6
    InterfaceToken token(0, simulation.getUniqueNumber(), 64);
    e->setInterfaceToken(token);

    // MTU: typical values are 576 (Internet de facto), 1500 (Ethernet-friendly),
    // 4000 (on some point-to-point links), 4470 (Cisco routers default, FDDI compatible)
    e->setMtu(4470);

    // capabilities
    e->setMulticast(true);
    e->setPointToPoint(true);

    // add
    InterfaceTable *ift = InterfaceTableAccess().get();
    ift->addInterface(e, this);

    return e;
}
```

EnviaRafaga::handleMessage

```
void EnviaRafagas::handleMessage(cMessage *msg)
{
	string strEndTxControl = "endTxControl";
	if (msg->arrivedOn("fromUnpack")) {
		// Si se ha recibido un paquete en unPack
		// fire notification
		// ... //
		delete msg;
	} else if (strEndTxControl.compare(msg->name())==0) {
		//Cuando se termine de transmitir el paquete de control, se manda la rafaga asociada
		ocupacionCanales[canalControl]=false;
		startTransmitting(siguienteRafaga[msg->kind()], msg->kind());
		delete msg;
	} else if (msg->arrivedOn("in")) {
		// Ha llegado rafaga para mandar

		if (ocupacionCanales[canalControl]==true) {
			// Meto paquete en cola
			txQueue.insert(msg);
		} else {
		// Ver si alguna lambda libre
		int puerta;
		int j=0;
```

EnviaRafaga::handleMessage

```
//Si todas ocupadas la guardo, si todas libres lo mando por una aleatoria, si alguna libre la mando por la 1ª libre
bool todasOcupadas=true;
bool todasLibres=true;
while(j<(lambda-1)) {
        if (ocupacionCanales[j]==true) todasLibres=false;
        else {
                puerta=j;
                todasOcupadas=false;
        }
        j++;
}

if (todasOcupadas==true){
        // ... //
        //Almacena el paquete en la cola de transmision
        txQueue.insert(msg);
} else {
        //Si todas las puertas estan libres la mando por una aleatoria
        if (todasLibres==true) puerta=intuniform(0, lambda-2);
        ocupacionCanales[puerta]=true;
```

EnviaRafaga::handleMessage

```
// Si hay en la cola guardo el recien llegado y cojo uno de ella
if (!txQueue.empty())
{
        txQueue.insert(msg);
        cMessage* rafaga = (cMessage *) txQueue.getTail();
        siguienteRafaga[puerta] = check_and_cast<OBSFrame*>(rafaga);
} else {
        siguienteRafaga[puerta] = check_and_cast<OBSFrame*>(msg);
}
// Enviar mensaje de control
OBSControlFrame *control=new OBSControlFrame("control");
control->setLambda(puerta);
control->setSize(siguienteRafaga[puerta]->length());
send(control, "out", canalControl);
ocupacionCanales[canalControl]=true;
// Controlo la finalizacion de la transmision
gateToWatch=gate("out", canalControl);
while (gateToWatch) {
        // ... //
        cMessage *endTxControl = new cMessage(strEndTxControl.c_str());
        endTxControl->setKind(puerta);
        scheduleAt(simTime()+tiempoTransmision, endTxControl);
}
```

EnviaRafaga::handleMessage

```
            }
        }
      } else {
        // Se ha terminado una transmision
        ocupacionCanales[puertaOrigen]=false;
        //Se manda el mensaje de fin de transmision
        int puertaOrigen=msg->kind();
        cMessage* endTxRafaga= new cMessage("endTxRafaga");
        send(endTxRafaga,"out",puertaOrigen);
        // ... //
        // Mirar si la cola tiene ráfaga
        if (!txQueue.empty())
        {
                // Saco ráfaga más vieja y envío
                // ........................ //
        }
        delete msg;
      }
}
```

EnviaRafaga::startTransmitting

```
void EnviaRafagas::startTransmitting(cMessage* msg, int puerta)
{
	//Manda el mensaje por la puerta indicada
	send(msg, "out", puerta);

	// ... //

	char *nombre=(char*)malloc(strlen("endTxEvent")+2);
	sprintf(nombre, "endTxEvent%d", puerta);
	cMessage *endTxEvent=new cMessage(nombre);
	//En el kind guardamos el destino
	endTxEvent->setKind(puerta);
	scheduleAt(simTime()+tiempoTransmision, endTxEvent);
}
```

EnviaRafaga

- Ráfaga sólo “transmitir” 1 vez, entre router de entrada y primer salto. En el resto de saltos (switches) sólo se revota a otra salida.
- Para simular:
  - Canales con ancho de banda infinito (se implementa indicando que el ancho de banda es 0) => el simulador envía ráfaga instantánea, sin tiempo de transmisión
  - Se manda un mensaje de control en el instante en que se supone que la ráfaga debería de terminar de transmitirse en el primer salto, y el router frontera destino no será capaz de procesar la ráfaga hasta recibir este mensaje, por que habrá de esperar un tiempo equivalente a una transmisión de la ráfaga antes de poder procesarla.

OBSUnpack

- Se encarga de recibir las ráfagas de la red, desempaquetar los paquetes y enviárselos a la capa superior

```
void OBSUnpack::initialize(int stage)
{
    lambda=par("lambda");
    memoriaRafaga = new OBSFrame *[lambda-1];
    int j;
    for(j=0;j<lambda-1;j++){
        memoriaRafaga[j] = NULL;
    }
}
```

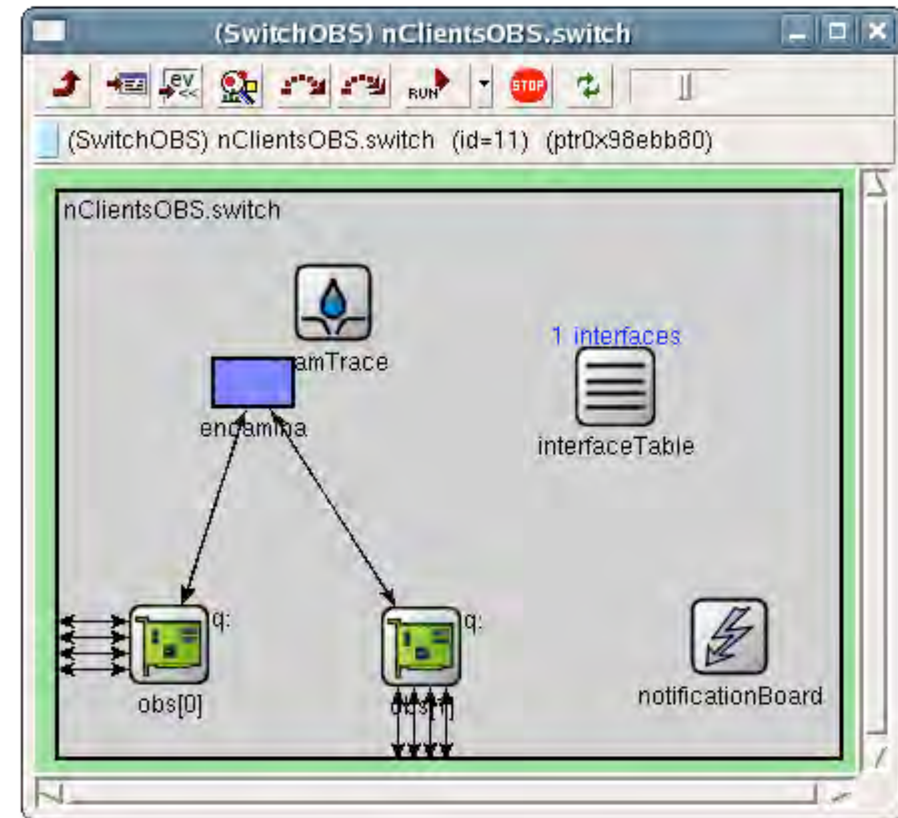
OBSUnpack::handleMessage

```
void OBSUnpack::handleMessage(cMessage *msg)
{
    if (dynamic_cast<OBSControlFrame *>(msg) != NULL) {
      OBSControlFrame *control=check_and_cast<OBSControlFrame*>(msg);
      int puerta=control->getLambda();
      delete msg;
    } else if (dynamic_cast<OBSFrame *>(msg) != NULL) {
      // Obtener id de la puerta de llegada
      const char *puertaLlegada=msg->arrivalGate()->fullName();
      // ... //
      memoriaRafaga[id] = check_and_cast<OBSFrame*>(msg);
    } else { //Si es un mensaje de fin de ráfaga
      // Obtener id de la puerta de llegada
      const char *puertaLlegada=msg->arrivalGate()->fullName();
      // ... //
      OBSFrame* rafaga = memoriaRafaga[id];
      memoriaRafaga[id] = NULL;
      if (rafaga->hasBitError()) {
            // Rafaga con error => desechar
      } else {
            int numPaquetes=rafaga->getNumPaquetes();
            for(i=0;i<numPaquetes;i++) {
                  avisoLlegada= new cMessage(*msg);
                  send(avisoLlegada, "toPack");
                  cMessage *payload = rafaga->nextMessage();
                  send(payload,"out");
            }
      }
      delete rafaga; delete msg;
    }
}
```

SwitchOBS

- Igual que Switch de INET
  - Conmutación entre entradas y salidas
- Pero SOLO interfaces OBS

- Todos los interfaces tienen que tener el mismo número de lambdas



SwitchOBS.ned

```
simple Encamina
    parameters:
      routingFile: string,
      lambda: numeric const,
      numInterfaces: numeric const;
    gates:
      in:in[];
      out:out[];
endsimple

module SwitchOBS
  parameters:
    routingFile: string,
    routingSwitch: string,
    numInterfaces: numeric const,
    lambda: numeric const;
  gates:
    in: OBSin[];
    out: OBSout[];
  submodules:
    encamina:Encamina;
      parameters:
          routingFile=routingSwitch,
          lambda=lambda,
          numInterfaces=numInterfaces;
    obs: OBSInterface2[numInterfaces];
      parameters:
        lambdaInterface = lambda;
      display: "q=l2queue;i=block/ifcard";
  connections nocheck:
    for i=0..numInterfaces-1,j=0..lambda-1 do
      obs[i].toSwitch++ --> encamina.in++;
      obs[i].fromSwitch++ <-- encamina.out++;
    endfor;
    for i=0..numInterfaces-1,j=0..lambda-1 do
      OBSin[i*lambda+j] --> obs[i].physIn++;
      OBSout[i*lambda+j] <-- obs[i].physOut++;
    endfor;
endmodule
```

OBSInterface2

OBSInterface2.ned

```
simple OBSInterface2
    parameters:
        lambdaInterface: numeric;
    gates:
        in: fromSwitch[];
        out: toSwitch[];
        in: physIn[];
        out: physOut[];
        in: netwIn;
        out: netwOut;
endsimple
```

OBSInterface.cc

```
void OBSInterface2::handleMessage(cMessage *msg)
{
        if (msg->arrivedOn("physIn")) {
          // llega mensaje del medio fisico => encaminador
          // Obtener la id de la puerta de llegada y salida
          // ... //
          send(msg, "toSwitch",id);
        } else if (msg->arrivedOn("fromSwitch")) {
          // llega mensaje del switch => medio fisico
          // Obtener la id de la puerta de llegada y salida
          // ... //
          send(msg, "physOut", id);
        } else {
          // No debería de pasar nunca => borrar
          delete msg;
        }
}
```

Encamina

```
void Encamina::handleMessage(cMessage *msg)
{
	//Si llega por la puerta in[X], hay que consultar la entrada X-floor(x/(lambda+1)) de la tabla de rutas
	// hay que restar porque X tiene en cuenta las lambdas de los canales de control, y esas entradas no aparecen en la
	tabla de rutas por ser constantes.

	// Obtener la id de la puerta de llegada y salida
	// ... //

	short indice=id;
	short idInterfazEntrada=tablaRutas[indice].idInterfazEntrada;
	short lambdaEntrada=tablaRutas[indice].lambdaEntrada;
	short interfazSalida=tablaRutas[indice].idInterfazSalida;
	short lambdaSalida=tablaRutas[indice].lambdaSalida;
	short idSalida=interfazSalida*lambda+lambdaSalida;

	send(msg, "out", idSalida);
}
```

```
//Esta estructura representa una entrada en la tabla de rutas
// Se rellena en la inicialización usando el fichero de rutas
struct entradaTabla {
	short idInterfazEntrada;
	short lambdaEntrada;
	short idInterfazSalida;
	short lambdaSalida;
};
```

SimuladorPerdidas

- Sirve para introducir pérdidas a ráfagas en el sistema OBS
- Por ahora sólo es capaz de generar pérdidas independientes
- Se basa en el mismo planteamiento del SwitchOBS, sustituyendo el "encaminador" por el módulo simple "PerdidasOBS"

```
simple PerdidasOBS
    parameters:
      lambdaInterface: numeric const,
      perdidas: numeric const;
    gates:
      in:in[];
      out:out[];
endsimple
```

PerdidasOBS

```
void PerdidasOBS::handleMessage(cMessage *msg)
{
        bool $borrado = false;

        // Obtener la id de la puerta de llegada y salida
        // ... //

        if (dynamic_cast<OBSControlFrame *>(msg) != NULL) {
               // Es un paquete de control => Calcular perdida
              if (dblrand() < perdidas) {
                     OBSControlFrame *control =
                     check_and_cast<OBSControlFrame *>(msg);
                     canalPerdidas[control->getLambda()] = true;
                     $borrado = true;
                     delete msg;
               }
        } else if (dynamic_cast<OBSFrame *>(msg) != NULL) {
               // Es una rafaga => mirar si en su canal perdidas
               if (canalPerdidas[id]==true) {
                     $borrado = true;
                     delete msg;
              }
            } else {
               // Mensaje fin de rafaga
               if (canalPerdidas[id]) {
                     $borrado = true;
                     delete msg;
               }
               // Inicializar canal
              canalPerdidas[id] = false;
         }

         if ($borrado == false) {
           // Puerta de salida
           int puerta;
           if (id < lambda) {
                     puerta = id + lambda;
           }
           else {
                     puerta = id - lambda;
           }

           send(msg, "out", puerta);
         }
}
```

Simulación OBS: parámetros

```
# identificadores de los interfaces OBS de los routers
**.r1.obs[0].obsPack.envia.idInterface=1;
**.r2.obs[0].obsPack.envia.idInterface=1;

# Formador de ráfaga por tamaño (en número de paquetes)
**.obs[*].obsPack.almacena[*].paqMax=3;
# Formador de ráfaga por temporizador (en segundos)
**.obs[*].obsPack.almacena[*].tiempoMax=1;

#Numero de lambdas (incluyendo la de control)
**.lambdaGeneral=4;

# colores de las lambdas
**.colores="1 2 3 4";

# Capacidad de la cola interna OBS para mantener las ráfagas
**.obs[*].obsPack.queueType = "DropTailQueue"
**.obs[*].queue.frameCapacity = 100    # in routers
```