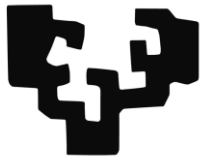
Simulación de equipos de transmisión en redes con enlaces bajo protocolo ARQ. Simulación de los protocolos, multiplexación de enlaces y función de conmutación.

Caso 3: Simulación de equipos y protocolos con OMNET++

Máster en Ingeniería de Telecomunicación



Iñigo Pérez Bada

Índice

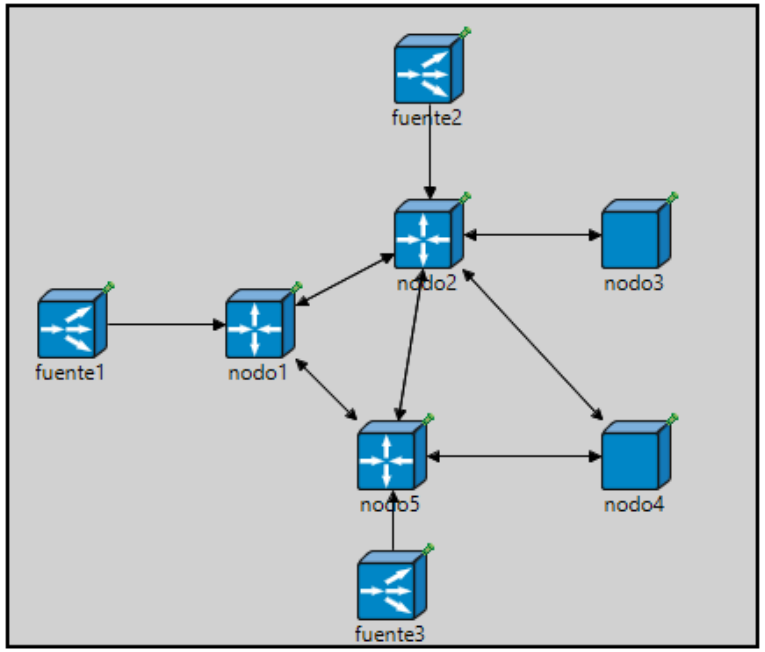
Introducción

Modelo

Fuente

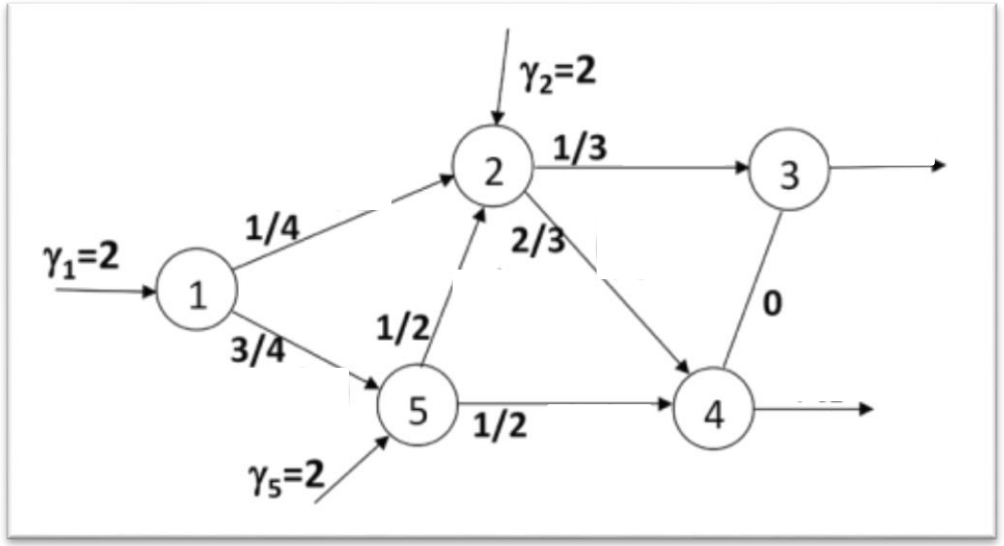
Nodo

Fin



Especificaciones

- Lambda: 2
- Capacidad: 3
- Fuentes: 3
- Nodos: 5



Metodología

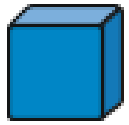
1. **Lógica a seguir**
2. **Definición de componentes**
3. **Definición del paquete**
4. **Implementación del protocolo**
5. **Estadísticas**

Desarrollo: Definiciones

MyNetwork.ned

```
simple Nodo
{
    parameters:
        @display("i=abstract/router2");
        double probability;
    gates:
        input inPort[];
        output outPort[];
}

simple Fin
{
    parameters:
        @display("i=abstract/server");
    gates:
        input inPort[];
        output outPort[];
}
simple Fuente
{
    parameters:
        @display("i=abstract/dispatcher");
        double tampaquete = default(1000/3);
        double lambda = default(2);
    gates:
        output outPort;
}
```



Omnetpp.ini

```
[General]
network = MyNet
MyNet.fuente1.lambda=2
MyNet.fuente1.tampaquete=1000
MyNet.fuente2.lambda=2
MyNet.fuente2.tampaquete=1000
MyNet.fuente3.lambda=2
MyNet.fuente3.tampaquete=1000
```

Desarrollo: Diseño de la red


```
network MyNet
{
    types:
        channel Channel extends ned.DatarateChannel
        {
            datarate = 1000bps;
            delay = 100us;
            per = 0.25;
        }
    submodules:
        nodo1: Nodo {probability = 1/4;gates:inPort[3];outPort[2];}
        nodo2: Nodo {probability = 1/3;gates:inPort[5];outPort[4]}
        nodo3: Fin {gates:inPort[1];outPort[1]}
        nodo4: Fin {gates:inPort[2];outPort[2];}
        nodo5: Nodo {probability = 1/2;gatesinPort[4];outPort[3];}
        fuente1: Fuente {}
        fuente2: Fuente {}
        fuente3: Fuente {}
```

```
connections:
    nodo1.outPort[0] --> Channel --> nodo2.inPort[2];
    nodo1.inPort[0] <-- Channel <-- nodo2.outPort[2];

    nodo1.outPort[1] --> Channel --> nodo5.inPort[2];
    nodo1.inPort[1] <-- Channel <-- nodo5.outPort[2];

    nodo2.outPort[0] --> Channel --> nodo3.inPort[0];
    nodo2.inPort[0] <-- Channel <-- nodo3.outPort[0];

    nodo2.outPort[1] --> Channel --> nodo4.inPort[0];
    nodo2.inPort[1] <-- Channel <-- nodo4.outPort[0];

    nodo5.outPort[0] --> Channel --> nodo2.inPort[3];
    nodo5.inPort[0] <-- Channel <-- nodo2.outPort[3];

    nodo5.outPort[1] --> Channel --> nodo4.inPort[1];
    nodo5.inPort[1] <-- Channel <-- nodo4.outPort[1];

    fuente1.outPort --> nodo1.inPort[2];
    fuente2.outPort --> nodo2.inPort[4];
    fuente3.outPort --> nodo5.inPort[3];
```

Desarrollo: Generación de paquetes

- int inicial;
- int numSeq;
- int origen;
- int ruta[4];
- simtime_t rutaTimes[4];

```
MyNetMsg* Fuente::getPacket() {
    std::string packetName = "packet::" + std::to_string(getId()) + "::"
            + std::to_string(numSeq);
    char *charPacketName = new char[packetName.length() + 1];
    strcpy(charPacketName, packetName.c_str());
    MyNetMsg *pkt = new MyNetMsg(charPacketName);
    pkt->setInicial(true);
    pkt->setKind(1);
    pkt->setNumSeq(numSeq);
    pkt->setOrigen(getId());
    pkt->setRuta(0, getId());
    pkt->setRutaTimes(0, simTime());
    numSeq++;
    return pkt;
}
```

Desarrollo: Generación de flujo

initialize()

```
lambda = (double) par("lambda");
tampaquete = (double) par("tampaquete");
// Get departure times, generate packets and schedule them
MyNetMsg *pkt = getPacket();
pkt->setBitLength(getLengths(tampaquete));
scheduleAt(simTime().dbl() + getDepartures(lambda), pkt);
```

handleMessage()

```
MyNetMsg *pkt = check_and_cast<MyNetMsg*>(msg);
send(pkt, "outPort");
MyNetMsg *myPkt = getPacket();
myPkt->setBitLength(getLengths(tampaquete));
scheduleAt(simTime().dbl() + getDepartures(lambda), myPkt);
```

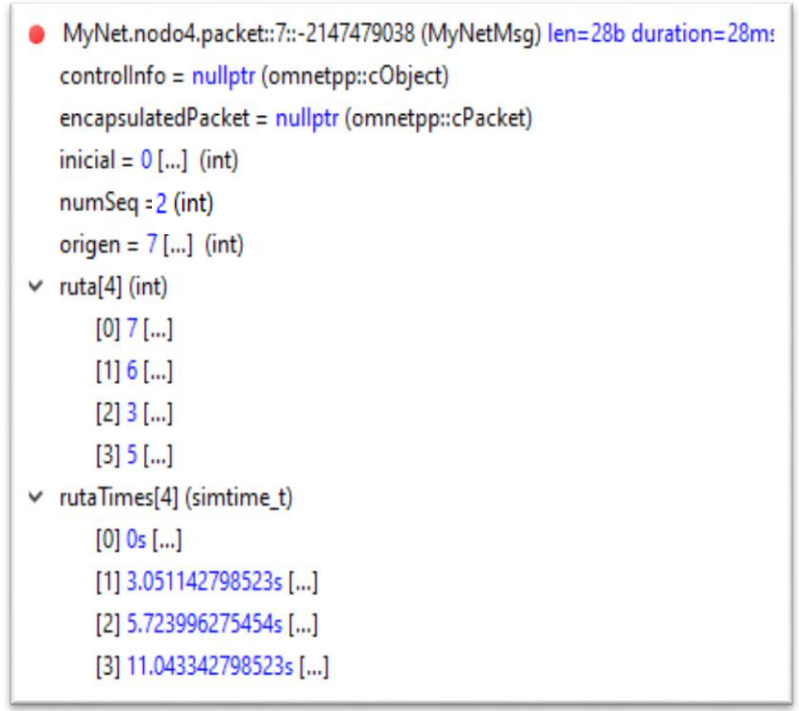
```
double Fuente::getDepartures(double lambda) {
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator (seed);
    std::uniform_real_distribution<double> uniformRandom(0.0, 1.0);
    double randomNumber = uniformRandom(generator);
    double number = (-1 / lambda) * log(randomNumber);
    return number;
}

double Fuente::getLengths(double tampaquete) {
    unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
    std::default_random_engine generator (seed);
    std::uniform_real_distribution<double> uniformRandom(0.0, 1.0);
    double randomNumber = uniformRandom(generator);
    double number = (-tampaquete) * log(randomNumber);
    return number;
}
```

Desarrollo: Seguimiento del paquete

- Registro de ruta
- Registro de timestamps en la ruta

```
for (int i = 0; i < 4; i++) {
    EV
              << "La posicion " + std::to_string(i)
                     + " en la ruta es: "
                     + std::to_string(pkt->getRuta(i));
    if (pkt->getRuta(i) == 0) {
        pkt->setRuta(i, getId());
        pkt->setRutaTimes(i, simTime());
        EV
                  << "Guardando en la posicion "
                         + std::to_string(i) + " de la ruta : "
                         + std::to_string(pkt->getId());
        break;
    }
}
```



Desarrollo: Implementación del protocolo

1. Envío de NAK
2. Envío de ACK

```
if (pkt->getKind() == 1) { // 1: Packet
    if (pkt->hasBitError()) {
        EV << "Packet arrived with error, send NAK\n";
        MyNetMsg *nak = new MyNetMsg("NAK");
        nak->setKind(3);
        send(nak, "outPort", arrivalGateIndex);
    } else {
        EV << "Packet arrived without error, send ACK\n";
        MyNetMsg *ack = new MyNetMsg("ACK");
        ack->setKind(2);
        send(ack, "outPort", arrivalGateIndex);
        EV << "Packet it's okav!";
```

3. Timeouts

```
if (uniform(0, 1) < 0.05) {
    EV << "\"Losing\" message.\n";
    bubble("message lost");  // making animation more informative...
} else {
```

Desarrollo: Implementación del protocolo

1. Reenvíos con Timeout

```
if (strcmp(msg->getName(), "timeoutEvent") == 0) {
    // If we receive the timeout event, that means the packet hasn't
    // arrived in time and we have to re-send it.
    EV << "Timeout expired, resending message and restarting timer\n";
    cGate *lastarrivalGate = lastpkt->getArrivalGate();
    int lastarrivalGateIndex = lastarrivalGate->getIndex();
    if (uniform(0, 1) < 0.33) {
        sendNext(0);
    }else{
        sendNext(1);
    }
    scheduleAt(simTime() + timeout, timeoutEvent);
```

1. Reenvíos con NAK

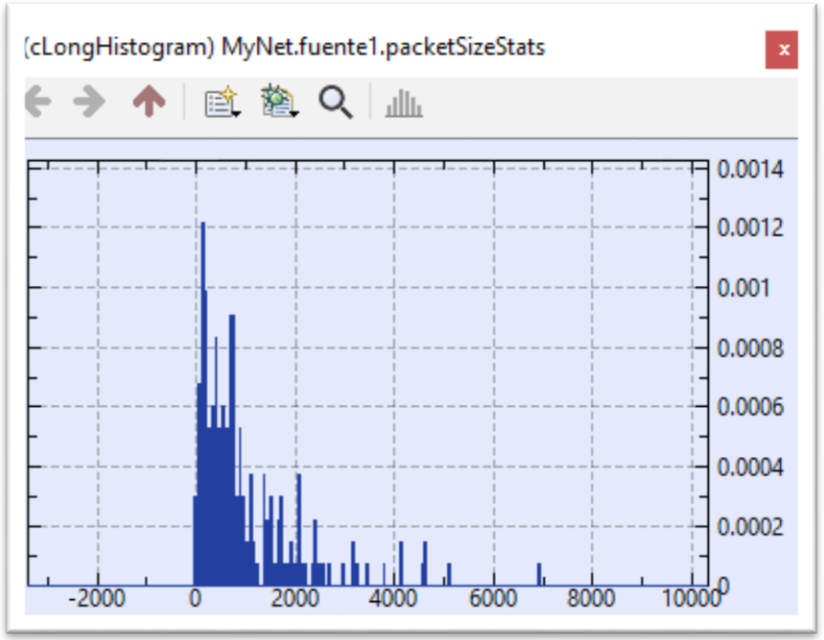
```
} else if (pkt->getKind() == 2) { // 2: ACK
    EV << "ACK from next node\n";
    if (queue[arrivalGateIndex]->isEmpty())
        EV
                << "WARNING: there are not packets in queue, but ACK arrived\n";
    else {
        // pop() removes queue's first packet
        queue[arrivalGateIndex]->pop();
        sendNext(arrivalGateIndex);
        scheduleAt(simTime() + timeout, timeoutEvent);
```

Estadísticas

1. Registro de mensajes enviados

2. Promedio de tamaño de paquetes y fidelidad de lambda.

```
MyNet.fuente1 (Fuente) id=7
  outPort (cGate) --> nodo1.inPort[2]
  tampaquete (cPar) 1000
  lambda (cPar) 2
  muestras (cPar) 1000
  lambdaVector (cOutVector) received 116 values, stored 116
  lambdaStats (cLongHistogram) count=116 mean=29 stddev=18.3881 min=0 max=60
  packetSize (cOutVector) received 116 values, stored 116
  packetSizeStats (cLongHistogram) count=116 mean=1047.73 stddev=1194.63 min=21 max=6869
  numSent (long) 116
```



Conclusiones

1. Orientado a la simulación de redes no como Mathematica.
2. Mayor support de funciones propias a la hora de trabajar con los paquetes.
3. Mayor complejidad de diseño que en Mathematica

¿Preguntas?

Caso 3: Simulación de equipos y protocolos con OMNET++

Máster en Ingeniería de Telecomunicación

Iñigo Pérez Bada