Universitat de les Illes Balears

escola politècnica superior

Treball Fi de Grau

GRAU D'ENGINYERIA TELEMÀTICA

Modelització i anàlisi de protocols ARQ amb OMNET++

MARTÍ PONS MAYOL

Tutor
Ignasi Furió Caldentey

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, 5 de juliol de 2018

SUMARI

ÍNDEX DE FIGURES

ACRÒNIMS

OSI Open System Interconnection

ARQ Automatic Repeat ReQuest

SW Stop-and-wait ARQ

GBN Go-Back-N ARQ

SR Selective Repeat ARQ

ACK Acknowledgment

NACK Negative Acknowledgement

IDE Integrated Development Environment

NED Network Description

BER Bit Error Rate

API Application Programming Interface

RESUM

Els protocols Automatic Repeat ReQuest (ARQ) s'utilitzen com un dels mecanismes de control d'errors tant a la capa d'enllaç de dades, com a la capa de transport de la pila de protocols Open System Interconnection (OSI).

Degut a la seva importància són protocols que encara que són senzills s'han d'aprendre i entendre bé al grau d'enginyeria telemàtica. En aquest treball es modela un simulador per a cada un dels protocols ARQ amb l'OMNET++, un framework basat en C++ molt potent creat per construir simuladors de xarxes. A partir d'aquests simuladors es pot fer un estudi de les prestacions i del rendiment que ofereixen els protocols així com veure amb una animació a temps real com funcionen.

CAPÍTOL 1

INTRODUCCIÓ

Al nostre grau i en els graus d'enginyeria en general s'estudien molts de conceptes teòrics complexes i difícils d'assimilar sense dedicar-hi esforç en el seu estudi. Generalment posar en pràctica aquests fonaments teòrics, ja sigui resolent problemes, o utilitzant programari acadèmic que permet fer simulacions o veure aquests conceptes de manera gràfica i amb resultats és la millor manera d'entendre definitivament i de repassar el que es vol ensenyar amb aquests fonaments.

Per tant, totes les eines que es descobreixin i es puguin utilitzar perquè els alumnes facin aquestes pràctiques són benvingudes.

En aquest treball analitzarem l'eina OMNET++ per determinar si és adecuada per emprar en pràctiques de les assignatures on s'estudien protocols de xarxes de telecomunicació com *Fonaments de Xarxes de Telecomunicació* o *Xarxes d'Àrea Local i Intranets*. En concret, l'objectiu és programar un entorn per simular intercanvis de trames utilitzant els següents protocols orientats a connexió de control d'errors:

- **Stop-and-wait ARQ:**
  l'emisor envia una sola trama cada transmissió, posa en marxa un temporitzador i espera a rebre un Acknowledgment (ACK). Si el temporitzador expira o l'ACK arriba amb bits erronis, és a dir, corrupte s'envia la trama que s'havia enviat abans, sinó, s'envia la següent trama.

- **Go-Back-N ARQ:**
  l'emisor té una finestra lliscant amb la que pot enviar N trames abans de rebre l'ACK. Envia les N trames i activa **un únic temporitzador**:

  – Si es reb un ACK corresponent a una de les trames de la finestra, es confirmen totes les trames anteriors a aquell ACK, es llisca la finestra fins aquella posició amb noves trames i es transmeten. Finalment, es reinicia el temporitzador.

  – Si es reb un ACK corrupte o que no correspon a una trama de la finestra d'aquell moment es descarta.

  - Si el temporitzador expira, es retransmeten totes les trames de la finestra d'aquell moment i es reinicia el temporitzador.

- **Selective Repeat ARQ:**
  A part de l'emisor, a diferència del GBN, **el receptor també té una finestra lliscant**, per tant es fa possible la retransmissió exclusiva de les trames perdudes (ens estalviam la retransmissió de la finestra sencera). L'emisor envia les N trames de la finestra i activa **un temporitzador per cada una de les trames**.

  - Si arriba un ACK corresponent a una trama de la finestra d'aquell moment es marca la trama com a confirmada.
  - Si es reb un ACK corrupte o que no correspon a una trama de la finestra d'aquell moment es descarta.
  - Si el receptor detecta que una trama no ha arribat bé envia un Negative Acknowledgement (NACK) i l'emisor quan reb aquest NACK tot d'una envia la trama corresponent.
  - l'emisor llisca la finestra quan la trama més antiga o les trames de la més antiga en davant consecutives de la finestra estan confirmades.

A partir d'aquests simuladors s'extrauran resultats amb gràfiques i paràmetres per poder comparar les prestacions de cada un.

Primer de tot, al capítol 2 explicarem de manera introductòria què és i com s'utilitza l'*OMNET++*. Seguirem amb el capítol 3 dedicat a la estructura del projecte creat amb l'*OMNET++*. Al capítol 4 ens fixarem en els simuladors en sí explicant els detalls importants i els diagrames de flux de cada un. Finalment, en els dos darrers capítols 5 i 6 explicarem respectivament els resultats obtinguts dels simuladors i el seu significat, i si l'eina resulta realment útil per els alumnes o no.

CAPÍTOL 2

INTRODUCCIÓ A L'*OMNET++*

2.1 Descripció de l'*OMNET++*

2.1.1 Introducció

L'*OMNET++* és una plataforma de simulacions de xarxes de telecomunicacions basada en el llenguatje de programació *C++*. Conté un Integrated Development Environment (IDE) propi fet a partir d'*Eclipse (una plataforma de software de codi obert per desenvolupar IDEs)* amb el qual es poden desenvolupar tots els aspectes dels projecte (no són necessaris programaris externs). Està destinat a una programació modular amb la qual es programen *mòduls* simples que envien i reben trames o generen esdeveniments, després s'utilitzen i s'interconnecten amb altres *mòduls* formant components i models que vendrien a ser xarxes d'aquests *mòduls*, així s'aconsegueix reusabilitat de codi i claretat en la seva estructura.

Un exemple perquè s'entengui bé seria programar un *host* i un *switch* com a mòduls i després utilitzar-los N vegades i connectar-los entre ells en un model per formar una LAN. Així és com funciona *OMNET++*. És compatible tant en Windows, com amb Mac OS X i Linux.

2.1.2 Executar les simulacions

L'IDE de l'*OMNET++* permet executar les simulacions de qualsevol manera possible, és a dir, des de executar-les veient amb una animació com els diferents *mòduls* s'intercanvien les trames (com es pot veure a la figura 2.1) i amb un log veient els events seqüencialment amb les seves marques de temps a la simulació; fins a configurar una gran quantitat de simulacions en paral·lel, utilitzant els diferents nuclis del microprocessador, utilitzant diferents valors parametritzables i simulant llargs intervals de temps com hores o dies en pocs segons o minuts d'execució de l'IDE per extreure'n resultats globals com es pot apreciar a la figura 2.2.

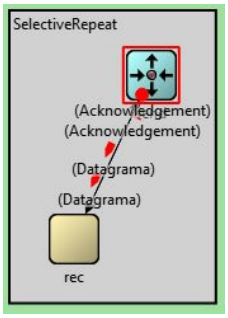

Figura 2.1: Animació d'una simulació

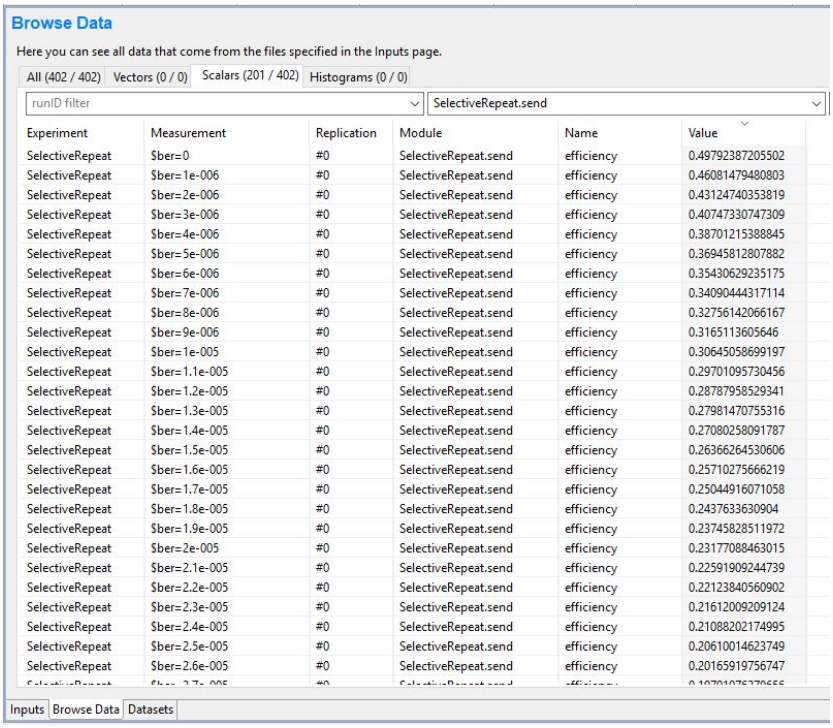

Figura 2.2: Resultats dels valors d'eficiència d'una execució de 201 simulacions amb concurrència amb valors diferents de BER per a cada simulació

2.1.3 Llibreries disponibles

Están disponibles amb total llibertat d'ús un gran ventall de llibreries creades per la comunitat realment útils que estalvien la feina d'haver de programar els mòduls per a realitzar simulacions entre les quals destac:

- INET Framework: més que una llibreria és un framework, el qual conté models dels protocols d'Internet (TCP, UDP, IPv4, IPv6, OSPF, BGP, ...), de protocols de la capa d'enllaç de dades com Ethernet, PPP, IEEE 802.11, ... i molts més.

- SimuLTE: conté models per simular xarxes mòbils com LTE, LTE Advanced, 3G, ...

Utilitzar l'*OMNET++* amb aquestes llibreries podria ser de gran valor perquè els alumnes facin pràctiques per estudiar aquests protocols ja que així, només s'hauria de configurar els models i fer les simulacions.

2.1.4 Comunitat

Aquesta eina té una comunitat molt activa, té dos fòrums principals:

- Un a través de *Google Groups* https://groups.google.com/forum/#!forum/omnetpp on van publicant les actualitzacions que van fent a l'aplicació, les millores i llibreries que van fent i resolen els dubtes que van demanant els usuaris.

- El segon és a través de stackoverflow amb l'etiqueta *OMNET++* enfocat més a dubtes de programació. https://stackoverflow.com/questions/tagged/omnet%2b%2b

Personalment he participat en els dos fòrums demanant petits dubtes i sempre m'han contestat ràpidament, la qual cosa és una avantatge a l'hora de desenvolupar el projecte.

2.2 Flux de feina d'un projecte amb l'OMNET++

Una vegada creat el projecte les passes per desenvolupar el nostre simulador són les següents:

1. Crear i programar els *mòduls* que tendrà el nostre model. Aquests mòduls es fan amb arxius *.cc*, és a dir, són "programets"de *C++* que han d'extendre de la classe *cSimpleModule* i han d'implementar com a mínim els següents mètodes obligatòriament:

    - **initialize():** Aquest mètode s'executa automàticament una vegada compilat el model, sense esperar cap esdeveniment. S'utilitza per inicialitzar les variables, per realitzar les funcions prèvies al primer intercanvi de trames, i per enviar les trames inicials per començar a posar en funcionament la xarxa que s'està simulant.
    - **handleMessage():** s'executa quan el *mòdul* reb un esdeveniment, ja sigui un temporitzador que ha expirat o bé una trames que ha arribat. S'utilitza per processar la trames, realitzar les funcions que pertoquin a partir d'aquest esdeveniment i enviar o no una nova trama.

2. Declarar i descriure la topologia de la xarxa, és a dir, el *model*. Aquesta passa es fa escrivint arxius .ned. El llenguatge Network Description (NED) permet a l'usuari declarar els *mòduls* que vol utilitzar en el *model*, i connectar-los ja sigui a través de canals o directament. També es permet crear mòduls compostos (mòduls formats amb la connexió de mòduls més senzills) a través d'arxius .ned.

3. Proporcionar l'arxiu .ini on s'indiquen els models que utilitzarà el nostre simulador, les configuracions i els paràmetres dels models. També es pot definir que es simuli N vegades amb valors diferents dels paràmetres.

4. Una vegada programat el simulador, podem compilar el projecte i executar-ho d'una de les maneres explicades al capítol 1.

5. Finalment podem obtenir els resultats de les simulacions a través de fitxers amb valors escalars i/o vectors de valors que podrem visualitzar amb gráfiques inclús emmagatzamar-los a una base de dades.

CAPÍTOL 3

L'ESTRUCTURA DEL PROJECTE

3.1 Vista general del projecte

A la figura 3.1 mostram una captura de la vista general de l'estructura del nostre projecte. Anem a explicar cada un dels seus elements:

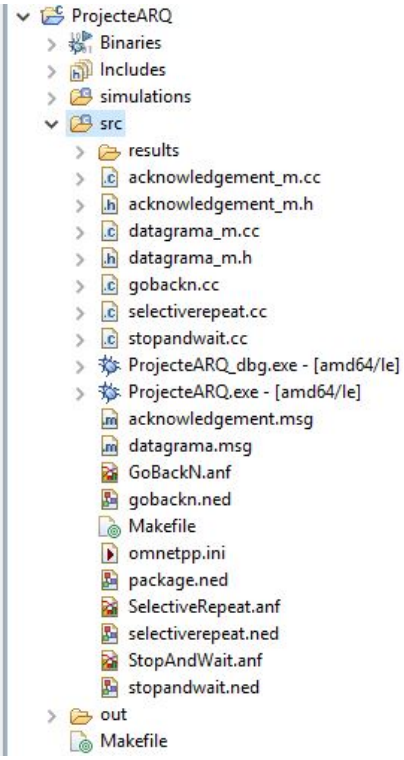

Figura 3.1: Estrctura del projecte

Com es pot observar hi ha el directori arrel del projecte *ProjecteARQ* on es troben quatre carpetes:

- **Binaries:** Executables generats automàticament per l'IDE.
- **Includes:** Llibreries utilitzades tant de *C++* com del propi *OMNET++*.
- **Simulations:** Configuracions de l'*OMNET++*.
- **SRC:** Aquí és on es troben els arxius que formen el projecte en sí.

3.2 Els elements de la font o "src" del projecte

El projecte està format per tres models: *StopAndWait*, *GoBackN*, i *Selective Repeat* on cada un simula un *host* emisor i un *host* receptor que s'intercanvien trames i ACKs utilitzant el seu protocol ARQ corresponent, sense cap dispositiu intermig tipus *switch, router, ...*

Hem decidit simplificar al màxim la topología de la xarxa per centrar tota l'atenció en el protocol en sí i no haver-se de fixar en els distints elements de la xarxa, també per llevar-mos complicacions de més a l'hora de programar els mòduls.

Així doncs, com es pot observar a la figura 3.1 tenim:

- **Els tres models .ned:** On s'indica:
  - **Els *mòduls* del model:** Un emisor i un receptor directament connectats amb un canal. Amb els paràmetres *longitud de la trama en bytes (packetLength), longitud de l'ACK en bytes (ackLengthParam)* en el cas del model *Stop and Wait* i afegint-hi la *longitud de la finestra lliscant en bytes (windowLength)* als altres dos.
  - **El canal:** amb els paràmetres *BER, retard de propagació (delay) en mil·lisegons, i taxa de transmissió (datarate) en Mbps*
- **Tres arxius .cc:** Són els *mòduls* de cada model on està programada la funcionalitat dels *hosts* seguint el protocol corresponent i utilitzant els paràmetres definits al seu model.
- **Dos arxius .msg:** Es defineix la estructura dels tipus de missatge (trama i *ACK*, és a dir, els camps que tenen).
- **Arxius .cc i .h autogenerats:** Són la traducció a codi dels arxius .msg que definim, es generen automàticament quan es compila el projecte.
- **Arxiu *omnetpp.ini*:** Aquest arxiu serveix per indicar els models que utilitzam en el projecte i inicialitzar, és a dir, donar els valors que volem, als paràmetres definits als models (arxius *.ned*). També es poden definir aquí altres tipus de configuracions que no he emprat al projecte.
- **Tres arxius .anf:** Un per a cada model, aquí es recullen els valors obtinguts dels resultats de la simulació i es poden filtrar i pintar amb una gràfica que també es pot personalitzar.

- **Arxius de configuració:** Són els arxius restants *Makefile, package.ned, Projecte-ARQ.exe, ...* autogenerats per l'IDE.

- **Carpeta results:** Conté tots els valors generats per la darrera simulació executada, cal eliminar aquesta carpeta després de cada simulació per a poder obtenir els nous resultats de la pròxima simulació.

CAPÍTOL 4

ELS MÒDULS DEL PROJECTE

4.1 Suposicions i detalls importants

- El *mòdul* de l'emisor i el del receptor estan programats al mateix arxiu .cc que tendrà el nom del seu corresponent protocol ARQ en dues classes, una per cada un.

- S'han programat els simuladors de manera que se suposa que l'emisor sempré té dades per transmetre, és a dir, en el cas de SW sempre que es reb l'ACK correctament es tendrá preparat i s'enviarà una nova trama sense cap període d'inactivitat que simuli que no hi ha dades a nivel de capa 3 *(capa de xarxa)* a transmetre. El mateix passa en les protocols de finestra lliscant, sempre que quedin posicions lliures a la finestra s'ompliran amb noves trames i s'enviaran.

- La duració dels temporitzador emprada és:

$$\text{timeout} = S_{\text{size}}\, t_{\text{transmissió}} + S_{\text{size}}\, t_{\text{transmissióAck}} + 2\, t_{\text{propagació}} \tag{4.1}$$

  On S_{size} és la quantitat de trames de la finestra.

- En el cas de GBN $S_{\text{size}} = 2^m - 1$ on m és el número de bits emprats per els números de seqüència de les trames i és un paràmetre que s'inicialitza a *omnetpp.ini*. És així degut a que en cas contrari, es desincronitzen l'emisor i el receptor. Ho explicaré amb un exemple: si $m = 2$ i $S_{\text{size}} = 2^m = 4$, s'envien les 4 primeres trames i es perden els 4 ACK corresponents, ara el receptor està esperant la trama 0 (del següent cicle) i com que l'emisor enviarà la trama 0 que havia enviat abans el receptor acceptarà aquesta trama 0 com la del següent cicle quan és una duplicada del primer.

- En el cas de SR passa el mateix però amb $S_{\text{size}} > 2^m/2$. Seguint l'exemple anterior: si $m = 2$ i $S_{\text{size}} = 3$ per exemple, s'envien les tres primeres trames i es perden els

tres ACK corresponents l'emisor enviarà la trama 0 duplicada i el receptor la rebrà com si fos una nova perquè el número de seqüència 0 estarà a la seva finestra: en canvi, si $m = 2$ i $S_{size} = 3$ l'emisor envia les dues primeres trames i els dos ACK es perden, quan l'emisor envia la trama 0 duplicada, el receptor la descarta correctament perquè aquell número de seqüència no es troba a la seva finestra.

- Seguint el llibre [2] el receptor, en el cas del protocol SW introdueix el número de seqüència que espera després de rebre la trama a l'ACK, en canvi, en els altres dos protocols de finestra lliscant, introdueix el número de seqüència de la trama que confirma.

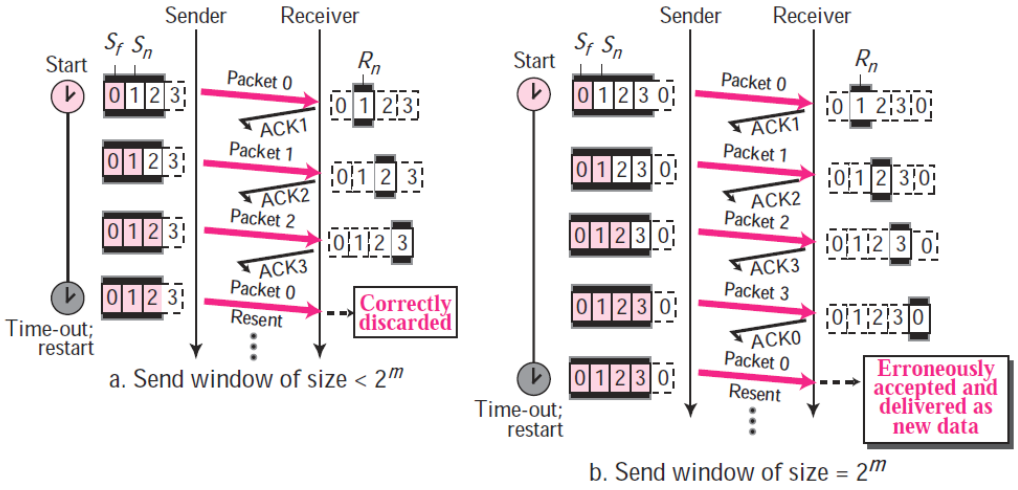

Figura 4.1: Problema de la longitud de la finestra de Go Back N, imatge treta de [1]

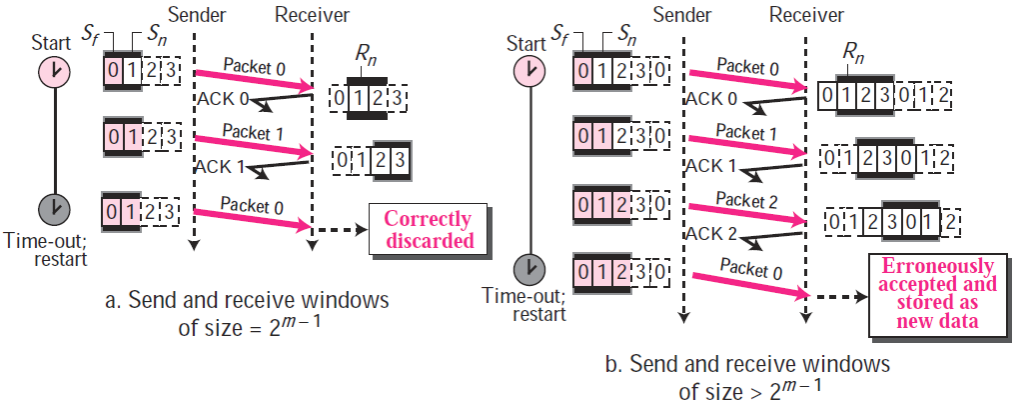

Figura 4.2: Problema de la longitud de la finestra de Selective Repeat, imatge treta de [1]

- Tant la pèrdua de trames com les trames arribades amb bits corruptes es calculen amb la funció *hasBitError()* de la classe *cPacket* que calcula si la trama és correcte aplicant el BER amb el valor aplicat a la simulació a cada bit de la trama.

- Els ACK sempre es reben correctament. S'ha programat així per evitarmos complicacions extres, cal dir que he intentat que els protocols funcionin amb pèrdues d'ACK incloses però en el cas de SR no ho he aconseguit.

En els següents diagrames de flux, cada procés s'ha identificat amb un nombre per poder fer-ne referència més fàcilment. Per tant, a l'hora d'explicar-los la numeració de la llista correspondra amb la numeració dels processos.

4.2 Programació del mòdul Stop-and-wait ARQ

El diagrama de flux del mòdul emisor del protocol SW es representa a la figura 4.3:

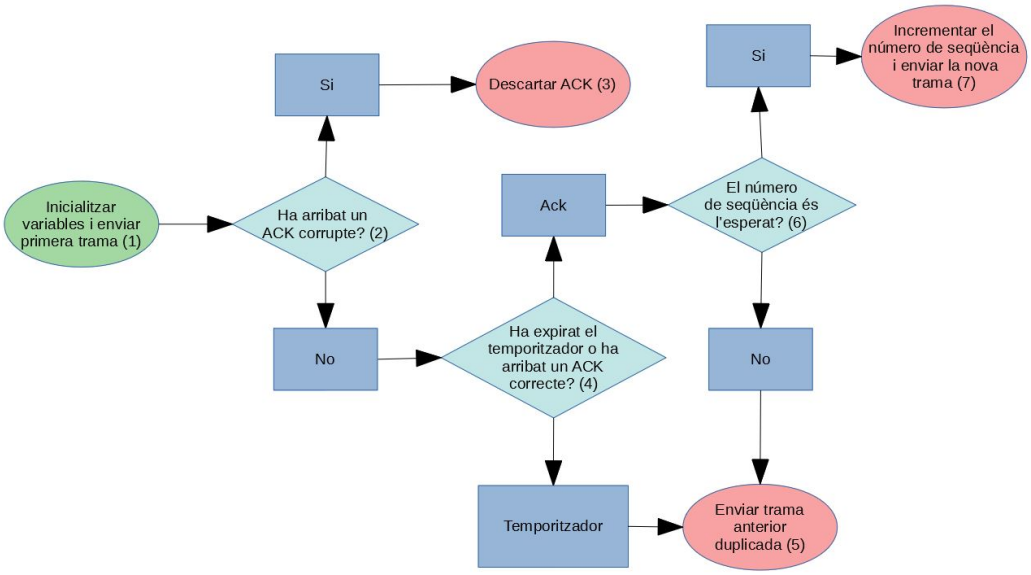

Figura 4.3: Flux emisor SW

Punts a destacar del diagrama:

1. Aquest procés està programat al mètode *initialize()* de la classe *Sender*. S'inicialitzen els següents paràmetres:

   - Un comptador de trames enviades amb èxit *effectiveTransmissions* a 0.
   - La longitud de la trama: Es recull el valor que s'ha definit a l'arxiu *omnetpp.ini*.
   - El retard de transmissió: Tant de la trama com del ACK i es calcula a partir de les seves longituds en bits dividides entre la taxa de transmissió, la qual també està definida com un paràmetre de l'*omnetpp.ini*.
   - El temporitzador: Amb la fórmula descrita al principi del capítol utilitzant el retard de transmissió definit abans, amb el retard de propagació com a paràmetre de l'*omnetpp.ini* i afegint un petit marge de temps més degut

a que si utilitzava exactament aquell interval saltava el temporitzador i arribava l'ACK al mateix temps i es desbaratava el programa.

Finalment es genera i s'envia la primera trama i s'activa el temporitzador perquè comenci el seu compte enrere.

Els següents processos s'executen al mètode *handleMessage()* de la classe *Sender*

2. S'obté l'esdeveniment que ha provocat l'execució del *handleMessage()* i es mira amb un condicional si prové de la recepció d'un ACK corrupte.

```
Datagrama* ack = dynamic_cast<Datagrama*>(msg);
if (ack->hasBitError()) { ...
```

3. S'elimina la trama rebuda i es reinicia el temporitzador si ja havia expirat.
4. **if** (msg == timeoutEvent) ...
5. S'envia la trama que s'ha enviat anteriorment duplicada i es reinicia el temporitzador.
6. Es mira el camp de número de seqüència de l'ACK per determinar si és l'esperat o no.
7. S'incrementa el número de seqüència, s'incrementa el comptador de trames enviades amb èxit, s'envia una nova trama amb aquest número i es reinicia el temporitzador.

Finalment tenim el mètode *finish()* que s'executa una vegada acabada la simulació, on és calcula l'eficiència de la simulació i es guarda el valor de la simulació d'aquell moment per després pintar la gràfica amb tots els valor d'eficiència de totes les simulacions.

```
void Sender::finish()
{
    double efficiencyRun = (effectiveTransmissions *
    transmissionTime.dbl()) / transmissionsTimeTotal.dbl();
    recordScalar("efficiency", efficiencyRun);
}
```

on *transmissionTime* és el retard de transmissió i *transmissionsTimeTotal* l'interval total de temps de la simulació.

El diagrama de flux del mòdul receptor del protocol SW es representa amb la figura 4.4:

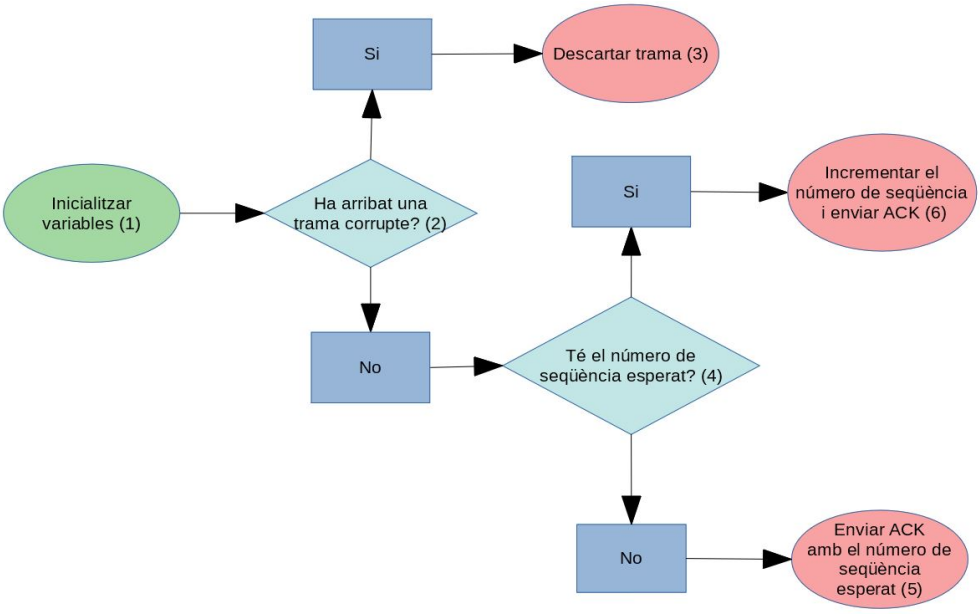

Figura 4.4: Flux receptor SW

1. És el mètode *initialize()* de la classe *Receiver*. Es recull el valor del paràmetre de la longitud en bits de l'ACK *ackLength*.
2. Es mira si ha arribat una trama corrupte.
3. S'elimina la trama.
4. Es comprova si el número de seqüència de la trama rebuda és igual al número de seqüència esperat, que és una variable de la classe *seqNumber*.
5. S'envia l'ACK amb el número de seqüència sense modificar és a dir, el que ja hi havia abans.
6. S'incrementa el número de seqüència, es genera un ACK amb aquest i la longitud definida a *initialize()* i s'envia.

4.3 Programació del mòdul Go-Back-N ARQ

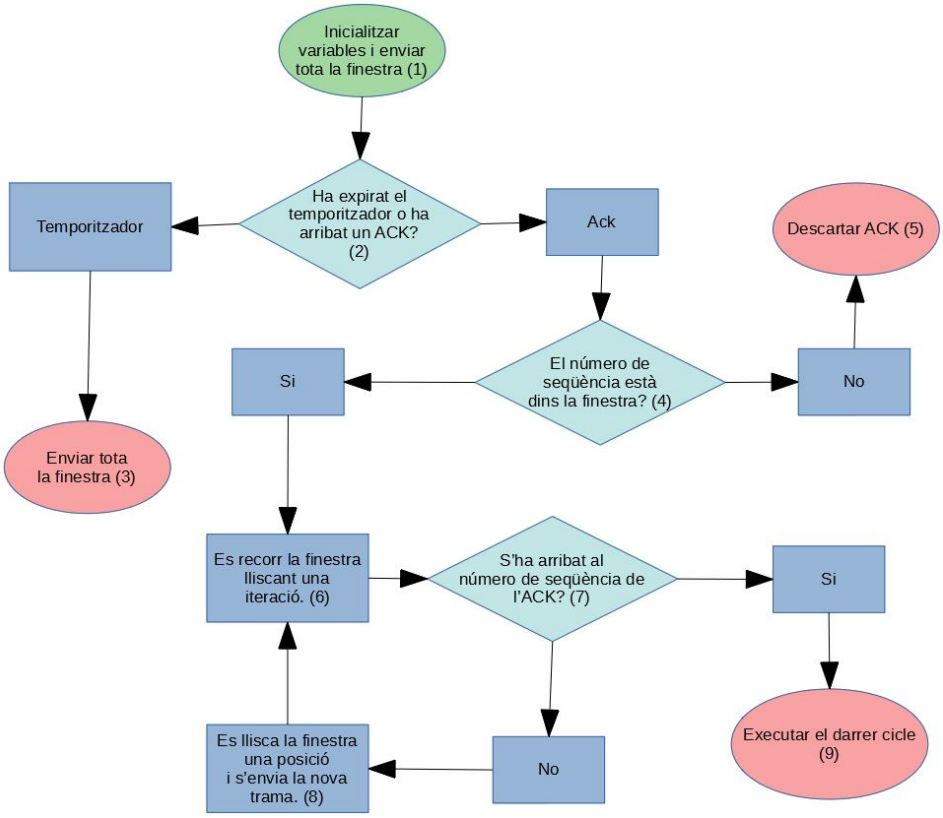

Figura 4.5: Flux emisor GBN

1. *initialize():*

   - Es recull el paràmetre *windowLengthBytes* de l'*omnetpp.ini* que és el número de bits emprats per definir la longitud de la finestra lliscant.

     ```
     m = par("windowLengthBytes");
     Ssize = pow(2, m) - 1;
     ```

   - S'inicialitza el comptador de trames enviades amb èxit *effectiveTransmissions* a 0.
   - Es recull la longitud de la trama.
   - Es recorr en bucle el número de trames que hi caben a la finestra lliscant: Es crea un vector o *array* amb els números de seqüència de les trames de la finestra *finestraIds*, es creen totes les trames, s'envien una darrera l'altra, es guarden els seus números de seqüència a *finestraIds* i es calcula el període d'expiració del temporitzador amb la fórmula de l'inici del capítol i es posa en marxa.

```
for ( int index = 0; index < Ssize; index++ ) {
    Datagrama *paquet = new Datagrama();
    paquet->setLabelId(Sn);
    paquet->setByteLength(packetLength);
    finestraIds.push_back(Sn);
    EV << "Paquet: ID = " + to_string (Sn) + "==>\n";
    //Ssize + 1 = 2^m
    Sn++;
    Sn = fmod(Sn, Ssize + 1);
    simtime_t transmissionDelay = channel->
        getTransmissionFinishTime();
    if ( transmissionDelay <= simTime() ){
        send(paquet, "out");
    } else {
        sendDelayed(paquet, transmissionDelay -
            simTime(), "out");
    }
    packetsSended++;
    EV << "Total de paquets enviats: " + to_string (
        packetsSended) + "\n";
    if ( !timeoutEvent->isScheduled() ) {
        Datagrama *paquetLength = new Datagrama();
        paquetLength->setByteLength(packetLength);
        transmissionTime = channel->calculateDuration(
            paquetLength);
        paquetLength->setByteLength((int)par("
            ackLength"));
        transmissionTimeAck = channel->
            calculateDuration(paquetLength);
        timeout = (Ssize * (transmissionTime) +
            channel->getDelay()) +
        (Ssize * (transmissionTimeAck) + channel->
            getDelay());
        scheduleAt(simTime()+timeout, timeoutEvent);
    }
}
```

- Es guarda el pròxim número de seqüència que s'utilitzarà amb mòdul $S_{\text{size}} + 1$.

```
Sn++;
Sn = fmod(Sn, Ssize + 1);
```

- Es calcula el període d'expiració del temporitzador amb la fórmula de l'inici del capítol i es posa en marxa.

```
timeout = (Ssize * (transmissionTime) + channel->getDelay())
+ (Ssize *(transmissionTimeAck) + channel->getDelay());
scheduleAt(simTime()+timeout, timeoutEvent);
```

2. **if** (msg == timeoutEvent) ...

3. S'elimina l'ACK rebut per destruïr l'objecte i no desperdiciar la memòria del sistema. Després es recorr tot *finestraIds*, s'obtenen tots els números de seqüència, i es generen i s'envien trames duplicades amb aquests números. Una vegada finalitzat el procés es reincia el temporitzador.

4. Es recorr *finestraIds*, es mira si el número de seqüència de l'ACK es troba dins el vector i es guarda l'index on es troba.

5. No es fa res, ja que l'ACK ja ha estat eliminat.

6. Aquí s'expliquen els punts 6, 7, 8 i 9: S'atura el temporitzador perquè no expiri i es crea un bucle que arriba fins l'índex abans guardat. A cada iteració s'elimina la primera posició de l'*array finestraIds*, es genera una nova trama amb un nou número de seqüència *Sn* i la longitud de trama concreta, es crea una nova posició al final de l'*array* i es guarda allà aquest nou número; així aconseguim lliscar la finestra una posició. Després enviam aquest nou paquet i incrementam *effectiveTransmissions*. Una vegada hem lliscat la finestra fins l'índex adecuat i hem enviat totes les trames reiniciam el temporitzador i el tornam a activar.

```
// Sf is the first message sent of the sender
//window that is waiting for acknowledgment.

Sf = ack->getLabelId();

delete msg;

//Check if the acknowledgment belongs
//to an outstanding packet.

int indexPacketConfirmed = -1;

int i = 0;
while ( i < Ssize && indexPacketConfirmed == -1 ) {
    if ( finestraIds[i] == Sf ) {
            indexPacketConfirmed = i;
    }
    i++;
 }
 //Slide the window and send packets
 //if there is a correct acknowledgment

 if ( indexPacketConfirmed != -1 ) {
    EV << "Ack rebut: ID = " + to_string (Sf) +"\n";
    cancelEvent(timeoutEvent);
    for ( int positionsToSlide = 0; positionsToSlide
    <= indexPacketConfirmed; positionsToSlide++ ) {
```

```
            //Delete first message of the window.

            finestraIds.erase(finestraIds.begin());

            //Insert the new message comming of the layer
            //of above at the end of the window and slide it.

            Datagrama *paquet = new Datagrama();
            paquet->setLabelId(Sn);
            paquet->setByteLength(packetLength);
            finestraIds.push_back(Sn);

            //Send the new message

            effectiveTransmissions++;
            transmissionsTimeTotal = simTime();
            EV << "Paquet: ID = " + to_string (Sn) + "==>\n";
            simtime_t transmissionDelay = channel->
                getTransmissionFinishTime();
            if ( transmissionDelay <= simTime() ){
                send(paquet, "out");
            } else {
                sendDelayed(paquet, transmissionDelay -
                    simTime(), "out");
            }
            //Ssize + 1 = 2^m
            Sn++;
            Sn = fmod(Sn, Ssize + 1);

            // Reset the timer

            if ( !timeoutEvent->isScheduled() ) {
                scheduleAt(simTime()+timeout, timeoutEvent);
            }
        }
    }
```

Per acabar tenim el mètode *finish()* que és idèntic al de SW.

El mòdul receptor del protocol GBN és exactament igual al del protocol SW, excepte que els números de seqüència que es van posant als ACK s'incrementen amb mòdul $S_{\text{size}} + 1$ així com les trames en aquest protocol, així que ja està explicat allà.

4.4 Programació del mòdul Selective Repeat ARQ

Els processos del mòdul emisor del protocol SR segueixen el flux que es mostra a la figura 4.6:

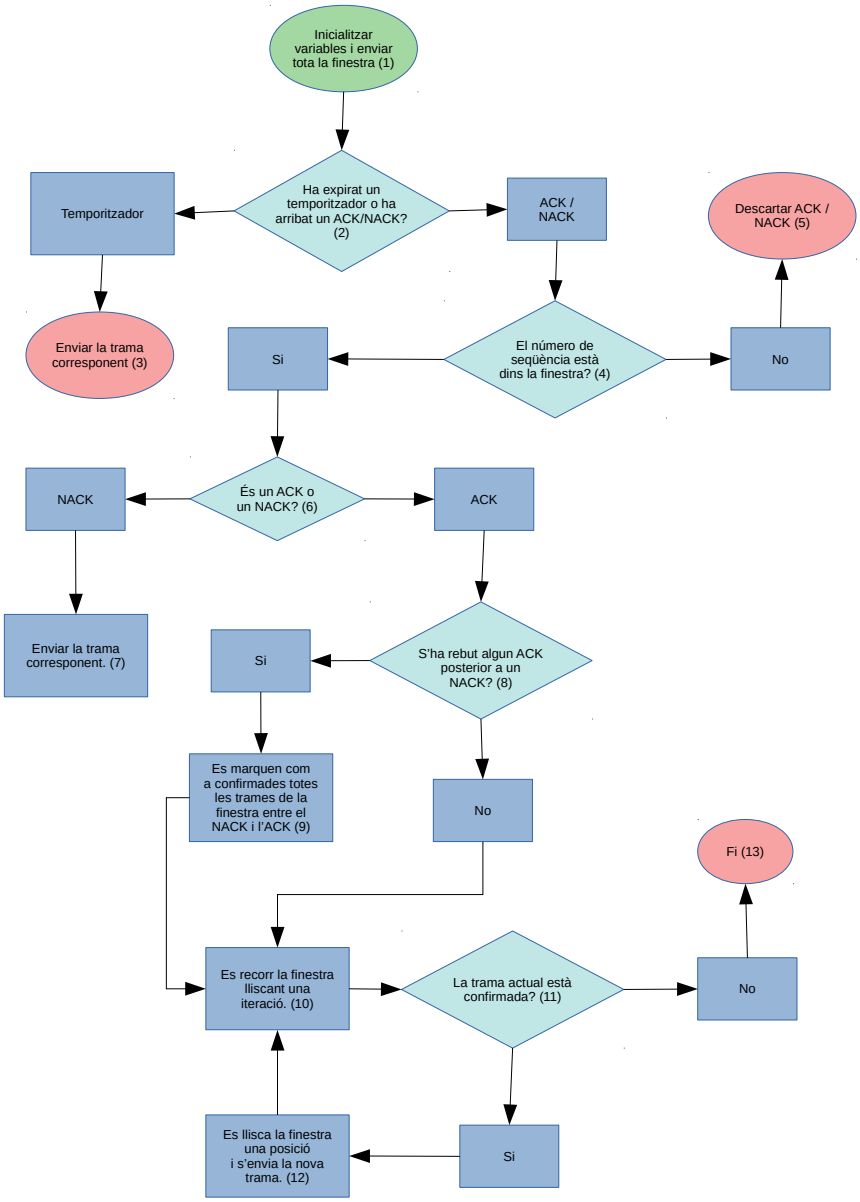

Figura 4.6: Flux emisor SR

1. *initialize():*

   - Recollir els valors de *windowLengthBits* i *packetLength* i calcular la longitud de la finestra S_{size}: Ssize = pow(2, m)/2;.
   - Inicialitzar *effectiveTransmissions* a 0.
   - Inicialitzar l'*array* de números de seqüència de la finestra lliscant de l'emisor. En aquest cas és un array bidimensional, de manera que en la primera fila tendrà tres possibles valors númerics a cada posició: el valor "0" si s'ha enviat la trama i encara no s'ha rebut confirmació; el valor "1" si ha arribat l'ACK per a aquella trama; i el valor "2" si s'ha rebut un NACK per a aquella trama. La segona fila tendrà el número de seqüència de la trama de cada posició de la finestra lliscant.
   - Inicialitzar l'*array* de temporitzadors *timersbufferEmisor*, un per a cada posició de la finestra lliscant.
   - Recórrer en bucle tota la finestra i a cada iteració: Generam una trama, omplim amb "0" i el número de seqüencia les respectives files de *finestraTramesEmisor*, incrementam amb mòdul $2S_{size}$ el pròxim número de seqüència, generam i activam el temporitzador d'aquella posició amb la fórmula també utilitzada al protocol GBN i el guardam al *timersbufferEmisor*. Així per cada posició de la finestra lliscant.

```
for ( int index = 0; index < Ssize; index++ ) {
Datagrama * paquet = new Datagrama();
paquet->setLabelId(Sn);
paquet->setByteLength(packetLength);
finestraTramesEmisor.at(0).push_back(0);
finestraTramesEmisor.at(1).push_back(Sn);
EV << "Paquet: ID = " + to_string (Sn) + "==>\n";
Datagrama * timeoutEvent = new Datagrama();
Sn++;
Sn = fmod(Sn, Ssize * 2);
simtime_t transmissionDelay = channel->
    getTransmissionFinishTime();
if ( transmissionDelay <= simTime() ){
    send(paquet, "out");
} else {
    sendDelayed(paquet, transmissionDelay - simTime(),
        "out");
}
packetsSended++;
EV << "Total de paquets enviats: " + to_string (
    packetsSended) + "\n";
if ( !timeoutEvent->isScheduled() ) {
    Datagrama *paquetLength = new Datagrama();
    paquetLength->setByteLength(packetLength);
    transmissionTime = channel->calculateDuration(
        paquetLength);
```

```
                paquetLength->setByteLength((int)par("ackLength"))
                    ;
                transmissionTimeAck = channel->calculateDuration(
                    paquetLength);
                timeout = (Ssize * (transmissionTime) + channel->
                    getDelay()) +
                (Ssize * (transmissionTimeAck) + channel->getDelay
                    ());
                scheduleAt(simTime()+timeout, timeoutEvent);
            }
            timersbufferEmisor.push_back(timeoutEvent);
        }
```

Pot semblar una mica extrany que els temporitzadors siguin objectes construïts amb la classe *Datagrama* però és així com funciona l'omnet, la classe *Datagrama* és una classe que he creat jo que exten de *cPacket* que inclou un *integer* utilitzat per assignar els números de seqüència; així doncs, els temporitzadors són com a "autotrames"que s'envia el mòdul a ell mateix passat l'interval de temps que es defineix a la funció *scheduleAt*.

2. A partir d'aquí explicarem els processos de *handleMessage()*: Es recorr l'array de temporitzadors *timersbufferEmisor* per veure si n'hi ha algun expirat.

3. Amb l'index de la posició del temporitzador que ha expirat, generam una trama duplicada i li assignam el número de seqüència que es troba a la posició d'aquest índex. Després reiniciam el temporitzador expirat per a que torni a estar actiu.

4. S'obté el número de seqüència de l'ACK rebut i el seu tipus (ACK o NACK); s'elimina l'ACK per alliberar memòria; i es recorr en bucle l'*array finestraTramesEmisor* per veure si l'ACK rebut confirma algun dels seus números de seqüència.

5. No es fa res perquè ja hem eliminat abans l'ACK / NACK.

6. S'atura el temporitzador de la trama que confirma el que que s'ha rebut, i es comprova el seu tipus si ACK o NACK.

7. Es genera i s'envia una trama duplicada amb el número de seqüència corresponent i s'ocupa la primera fila de *finestraTramesEmisor* en aquella posició amb un "2" per indicar que s'ha rebut un Nack per a aquella trama. Després es reinicia el seu temporitzador de nou i es guarda a l'*array* de temporitzadors.

8. Es recorr la primera fila de *finestraTramesEmisor* per veure si hi ha NACKs rebuts i ACKs rebuts posteriors a aquests NACKs.

9. S'aturen tots els temporitzadors i es marquen com a rebuts (és a dir, "1" a la primera línia de *finestraTramesEmisor*) del primer NACK rebut fins el primer ACK posterior al NACK.

10. Aquí s'explicaran els punts *10*, *11* i *12*: Es recorr la primera línia de *finestraTramesEmisor* en bucle amb la condició de que siguin ACK rebuts, o sigui, "1"s

consecutius. A cada iteració s'elimina la primera posició de l'*array* i del *timersbufferEmisor*; s'incrementa *effectiveTransmissions*; es genera una nova trama amb un nou número de seqüència en mòdul $2S_{size}$; s'afegeix una nova posició al final de *finestraTramesEmisor* i s'omple a la primera fila amb un "0" i a la segona amb el número generat; s'envia la trama generada i es genera, s'activa, i s'afegeix al final de *timersbufferEmisor* un temporitzador per a aquesta nova trama.

11. Una vegada ja no es troben més uns consecutius a *finestraTramesEmisor* es finalitza el procés de *handleMessage()*

12. Finalment quan s'acaba la simulació es calcula l'eficiència i es recull el valor al mètode *finish* exactament de la mateixa manera que en el protocol *GBN*.

El codi d'aquests processos el trobarem als annexos ja que és molt extens.

Finalment el darrer mòdul que s'ha creat és el receptor del protocol SR que segueix el següent flux:

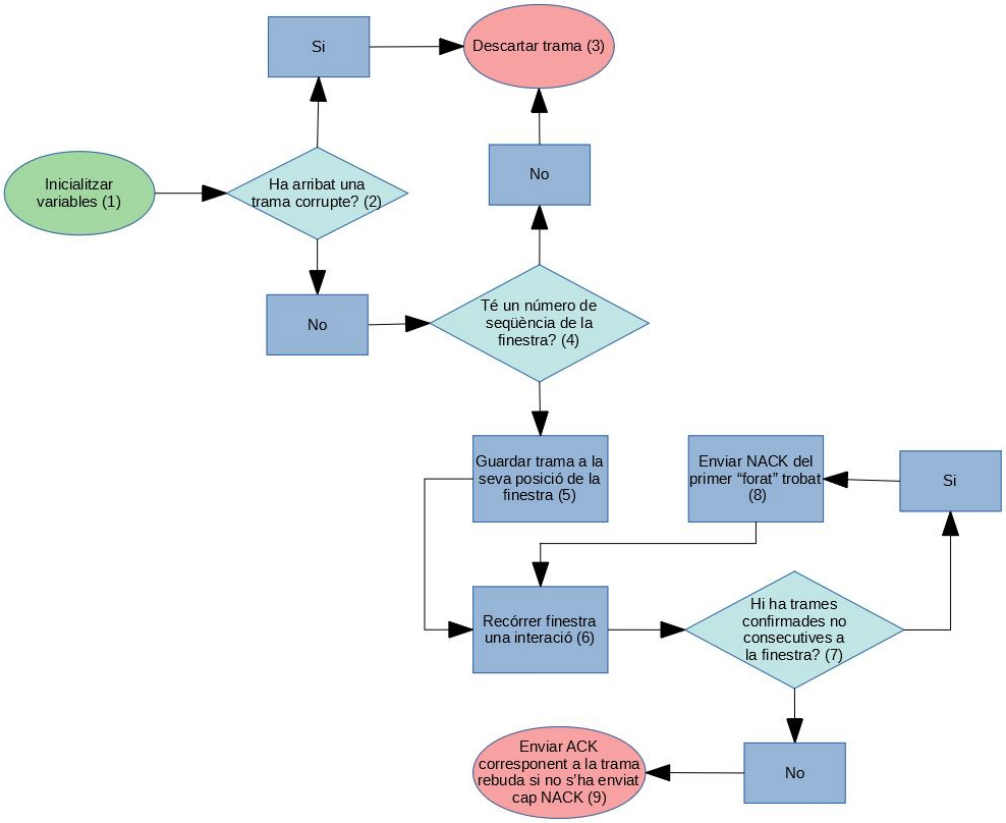

Figura 4.7: Flux receptor SR

1. *initialize():* Recull els valors de *windowLengthBits* i *ackLength*; inicialitza la longitud de la finestra lliscant del receptor S_{size} i crea i omple una array bidimensional *finestraTramesReceptor* exactament igual i amb els mateixos números de seqüència que a l'*initialize()* de l'emisor.

2. **if** (paquetRebut−>hasBitError())

3. No es fa res.

4. Es recorr la segona fila de *finestraTramesReceptor* per mirar i el número de seqüència de la trama rebuda es troba dins la finestra lliscant del receptor.

5. Es mira la primera fila de *finestraTramesReceptor* a la posició on es troba el número de seqüència de la trama rebuda. Si ja s'havia rebut, és a dir si hi ha un "1" no es fa res; si nó, és a dir s'hi no s'havia rebut encara prèviament "0" o s'havia enviat un NACK per aquesta trama "2" es canvia a l'"1" indicant que ja s'ha rebut.

6. En aquest punt explicam els processos de *6, 7, 8 i 9*: Es recorr la primera fila de *finestraTramesReceptor* per veure quina és la darrera trama que s'ha rebut. Després es recorr en bucle la finestra fins a aquell index. A cada iteració es llisca la finestra i es posa el següent número de seqüència nou (mòdul $2S_{\text{size}}$) al final si a la primera fila hi ha un "1" consecutiu, és a dir, si desde que ha començat el bucle no s'ha enviat cap NACK; sinó, s'atura de lliscar la finestra i s'envien tants NACKs com "0"s hi hagi fins el darrer "1" de la primera fila de *finestraTramesReceptor*. Si no hi ha "0"s entre els "1"s s'envia un ACK amb el número de seqüència del primer "0" de la finestra lliscant del receptor.

CAPÍTOL 5

RESULTATS

5.1 Tipus d'execució

Com s'ha vist al capítol 1, amb l'*OMNET++* podem executar les nostres simulacions de vàries maneres, cada una d'elles destinades a obtenir un tipus de resultat o un altre. L'execució animada a temps real pot ser extramadament útil per als alumnes que estudien protocols de xarxes de telecomunicacions, perquè així poden veure com funcionen realment els protocols d'una manera molt intuïtiva; però en el meu cas, lo realment útil és executar moltes simulacions autòmatiques en paral·lel per veure d'un cop d'ull les prestacions i mancances de cada protocol.

5.2 Resultats obtinguts dels simuladors implementats

Per a poder comparar els protocols he utilitzat la fórmula :

$$\text{efficiency} = \frac{\text{effectiveTransmissions} \cdot \text{transmissionTime}}{\text{transmissionsTimeTotal}} \tag{5.1}$$

Que s'ha explicat al capítol 4 on vèiem com es programa aquesta fórmula per obtenir el resultat al mètode *finish()* de cada arxiu *.cc*. El funcionament és que a *l'omnet.ini* de cada model tenim el paràmetre de BER com un array de dos-cents valors que va de 0 a 0.0002 incrementant-se a raó de $1 \cdot 10^{-6}$ a cada valor. Si l'*OMNET++* detecta que s'estan utilitzant paràmetres d'aquests tipus el que fa és preparar una simulació per a cada un dels valors del paràmetre, per tant, obtindrem un resultat d'eficiència per a cada una de les simulacions, així podrem dibuixar una gràfica amb aquests dos-cents valors d'eficiència respecte el seu valor de BER; per tant, ja tenim dos-centes simulacions una per a cada valor. Ara bé, haurem de configurar el temps d'execució que volem per cada simulació perquè siguin finites i puguem extreure'n els resultats.

Aquesta configuració es troba a la barra d'eines *Al botó de Run src / Run Configurations...*

Com podem veure a la següent figura, hem seleccionat que l'IDE executi el model de SR (al selector de *Config name*), que utilitzi els quatre nuclis del meu processador en paral·lel i que simuli un funcionament de la xarxa de 15000 segons.

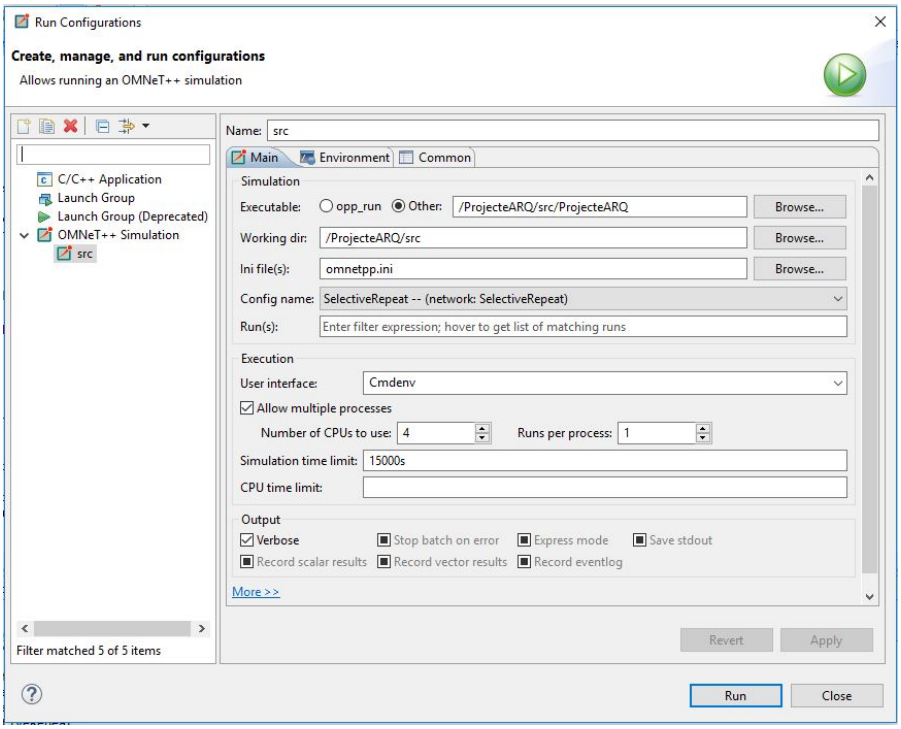

Figura 5.1: Configuració de l'execució

Finalment, a l'arxiu *omnet.ini* definim els valors de tots els paràmetres que volem que la execució utilitzi, com podem veure a continuació, per els resultats que es mostraran seguidament he utilitzat aquests:

```
[General]

[Config StopAndWait]

network = StopAndWait
*.packetLengthParam = 1500B
*.ackLengthParam = 100B
*.berParam = ${ber=0..2e-4 step 1e-6 }
*.datarateParam = 5 Mbps
*.delayParam = 18 ms


[Config SelectiveRepeat]
network = SelectiveRepeat
#############################################################################################
# In case of Selective repeat, the window length is (2^m)/2  --> if m = 4, then window length is 8. #
#############################################################################################

*.windowLengthBitsParam = 4
*.packetLengthParam = 1500B
*.ackLengthParam = 100B
*.berParam = ${ber=0..2e-4 step 1e-6 }
*.datarateParam = 5 Mbps
*.delayParam = 18 ms


[Config GoBackN]
network = GoBackN
############################################################################################
# In case of Go back n, the window length is (2^m) - 1  --> if m = 3, then window length is 7. #
############################################################################################

*.windowLengthBitsParam = 3
*.packetLengthParam = 1500B
*.ackLengthParam = 100B
*.berParam = ${ber=0..2e-4 step 1e-6 }
*.datarateParam = 5 Mbps
*.delayParam = 18 ms
```

Figura 5.2: Valors de l'omnet.ini

La funció o significat de cada paràmetre està explicat al capítol 4.

Ara sí, ja es pot executar el conjunt de simulacions per a cada model. Una vegada completada cada simulació, s'obtenen tots els valors de la fórmula *efficiencyRun* al seu respectiu arxiu *.anf, per exemple, en el cas de simular per el model SR l'arxiu resultat seria SelectiveRepeat.anf.* I a partir d'aquest arxiu *Browse Data / Seleccionar tots els valor efficiency / click dret i Plot* podrem dibuixar la gràfica. En el nostre cas, executant els tres models un per un obtenim les tres següents figures:

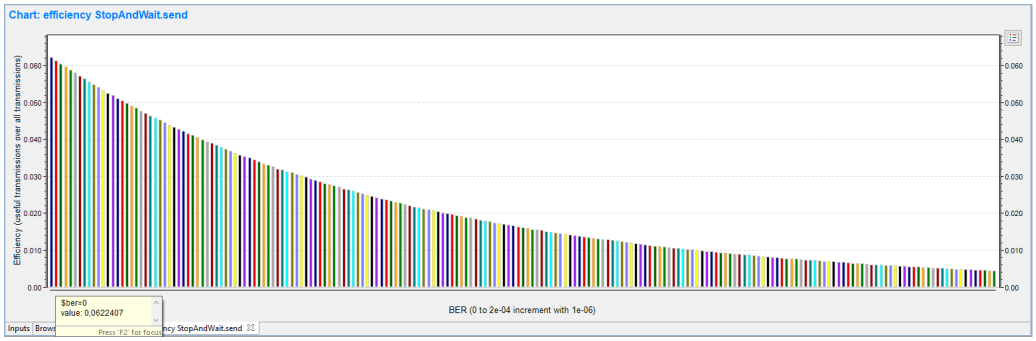

Figura 5.3: Eficiència del protocol SW

Com es pot apreciar tenim una columna de colors per diferenciar-les per a cada un dels diferents valors del BER.

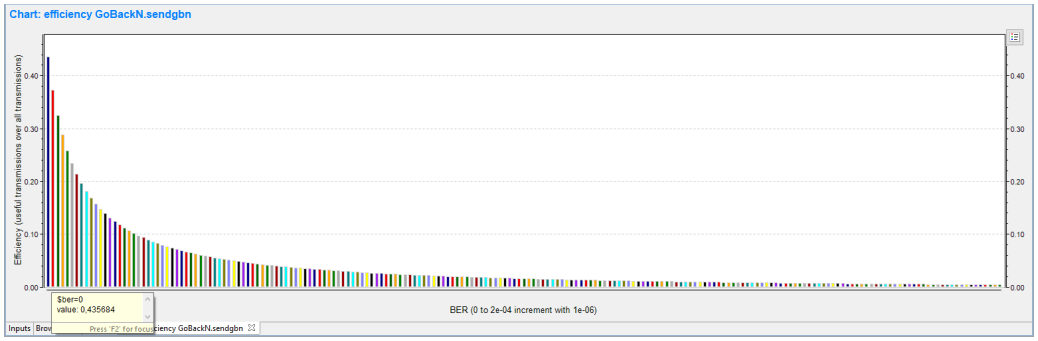

Figura 5.4: Eficiència del protocol GBN

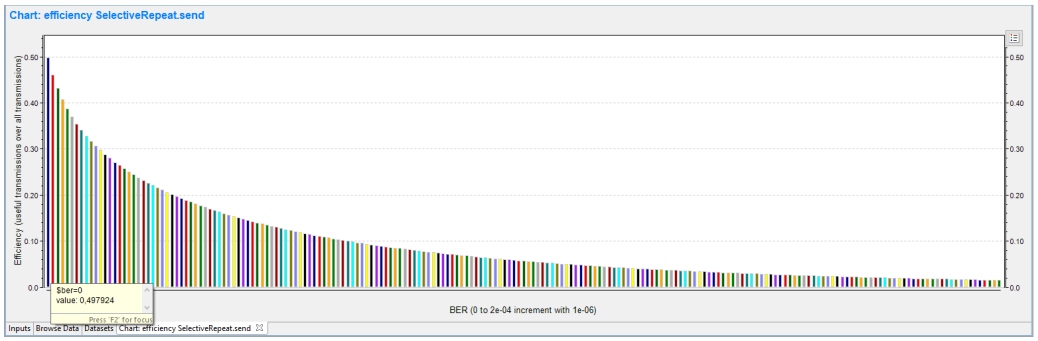

Figura 5.5: Eficiència del protocol SR

En el cas de SW tenim la descendència més lineal d'eficiència a mesura que va pujant el BER, no obstant això, encara que els nivells de BER siguin alts, el protocol en sí és ineficient; s'aconsegueix una eficiència que va de 0,0622407 (6,22%) en BER = 0 fins a 0,00440463 (0,44%) en BER = 0,0002.

Per al GBN podem dir que és el protocol que més sofreix el BER ja que és on descendeix més la eficiència a mesura que aquest va augmentant, això és degut a la ineficient forma de respondre a les trames corruptes o perdudes ja que es torna a retransmetre tota la finestra sencera cada vegada quan alomillor ja s'havien rebut bé la majoria de trames de la finestra, la qual cosa fa que les retransmissions malgastin molta amplada de banda. Els nivells obtinguts són: una eficiència de 0,435684 (43,56%) per a un BER = 0 fins a 0,00413814 (0,41%) per al BER = 0,0002. Per a BERs alts és moltissim més eficient que SW en canvi, per a BERs notablement baixos és inclús un poc menys eficient que el SW.

Finalment del protocol SR extraiem la conclusió de que és molt eficient tant per a nivells de BER baixos com robust en termes d'eficiència quan els nivells de BER pujen, ja que només es retransmeten únciament les trames perdudes o corruptes, no tota la finestra. En el context que hem descrit amb els paràmetres de *omnet.ini* obtenim una eficiència de 0,497924 (49,79%) per al BER = 0, un poc més elevada que al protocol GBN fins a una eficiència del 0,0149771 (1,49 %). L'inconvenient del protocol SR és que necessita una finestra lliscant tant per l'emisor com per el receptor i això pot afegir

complexitat al sistema.

Òbviament si canviam els valors dels paràmetres aquestes eficiències augmentaran o disminuiran, per exemple, si disminuïm el retard de propagació, les eficiències augmenten ja que s'està més temps transmetent informació efectiva respecte al total del temps.

En definitiva tenim un protocol molt simple i fàcil d'implementar però molt ineficient sigui quin sigui el nivell de BER com és el SW; un protocol molt eficient i que funciona molt bé per a canals amb un BER elevat que només necessit una finestra lliscant a l'emisor; i un protocol més complexe però molt eficient sigui quin sigui el nivell de BER del canal.

CAPÍTOL 6

CONCLUSIONS

En aquest document s'ha explicat com funciona l'eina de modelització de sistemes d'esdeveniments discrets *OMNET++* i quines passes s'han de seguir a l'hora d'utilitzar-lo per crear un simulador d'una xarxa de telecomunicacions que es comunica amb un protocol utilitzat al model TCP/IP, amb l'objectiu d'estudiar i analitzar el protocol en qüestió, que ha estat l'ARQ.

Tot això amb la finalitat de decidir si aquesta eina és apta i interessant per utilitzar-la en pràctiques de laboratori del grau d'enginyeria telemàtica.

6.1 El que s'ha après

La capacitat d'adaptabilitat que té aquesta eina a l'hora de crear un simulador d'esdevendiments discrets, d'analitzar-lo i extreure'n resultats és gairebé il·limitada, normalment això sol tenir la conseqüència de que és complexe i té una corba d'aprenentatge dura. Ara bé, gràcies a l'estructura modular que obliga a seguir la eina fa que crear models de xarxes per grosses i complexes que siguin, encara que tenguin molts de nodes o *mòduls* sigui bastant senzill ja que només es tracta de dibuixar la xarxa i connectar els *mòduls*; la única etapa realment difícil és la de la programació dels *mòduls* en *C++* la qual no la recoman per a què hi dediquin temps i esforços els alumnes ja que es requereix tant un cert nivell d'habilitat a l'hora de programar en el llenguatje *C++* com en mirar i estudiar-se un poc la Application Programming Interface (API) [3] de les classes més importants del *framework OMNET++* ja que són el que s'utilitza per programar els *mòduls*. [4]

6.2 El que falta per aprendre

Com s'ha vist al llarg d'aquest document, la virtud de la modularització de l'eina realment no s'ha explotat en aquest treball ja que ens hem centrat a crear la xarxa més senzilla possible per veure si es podien programar i simular els protocols ARQ i quant

difícil era. Una vegada demostrat que es pot fer, el que queda per fer és explotar l'eina utilitzant directament els mòduls de les llibreries existents com la de *INET Framework* per així poder crear grans xarxes ràpidament i fàcilment i poder elaborar simuladors que serien d'allò més interessants i educatius per als alumnes del nostre grau sense haver de perdre temps programant mòduls que facin el que volem, ja que aquests mòduls, si són de protocols utilitzats al model *TCP/IP* entre altre coses, ja existeixen a les llibreries i segurament estan millor programats i optimitzats del què podríem fer nosaltres.

ANNEXOS

A.1 Codi

A.1.1 stopandwait.cc

```
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include "datagrama_m.h"

using namespace omnetpp;
using namespace std;

/**
 * Let us take a step back, and remove random delaying from the
    code.
 * We'll leave in, however, losing the packet with a small
    probability.
 * And, we'll we do something very common in telecommunication
    networks:
 * if the packet doesn't arrive within a certain period, we'll
    assume it
 * was lost and create another one. The timeout will be handled
    using
 * (what else?) a self-message.
 */
class Sender: public cSimpleModule {
private:
    int effectiveTransmissions;
    int packetLength;
```

```
    simtime_t transmissionTime;
    simtime_t transmissionTimeAck;
    simtime_t transmissionsTimeTotal;
    simtime_t timeout;  // timeout
    Datagrama *timeoutEvent;  // holds pointer to the timeout self
        -message
    Datagrama *msgEnviat;
    int seqNumber = 0;

public:
    Sender();
    virtual ~Sender();

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void finish() override;
};

Define_Module(Sender);

Sender::Sender() {
    timeoutEvent = nullptr;
}

Sender::~Sender() {
    cancelAndDelete(timeoutEvent);
}
void Sender::initialize() {
    cDatarateChannel * channel = check_and_cast<cDatarateChannel
        *>(
            gate("out")->getChannel());
    // Initialize variables.
    timeoutEvent = new Datagrama("timeoutEvent");
    effectiveTransmissions = 0;
    // Generate and send initial message.
    EV << "Enviat el paquet inicial. \n";
    msgEnviat = new Datagrama("Datagrama");
    msgEnviat->setLabelId(seqNumber);
    packetLength = par("packetLength");
    msgEnviat->setByteLength(packetLength);
    EV << "Paquet: ID = " + to_string(seqNumber) + "==>\n";
    transmissionTime = channel->calculateDuration(msgEnviat);
    msgEnviat->setByteLength((int) par("ackLength"));
    transmissionTimeAck = channel->calculateDuration(msgEnviat);
    send(msgEnviat, "out");
```

```
    timeout = transmissionTime + transmissionTimeAck + (2 *
        channel->getDelay())
            + 0.1
                    * (transmissionTime + transmissionTimeAck
                            + (2 * channel->getDelay()));
    scheduleAt(simTime() + timeout, timeoutEvent);
}

void Sender::handleMessage(cMessage *msg) {
    Datagrama* ack = dynamic_cast<Datagrama*>(msg);
    if (ack->hasBitError()) {
        bubble("Ack perdut.");  // making animation more
            informative...
        delete msg;
        if (msg == timeoutEvent) {
            cDatarateChannel * channel = check_and_cast<
                cDatarateChannel*>(
                    gate("out")->getChannel());
            timeout = transmissionTime + transmissionTimeAck
                    + (2 * channel->getDelay())
                    + 0.1
                            * (transmissionTime +
                                transmissionTimeAck
                                    + (2 * channel->getDelay()));
            timeoutEvent = new Datagrama("timeoutEvent");
            scheduleAt(simTime() + timeout, timeoutEvent);
        }
    } else {
        if (msg == timeoutEvent) {
            // If we receive the timeout event, that means the
                packet hasn't
            // arrived in time and we have to re-send it.
            EV
                    << "El temporitzador ha expirat, re-enviant
                        el paquet i restaurant el temporitzador.\
                        n";
            EV << "Paquet: ID = " + to_string(seqNumber) + "==>\n"
                ;
            msgEnviat = new Datagrama("Datagrama");
            msgEnviat->setLabelId(seqNumber);
            msgEnviat->setByteLength(packetLength);
            send(msgEnviat, "out");
            scheduleAt(simTime() + timeout, timeoutEvent);
        } else {
            // message arrived
            if (ack->getLabelId() == seqNumber + 1) {
                seqNumber++;
```

```
                // Save data to calculate efficiency
                effectiveTransmissions++;
                transmissionsTimeTotal = simTime();
            }
            // Acknowledgment received -- delete the received ack
                and cancel
            // the timeout event.
            cancelEvent(timeoutEvent);
            delete msg;
            msgEnviat = new Datagrama("Datagrama");
            msgEnviat->setLabelId(seqNumber);
            msgEnviat->setByteLength(packetLength);
            EV << "Paquet: ID = " + to_string(seqNumber) + "==>\n"
                ;
            send(msgEnviat, "out");
            scheduleAt(simTime() + timeout, timeoutEvent);
        }
    }
}

void Sender::finish() {
    double efficiencyRun = (effectiveTransmissions *
        transmissionTime.dbl())
            / transmissionsTimeTotal.dbl();
    recordScalar("efficiency", efficiencyRun);
}

/**
 * Sends back an acknowledgment -- or not.
 */
class Receiver: public cSimpleModule {
private:
    int seqNumber = 0;
    int ackLength;

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(Receiver);

void Receiver::initialize() {
    ackLength = par("ackLength");
}

void Receiver::handleMessage(cMessage *msg) {
```

```
    Datagrama* msgRebut = dynamic_cast<Datagrama*>(msg);
    if (msgRebut->hasBitError()) {
        bubble("Paquet perdut.");  // making animation more
            informative...
        delete msg;
    } else {
        int actualSeqNumber = msgRebut->getLabelId();
        Datagrama *ack = new Datagrama("Datagrama");
        if (actualSeqNumber == seqNumber) {
            EV
                    << "Paquet rebut no duplicat...Transmetent-
                        ho a la capa superior.\n";
            seqNumber++;
        } else {
            EV << "Paquet rebut duplicat...Eliminant-lo.\n";
        }
        delete msg;
        ack->setLabelId(seqNumber);
        ack->setByteLength(ackLength);
        EV << "<== Ack: ID = " + to_string(ack->getLabelId()) + "\
            n";
        send(ack, "out");
    }
}
```

A.1.2 gobackn.cc

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include <math.h>
#include "datagrama_m.h"
#include <vector>

using namespace omnetpp;
using namespace std;

class SenderGBN: public cSimpleModule {
private:
    simtime_t timeout; // timeout
    int effectiveTransmissions;
    int packetLength;
    simtime_t transmissionTime;
    simtime_t transmissionTimeAck;
    simtime_t transmissionsTimeTotal;
    Datagrama *timeoutEvent;  // holds pointer to the timeout self
```

```
        -message
    int m; // Number of bits to represent the sequence number of
        the sliding window
    int Ssize; // Size of the sliding window
    int Sf = 0; // Sf = First message sent of the sliding window (
        sequence number )
    int Sn = 0;      // Sn = Next message to be sent
    int packetsSended = 0;
    int packetsAcknowledged = 0;
    vector<int> finestraIds;

public:
    SenderGBN();
    virtual ~SenderGBN();

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void finish() override;
};

Define_Module(SenderGBN);

SenderGBN::SenderGBN() {
    timeoutEvent = nullptr;
}

SenderGBN::~SenderGBN() {
    cancelAndDelete(timeoutEvent);

}
void SenderGBN::initialize() {
    m = par("windowLengthBits");
    Ssize = pow(2, m) - 1;
    // Initialize variables.
    cDatarateChannel * channel = check_and_cast<cDatarateChannel
        *>(
            gate("out")->getChannel());
    timeoutEvent = new Datagrama("timeoutEvent");
    effectiveTransmissions = 0;
    packetLength = par("packetLength");
    for (int index = 0; index < Ssize; index++) {
        Datagrama *paquet = new Datagrama();
        paquet->setLabelId(Sn);
        paquet->setByteLength(packetLength);
        finestraIds.push_back(Sn);
        EV << "Paquet: ID = " + to_string(Sn) + "==>\n";
```

```
        //Ssize + 1 = 2^m
        Sn++;
        Sn = fmod(Sn, Ssize + 1);
        simtime_t transmissionDelay = channel->
            getTransmissionFinishTime();
        if (transmissionDelay <= simTime()) {
            send(paquet, "out");
        } else {
            sendDelayed(paquet, transmissionDelay - simTime(), "
                out");
        }
        packetsSended++;
        EV << "Total de paquets enviats: " + to_string(
            packetsSended) + "\n";
        if (!timeoutEvent->isScheduled()) {
            Datagrama *paquetLength = new Datagrama();
            paquetLength->setByteLength(packetLength);
            transmissionTime = channel->calculateDuration(
                paquetLength);
            paquetLength->setByteLength((int) par("ackLength"));
            transmissionTimeAck = channel->calculateDuration(
                paquetLength);
            timeout = (Ssize * (transmissionTime) + channel->
                getDelay())
                    + (Ssize * (transmissionTimeAck) + channel->
                        getDelay());
            scheduleAt(simTime() + timeout, timeoutEvent);
        }
    }
    //Print in console the status of the sliding window
    EV << "Estat de la finestra lliscant: ";
    for (int index = 0; index < Ssize; index++) {
        EV << to_string(finestraIds[index]) + " ";
    }
    EV << "\n";
}

void SenderGBN::handleMessage(cMessage *msg) {
    Datagrama* ack = dynamic_cast<Datagrama*>(msg);
//     if (ack->hasBitError()) {
//             bubble("Ack perdut.");  // making animation more
    informative...
//             delete msg;
//             if (msg == timeoutEvent) {
//               cDatarateChannel * channel = check_and_cast<
    cDatarateChannel*>(gate("out")->getChannel());
//               timeout = (Ssize * (transmissionTime) + channel->
```

```
    getDelay()) + (Ssize * (transmissionTimeAck) + channel->
    getDelay());
//                timeoutEvent = new Datagrama("timeoutEvent");
//                scheduleAt(simTime()+timeout, timeoutEvent);
//              }
//    }
//    else {
    cDatarateChannel * channel = check_and_cast<cDatarateChannel
        *>(
            gate("out")->getChannel());

    if (msg == timeoutEvent) {
        // If we receive the timeout event, that means the packet
           hasn't arrived in time and we have to re-send it.
        EV
                << "El temporitzador ha expirat, re-enviant els
                   paquets pendents de confirmacio de la
                   finestraIds llisacant i restaurant el
                   temporitzador.\n";
        // Delete all lost paquets and recreate it with the same
           ID instead of just resend them for c++ destructor
           purpose.
        for (int index = 0; index < Ssize; index++) {

            Datagrama *paquet = new Datagrama();
            paquet->setLabelId(finestraIds[index]);
            paquet->setByteLength(packetLength);

            EV << "Paquet: ID = " + to_string(finestraIds[index])
               + "==>\n";
            simtime_t transmissionDelay = channel->
               getTransmissionFinishTime();
            if (transmissionDelay <= simTime()) {
                send(paquet, "out");
            } else {
                sendDelayed(paquet, transmissionDelay - simTime(),
                    "out");
            }
            packetsSended++;
            EV
                    << "Total de paquets enviats: " + to_string(
                       packetsSended)
                            + "\n";
            if (!timeoutEvent->isScheduled()) {
                scheduleAt(simTime() + timeout, timeoutEvent);
            }
        }
```

```
        //Print in console the status of the sliding window
        EV << "Estat␣de␣la␣finestra␣lliscant:␣";
        for (int index = 0; index < Ssize; index++) {
            EV << to_string(finestraIds[index]) + "␣";
        }
        EV << "\n";
    } else {

        // Sf is the first message sent of the sender window that
            is waiting for acknowledgment.

        Sf = ack->getLabelId();

        delete msg;

        //Check if the acknowledgment belongs to an outstanding
            packet.

        int indexPacketConfirmed = -1;

        int i = 0;
        while (i < Ssize && indexPacketConfirmed == -1) {
            if (finestraIds[i] == Sf) {
                indexPacketConfirmed = i;
            }
            i++;
        }
        //Slide the window and send packets if there is a correct
            acknowledgment

        if (indexPacketConfirmed != -1) {
            EV << "Ack␣rebut:␣ID␣=␣␣" + to_string(Sf) + "\n";
            cancelEvent(timeoutEvent);
            for (int positionsToSlide = 0;
                    positionsToSlide <= indexPacketConfirmed;
                    positionsToSlide++) {

                //Print in console the status of the sliding
                    window
                EV << "Estat␣de␣la␣finestra␣lliscant:␣";
                for (int index = 0; index < Ssize; index++) {
                    EV << to_string(finestraIds[index]) + "␣";
                }
                EV << "\n";

                //Delete first message of the window.
```

```
        finestraIds.erase(finestraIds.begin());

        //Insert the new message comming of the layer of
            above at the end of the window and slide it.

        Datagrama *paquet = new Datagrama();
        paquet->setLabelId(Sn);
        paquet->setByteLength(packetLength);
        finestraIds.push_back(Sn);

        //Send the new message

        effectiveTransmissions++;
        transmissionsTimeTotal = simTime();
        EV << "Paquet: ID = " + to_string(Sn) + "==>\n";
        simtime_t transmissionDelay =
                channel->getTransmissionFinishTime();
        if (transmissionDelay <= simTime()) {
            send(paquet, "out");
        } else {
            sendDelayed(paquet, transmissionDelay -
                simTime(), "out");
        }
        //Ssize + 1 = 2^m
        Sn++;
        Sn = fmod(Sn, Ssize + 1);

        packetsAcknowledged++;
        EV
                  << "Total de paquets enviats i
                      confirmats: "
                          + to_string(packetsAcknowledged)
                              + "\n";

        packetsSended++;
        EV
                  << "Total de paquets enviats: "
                          + to_string(packetsSended) + "\n
                              ";

        // Reset the timer

        if (!timeoutEvent->isScheduled()) {
            scheduleAt(simTime() + timeout, timeoutEvent);
        }
    }
```

```
        }
    }
//    }
}

void SenderGBN::finish() {
    double efficiencyRun = (effectiveTransmissions *
        transmissionTime.dbl())
            / transmissionsTimeTotal.dbl();
    recordScalar("efficiency", efficiencyRun);
}

/**
 * Sends back an acknowledgment -- or not.
 */
class ReceiverGBN: public cSimpleModule {
private:
    int m = 3;
    int Ssize = pow(2.0, m) - 1;
    int Rn = 0;
    int ackLength;

public:
    virtual ~ReceiverGBN();

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(ReceiverGBN);

void ReceiverGBN::initialize() {
    ackLength = par("ackLength");
}

ReceiverGBN::~ReceiverGBN() {
    cOwnedObject *Del = NULL;
    int OwnedSize = this->defaultListSize();
    for (int i = 0; i < OwnedSize; i++) {
        Del = this->defaultListGet(0);
        this->drop(Del);
        delete Del;
    }
}

//In GBN an ackNo is cumulative;
```

```
//it defines the sequence number of the last packet confirmed,
    confirming that all
//previous packets have been received safe and sound.
void ReceiverGBN::handleMessage(cMessage *msg) {
    //Message lost.
    Datagrama* datagramaRebut = dynamic_cast<Datagrama*>(msg);
    if (datagramaRebut->hasBitError()) {
        bubble("Paquet perdut.");  // making animation more
            informative...
        Datagrama* ackRebut = dynamic_cast<Datagrama*>(msg);
        EV << "Paquet perdut, ID: " + to_string(ackRebut->
            getLabelId()) + " \n";
        delete msg;
    }
    //Message received.
    else {
        int actualSeqNumber = datagramaRebut->getLabelId();
        Datagrama *ack = new Datagrama("Datagrama");
        //Message expected.
        if (actualSeqNumber == Rn) {

            //Send the ack

            //If this were a real ARQ implementation the message
                would be delivered to the layer of above (network
                layer) instead of delete the message.
            delete msg;
            EV
                    << "Paquet rebut no duplicat...Transmetent-
                        ho a la capa superior.\n";
            ack->setLabelId(Rn);
            ack->setByteLength(ackLength);
            EV << "<== Ack: ID = " + to_string(ack->getLabelId())
                + "\n";
            send(ack, "out");
            Rn++;
            Rn = fmod((Rn), Ssize + 1);
        } else {
            //Message unexpected.
            EV << "Paquet rebut inesperat...Descartant-lo.\n";
            delete msg;
        }
    }
}
```

A.1.3 selectiverepeat.cc

```
#include <iostream>
#include <stdio.h>
#include <string.h>
#include <omnetpp.h>
#include <math.h>
#include "datagrama_m.h"
#include "acknowledgement_m.h"
#include <vector>

using namespace omnetpp;
using namespace std;

class SenderSR: public cSimpleModule {
private:
    simtime_t timeout; // timeout
    int effectiveTransmissions;
    int packetLength;
    simtime_t transmissionTime;
    simtime_t transmissionTimeAck;
    simtime_t transmissionsTimeTotal;
    int m; // Number of bits to represent the sequence number of
        the sliding window
    int Ssize; // Size of the sliding window
    int Sn = 0;      // Sn = Next message to be sent
    vector<Datagrama*> timersbufferEmisor; // Buffer of the timers
         for each message
    //of the sliding window
    // table with two rows:
    // In the first, there are 0 if the message is not
        acknowledged yet, 1 if the
    //message have been acknowledged, 2 if there are a Nack
        message sent for it.
    // In the second row, there are the sequence numbers of the
        messages that are
    //in the sliding window.
    vector<vector<int> > finestraTramesEmisor;
    int packetsSended = 0;
    int packetsAcknowledged = 0;
public:
    SenderSR();
    virtual ~SenderSR();

protected:
    virtual void initialize() override;
    virtual void handleMessage(cMessage *msg) override;
    virtual void finish() override;
};
```

```
Define_Module(SenderSR);

SenderSR::SenderSR() {

}

SenderSR::~SenderSR() {
    cOwnedObject *Del = NULL;
    int OwnedSize = this->defaultListSize();
    for (int i = 0; i < OwnedSize; i++) {
        Del = this->defaultListGet(0);
        this->drop(Del);
        delete Del;
    }
}
void SenderSR::initialize() {
    // Initialize variables.
    m = par("windowLengthBits");
    Ssize = pow(2, m) / 2;
    cDatarateChannel * channel = check_and_cast<cDatarateChannel
        *>(
            gate("out")->getChannel());
    effectiveTransmissions = 0;
    packetLength = par("packetLength");
    vector<int> row0;
    vector<int> row1;
    finestraTramesEmisor.push_back(row0);
    finestraTramesEmisor.push_back(row1);
    for (int index = 0; index < Ssize; index++) {
        Datagrama * paquet = new Datagrama();
        paquet->setLabelId(Sn);
        paquet->setByteLength(packetLength);
        finestraTramesEmisor.at(0).push_back(0);
        finestraTramesEmisor.at(1).push_back(Sn);
        EV << "Paquet: ID = " + to_string(Sn) + "==>\n";
        Datagrama * timeoutEvent = new Datagrama();
        Sn++;
        Sn = fmod(Sn, Ssize * 2);
        simtime_t transmissionDelay = channel->
            getTransmissionFinishTime();
        if (transmissionDelay <= simTime()) {
            send(paquet, "out");
        } else {
            sendDelayed(paquet, transmissionDelay - simTime(), "
                out");
        }
```

```
        packetsSended++;
        EV << "Total␣de␣paquets␣enviats:␣" + to_string(
            packetsSended) + "\n";
        if (!timeoutEvent->isScheduled()) {
            Datagrama *paquetLength = new Datagrama();
            paquetLength->setByteLength(packetLength);
            transmissionTime = channel->calculateDuration(
                paquetLength);
            paquetLength->setByteLength((int) par("ackLength"));
            transmissionTimeAck = channel->calculateDuration(
                paquetLength);
            timeout = (Ssize * (transmissionTime) + channel->
                getDelay())
                    + (Ssize * (transmissionTimeAck) + channel->
                        getDelay());
            scheduleAt(simTime() + timeout, timeoutEvent);
        }
        timersbufferEmisor.push_back(timeoutEvent);
    }
}

void SenderSR::handleMessage(cMessage *msg) {
    Acknowledgement* ack = dynamic_cast<Acknowledgement*>(msg);
//     if (ack->hasBitError()) {
//             bubble("Ack perdut."); // making animation more
    informative...
//             delete msg;
//             int i = 0;
//             int indexTimerExpired = -1;
//             while ( i < Ssize && indexTimerExpired == -1 ) {
//                 if ( msg == timersbufferEmisor.at(i) ) {
//                     indexTimerExpired = i;
//                 }
//                 i++;
//             }
//             if ( indexTimerExpired != -1 ) {
//                 cDatarateChannel * channel = check_and_cast<
    cDatarateChannel*>
//                 (gate("out")->getChannel());
//                 timeout = (Ssize * (transmissionTime) + channel
    ->getDelay()) +
//                 (Ssize * (transmissionTimeAck) + channel->
    getDelay());
//                 Datagrama *timeoutEvent = new Datagrama("
    timeoutEvent");
//                 timersbufferEmisor.at(indexTimerExpired) =
    timeoutEvent;
```

```
//                  scheduleAt(simTime()+timeout, timeoutEvent);
//              }
//    }
//    else {
    //Search into the buffer of timers if the handleMessage have
        been executed
    //because of the expiration of some timer
    cDatarateChannel * channel = check_and_cast<cDatarateChannel
        *>(
            gate("out")->getChannel());
    int i = 0;
    int indexTimerExpired = -1;
    while (i < Ssize && indexTimerExpired == -1) {
        if (msg == timersbufferEmisor.at(i)) {
            indexTimerExpired = i;
        }
        i++;
    }

    if (indexTimerExpired != -1) {
        // If we receive the timeout event, that means the packet
            hasn't
        //arrived in time and we have to re-send it.
        EV
                << "Un dels temporitzadors ha expirat, re-
                    enviant el paquet corresponent"
                        "a aquell temporitzador.\n";
        EV
                << "Paquet: ID = "
                        + to_string(
                                finestraTramesEmisor.at(1).at(
                                        indexTimerExpired)) + "
                                            ==>\n";

        //Print in console the status of the sliding window
        EV << "Estat de la finestra lliscant de l'emisor:\n";
        for (int index = 0; index < Ssize; index++) {
            EV << to_string(finestraTramesEmisor.at(0).at(index))
                + " ";
        }
        EV << "\n";
        for (int index = 0; index < Ssize; index++) {
            EV << to_string(finestraTramesEmisor.at(1).at(index))
                + " ";
        }
        EV << "\n";
        Datagrama *paquet = new Datagrama();
```

```
        paquet->setLabelId(finestraTramesEmisor.at(1).at(
            indexTimerExpired));
        paquet->setByteLength(packetLength);
        packetsSended++;
        EV << "Total de paquets enviats: " + to_string(
            packetsSended) + "\n";
        simtime_t transmissionDelay = channel->
            getTransmissionFinishTime();
        if (transmissionDelay <= simTime()) {
            send(paquet, "out");
        } else {
            sendDelayed(paquet, transmissionDelay - simTime(), "
                out");
        }
        scheduleAt(simTime() + timeout,
                timersbufferEmisor.at(indexTimerExpired));
    } else {
        // Ack received.
        int ackNo = ack->getLabelId();
        bool ackType = ack->getAckType();
        delete msg;

        //Check if the acknowledgment belongs to an outstanding
            packet.

        int indexPacketConfirmed = -1;
        int index = 0;

        while (index < Ssize && indexPacketConfirmed == -1) {
            if (finestraTramesEmisor.at(1).at(index) == ackNo) {
                indexPacketConfirmed = index;
            }
            index++;
        }

        //Slide the window and send packets if there is a correct
            acknowledgment

        if (indexPacketConfirmed != -1) {
            cancelAndDelete(timersbufferEmisor.at(
                indexPacketConfirmed));
            if (ackType == false) {

                // A NACK have been received, send the
                    corresponding message and
                //don't stop his timer
```

```
            EV << "Paquet: ID = " + to_string(ackNo) + "==>\n"
                ;

            //Print in console the status of the sliding
                window
            EV << "Estat de la finestra lliscant de l'emisor:\
                n";
            for (int index = 0; index < Ssize; index++) {
                EV << to_string(finestraTramesEmisor.at(0).at(
                    index)) + " ";
            }
            EV << "\n";
            for (int index = 0; index < Ssize; index++) {
                EV << to_string(finestraTramesEmisor.at(1).at(
                    index)) + " ";
            }
            EV << "\n";

            Datagrama *paquet = new Datagrama();
            paquet->setLabelId(
                    finestraTramesEmisor.at(1).at(
                        indexPacketConfirmed));
            paquet->setByteLength(packetLength);
            simtime_t transmissionDelay =
                    channel->getTransmissionFinishTime();
            if (transmissionDelay <= simTime()) {
                send(paquet, "out");
            } else {
                sendDelayed(paquet, transmissionDelay -
                    simTime(), "out");
            }
            finestraTramesEmisor.at(0).at(indexPacketConfirmed
                ) = 2;
            packetsSended++;
            EV
                      << "Total de paquets enviats: "
                              + to_string(packetsSended) + "\n
                                  ";
            Datagrama *timeoutEvent = new Datagrama();
            if (!timeoutEvent->isScheduled()) {
                scheduleAt(simTime() + timeout, timeoutEvent);
            }
            timersbufferEmisor.at(indexPacketConfirmed) =
                timeoutEvent;
        } else {
            if (finestraTramesEmisor.at(0).at(
                indexPacketConfirmed) == 0
```

```
            || finestraTramesEmisor.at(0).at(
                indexPacketConfirmed)
                    == 2) {
        finestraTramesEmisor.at(0).at(
            indexPacketConfirmed) = 1;
    }

    int indexNack = -1;
    int indexReceived = -1;
    int i = 0;

    //Point out the messages have been received since
        the Nack message
    //have been received
    while (indexReceived == -1 && i < Ssize) {
        if (finestraTramesEmisor.at(0).at(i) == 2
                && indexNack == -1) {
            indexNack = i;
        } else if (indexNack != -1
                && finestraTramesEmisor.at(0).at(i) ==
                    1) {
            indexReceived = i;
        }
        i++;
    }
    //Stop his timers
    if (indexReceived != -1) {
        for (int index = indexNack; index <
            indexReceived;
                index++) {
            cancelAndDelete(timersbufferEmisor.at(
                index));
            finestraTramesEmisor.at(0).at(index) = 1;
        }
    }
    EV << "Ack␣rebut:␣ID␣=␣␣" + to_string(ackNo) + "\n
        ";

    i = 0;

    while (i < Ssize && finestraTramesEmisor.at(0).at
        (0) == 1) {
        //Slide the buffer of packets and send the
            packets of the layer of above
        //Print in console the status of the sliding
            window
        EV << "Estat␣de␣la␣finestra␣lliscant␣de␣l'
```

```
    emisor:\n";
for (int index = 0; index < Ssize; index++) {
    EV
            << to_string(
                    finestraTramesEmisor.at
                        (0).at(index))
                    + "␣";
}
EV << "\n";
for (int index = 0; index < Ssize; index++) {
    EV
            << to_string(
                    finestraTramesEmisor.at
                        (1).at(index))
                    + "␣";
}
EV << "\n";
timersbufferEmisor.erase(timersbufferEmisor.
    begin());
finestraTramesEmisor.at(0).erase(
        finestraTramesEmisor.at(0).begin());
finestraTramesEmisor.at(1).erase(
        finestraTramesEmisor.at(1).begin());
Datagrama *paquet = new Datagrama();
Datagrama *timeoutEvent = new Datagrama();
paquet->setLabelId(Sn);
paquet->setByteLength(packetLength);
finestraTramesEmisor.at(0).push_back(0);
finestraTramesEmisor.at(1).push_back(Sn);

EV << "Paquet:␣ID␣=␣" + to_string(Sn) + "==>\n
    ";

effectiveTransmissions++;
transmissionsTimeTotal = simTime();
simtime_t transmissionDelay =
        channel->getTransmissionFinishTime();
if (transmissionDelay <= simTime()) {
    send(paquet, "out");
} else {
    sendDelayed(paquet, transmissionDelay -
        simTime(),
            "out");
}
Sn++;
Sn = fmod(Sn, Ssize * 2);
```

```
                    packetsAcknowledged++;
                    EV
                            << "Total de paquets enviats i
                                confirmats: "
                                    + to_string(
                                        packetsAcknowledged) + "\
                                        n";

                    packetsSended++;
                    EV
                            << "Total de paquets enviats: "
                                    + to_string(packetsSended) +
                                        "\n";
                    if (!timeoutEvent->isScheduled()) {
                        scheduleAt(simTime() + timeout,
                            timeoutEvent);
                    }
                    timersbufferEmisor.push_back(timeoutEvent);
                    i++;
                }
            }
        }
    }
//     }
}

void SenderSR::finish() {
    double efficiencyRun = (effectiveTransmissions *
        transmissionTime.dbl())
            / transmissionsTimeTotal.dbl();
    recordScalar("efficiency", efficiencyRun);
}

/**
 * Sends back an acknowledgment -- or not.
 */
class ReceiverSR: public cSimpleModule {
private:
    int m;
    int Ssize;
    int Rn = -1;
    vector<vector<int> > finestraTramesReceptor;
    int ackLength;
public:
    virtual ~ReceiverSR();
protected:
    virtual void initialize() override;
```

```
    virtual void handleMessage(cMessage *msg) override;
};

Define_Module(ReceiverSR);

ReceiverSR::~ReceiverSR() {
    cOwnedObject *Del = NULL;
    int OwnedSize = this->defaultListSize();
    for (int i = 0; i < OwnedSize; i++) {
        Del = this->defaultListGet(0);
        this->drop(Del);
        delete Del;
    }
}
//The semantics of acknowledgment
//is different in SR. In SR, an ackNo defines the sequence number
    of one single packet
//that is received safe and sound; there is no feedback for any
    other.
void ReceiverSR::initialize() {
    m = par("windowLengthBits");
    Ssize = pow(2, m) / 2;
    ackLength = par("ackLength");
    vector<int> row0;
    vector<int> row1;
    finestraTramesReceptor.push_back(row0);
    finestraTramesReceptor.push_back(row1);
    // Initialize variables.
    for (int index = 0; index < Ssize; index++) {
        finestraTramesReceptor.at(0).push_back(0);
        finestraTramesReceptor.at(1).push_back(index);
    }
}

void ReceiverSR::handleMessage(cMessage *msg) {
    cDatarateChannel * channel = check_and_cast<cDatarateChannel
        *>(
            gate("out")->getChannel());
    Datagrama* paquetRebut = dynamic_cast<Datagrama*>(msg);
    int actualSeqNumber = paquetRebut->getLabelId();

    bool outstanding = false;
    int indexPacketConfirmed = 0;

    for (int index = 0; index < Ssize; index++) {
        if (finestraTramesReceptor.at(1).at(index) ==
            actualSeqNumber) {
```

```
            outstanding = true;
            indexPacketConfirmed = index;
            break;
        }
    }

    if (paquetRebut->hasBitError()) {
        bubble("Paquet perdut.");  // making animation more
            informative...
        EV << "Paquet perdut, ID: " + to_string(actualSeqNumber) +
             " \n";
    } else if (outstanding == true) {
        if (finestraTramesReceptor.at(0).at(indexPacketConfirmed)
            == 0
                || finestraTramesReceptor.at(0).at(
                    indexPacketConfirmed) == 2) {
            //Save the received message and point out the receiver
                sliding window
            EV
                    << "Paquet rebut no duplicat. (ID: "
                            + to_string(
                                    finestraTramesReceptor.at(1)
                                        .at(
                                            indexPacketConfirmed
                                                )) + ")\n";
            finestraTramesReceptor.at(0).at(indexPacketConfirmed)
                = 1;
            int lastIndexWindow = -1;
            int lastReceivedPacket = -1;
            EV << "Estat de la finestra lliscant del receptor: ";
            EV << "\n";
            for (int index = 0; index < Ssize; index++) {
                EV << to_string(finestraTramesReceptor.at(0).at(
                    index)) + " ";
            }
            EV << "\n";
            for (int index = 0; index < Ssize; index++) {
                EV << to_string(finestraTramesReceptor.at(1).at(
                    index)) + " ";
            }
            EV << "\n";
            for (int i = 0; i < Ssize; i++) {
                if (finestraTramesReceptor.at(0).at(i) == 1) {
                    lastReceivedPacket = i;
                }
            }
```

```
if (lastReceivedPacket != -1) {
    int index = 0;
    bool nackSended = false;
    while (index <= lastReceivedPacket) {
        //Message received successfully, slide the
            window.
        if (finestraTramesReceptor.at(0).at(0) == 1
                && nackSended == false) {
            Rn = finestraTramesReceptor.at(1).at(0);
            //Slide the window
            finestraTramesReceptor.at(0).erase(
                    finestraTramesReceptor.at(0).begin
                        ());
            finestraTramesReceptor.at(1).erase(
                    finestraTramesReceptor.at(1).begin
                        ());
            lastIndexWindow = finestraTramesReceptor.
                at(1).back()
                    + 1;
            lastIndexWindow = fmod(lastIndexWindow,
                Ssize * 2);
            finestraTramesReceptor.at(1).push_back(
                lastIndexWindow);
            finestraTramesReceptor.at(0).push_back(0);
            lastReceivedPacket--;
        }
        //Hole confirmed, send Nack.
        else if (finestraTramesReceptor.at(0).at(index
            ) == 0) {
            Acknowledgement *Nack = new
                Acknowledgement();
            Nack->setLabelId(
                    finestraTramesReceptor.at(1).at(
                        index));
            Nack->setAckType(false);
            Nack->setByteLength(ackLength);

            //Print in console the status of the
                sliding window
            EV
                    << "<==␣Nack:␣ID␣=␣"
                            + to_string(Nack->
                                getLabelId())
                            + "\n";

            nackSended = true;
            simtime_t transmissionDelay =
```

```
                                channel->getTransmissionFinishTime
                                    ();
                            if (transmissionDelay <= simTime()) {
                                send(Nack, "out");
                            } else {
                                sendDelayed(Nack, transmissionDelay -
                                    simTime(),
                                        "out");
                            }
                            finestraTramesReceptor.at(0).at(index) =
                                2;
                            index++;
                        } else {
                            index++;
                        }
                    }
                    if (!nackSended) {
                        Acknowledgement *ack = new Acknowledgement();
                        ack->setLabelId(Rn);
                        ack->setAckType(true);
                        ack->setByteLength(ackLength);
                        EV
                                << "<==␣Ack:␣ID␣=␣" + to_string(ack
                                    ->getLabelId())
                                        + "\n";

                        simtime_t transmissionDelay =
                                channel->getTransmissionFinishTime();
                        if (transmissionDelay <= simTime()) {
                            send(ack, "out");
                        } else {
                            sendDelayed(ack, transmissionDelay -
                                simTime(), "out");
                        }
                    }
                }
            }
        }
    }
}
```

BIBLIOGRAFIA

[1] B. A. Forouzan, *TCP/IP protocol suite / Behrouz A. Forouzan.*, 4th ed. McGraw-Hill, 2010. (document), 4.1, 4.2

[2] A. S. TANENBAUM, *Redes de computadoras*, 4th ed. Pearson Educación, México, 2003. 4.1

[3] "Api de l'omnet++." [Online]. Available: https://www.omnetpp.org/doc/omnetpp/api/index.html 6.1

[4] "Tictoc tutorial." [Online]. Available: https://docs.omnetpp.org/tutorials/tictoc/6.1