Apéndice A

Índice

A. Códigos Utilizados

A continuación se muestran los códigos utilizados para las simulaciones. El generador de números aleatorios es el llamado generador de Parisi Rapuano[1].Los archivos de las redes utilizadas no se muestran, son archivos en los que en la primera linea aparece el número de nodos y links que hay en la red y a continuación se nombran, en cada línea, las uniones existentes. Por ejemplo:

1000 2994 (Una red de 1000 nodos 2994 links) 1 17 (El nodo 1 está unido al nodo 17)

A.1. Ciclo de histéresis

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <malloc.h>

//////////////////////

//#define EMPIEZAEN1
#define nCaracteres 256
#define tiempoPrevio     3000
#define tiempoEjecucion 500
#define creadosCadaMax 100
//#define ER
#define SF
//#define creadosCada 1
```

[1]Implementado de la misma forma que en se muestra en los apuntes de la asignatura *Física Computacional* proporcionados por Dr. Alfonso Tarancón Lafita.

```
//////////////////////


//Numero de nodos de nuestra red
int nNodos, nLinks;
int maximoGrado;
int creadosCada, creadosCadaSiguiente;
float p, saltoP = 0.02;
int tiempo, prorroga=0, iFichero;
int probabilidadesMax = 17;
#ifdef ER
int tiempos[16]=            {500, 1000, 1250, 2000 ,3000 ,3500 ,4000 ,
   → 4500, 5000, 5500 ,30000, 60000, 140000,290000, 380000,
   → 480000}; int iTiempo;
float probabilidades[16] = {0.07, 0.09,0.11, 0.13, 0.15, 0.17, 0.19,
   → 0.17, 0.15, 0.13, 0.11, 0.09, 0.07, 0.05, 0.03, 0.01}; int
   → iProbabilidad;
#endif // ER

#ifdef SF
int tiempos[17]=            {500, 1000, 1250, 2000 ,5000 ,7000 ,10000
   → , 11000, 12000, 13000 ,14000, 18000, 21000, 23000, 25000,
   → 27000, 29000}; int iTiempo;
float probabilidades[17] = {0.035, 0.037,0.039, 0.041, 0.043, 0.045,
   → 0.047, 0.045, 0.043, 0.041, 0.039, 0.037, 0.035, 0.030, 0.02,
   → 0.01, 0.005}; int iProbabilidad;
#endif // SF
float h=0.7, epsilon = 0.05;
int nPaquetesActivos = 0, nPaquetesActivosAntes = 0;
#include "Biblioteca.h"
int main(){
    fclose(probs);
    #ifdef ER
    char nombreRed[nCaracteres]="Red_ER_2.txt";
    #endif
    #ifdef SF
    char nombreRed[nCaracteres]="Red_SF_1.txt";
    #endif // SF
    //Calculamos el grado maximo de la red
    printf("Vamos␣a␣calcular␣el␣grado␣maximo\n");
    MaximoGrado(nombreRed);
    printf("EL␣nombre␣es:␣%s\n", nombreRed);
    int *conCuantosCada=(int*) malloc(nNodos*sizeof(int));
    int *conQuienesCada=(int*) malloc(nNodos*maximoGrado*sizeof(int))
       → ;
```

```
CreaMatrices(nombreRed, conCuantosCada, conQuienesCada);
int *distancias= (int*) malloc(nNodos*nNodos*sizeof(int));
DistanciasMinimas(conCuantosCada, conQuienesCada, distancias);

//
   → ////////////////////////////////////////////////////////////////////////
   →

//Arrays necesarios para paquetes (nos dan info de cada paquete)
int *dondeEsta;
int *dondeVa;
int *quienVaDespues;

int *dondeEstaLuego;
int *dondeVaLuego;
int *quienVaDespuesLuego;
char bufferP[50], bufferH[50];

FILE *ficheroP;

//Arrays que nos dan informacion sobre cada nodo

int *cuantosEnMiCola = (int*) malloc(nNodos*sizeof(int));
int *cuantosEnMiColaLuego = (int*) malloc(nNodos*sizeof(int));
int *primero = (int*) malloc(nNodos*sizeof(int));
int *ultimo = (int*) malloc(nNodos*sizeof(int));
int *primeroLuego = (int*) malloc(nNodos*sizeof(int));
int *ultimoLuego = (int*) malloc(nNodos*sizeof(int));
int *paquetes; int ventana = 20, iVentana=0, nVentanas=
   → tiempoEjecucion/ventana;float rho, diferenciaPaquetes;

int i, iNodoAux, jNodoAux, iPaqueteAux;

int iCreado;
    #ifdef ER
    sprintf(bufferP, "ER_HisteresisH%.2f.txt", h);
    #endif // ER
    #ifdef SF
    sprintf(bufferP, "SF_2PaquetesTiempoH%.2fP%.3f.txt", h,p);
    #endif // SF
    ficheroP=fopen(bufferP, "w");
    p=0.035;
    creadosCada=(int)(p*nNodos);
    printf("Creados cada vale: %d\n",creadosCada );
    InicializaNodos(cuantosEnMiCola, primero, ultimo);
    dondeEsta = (int*) malloc((creadosCada)*sizeof(int));
```

```
        dondeVa = (int*) malloc((creadosCada)*sizeof(int));
        quienVaDespues = (int*) malloc((creadosCada)*sizeof(int));
        paquetes = (int*) malloc(nVentanas*sizeof(int));
        tiempo = 0;
    for(iProbabilidad = 0;iProbabilidad < probabilidadesMax; iTiempo
       → ++, iProbabilidad++){
        p=probabilidades[iProbabilidad];
        #ifdef ER
        sprintf(bufferP, "ER_HisteresisP%d.txt", iProbabilidad);
        #endif // ER
        #ifdef SF
        sprintf(bufferP, "SF_HisteresisP%d.txt", iProbabilidad);
        #endif // SF
        ficheroP=fopen(bufferP, "w");
        printf("La probabilidad pasa a ser %f e iProbabilidad vale %d
           → \n", probabilidades[iProbabilidad], iProbabilidad);
//          creadosCada=(int)(p*nNodos);
        printf("Creados cada vale: %d y hay %d paquetes\n",
           → creadosCada, nPaquetesActivos );
//          Histeresis (int* dondeEsta, int* dondeVa, int*
   → quienVaDespues, int nPaquetesActivos, int creadosCada);
        for (; tiempo < tiempos[iTiempo]; tiempo++){
            for(iCreado = 0; iCreado < creadosCada; iCreado++){
                CreaPaquete(dondeEsta, dondeVa, cuantosEnMiCola,
                   → ultimo, quienVaDespues, primero);
            }
            if(tiempo==tiempos[iTiempo-1]){
//              creadosCada = 1;
            nPaquetesActivosAntes=nPaquetesActivos;
            dondeEstaLuego = (int*) malloc((nPaquetesActivos)*sizeof(
               → int));
            dondeVaLuego = (int*) malloc((nPaquetesActivos)*sizeof(
               → int));
            quienVaDespuesLuego = (int*) malloc((nPaquetesActivos)*
               → sizeof(int));
            MuevePrimeros(dondeEsta, dondeVa, conCuantosCada,
               → conQuienesCada, distancias, primero,
               → cuantosEnMiCola, quienVaDespues, ultimo,
               → dondeEstaLuego, quienVaDespuesLuego, dondeVaLuego,
               → primeroLuego, ultimoLuego, cuantosEnMiColaLuego);//
               → La info pasa a los LUEGO
            free(dondeEsta); free(dondeVa); free(quienVaDespues);//
               → Liberamos los NOLUEGO
            //redefinimos los que son NOLUEGO
            if(tiempo == tiempos[iTiempo]-1){
                creadosCada = (int) (probabilidades[iProbabilidad+1]*
```

```
                    → nNodos);
                dondeEsta = (int*) malloc((nPaquetesActivos+
                    → creadosCada)*sizeof(int));
                dondeVa = (int*) malloc((nPaquetesActivos+creadosCada
                    → )*sizeof(int));
                quienVaDespues = (int*) malloc((nPaquetesActivos+
                    → creadosCada)*sizeof(int));
            }
            else{
                dondeEsta = (int*) malloc((nPaquetesActivos+
                    → creadosCada)*sizeof(int));
                dondeVa = (int*) malloc((nPaquetesActivos+creadosCada
                    → )*sizeof(int));
                quienVaDespues = (int*) malloc((nPaquetesActivos+
                    → creadosCada)*sizeof(int));
            }

            CambiaArrays(dondeEstaLuego, dondeEsta, dondeVaLuego,
                → dondeVa, quienVaDespuesLuego, quienVaDespues,
                → primero, ultimo, cuantosEnMiCola, primeroLuego,
                → ultimoLuego, cuantosEnMiColaLuego);//La informacion
                →  pasa de los Luego a los NOLUEGO
            free(dondeEstaLuego);
            free(dondeVaLuego);
            free(quienVaDespuesLuego);
            if (tiempo%5==0){
//                  printf("Escribo en el archivo para tiempo %d\n",
   → tiempo);
                fprintf(ficheroP, "%d\t %d\n", tiempo,
                    → nPaquetesActivos);
            }
        }
        nPaquetesActivosAntes=0;
        fclose(ficheroP);
        free(paquetes);
    }
//    fclose(ficheroH);
//}

    //Cosas de nodos (no se liberan cada vez)
    free(primero);
    free(ultimo);
    free(primeroLuego);
    free(ultimoLuego);
    free(cuantosEnMiCola);
    free(cuantosEnMiColaLuego);
```

```
    free(conCuantosCada);
    free(conQuienesCada);
    free(distancias);
    //Cosas de paquetes, se van liberando
    free(dondeEsta);
    free(dondeVa);
    free(quienVaDespues);
    printf("\nFin\n");
    return 0;
}
```

A.1.1. Librería utilizada

```
float generaRand(void);
void MaximoGrado (char*); //Le damos el fichero de la matriz y
    → devuelve el maximo grado de la red
void CreaMatrices ( char*, int*, int *);//Crea matriz vecinos y
    → vector numero de vecinos
void ReescribeRed(char*);//Pasa la matriz a un txt desde el .net
void ReiniciaMenosUno(int etiquetas[]); //Nos pone un vector de
    → nNodos componentes todo a -1
void DistanciasMinimas (int conCuantosCada[], int conQuienesCada[],
    → int distancias[]);//Calcula la matriz de distancias
void ImprimeMatrizEnFichero (char nombre[],int lado, int matriz[]);//
    → Imprime una matriz en un fichero
void InicializaNodos (int *cuantosEnMiCola, int *primero, int *ultimo
    → );
void CreaPaquete (int *dondeEsta, int *dondeVa, int* cuantosEnMiCola,
    →  int* ultimo, int *quienVaDespues, int *primero);
void CongestionAware(int* llegamos, int* dondeMovemos, int iNodo, int
    →  * dondeVa, int *conCuantosCada,int *conQuienesCada, int *
    → distancias, int *primero, int *cuantosEnMiCola);

float generaRand(void){
    int i, INI, FACTOR, SUM, SEMILLA = 12345;
    static int veces = 0;
    static unsigned char ind_ran, ig1, ig2, ig3;
    static unsigned int rueda[256], ir1;
    if (veces == 0){
        srand(SEMILLA);
        INI = SEMILLA;
        FACTOR = 67397;
        SUM = 7364893;
        for(i = 0; i < 256; i++){
            rueda[i] = (rand() << 16) + rand();
        }
```

```
        ind_ran = 0;
        veces = 1;
    }
    #define NormRANu (2.3283063671E-10F)
    float r;
    ig1 = ind_ran - 24;
    ig2 = ind_ran - 55;
    ig3 = ind_ran - 61;
    rueda[ind_ran] = rueda[ig1] + rueda[ig2];
    ir1 = (rueda[ind_ran]^rueda[ig3]);
    ind_ran++;
    r = ir1*NormRANu;
    return r;
}

void MaximoGrado (char nombreFichero[]){
    FILE *archivo;
    archivo = fopen (nombreFichero, "r");
    int iNodo, iLectura; int actual1, actual2;

    /*Creamos la rutina que lee el fichero y crea el array*/
    //Leemos la primera linea que nos da informacion sobre la red
    fscanf(archivo, "%d %d\n", &nNodos, &nLinks);
    //Inicializamos a 0 el array, de forma que luego iremos sumando
       → uno
    int conCuantosCada[nNodos];
    for(iNodo = 0; iNodo < nNodos; iNodo ++){
        conCuantosCada[iNodo] = 0;
    }
    //Vamos leyendo los links y guardando los datos
    /*
    Este if lo que hace es lo siguiente: nos resta 1 a todos los
       → links. Por que? El codigo
    esta hecho para funcionar en redes cuyo primer indice es el 0, si
       →  es el 1, tenemos que
    reescribir la red, asi evitamos duplicar todo el codigo
    */
    #ifdef EMPIEZAEN1
        char nombreFicheroC[nCaracteres];
        int nLetra;
        for(nLetra=0;nLetra<nCaracteres;nLetra++){
            nombreFicheroC[nLetra]= nombreFichero[nLetra];
        }
        sprintf(nombreFichero, "c_%s", nombreFicheroC);
        fclose(archivo);
        FILE *archivoViejo;
```

```
        archivo = fopen(nombreFichero, "w");
        archivoViejo = fopen(nombreFicheroC, "r");
        fscanf(archivoViejo, "%d %d\n",&actual1, &actual2 );
        fprintf(archivo, "%d %d\n",actual1, actual2 );
        for(iLectura=0;iLectura<nLinks; iLectura++){
            fscanf(archivoViejo, "%d %d\n",&actual1, &actual2 );
            actual1=actual1-1;
            actual2=actual2-1;
            fprintf(archivo, "%d %d\n",actual1, actual2 );
        }
        fclose(archivo);
        fclose(archivoViejo);
        archivo=fopen(nombreFichero, "r");
        fscanf(archivo, "%d %d\n",&actual1, &actual2 );


    #endif // EMPIEZAEN1
    for(iLectura=0; iLectura < nLinks; iLectura++){
        fscanf(archivo, "%d %d\n",&actual1, &actual2 );
        conCuantosCada[actual1]+=1;
        conCuantosCada[actual2]+=1;
    }
    for(iNodo=0;iNodo < nNodos; iNodo++){
        if(conCuantosCada[iNodo]>maximoGrado){
            maximoGrado = conCuantosCada[iNodo];
        }
    }

    printf("La red tiene %d nodos y %d links y el maximo grado es: %d
       → \n", nNodos, nLlinks, maximoGrado);
    fclose(archivo);

}

void CreaMatrices ( char *nombreFichero,int *conCuantosCada, int
   → conQuienesCada[]){
    FILE *archivo;
    archivo = fopen (nombreFichero, "r");
    int iNodo,iLectura,jNodo,actual1, actual2;

    /*Creamos la rutina que lee el fichero y crea el array*/
    //Leemos la primera linea que nos da informacion sobre la red
    fscanf(archivo, "%d %d\n", &nNodos, &nLinks);
    //Inicializamos a 0 el array, de forma que luego iremos sumando
       → uno
```

```
    for(iNodo = 0; iNodo < nNodos; iNodo ++){
        conCuantosCada[iNodo] = 0;
        for(jNodo = 0; jNodo < maximoGrado; jNodo++){
            conQuienesCada[iNodo*maximoGrado+jNodo]=-1;
        }
    }
    //Vamos leyendo los links y guardando los datos
    for(iLectura=0; iLectura < nLinks; iLectura++){
        fscanf(archivo, "%d %d\n",&actual1, &actual2 );
        conQuienesCada[actual1*maximoGrado+conCuantosCada[actual1]] =
            → actual2;
        conQuienesCada[actual2*maximoGrado+conCuantosCada[actual2]] =
            → actual1;
        conCuantosCada[actual1]+=1;
        conCuantosCada[actual2]+=1;
    }
//     for(iNodo=0;iNodo<nNodos;iNodo++){
//         for(jNodo=0;jNodo<maximoGrado;jNodo++){
//             printf("%d    ", conQuienesCada[iNodo*maximoGrado+jNodo
    → ]);
//         }
//         printf("\n");
//     }

    fclose(archivo);

}

void ReescribeRed(char* nombreRed){
    FILE*redDada;
    FILE*redEscrita;

    redDada = fopen(nombreRed, "r");
    redEscrita = fopen("ReescribeRed.txt", "w");
    int actual1, actual2, iLectura;
    for(iLectura = 0; iLectura<nLinks+1; iLectura++){
        fscanf(redDada, "%d %d\n", &actual1, &actual2);
        fprintf(redEscrita, "%d %d\n", actual1, actual2);
    }
    fclose(redDada);
    fclose(redEscrita);


}

void ReiniciaMenosUno (int etiquetas[]){
```

```
    int iNodo;
    for(iNodo = 0 ; iNodo < nNodos ; iNodo++){
        etiquetas[iNodo]=-1;
    }
}

void DistanciasMinimas (int conCuantosCada[], int conQuienesCada[],
    → int distancias[]){

    int nodoCalculamos, iNodo, iVecino, cuantosVecinosTiene, iColumna
        → ;
    int etiquetas[nNodos];
    int visitar[nNodos];
    int hastaDondeVisitar, g;

    for (nodoCalculamos = 0 ; nodoCalculamos < nNodos ;
        → nodoCalculamos++){
        ReiniciaMenosUno(etiquetas);//Todos vuelven a no estar
            → marcados
        ReiniciaMenosUno(visitar);//No sabemos cuales tenemos que
            → visitar
        etiquetas[nodoCalculamos]=0;//La distancia a si mismo es nula
        visitar[0]=nodoCalculamos;//El primero que se visita es en el
            →  que estamos
        hastaDondeVisitar=1;//Para visitar por lo menos el primero
            → ([0])
        for(g = 0 ; g < hastaDondeVisitar ; g++){
            //Vemos cuales son los vecinos del nodo que observamos
            cuantosVecinosTiene=conCuantosCada[visitar[g]];
            for(iVecino = 0 ; iVecino < cuantosVecinosTiene ; iVecino
                → ++){
                if(etiquetas[conQuienesCada[visitar[g]*maximoGrado+
                    → iVecino]]==-1){
                    visitar[hastaDondeVisitar]=conQuienesCada[visitar
                        → [g]*maximoGrado+iVecino];
                    hastaDondeVisitar += 1;
                    etiquetas[conQuienesCada[visitar[g]*maximoGrado+
                        → iVecino]]=etiquetas[visitar[g]]+1;
                }
            }

        }

    //Ya sabemos la distancia de cualquier nodo a nodoCalculamos,
        → ahora vamos a ir rellenando la matriz de distancias
        for(iColumna = 0; iColumna < nNodos; iColumna++){
```

```
                distancias[nodoCalculamos*nNodos+iColumna]=etiquetas[
                    → iColumna];
            }
        }
}

void ImprimeMatrizEnFichero( char* nombreFichero,int lado, int matriz
    → []){
    #ifdef EMPIEZAEN1


    #else
        FILE* fichero;
    char nombre2[nCaracteres];
    sprintf(nombre2, "Matriz_%s.txt", nombreFichero);
    fichero = fopen(nombre2, "w");
    int iFila, iColumna;
    for(iColumna = 0; iColumna < lado; iColumna++){
        for(iFila = 0; iFila < lado; iFila++){
            fprintf(fichero, "%d␣", matriz[iFila*lado+iColumna]);
        }
        fprintf(fichero, "\n");
    }
    fclose(fichero);
    #endif

}

void CreaPaquete (int *dondeEsta, int *dondeVa, int* cuantosEnMiCola,
    →  int* ultimo, int *quienVaDespues, int *primero){
    dondeEsta[nPaquetesActivos] = (int) nNodos*generaRand();
    do{
    dondeVa[nPaquetesActivos] = (int) nNodos*generaRand();
    }while(dondeVa[nPaquetesActivos]==dondeEsta[nPaquetesActivos]);
    quienVaDespues[nPaquetesActivos] = -1;
    cuantosEnMiCola[dondeEsta[nPaquetesActivos]] += 1;
    if(cuantosEnMiCola[dondeEsta[nPaquetesActivos]]!= 1){
        quienVaDespues[ultimo[dondeEsta[nPaquetesActivos]]] =
            → nPaquetesActivos;
    }
    ultimo[dondeEsta[nPaquetesActivos]] = nPaquetesActivos;
    if(cuantosEnMiCola[dondeEsta[nPaquetesActivos]] == 1){
        primero[dondeEsta[nPaquetesActivos]] = nPaquetesActivos;
    }
    nPaquetesActivos++;
}
```

```
void InicializaNodos (int *cuantosEnMiCola, int *primero, int *ultimo
    → ){
     int i;
     for(i = 0; i < nNodos; i++){
          cuantosEnMiCola[i]=0;
          primero[i] = -1;
          ultimo[i] = -1;
     }
}

void CambiaArrays (int *dondeEsta, int *dondeEstaNuevo, int *dondeVa,
    →  int *dondeVaNuevo, int *quienVaDespues, int *
    → quienVaDespuesNuevo, int *primero, int * ultimo, int *
    → cuantosEnMiCola, int *primeroLuego, int *ultimoLuego, int *
    → cuantosEnMiColaLuego){
     int iPaquete, cuantosVoy, iNodo;
     int *cambios = (int*) malloc(2*nPaquetesActivosAntes*sizeof(int))
        → ;
     int aux;
     for(iPaquete = 0, cuantosVoy = 0; iPaquete <
        → nPaquetesActivosAntes; iPaquete++){
          cambios[iPaquete] = iPaquete;
          cambios[nPaquetesActivosAntes + iPaquete] = -1;
          if(dondeVa[iPaquete] != dondeEsta[iPaquete]){
               cambios[nPaquetesActivosAntes+iPaquete] = cuantosVoy;
               cuantosVoy++;
          }
     }
     for(iPaquete = 0, cuantosVoy = 0; iPaquete <
        → nPaquetesActivosAntes; iPaquete++){
          if(dondeEsta[iPaquete] != dondeVa[iPaquete]){
               dondeEstaNuevo[cuantosVoy] = dondeEsta[iPaquete];
               dondeVaNuevo[cuantosVoy] = dondeVa[iPaquete];
               quienVaDespuesNuevo[cuantosVoy]=-1;
               cuantosVoy++;
          }
          if(quienVaDespues[iPaquete] != -1){
               quienVaDespuesNuevo[cuantosVoy-1] = cambios[
                   → nPaquetesActivosAntes+quienVaDespues[iPaquete]];
          }
     }
     for(iNodo = 0; iNodo < nNodos; iNodo++){
          primero[iNodo] = -1;
          ultimo[iNodo] = -1;
          cuantosEnMiCola[iNodo] = cuantosEnMiColaLuego[iNodo];
```

```
        if(cuantosEnMiCola[iNodo]!=0){
            if(primeroLuego[iNodo]==-1){
                    printf("::::::::::::El nodo %d tiene gente en cola
                       → y el primero es -1\n", iNodo);
            }
            primero[iNodo] = cambios[nPaquetesActivosAntes+
               → primeroLuego[iNodo]];
            ultimo[iNodo] = cambios[nPaquetesActivosAntes+ultimoLuego
               → [iNodo]];
        }
    }
    free(cambios);
};

void MuevePrimeros(int * dondeEsta, int * dondeVa, int *
   → conCuantosCada,int *conQuienesCada, int *distancias, int *
   → primero, int *cuantosEnMiCola, int *quienVaDespues, int* ultimo
   → , int *dondeEstaLuego, int *quienVaDespuesLuego, int *
   → dondeVaLuego, int *primeroLuego, int *ultimoLuego, int *
   → cuantosEnMiColaLuego){

    int iNodo, iVecino, quienMuevo, cuantosBuenos = 0, cuantosMalos =
       →  0, llegamos = 0, dondeMovemos;
    int distanciaVecinos[maximoGrado], vecinosBuenos[maximoGrado],
       → vecinosMalos[maximoGrado];
    int iPaquete;
    for(iPaquete=0; iPaquete < nPaquetesActivos; iPaquete++ ){
        dondeVaLuego[iPaquete] = dondeVa[iPaquete];
        dondeEstaLuego[iPaquete] = dondeEsta[iPaquete];
        quienVaDespuesLuego[iPaquete] = quienVaDespues[iPaquete];
    }
    for(iNodo = 0; iNodo < nNodos; iNodo++){
        cuantosEnMiColaLuego[iNodo] = cuantosEnMiCola[iNodo];
        primeroLuego[iNodo] = primero[iNodo];
        ultimoLuego[iNodo] = ultimo[iNodo];
    }
    for(iNodo = 0; iNodo < nNodos; iNodo++, quienMuevo = 0,
       → cuantosBuenos = 0, cuantosMalos = 0, llegamos = 0){
        if(primero[iNodo] != -1){
            quienMuevo = primero[iNodo];
            cuantosEnMiColaLuego[iNodo]--;
            if(cuantosEnMiColaLuego[iNodo]==0){
                ultimoLuego[iNodo] = -1;
                primeroLuego[iNodo] = -1;
            }
            if(cuantosEnMiColaLuego[iNodo]>0){
```

```
                primeroLuego[iNodo] = quienVaDespuesLuego[quienMuevo
                    → ];
            }
            CongestionAware( &llegamos, &dondeMovemos, iNodo, dondeVa
                → , conCuantosCada, conQuienesCada, distancias,
                → primero, cuantosEnMiCola);
            if(llegamos==1){
                quienVaDespuesLuego[quienMuevo] = -1;
                nPaquetesActivos--;
                dondeEstaLuego[quienMuevo]=dondeVa[quienMuevo];
            }
            if((llegamos == 0)){
                dondeEstaLuego[quienMuevo] = dondeMovemos;

            }
            if(llegamos == 0){
                if(cuantosEnMiColaLuego[dondeEstaLuego[quienMuevo]]
                    → == 0){

                    primeroLuego[dondeEstaLuego[quienMuevo]] =
                        → quienMuevo;
                    ultimoLuego[dondeEstaLuego[quienMuevo]] =
                        → quienMuevo;
                    quienVaDespuesLuego[quienMuevo] = -1;
                    cuantosEnMiColaLuego[dondeEstaLuego[quienMuevo
                        → ]]++;
                }else{
                    quienVaDespuesLuego[ultimoLuego[dondeEstaLuego[
                        → quienMuevo]]] = quienMuevo;
                    ultimoLuego[dondeEstaLuego[quienMuevo]] =
                        → quienMuevo;
                    cuantosEnMiColaLuego[dondeEstaLuego[quienMuevo
                        → ]]++;
                }
            }
        }
    }
}

void CongestionAware(int* llegamos, int* dondeMovemos, int iNodo, int
    →  * dondeVa, int *conCuantosCada,int *conQuienesCada, int *
    → distancias, int *primero, int *cuantosEnMiCola){
        int iVecino, mejorVecino, cuantosMejores=0, vecinoElegido;
        float *distanciaReal = (float*) malloc(conCuantosCada[iNodo]*
            → sizeof(float));
        int *mejoresVecinos = (int*) malloc(conCuantosCada[iNodo]*
```

```
        → sizeof(float));
    float minimaDistancia;
    for(iVecino=0; iVecino< conCuantosCada[iNodo]; iVecino++){
        distanciaReal[iVecino]=h*distancias[conQuienesCada[iNodo*
            → maximoGrado+iVecino]*nNodos+dondeVa[primero[iNodo
            → ]]]+(1.0-h)*cuantosEnMiCola[conQuienesCada[iNodo*
            → maximoGrado+iVecino]];
        if(iVecino==0){
            minimaDistancia=distanciaReal[iVecino];
        }
        if(conQuienesCada[iNodo*maximoGrado+iVecino]==dondeVa[
            → primero[iNodo]]){
            *llegamos = 1;
            mejorVecino=dondeVa[primero[iNodo]];
            distanciaReal[iVecino] = 0;
            minimaDistancia = 0;
            *dondeMovemos = mejorVecino;
        }
        if(distanciaReal[iVecino]<minimaDistancia){
            minimaDistancia=distanciaReal[iVecino];
        }
    }
    if(*llegamos==0){
        for(iVecino = 0, cuantosMejores = 0; iVecino <
            → conCuantosCada[iNodo]; iVecino++){
            if((distanciaReal[iVecino]>minimaDistancia-epsilon)
                → &&(distanciaReal[iVecino]<minimaDistancia+
                → epsilon)){
                mejoresVecinos[cuantosMejores] = iVecino;
                cuantosMejores++;
            }
        }
        int aleatorio= (int) (cuantosMejores*generaRand());
        vecinoElegido =  mejoresVecinos[aleatorio];
        mejorVecino= conQuienesCada[iNodo*maximoGrado+
            → vecinoElegido];
        *dondeMovemos=mejorVecino;
    }
    free(distanciaReal);
    free(mejoresVecinos);

}
```

A.2. Congestion Aware

```
#include <stdio.h>
#include <math.h>
```

```
#include <time.h>
#include <stdlib.h>
#include <malloc.h>

//////////////////////

//#define EMPIEZAEN1
#define nCaracteres 256
#define tiempoPrevio    1500
#define tiempoEjecucion 500
#define creadosCadaMax 100
//#define ER
#define SF
//#define creadosCada 1

//////////////////////


//Numero de nodos de nuestra red
int nNodos, nLinks;
int maximoGrado;
int creadosCada;
float p;
int tiempo, prorroga=0;
float h=0.6, epsilon = 0.05;

int nPaquetesActivos = 0, nPaquetesActivosAntes = 0;

#include "LibreriaCA.h"

int main(){

#ifdef ER
char nombreRed[nCaracteres]="red_ER_1000.txt";
#endif
#ifdef SF
char nombreRed[nCaracteres]="red_SF_1.txt";
#endif // SF
    //En primer lugar leemos la red
    printf("Vamos a calcular el grado maximo\n");
    MaximoGrado(nombreRed);
    printf("EL nombre es: %s\n", nombreRed);
    //Creamos los arrays que nos daran informacion sobre los nodos
    int *conCuantosCada=(int*) malloc(nNodos*sizeof(int));
    int *conQuienesCada=(int*) malloc(nNodos*maximoGrado*sizeof(int))
        → ;
```

```
CreaMatrices(nombreRed, conCuantosCada, conQuienesCada);
int *distancias= (int*) malloc(nNodos*nNodos*sizeof(int));
//Creamos una matriz con las distancias de cualquier nodo a
   → cualquier otro por el camino mas corto
DistanciasMinimas(conCuantosCada, conQuienesCada, distancias);

//Arrays necesarios para paquetes. Nos dan info de cada paquete
int *dondeEsta;
int *dondeVa;
int *quienVaDespues;

int *dondeEstaLuego;
int *dondeVaLuego;
int *quienVaDespuesLuego;
char bufferP[50], bufferH[50];

FILE *ficheroP;

//Arrays que nos dan informacion sobre cada nodo

int *cuantosEnMiCola = (int*) malloc(nNodos*sizeof(int));
int *cuantosEnMiColaLuego = (int*) malloc(nNodos*sizeof(int));
int *primero = (int*) malloc(nNodos*sizeof(int));
int *ultimo = (int*) malloc(nNodos*sizeof(int));
int *primeroLuego = (int*) malloc(nNodos*sizeof(int));
int *ultimoLuego = (int*) malloc(nNodos*sizeof(int));
int *paquetes; int ventana = 20, iVentana=0, nVentanas=
   → tiempoEjecucion/ventana;float rho, diferenciaPaquetes;

int i, iNodoAux, jNodoAux, iPaqueteAux;
int iCreado;

for(h=1;h>0.59;h=h-0.1){

    FILE *ficheroH;
    #ifdef ER
    sprintf(bufferH, "ERCongestionH%.2f.txt", h);
    #endif // ER
    #ifdef SF
    sprintf(bufferH, "SFCongestionH%.2f.txt", h);
    #endif // SF

    ficheroH = fopen (bufferH, "w");
    for(p=0.001;p<=0.1; p=p+0.002, iVentana=0){
        #ifdef ER
```

```
sprintf(bufferP, "ERPaquetesTiempoH%.2fP%.2f.txt", h,p);
#endif // ER
#ifdef SF
sprintf(bufferP, "SFPaquetesTiempoH%.2fP%.3f.txt", h,p);
#endif // SF
ficheroP=fopen(bufferP, "w");
creadosCada=(int)(p*nNodos);
InicializaNodos(cuantosEnMiCola, primero, ultimo);
dondeEsta = (int*) malloc((creadosCada)*sizeof(int));
dondeVa = (int*) malloc((creadosCada)*sizeof(int));
quienVaDespues = (int*) malloc((creadosCada)*sizeof(int))
    → ;
paquetes = (int*) malloc(nVentanas*sizeof(int));
for (tiempo = 0; tiempo < (tiempoPrevio+tiempoEjecucion+
    → prorroga); tiempo++){
    for(iCreado = 0; iCreado < creadosCada; iCreado++){
        CreaPaquete(dondeEsta, dondeVa, cuantosEnMiCola,
            → ultimo, quienVaDespues, primero);
    }
    nPaquetesActivosAntes=nPaquetesActivos;
    dondeEstaLuego = (int*) malloc((nPaquetesActivos)*
        → sizeof(int));
    dondeVaLuego = (int*) malloc((nPaquetesActivos)*
        → sizeof(int));
    quienVaDespuesLuego = (int*) malloc((nPaquetesActivos
        → )*sizeof(int));
    MuevePrimeros(dondeEsta, dondeVa, conCuantosCada,
        → conQuienesCada, distancias, primero,
        → cuantosEnMiCola, quienVaDespues, ultimo,
        → dondeEstaLuego, quienVaDespuesLuego,
        → dondeVaLuego, primeroLuego, ultimoLuego,
        → cuantosEnMiColaLuego);//La info pasa a los
        → LUEGO
    /*Tras mover los primeros, la informacion relevante
        → esta en los arrays con "Luego" al final,
    los otros ademas tendran que cambiar de tamano,
        → reduciendose por los paquetes que llegan y
    aumentandose por los que se crean en cada paso de
        → tiempo.*/
    free(dondeEsta); free(dondeVa); free(quienVaDespues);
        → //Liberamos los NOLUEGO
    //Volvemos a crear los arrays, copiamos la
        → informacion (aqui muchos paquetes cambian de
        → numero, ya que
    //los paquetes que llegan tienen informacion
        → irrelevante y la destruimos
```

```
                dondeEsta = (int*) malloc((nPaquetesActivos+
                    → creadosCada)*sizeof(int));
                dondeVa = (int*) malloc((nPaquetesActivos+creadosCada
                    → )*sizeof(int));
                quienVaDespues = (int*) malloc((nPaquetesActivos+
                    → creadosCada)*sizeof(int));
                CambiaArrays(dondeEstaLuego, dondeEsta, dondeVaLuego,
                    →  dondeVa, quienVaDespuesLuego, quienVaDespues,
                    → primero, ultimo, cuantosEnMiCola, primeroLuego,
                    →  ultimoLuego, cuantosEnMiColaLuego);//La
                    → informacion pasa de los Luego a los NOLUEGO
                free(dondeEstaLuego);F
                free(dondeVaLuego);
                free(quienVaDespuesLuego);
                if((tiempo>tiempoPrevio)&&((tiempo-tiempoPrevio)%
                    → ventana==0)){
                    paquetes[iVentana]=nPaquetesActivos;
    //                  printf("nVentanas vale %d y en %d hay %d
        → paquetes\n", nVentanas, iVentana, paquetes[iVentana]);
                    iVentana++;
                }
                if (tiempo%5==0){
                    fprintf(ficheroP, "%d\t %d\n", tiempo,
                        → nPaquetesActivos);
                }
            }
            for(iVentana = 0, rho = 0;iVentana<nVentanas-2;iVentana
                → ++){
                diferenciaPaquetes = (float)(paquetes[iVentana+1]-
                    → paquetes[iVentana]);
                rho = rho +(diferenciaPaquetes/(ventana*p*nNodos));
            }
            rho=rho/nVentanas;
            printf("El parametro de orden para %d creados y h= %f
                → vale %f y hay %d paquetes activos\n", creadosCada,h
                → , rho, nPaquetesActivos);
            fprintf(ficheroH, "%f\t %f\n", p, rho);
            nPaquetesActivos=0;
            nPaquetesActivosAntes=0;
            fclose(ficheroP);
            free(paquetes);
        }
        fclose(ficheroH);
}
    free(primero);
    free(ultimo);
```

```
    free(primeroLuego);
    free(ultimoLuego);
    free(cuantosEnMiCola);
    free(cuantosEnMiColaLuego);
    free(conCuantosCada);
    free(conQuienesCada);
    free(distancias);
    free(dondeEsta);
    free(dondeVa);
    free(quienVaDespues);
    return 0;
}
```

A.2.1. Librería utilizada

En este caso la librería es la misma que la mostrada en A.1.1.

A.3. Mejor de cada nodo

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <malloc.h>

//////////////////////

//#define EMPIEZAEN1
#define nCaracteres 256
int tiempoPrevio=    2000;
#define tiempoEjecucion 1000
#define creadosCadaMax 100
//#define ER
#define SF
//#define creadosCada 1

//////////////////////


//Numero de nodos de nuestra red
int nNodos, nLinks;
int maximoGrado;
int creadosCada;
float p;
int tiempo, prorroga=0;
float h=0.6, epsilon = 0.05;
int ventanaCambios = 5;
```

```
int nPaquetesActivos = 0, nPaquetesActivosAntes = 0;

#include "BiblioShort.h"

int main(){

#ifdef ER
char nombreRed[nCaracteres]="red_ER_1.txt";
#endif
#ifdef SF
char nombreRed[nCaracteres]="red_SF_1.txt";
#endif // SF
    //En primer lugar leemos la red
    //Calculamos el grado maximo de la red (Esto nos dara info sobre
       → el tamano de algunos arrays
    printf("Vamos a calcular el grado maximo\n");
    MaximoGrado(nombreRed);
    printf("EL nombre es: %s\n", nombreRed);
    //Creamos los arrays que nos daran informacion sobre los nodos
    int *conCuantosCada=(int*) malloc(nNodos*sizeof(int));
    int *conQuienesCada=(int*) malloc(nNodos*maximoGrado*sizeof(int))
       → ;
    float *mejoresPayOff = (float*) malloc(nNodos*sizeof(float));
    CreaMatrices(nombreRed, conCuantosCada, conQuienesCada);
    int *distancias= (int*) malloc(nNodos*nNodos*sizeof(int));
    //Creamos una matriz con las distancias de cualquier nodo a
       → cualquier otro por el camino mas corto
    DistanciasMinimas(conCuantosCada, conQuienesCada, distancias);

    //Arrays necesarios para paquetes (nos dan info de cada paquete)
    int *dondeEsta, *dondeVa, *quienVaDespues, *cuantasVentanas, *
       → tipoPaquete, *distanciasAnteriores, *distanciasActuales;
    float *payOff;
    int *dondeEstaLuego, *dondeVaLuego, *quienVaDespuesLuego, *
       → cuantasVentanasLuego, *tipoPaqueteLuego,
       → distanciasAnteriosesLuego;
    float *payOffLuego;
    char bufferP[50], bufferH[50], bufferTipo[50];

    FILE *ficheroP;

    //Arrays que nos dan informacion sobre cada nodo

    int *cuantosEnMiCola = (int*) malloc(nNodos*sizeof(int));
    int *cuantosEnMiColaLuego = (int*) malloc(nNodos*sizeof(int));
```

```
//     int *puntuacionVecinos = (int*) malloc(maximoGrado*sizeof(int))
   → ;
    int *primero = (int*) malloc(nNodos*sizeof(int));
    int *tieneUtiles = (int*) malloc(nNodos*sizeof(int));
    int *elMejorPaquete = (int*)malloc(nNodos*sizeof(int));
    int *ultimo = (int*) malloc(nNodos*sizeof(int));
    int *primeroLuego = (int*) malloc(nNodos*sizeof(int));
    int *ultimoLuego = (int*) malloc(nNodos*sizeof(int));
    int *paquetes; int ventana = 20, iVentana=0, nVentanas=
       → tiempoEjecucion/ventana;float rho, diferenciaPaquetes;


    int i, iNodoAux, jNodoAux, iPaqueteAux;
    float unos, ceros;
    int iCreado;
        FILE *ficheroH;
        FILE *ficheroTipo;

    //Empezamos con elprograma en si
    for(h= 0.7;h>0.59;h=h-0.1){


        #ifdef ER
        sprintf(bufferH, "ERCongestionH%.2f.txt", h);
        #endif // ER
        #ifdef SF
        sprintf(bufferH, "SFCongestionH%.2f.txt", h);
        #endif // SF

        ficheroH = fopen (bufferH, "w");
        for(p=0.001;p<=0.06; p=p+0.002, iVentana=0){
            #ifdef ER
            sprintf(bufferP, "ERPaquetesTiempoH%.2fP%.2f.txt", h,p);
            sprintf(bufferTipo, "ERTipoTiempoH%.2fP%.2f.txt", h, p);
            #endif // ER
            #ifdef SF
            sprintf(bufferP, "SFPaquetesTiempoH%.2fP%.3f.txt", h,p);
            sprintf(bufferTipo, "SFTipoTiempoH%.2fP%.3f.txt", h, p);
            #endif // SF

            ficheroP=fopen(bufferP, "w");
            ficheroTipo = fopen(bufferTipo, "w");

            creadosCada=(int)(p*nNodos);
```

```
            printf("Creados cada vale: %d\n",creadosCada ); //
               → Comentario para ver que el programa va avanzando
            InicializaNodos(cuantosEnMiCola, primero, ultimo);
//                printf("Definiremos los arrays\n");
            dondeEsta = (int*) malloc((creadosCada)*sizeof(int));
            dondeVa = (int*) malloc((creadosCada)*sizeof(int));
            quienVaDespues = (int*) malloc((creadosCada)*sizeof(int))
               → ;
            paquetes = (int*) malloc(nVentanas*sizeof(int));
            cuantasVentanas = (int*) malloc((creadosCada)*sizeof(int)
               → );
            payOff = (float*) malloc((creadosCada)*sizeof(float));
            tipoPaquete = (int*) malloc((creadosCada)*sizeof(int));
            distanciasAnteriores = (int*) malloc((creadosCada)*sizeof
               → (int));
            tiempoPrevio = 1000;
            if(p<0.052){
                 tiempoPrevio = 3000;
            }

            for (tiempo = 0; tiempo < (tiempoPrevio+tiempoEjecucion+
               → prorroga); tiempo++){


                for(iCreado = 0; iCreado < creadosCada; iCreado++){
                     CreaPaquete(dondeEsta, dondeVa, cuantosEnMiCola,
                        → ultimo, quienVaDespues, primero,
                        → tipoPaquete, cuantasVentanas, distancias,
                        → distanciasAnteriores);
                }
                nPaquetesActivosAntes=nPaquetesActivos;
                dondeEstaLuego = (int*) malloc((nPaquetesActivos)*
                   → sizeof(int));
                dondeVaLuego = (int*) malloc((nPaquetesActivos)*
                   → sizeof(int));
                quienVaDespuesLuego = (int*) malloc((nPaquetesActivos
                   → )*sizeof(int));
                cuantasVentanasLuego = (int*) malloc((
                   → nPaquetesActivos)*sizeof(int));
                payOffLuego = (float*) malloc((nPaquetesActivos)*
                   → sizeof(float));
                tipoPaqueteLuego = (int*) malloc((nPaquetesActivos)*
                   → sizeof(int));
                distanciasAnterioresLuego = (int*) malloc((
                   → nPaquetesActivos)*sizeof(int));
```

```
IgualaArrays(cuantasVentanas, cuantasVentanasLuego,
    → payOff, payOffLuego, tipoPaquete,
    → tipoPaqueteLuego, distanciasAnteriores,
    → distanciasAnterioresLuego);
MuevePrimeros(dondeEsta, dondeVa, conCuantosCada,
    → conQuienesCada, distancias, primero,
    → cuantosEnMiCola, quienVaDespues, ultimo,
    → dondeEstaLuego, quienVaDespuesLuego,
    → dondeVaLuego, primeroLuego, ultimoLuego,
    → cuantosEnMiColaLuego, tipoPaquete);//La info
    → pasa a los LUEGO
/*Tras mover los primeros, la informacion relevante
    → esta en los arrays con "Luego" al final,
los otros ademas tendran que cambiar de tamano,
    → reduciendose por los paquetes que llegan y
aumentandose por los que se crean en cada paso de
    → tiempo.*/
free(dondeEsta); free(dondeVa); free(quienVaDespues);
    → //Liberamos los NOLUEGO
free(cuantasVentanas);free(payOff);free(tipoPaquete);
    → free(distanciasAnteriores);
//Volvemos a crear los arrays, copiamos la
    → informacion (aqui muchos paquetes cambian de
    → numero, ya que
//los paquetes que llegan tienen informacion
    → irrelevante y la destruimos
dondeEsta = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
dondeVa = (int*) malloc((nPaquetesActivos+creadosCada
    → )*sizeof(int));
quienVaDespues = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
cuantasVentanas = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
payOff = (float*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(float));
tipoPaquete = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
distanciasAnteriores = (int*) malloc((
    → nPaquetesActivos+creadosCada)*sizeof(int));
CambiaArrays(dondeEstaLuego, dondeEsta, dondeVaLuego,
    →  dondeVa, quienVaDespuesLuego, quienVaDespues,
    → primero, ultimo, cuantosEnMiCola, primeroLuego,
    →  ultimoLuego, cuantosEnMiColaLuego,
    → cuantasVentanas, payOff, tipoPaquete,
    → cuantasVentanasLuego, payOffLuego,
```

```
            → tipoPaqueteLuego, distanciasAnteriores,
            → distanciasAnterioresLuego);
        free(dondeEstaLuego);free(dondeVaLuego);free(
            → quienVaDespuesLuego);
        free(cuantasVentanasLuego);free(payOffLuego);free(
            → tipoPaqueteLuego);free(
            → distanciasAnterioresLuego);
        if((tiempo>tiempoPrevio)&&((tiempo-tiempoPrevio)%
            → ventana==0)){
            paquetes[iVentana]=nPaquetesActivos;
            iVentana++;
        }
        if (tiempo%5==0){
            fprintf(ficheroP, "%d\t %d\n", tiempo,
                → nPaquetesActivos);
        }
        if(tiempo%ventanaCambios==0&& tiempo !=0){
            distanciasActuales = (int*) malloc(
                → nPaquetesActivos*sizeof(int));
            DistanciasActuales(dondeEsta, dondeVa, distancias
                → , distanciasActuales);
            CalculaPayOff(distanciasActuales,
                → distanciasAnteriores, payOff, mejoresPayOff
                → , dondeEsta, primero, cuantasVentanas,
                → tieneUtiles, quienVaDespues, elMejorPaquete
                → );
            free(distanciasActuales);
            CambiaEstrategias(tipoPaquete, mejoresPayOff,
                → payOff, dondeEsta, tieneUtiles,
                → elMejorPaquete);
            ActualizaArrays(cuantasVentanas,
                → distanciasAnteriores, distancias, dondeEsta
                → , dondeVa);
            CuentaPaquetes(tipoPaquete, &unos, &ceros);
            fprintf(ficheroTipo, "%d\t %f\t %f\t\n", tiempo,
                → unos, ceros)

        }
}


for(iVentana = 0, rho = 0;iVentana<nVentanas-2;iVentana
    → ++){
    diferenciaPaquetes = (float)(paquetes[iVentana+1]-
        → paquetes[iVentana]);
    rho = rho +(diferenciaPaquetes/(ventana*p*nNodos));
```

```
                }
                rho=rho/nVentanas;
                printf("El parametro de orden para %d creados y h= %f
                    → vale %f y hay %d paquetes activos\n", creadosCada,h
                    → , rho, nPaquetesActivos);
                fprintf(ficheroH, "%f\t %f\n", p, rho);
                nPaquetesActivos=0;
                nPaquetesActivosAntes=0;
                fclose(ficheroP);
                fclose(ficheroTipo);
                free(paquetes);
            }
            fclose(ficheroH);
}
    //Cosas de nodos (no se liberan cada vez)
    free(primero);
    free(ultimo);
    free(tieneUtiles);
    free(elMejorPaquete);
    free(primeroLuego);
    free(ultimoLuego);
    free(cuantosEnMiCola);
    free(cuantosEnMiColaLuego);
    free(conCuantosCada);
    free(conQuienesCada);
    free(distancias);
    free(mejoresPayOff);


    //Cosas de paquetes, se van liberando
    free(dondeEsta);
    free(dondeVa);
    free(quienVaDespues);
    free(cuantasVentanas);
    free(payOff);
    free(tipoPaquete);
    free(distanciasAnteriores);;
    printf("\nFin\n");
    return 0;
}
```

A.3.1. Librería utilizada

A la librería mostrada en A.1.1 se añaden las funciones:

```
void IgualaArrays(int *cuantasVentanas, int *cuantasVentanasLuego,
    → float *payOff, float *payOffLuego, int *tipoPaquete, int *
    → tipoPaqueteLuego, int *distanciasAnteriores, int *
```

```
    → distanciasAnteriores Luego);
void CalculaPayOff(int *distanciasActuales, int *distanciasAnteriores
    → , float *payOff, float *mejoresPayOff, int *dondeEsta,int *
    → primero, int *cuantasVentanas, int *tieneUtiles, int *
    → quienVaDespues, int *elMejorPaquete);
void CambiaEstrategias(int *tipoPaquete, float *mejoresPayoff, float
    → *payOff, int *dondeEsta, int *tieneUtiles, int* elMejorPaquete)
    → ;
void CuentaPaquetes(int *tipoPaquete, float *unos, float *ceros);


void DistanciasActuales(int *dondeEsta, int *dondeVa, int *distancias
    → , int *distanciasActuales){
     int iPaquete;
     for (iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
          distanciasActuales[iPaquete] = distancias[dondeEsta[iPaquete
             → ]*nNodos+dondeVa[iPaquete]];
     }


}


void CalculaPayOff(int *distanciasActuales, int *distanciasAnteriores
    → , float *payOff, float *mejoresPayOff, int *dondeEsta,int *
    → primero, int *cuantasVentanas, int *tieneUtiles,  int *
    → quienVaDespues, int *elMejorPaquete){
     int iPaquete, iNodo, visitamos, salimos;
     for(iNodo = 0; iNodo<nNodos; iNodo++, visitamos = 0 ){
          visitamos = primero[iNodo];
          if(visitamos != -1){
          }

          tieneUtiles[iNodo] = 0;
          salimos  = 0;
          for(visitamos = primero[iNodo]; visitamos != -1 && salimos ==
             →  0;){
               if(cuantasVentanas[visitamos] != 2){
                    visitamos = quienVaDespues[visitamos];

               }
               else{
                    salimos = 1;
                    tieneUtiles[iNodo] = 1;
                    elMejorPaquete[iNodo] = visitamos;
```

```
                mejoresPayOff[iNodo] = ((float)(distanciasAnteriores[
                    → visitamos]-distanciasActuales[visitamos]))/
                    → ventanaCambios;

            }

        }
    }

    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){

        payOff[iPaquete] = ((float)(distanciasAnteriores[iPaquete]-
            → distanciasActuales[iPaquete]))/((float)(ventanaCambios)
            → );
        if(payOff[iPaquete] > mejoresPayOff[dondeEsta[iPaquete]]&&
            → cuantasVentanas[iPaquete] ==2){
            mejoresPayOff[dondeEsta[iPaquete]] = payOff[iPaquete];
            elMejorPaquete[dondeEsta[iPaquete]] = iPaquete;
        }
    }


}

void CambiaEstrategias(int *tipoPaquete, float *mejoresPayoff, float
    → *payOff, int *dondeEsta, int *tieneDisponibles, int *
    → elMejorPaquete){
    int iPaquete; float fermi; float alea, siONo;

    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        siONo = (float) tieneDisponibles[dondeEsta[iPaquete]];
        fermi=(float) (1.0/(1.0+exp(((double)(payOff[iPaquete]-
            → mejoresPayoff[dondeEsta[iPaquete]])))));
        if(generaRand()<fermi*siONo){
            if(tipoPaquete[elMejorPaquete[dondeEsta[iPaquete]]]!=
                → tipoPaquete[iPaquete]){
                if(tipoPaquete[elMejorPaquete[dondeEsta[iPaquete
                    → ]]]==1){
                    tipoPaquete[iPaquete] == 1;
                }
                else{
                    tipoPaquete[iPaquete] == 0;
                }
            }
        }
    }
```

```
}

void ActualizaArrays(int *cuantasVentanas, int *distanciasAnteriores,
    → int *distancias, int *dondeEsta, int *dondeVa){
    int iPaquete;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        distanciasAnteriores[iPaquete]=distancias[dondeEsta[iPaquete
            → ]*nNodos+dondeVa[iPaquete]];
        if(cuantasVentanas[iPaquete]<2){
            cuantasVentanas[iPaquete]++;
        }
    }
}
void CuentaPaquetes(int *tipoPaquete, float *unos, float *ceros){
    int nCeros = 0, nUnos = 0;
    int iPaquete;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        if(tipoPaquete[iPaquete]==0){
            nCeros++;
        }
        if(tipoPaquete[iPaquete] == 1){
            nUnos++;
        }
        if(tipoPaquete[iPaquete] != 0&& tipoPaquete[iPaquete]!=1){
            printf("::::::::::::::::");
        }
    }
    *ceros = (float) nCeros/nPaquetesActivos;
    *unos = (float) nUnos/nPaquetesActivos;
}
void RevisaOrden (int *quienVaDespues, int *dondeEsta, int *primero){
    int iNodo, visitamos;
    for(iNodo = 0; iNodo < nNodos; iNodo++){
        for(visitamos = primero[iNodo]; visitamos != -1;visitamos =
            → quienVaDespues[visitamos]){

            if((dondeEsta[visitamos] != dondeEsta[quienVaDespues[
                → visitamos]])&& (quienVaDespues[visitamos]!=-1)){
                printf("Visitamos␣el␣paquete␣%d␣y␣esta␣en␣el␣nodo␣%d.
                    → ␣Despues␣va␣%d\n", visitamos, dondeEsta[
                    → visitamos], quienVaDespues[visitamos]);

                printf("::::::::::::::::::VAYA␣VAYA
                    → ::::::::::::::::::\n");
            }
        }
```

```
    }
}
```

A.4. Mejor de la vecindad

```
#include <stdio.h>
#include <math.h>
#include <time.h>
#include <stdlib.h>
#include <malloc.h>

//////////////////////

//#define EMPIEZAEN1
#define nCaracteres 256
int tiempoPrevio = 1000;
#define tiempoEjecucion 1000
#define creadosCadaMax 100
//#define ER
#define SF
//#define creadosCada 1

//////////////////////


//Numero de nodos de nuestra red
int nNodos, nLinks;
int maximoGrado;
int creadosCada;
float p;
int tiempo, prorroga=0;
float h=0.6, epsilon = 0.05;
int ventanaCambios = 3;
int radioAccion = 2;

int nPaquetesActivos = 0, nPaquetesActivosAntes = 0;

#include "BiblioShort.h"

int main(){

#ifdef ER
char nombreRed[nCaracteres]="red_ER_1.txt";
#endif
#ifdef SF
char nombreRed[nCaracteres]="red_SF_1.txt";
```

```
#endif // SF
    //En primer lugar leemos la red
    //Calculamos el grado maximo de la red (Esto nos dara info sobre
       → el tamano de algunos arrays
    printf("Vamos a calcular el grado maximo\n");
    MaximoGrado(nombreRed);
    printf("EL nombre es: %s\n", nombreRed);
    //Creamos los arrays que nos daran informacion sobre los nodos
    int *conCuantosCada=(int*) malloc(nNodos*sizeof(int));
    int *conQuienesCada=(int*) malloc(nNodos*maximoGrado*sizeof(int))
       → ;
    float *mejoresPayOff = (float*) malloc(nNodos*sizeof(float));
    CreaMatrices(nombreRed, conCuantosCada, conQuienesCada);
    int *distancias= (int*) malloc(nNodos*nNodos*sizeof(int));
    //Creamos una matriz con las distancias de cualquier nodo a
       → cualquier otro por el camino mas corto
    DistanciasMinimas(conCuantosCada, conQuienesCada, distancias);

    //Arrays necesarios para paquetes (nos dan info de cada paquete)
    int *dondeEsta;
    int *dondeVa;
    int *quienVaDespues;
    int *cuantasVentanas;
    float *payOff;
    int *tipoPaquete;
    int *distanciasAnteriores;
    int *distanciasActuales;


    int *dondeEstaLuego;
    int *dondeVaLuego;
    int *quienVaDespuesLuego;
    int *cuantasVentanasLuego;
    float *payOffLuego;
    int *tipoPaqueteLuego;
    int *distanciasAnterioresLuego;

    char bufferP[50], bufferH[50], bufferTipo[50];

    FILE *ficheroP;

    //Arrays que nos dan informacion sobre cada nodo

    int *cuantosEnMiCola = (int*) malloc(nNodos*sizeof(int));
    int *cuantosEnMiColaLuego = (int*) malloc(nNodos*sizeof(int));
```

```
//     int *puntuacionVecinos = (int*) malloc(maximoGrado*sizeof(int))
    → ;
    int *primero = (int*) malloc(nNodos*sizeof(int));
    int *tieneUtiles = (int*) malloc(nNodos*sizeof(int));
    int *elMejorPaquete = (int*)malloc(nNodos*sizeof(int));
    int *ultimo = (int*) malloc(nNodos*sizeof(int));
    int *primeroLuego = (int*) malloc(nNodos*sizeof(int));
    int *ultimoLuego = (int*) malloc(nNodos*sizeof(int));
    int *paquetes; int ventana = 20, iVentana=0, nVentanas=
       → tiempoEjecucion/ventana;float rho, diferenciaPaquetes;
    int i, iNodoAux, jNodoAux, iPaqueteAux;
    float unos, ceros;
    int iCreado;
        FILE *ficheroH;
        FILE *ficheroTipo;
    //Empezamos con elprograma en si
    for (radioAccion = 1; radioAccion < 2; radioAccion++){
    for(h= 0.9;h>0.59;h=h-0.1){
        #ifdef ER
        sprintf(bufferH, "ERCongestionH%.2f.txt", h);
        #endif // ER
        #ifdef SF
        sprintf(bufferH, "SFCongestionR%dH%.1f.txt",radioAccion, h);
        #endif // SF
        ficheroH = fopen (bufferH, "w");
        for(p=0.001;p<=0.06; p=p+0.002, iVentana=0){
            #ifdef ER
            sprintf(bufferP, "ERPaquetesTiempoR%dH%.2fP%.2f.txt",
               → radioAccion h,p);
            sprintf(bufferTipo, "ERTipoTiempoR%dH%.2fP%.2f.txt",
               → radioAccion, h, p);
            #endif // ER
            #ifdef SF
            sprintf(bufferP, "SFPaquetesTiempoR%dH%.1fP%.3f.txt",
               → radioAccion, h,p);
            sprintf(bufferTipo, "SFTipoTiempoR%dH%.1fP%.3f.txt",
               → radioAccion, h, p);
            #endif // SF
            ficheroP=fopen(bufferP, "w");
            ficheroTipo = fopen(bufferTipo, "w");
            creadosCada=(int)(p*nNodos);
            printf("Creados cada vale: %d\n",creadosCada ); //
               → Comentario para ver que el programa va avanzando
            InicializaNodos(cuantosEnMiCola, primero, ultimo);
            dondeEsta = (int*) malloc((creadosCada)*sizeof(int));
            dondeVa = (int*) malloc((creadosCada)*sizeof(int));
```

```
quienVaDespues = (int*) malloc((creadosCada)*sizeof(int))
    → ;
paquetes = (int*) malloc(nVentanas*sizeof(int));
cuantasVentanas = (int*) malloc((creadosCada)*sizeof(int)
    → );
payOff = (float*) malloc((creadosCada)*sizeof(float));
tipoPaquete = (int*) malloc((creadosCada)*sizeof(int));
distanciasAnteriores = (int*) malloc((creadosCada)*sizeof
    → (int));
tiempoPrevio = 1000;
if(p<0.052){
     tiempoPrevio = 3000;
}
for (tiempo = 0; tiempo < (tiempoPrevio+tiempoEjecucion+
    → prorroga); tiempo++){
    for(iCreado = 0; iCreado < creadosCada; iCreado++){
        CreaPaquete(dondeEsta, dondeVa, cuantosEnMiCola,
            → ultimo, quienVaDespues, primero,
            → tipoPaquete, cuantasVentanas, distancias,
            → distanciasAnteriores);
    }
    nPaquetesActivosAntes=nPaquetesActivos;
    dondeEstaLuego = (int*) malloc((nPaquetesActivos)*
        → sizeof(int));
    dondeVaLuego = (int*) malloc((nPaquetesActivos)*
        → sizeof(int));
    quienVaDespuesLuego = (int*) malloc((nPaquetesActivos
        → )*sizeof(int));
    cuantasVentanasLuego = (int*) malloc((
        → nPaquetesActivos)*sizeof(int));
    payOffLuego = (float*) malloc((nPaquetesActivos)*
        → sizeof(float));
    tipoPaqueteLuego = (int*) malloc((nPaquetesActivos)*
        → sizeof(int));
    distanciasAnterioresLuego = (int*) malloc((
        → nPaquetesActivos)*sizeof(int));
    IgualaArrays(cuantasVentanas, cuantasVentanasLuego,
        → payOff, payOffLuego, tipoPaquete,
        → tipoPaqueteLuego, distanciasAnteriores,
        → distanciasAnterioresLuego);
    MuevePrimeros(dondeEsta, dondeVa, conCuantosCada,
        → conQuienesCada, distancias, primero,
        → cuantosEnMiCola, quienVaDespues, ultimo,
        → dondeEstaLuego, quienVaDespuesLuego,
        → dondeVaLuego, primeroLuego, ultimoLuego,
        → cuantosEnMiColaLuego, tipoPaquete);//La info
```

```
    → pasa a los LUEGO
/*Tras mover los primeros, la informacion relevante
    → esta en los arrays con "Luego" al final,
los otros ademas tendran que cambiar de tamano,
    → reduciendose por los paquetes que llegan y
aumentandose por los que se crean en cada paso de
    → tiempo.*/
free(dondeEsta); free(dondeVa); free(quienVaDespues);
    → //Liberamos los NOLUEGO
free(cuantasVentanas);free(payOff);free(tipoPaquete);
    → free(distanciasAnteriores);
//Volvemos a crear los arrays, copiamos la
    → informacion (aqui muchos paquetes cambian de
    → numero, ya que
//los paquetes que llegan tienen informacion
    → irrelevante y la destruimos
dondeEsta = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
dondeVa = (int*) malloc((nPaquetesActivos+creadosCada
    → )*sizeof(int));
quienVaDespues = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
cuantasVentanas = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
payOff = (float*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(float));
tipoPaquete = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
distanciasAnteriores = (int*) malloc((
    → nPaquetesActivos+creadosCada)*sizeof(int));
CambiaArrays(dondeEstaLuego, dondeEsta, dondeVaLuego,
    →  dondeVa, quienVaDespuesLuego, quienVaDespues,
    → primero, ultimo, cuantosEnMiCola, primeroLuego,
    →  ultimoLuego, cuantosEnMiColaLuego,
    → cuantasVentanas, payOff, tipoPaquete,
    → cuantasVentanasLuego, payOffLuego,
    → tipoPaqueteLuego, distanciasAnteriores,
    → distanciasAnterioresLuego);//La informacion
    → pasa de los Luego a los NOLUEGO
free(dondeEstaLuego);free(dondeVaLuego);free(
    → quienVaDespuesLuego);
free(cuantasVentanasLuego);free(payOffLuego);free(
    → tipoPaqueteLuego);free(
    → distanciasAnterioresLuego);
if((tiempo>tiempoPrevio)&&((tiempo-tiempoPrevio)%
    → ventana==0)){
```

```
            paquetes[iVentana]=nPaquetesActivos;
            iVentana++;
        }
        if (tiempo%5==0){
            fprintf(ficheroP, "%d\t %d\n", tiempo,
                → nPaquetesActivos);
        }
        if(tiempo%ventanaCambios==0&& tiempo !=0){
            distanciasActuales = (int*) malloc(
                → nPaquetesActivos*sizeof(int));
            DistanciasActuales(dondeEsta, dondeVa, distancias
                → , distanciasActuales);
            CalculaPayOff(distanciasActuales,
                → distanciasAnteriores, payOff, mejoresPayOff
                → , dondeEsta, primero, cuantasVentanas,
                → tieneUtiles, quienVaDespues, elMejorPaquete
                → );
            free(distanciasActuales);
            ElMejorPayoff(mejoresPayOff, radioAccion,
                → conQuienesCada, conCuantosCada,
                → elMejorPaquete, dondeEsta, tieneUtiles,
                → cuantosEnMiCola);
            CambiaEstrategias(tipoPaquete, mejoresPayOff,
                → payOff, dondeEsta, tieneUtiles,
                → elMejorPaquete);
            ActualizaArrays(cuantasVentanas,
                → distanciasAnteriores, distancias, dondeEsta
                → , dondeVa);
            CuentaPaquetes(tipoPaquete, &unos, &ceros);
            fprintf(ficheroTipo, "%d\t %f\t %f\t\n", tiempo,
                → unos, ceros);
        }
    }

    for(iVentana = 0, rho = 0;iVentana<nVentanas-2;iVentana
        → ++){
        diferenciaPaquetes = (float)(paquetes[iVentana+1]-
            → paquetes[iVentana]);
        rho = rho +(diferenciaPaquetes/(ventana*p*nNodos));
    }
    rho=rho/nVentanas;
    fprintf(ficheroH, "%f\t %f\n", p, rho);
    nPaquetesActivos=0;
    nPaquetesActivosAntes=0;
    fclose(ficheroP);
    fclose(ficheroTipo);
```

```
            free(paquetes);
        }
        fclose(ficheroH);
}
}
    free(primero);
    free(ultimo);
    free(tieneUtiles);
    free(elMejorPaquete);
    free(primeroLuego);
    free(ultimoLuego);
    free(cuantosEnMiCola);
    free(cuantosEnMiColaLuego);
    free(conCuantosCada);
    free(conQuienesCada);
    free(distancias);
    free(mejoresPayOff);

    free(dondeEsta);
    free(dondeVa);
    free(quienVaDespues);
    free(cuantasVentanas);
    free(payOff);
    free(tipoPaquete);
    free(distanciasAnteriores);;
    return 0;
}
```

A.4.1. Librería utilizada

A la librería mostrada en A.1.1 se añaden las funciones:

```
void IgualaArrays(int *cuantasVentanas, int *cuantasVentanasLuego,
    → float *payOff, float *payOffLuego, int *tipoPaquete, int *
    → tipoPaqueteLuego, int *distanciasAnteriores, int *
    → distanciasAnterioresLuego);
void CalculaPayOffMejor(int *distanciasActuales, int *
    → distanciasAnteriores, float *payOff, float *mejoresPayOff, int
    → *dondeEsta,int *primero, int *cuantasVentanas, int *tieneUtiles
    → , int *quienVaDespues, int *elMejorPaquete);
void CambiaEstrategias(int *tipoPaquete, float *mejoresPayoff, float
    → *payOff, int *dondeEsta, int *tieneUtiles, int* elMejorPaquete)
    → ;
void CuentaPaquetes(int *tipoPaquete, float *unos, float *ceros);

void IgualaArrays(int *cuantasVentanas, int *cuantasVentanasLuego,
    → float *payOff, float *payOffLuego, int *tipoPaquete, int *
    → tipoPaqueteLuego, int *distanciasAnteriores, int *
```

```
    → distanciasAnterioresLuego){

     int iPaquete;
     for(iPaquete = 0; iPaquete< nPaquetesActivos; iPaquete++){
          cuantasVentanasLuego[iPaquete] = cuantasVentanas[iPaquete];
          payOffLuego[iPaquete] = payOff[iPaquete];
          tipoPaqueteLuego[iPaquete] = tipoPaquete[iPaquete];
          distanciasAnterioresLuego[iPaquete] = distanciasAnteriores[
              → iPaquete];

     }

}

void DistanciasActuales(int *dondeEsta, int *dondeVa, int *distancias
    → , int *distanciasActuales){
     int iPaquete;
//         printf("Entro a DistanciasActuales\n");
     for (iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
//                 printf("El paquete esta en %d y va a %d\n",
    → dondeEsta[iPaquete], dondeVa[iPaquete]);
          distanciasActuales[iPaquete] = distancias[dondeEsta[iPaquete
              → ]*nNodos+dondeVa[iPaquete]];
     }


}

void CalculaPayOff(int *distanciasActuales, int *distanciasAnteriores
    → , float *payOff, float *mejoresPayOff, int *dondeEsta,int *
    → primero, int *cuantasVentanas, int *tieneUtiles,  int *
    → quienVaDespues, int *elMejorPaquete){
     int iPaquete, iNodo, visitamos, salimos;
     for(iNodo = 0; iNodo<nNodos; iNodo++, visitamos = 0 ){
          visitamos = primero[iNodo];
          elMejorPaquete[iNodo] = -1;
          if(visitamos != -1){
          }
          tieneUtiles[iNodo] = 0;
          salimos  = 0;
          for(visitamos = primero[iNodo]; visitamos != -1 && salimos ==
              →  0;){
               if(cuantasVentanas[visitamos] != 2){
                    visitamos = quienVaDespues[visitamos];
               }
               else{
```

```
                salimos = 1;
                tieneUtiles[iNodo] = 1;
                elMejorPaquete[iNodo] = visitamos;
                mejoresPayOff[iNodo] = ((float)(distanciasAnteriores[
                    → visitamos]-distanciasActuales[visitamos]))/
                    → ventanaCambios;
            }
        }
    }
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        payOff[iPaquete] = ((float)(distanciasAnteriores[iPaquete]-
            → distanciasActuales[iPaquete]))/((float)(ventanaCambios)
            → );
        if((payOff[iPaquete] > mejoresPayOff[dondeEsta[iPaquete]])&&
            → (cuantasVentanas[iPaquete] ==2)){
            mejoresPayOff[dondeEsta[iPaquete]] = payOff[iPaquete];
            elMejorPaquete[dondeEsta[iPaquete]] = iPaquete;
        }
    }
}

void CambiaEstrategias(int *tipoPaquete, float *mejoresPayoff, float
    → *payOff, int *dondeEsta, int *tieneDisponibles, int *
    → elMejorPaquete){
    int iPaquete; float fermi; float alea, siONo;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        siONo = (float) tieneDisponibles[dondeEsta[iPaquete]];
        fermi=(float) (1.0/(1.0+exp(((double)(payOff[iPaquete]-
            → mejoresPayoff[dondeEsta[iPaquete]])))));
        fermi = 1.0;
        if(generaRand()<fermi*siONo){
            if(tipoPaquete[elMejorPaquete[dondeEsta[iPaquete]]]!=
                → tipoPaquete[iPaquete]){
                if(tipoPaquete[elMejorPaquete[dondeEsta[iPaquete
                    → ]]]==1){
                    tipoPaquete[iPaquete] == 1;
                }
                else{
                    tipoPaquete[iPaquete] == 0;
                }
            }
        }
    }
}
```

```
void ActualizaArrays(int *cuantasVentanas, int *distanciasAnteriores,
    → int *distancias, int *dondeEsta, int *dondeVa){
    int iPaquete;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        distanciasAnteriores[iPaquete]=distancias[dondeEsta[iPaquete
            → ]*nNodos+dondeVa[iPaquete]];
        if(cuantasVentanas[iPaquete]<2){
            cuantasVentanas[iPaquete]++;
        }
        else{
        }
    }
}
void CuentaPaquetes(int *tipoPaquete, float *unos, float *ceros){
    int nCeros = 0, nUnos = 0;
    int iPaquete;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        if(tipoPaquete[iPaquete]==0){
            nCeros++;
        }
        if(tipoPaquete[iPaquete] == 1){
            nUnos++;
        }
        if(tipoPaquete[iPaquete] != 0&& tipoPaquete[iPaquete]!=1){
            printf("::::::::::::::::");
        }
    }
    *ceros = (float) nCeros/nPaquetesActivos;
    *unos = (float) nUnos/nPaquetesActivos;
}
void RevisaOrden (int *quienVaDespues, int *dondeEsta, int *primero){
    int iNodo, visitamos;
    for(iNodo = 0; iNodo < nNodos; iNodo++){
        for(visitamos = primero[iNodo]; visitamos != -1;visitamos =
            → quienVaDespues[visitamos]){
            if((dondeEsta[visitamos] != dondeEsta[quienVaDespues[
                → visitamos]])&& (quienVaDespues[visitamos]!=-1)){
                printf("Visitamos el paquete %d y esta en el nodo %d.
                    → Despues va %d\n", visitamos, dondeEsta[
                    → visitamos], quienVaDespues[visitamos]);
                printf("::::::::::::::::::VAYA VAYA
                    → ::::::::::::::::::\n");
            }
        }
    }
}
```

```
void ElMejorPayoff(float *mejoresPayOff, int R, int *conQuienesCada,
   → int *conCuantosCada, int *elMejorPaquete, int *dondeEsta, int *
   → tieneUtiles, int *cuantosEnMiCola){
    int *cualesCuentan = (int*) malloc(nNodos*sizeof(int));
    int *etiquetas = (int*) malloc(nNodos*sizeof(int));
    int iNodo, visitamos, visitamosFinal, colocar, veces, i, j,
       → hastaDonde;
    float *mejoresPayOffViejos  = (float*) malloc(nNodos*sizeof(float
       → ));
    int *elMejorPaqueteViejo = (int*) malloc (nNodos*sizeof(int));
    int *tieneUtilesViejo = (int*) malloc (nNodos*sizeof(int));


    for(iNodo = 0; iNodo < nNodos; iNodo++){
        mejoresPayOffViejos[iNodo] = mejoresPayOff[iNodo];
        tieneUtilesViejo[iNodo] = tieneUtiles[iNodo];

        elMejorPaqueteViejo[iNodo] = elMejorPaquete[iNodo];

        etiquetas[iNodo] = -1;
    }

    for(iNodo = 0; iNodo < nNodos; iNodo++){
        if(cuantosEnMiCola[iNodo] != 0){
            for(i = 0; i < conCuantosCada[iNodo]; i++){
                if((mejoresPayOffViejos[conQuienesCada[iNodo*
                   → maximoGrado+i]] > mejoresPayOffViejos[iNodo])&&
                   →  tieneUtilesViejo[conQuienesCada[iNodo*
                   → maximoGrado+i]] == 1){
                        mejoresPayOff[iNodo] = mejoresPayOffViejos[
                           → conQuienesCada[iNodo*maximoGrado+i]];
                        elMejorPaquete[iNodo] = elMejorPaqueteViejo[
                           → conQuienesCada[iNodo*maximoGrado+i]];
                        tieneUtiles[iNodo] = 1;
                }
            }
        }
    }
        free(etiquetas); free(cualesCuentan); free(
           → mejoresPayOffViejos); free(elMejorPaqueteViejo);
}
```

A.5. Mejor desde Actualización

```
#include <stdio.h>
#include <math.h>
```

```
#include <time.h>
#include <stdlib.h>
#include <malloc.h>

//////////////////////

//#define EMPIEZAEN1
#define nCaracteres 256
int tiempoPrevio = 1500;
#define tiempoEjecucion 1000
#define creadosCadaMax 100
//#define ER
#define SF
//#define creadosCada 1

//////////////////////


//Numero de nodos de nuestra red
int nNodos, nLinks;
int maximoGrado;
int creadosCada;
float p;
int tiempo, prorroga=0;
float h=0.6, epsilon = 0.05;
int ventanaCambios = 3;
int flagParaFicheros;

int nPaquetesActivos = 0, nPaquetesActivosAntes = 0;

#include "BiblioShort.h"

int main(){

#ifdef ER
char nombreRed[nCaracteres]="red_ER_1.txt";
#endif
#ifdef SF
char nombreRed[nCaracteres]="red_SF_1.txt";
#endif // SF
    //En primer lugar leemos la red
    //Calculamos el grado maximo de la red (Esto nos dara info sobre
        → el tamano de algunos arrays
    printf("Vamos a calcular el grado maximo\n");
    MaximoGrado(nombreRed);
    printf("EL nombre es: %s\n", nombreRed);
```

```
    //Creamos los arrays que nos daran informacion sobre los nodos
    int *conCuantosCada=(int*) malloc(nNodos*sizeof(int));
    int *conQuienesCada=(int*) malloc(nNodos*maximoGrado*sizeof(int))
       → ;
    float *mejoresPayOff = (float*) malloc(nNodos*sizeof(float));
    CreaMatrices(nombreRed, conCuantosCada, conQuienesCada);
    int *distancias= (int*) malloc(nNodos*nNodos*sizeof(int));
    //Creamos una matriz con las distancias de cualquier nodo a
       → cualquier otro por el camino mas corto
    DistanciasMinimas(conCuantosCada, conQuienesCada, distancias);
    //Arrays necesarios para paquetes (nos dan info de cada paquete)
    int *dondeEsta;
    int *dondeVa;
    int *quienVaDespues;
    int *cuantasVentanas;
    float *payOff;
    int *tipoPaquete; //tipo 1 es aware, tipo 0 shortest path
    int *distanciasAnteriores;
    int *distanciasActuales;
    int *tiempoUpdate;
    int *dondeEstaLuego;
    int *dondeVaLuego;
    int *quienVaDespuesLuego;
    int *cuantasVentanasLuego;
    float *payOffLuego;
    int *tipoPaqueteLuego;
    int *distanciasAnterioresLuego;
    int *tiempoUpdateLuego;
    char bufferP[50], bufferH[50], bufferTipo[50];
    FILE *ficheroP;
    //Arrays que nos dan informacion sobre cada nodo
    int *cuantosEnMiCola = (int*) malloc(nNodos*sizeof(int));
    int *cuantosEnMiColaLuego = (int*) malloc(nNodos*sizeof(int));
//     int *puntuacionVecinos = (int*) malloc(maximoGrado*sizeof(int))
   → ;
    int *primero = (int*) malloc(nNodos*sizeof(int));
    int *tieneUtiles = (int*) malloc(nNodos*sizeof(int));
    int *elMejorPaquete = (int*)malloc(nNodos*sizeof(int));
    int *ultimo = (int*) malloc(nNodos*sizeof(int));
    int *primeroLuego = (int*) malloc(nNodos*sizeof(int));
    int *ultimoLuego = (int*) malloc(nNodos*sizeof(int));
    int *paquetes; int ventana = 20, iVentana=0, nVentanas=
       → tiempoEjecucion/ventana;float rho, diferenciaPaquetes;
    int i, iNodoAux, jNodoAux, iPaqueteAux;
    float unos, ceros;
    int iCreado;
```

```
    FILE *ficheroH;
    FILE *ficheroTipo;
//Empezamos con elprograma en si
flagParaFicheros =1;
for(h= 0.9;h>0.59;h=h-0.1){
    #ifdef ER
    sprintf(bufferH, "ERCongestionH%.2f.txt", h);
    #endif // ER
    #ifdef SF
    sprintf(bufferH, "SF%dCongestionNaiveC%dH%.2f.txt",
        → flagParaFicheros, ventanaCambios, h);
    #endif // SF
    ficheroH = fopen (bufferH, "w");
    for(p=0.001;p<=0.06; p=p+0.002, iVentana=0){
        #ifdef ER
        sprintf(bufferP, "ER%dPaquetesTiempoH%.2fP%.2f.txt",
            → flagParaFicheros, h,p);
        sprintf(bufferTipo, "ERTipoTiempoH%.2fP%.2f.txt", h, p);
        #endif // ER
        #ifdef SF
        sprintf(bufferP, "SF%dPaquetesTiempoNaiveC%dH%.2fP%.3f.
            → txt",flagParaFicheros,ventanaCambios, h,p );
        sprintf(bufferTipo, "SF%dTipoTiempoNaiveC%dH%.2fP%.3f.txt
            → ",flagParaFicheros, ventanaCambios,h, p );
        #endif // SF
        ficheroP=fopen(bufferP, "w");
        ficheroTipo = fopen(bufferTipo, "w");
        creadosCada=(int)(p*nNodos);
        InicializaNodos(cuantosEnMiCola, primero, ultimo);
        dondeEsta = (int*) malloc((creadosCada)*sizeof(int));
        dondeVa = (int*) malloc((creadosCada)*sizeof(int));
        quienVaDespues = (int*) malloc((creadosCada)*sizeof(int))
            → ;
        paquetes = (int*) malloc(nVentanas*sizeof(int));
        cuantasVentanas = (int*) malloc((creadosCada)*sizeof(int)
            → );
        payOff = (float*) malloc((creadosCada)*sizeof(float));
        tipoPaquete = (int*) malloc((creadosCada)*sizeof(int));
        distanciasAnteriores = (int*) malloc((creadosCada)*sizeof
            → (int));
        tiempoUpdate = (int*) malloc((creadosCada)*sizeof(int));
        tiempoPrevio = 1000;
        if(p<0.052){
            tiempoPrevio = 3000;
        }
        for (tiempo = 0; tiempo < (tiempoPrevio+tiempoEjecucion+
```

```
→ prorroga); tiempo++){
 for(iCreado = 0; iCreado < creadosCada; iCreado++){
      CreaPaquete(dondeEsta, dondeVa, cuantosEnMiCola,
          → ultimo, quienVaDespues, primero,
          → tipoPaquete, cuantasVentanas, distancias,
          → distanciasAnteriores, tiempoUpdate);
 }
 nPaquetesActivosAntes=nPaquetesActivos;
 dondeEstaLuego = (int*) malloc((nPaquetesActivos)*
     → sizeof(int));
 dondeVaLuego = (int*) malloc((nPaquetesActivos)*
     → sizeof(int));
 quienVaDespuesLuego = (int*) malloc((nPaquetesActivos
     → )*sizeof(int));
 cuantasVentanasLuego = (int*) malloc((
     → nPaquetesActivos)*sizeof(int));
 payOffLuego = (float*) malloc((nPaquetesActivos)*
     → sizeof(float));
 tipoPaqueteLuego = (int*) malloc((nPaquetesActivos)*
     → sizeof(int));
 distanciasAnterioresLuego = (int*) malloc((
     → nPaquetesActivos)*sizeof(int));
 tiempoUpdateLuego = (int*) malloc((nPaquetesActivos)*
     → sizeof(int));
 IgualaArrays(cuantasVentanas, cuantasVentanasLuego,
     → payOff, payOffLuego, tipoPaquete,
     → tipoPaqueteLuego, distanciasAnteriores,
     → distanciasAnterioresLuego, tiempoUpdate,
     → tiempoUpdateLuego);
 RevisaOrden(quienVaDespues, dondeEsta, primero);
 MuevePrimeros(dondeEsta, dondeVa, conCuantosCada,
     → conQuienesCada, distancias, primero,
     → cuantosEnMiCola, quienVaDespues, ultimo,
     → dondeEstaLuego, quienVaDespuesLuego,
     → dondeVaLuego, primeroLuego, ultimoLuego,
     → cuantosEnMiColaLuego, tipoPaquete);//La info
     → pasa a los LUEGO
 /*Tras mover los primeros, la informacion relevante
     → esta en los arrays con "Luego" al final,
 los otros ademas tendran que cambiar de tamano,
     → reduciendose por los paquetes que llegan y
 aumentandose por los que se crean en cada paso de
     → tiempo.*/
 free(dondeEsta); free(dondeVa); free(quienVaDespues);
     → //Liberamos los NOLUEGO
 free(cuantasVentanas);free(payOff);free(tipoPaquete);
```

```
    → free(distanciasAnteriores); free(tiempoUpdate);
//Volvemos a crear los arrays, copiamos la
    → informacion (aqui muchos paquetes cambian de
    → numero, ya que
//los paquetes que llegan tienen informacion
    → irrelevante y la destruimos
dondeEsta = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
dondeVa = (int*) malloc((nPaquetesActivos+creadosCada
    → )*sizeof(int));
quienVaDespues = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
cuantasVentanas = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
payOff = (float*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(float));
tipoPaquete = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
distanciasAnteriores = (int*) malloc((
    → nPaquetesActivos+creadosCada)*sizeof(int));
tiempoUpdate = (int*) malloc((nPaquetesActivos+
    → creadosCada)*sizeof(int));
CambiaArrays(dondeEstaLuego, dondeEsta, dondeVaLuego,
    →  dondeVa, quienVaDespuesLuego, quienVaDespues,
    → primero, ultimo, cuantosEnMiCola, primeroLuego,
    →  ultimoLuego, cuantosEnMiColaLuego,
    → cuantasVentanas, payOff, tipoPaquete,
    → cuantasVentanasLuego, payOffLuego,
    → tipoPaqueteLuego, distanciasAnteriores,
    → distanciasAnterioresLuego, tiempoUpdate,
    → tiempoUpdateLuego);//La informacion pasa de los
    →  Luego a los NOLUEGO

free(dondeEstaLuego);free(dondeVaLuego);free(
    → quienVaDespuesLuego);
free(cuantasVentanasLuego);free(payOffLuego);free(
    → tipoPaqueteLuego);free(
    → distanciasAnterioresLuego);
free(tiempoUpdateLuego);
if((tiempo>tiempoPrevio)&&((tiempo-tiempoPrevio)%
    → ventana==0)){
  paquetes[iVentana]=nPaquetesActivos;
  iVentana++;
}
if (tiempo%5==0){
  fprintf(ficheroP, "%d\t %d\n", tiempo,
```

```
                    → nPaquetesActivos);
                }
                if(tiempo%ventanaCambios==0&& tiempo !=0){
                    distanciasActuales = (int*) malloc(
                    → nPaquetesActivos*sizeof(int));
                    DistanciasActuales(dondeEsta, dondeVa, distancias
                    → , distanciasActuales);
                    CalculaPayOff(distanciasActuales,
                    → distanciasAnteriores, payOff, mejoresPayOff
                    → , dondeEsta, primero, cuantasVentanas,
                    → tieneUtiles, quienVaDespues, elMejorPaquete
                    → , tiempoUpdate);
                    free(distanciasActuales);
                    CambiaEstrategias(tipoPaquete, mejoresPayOff,
                    → payOff, dondeEsta, tieneUtiles,
                    → elMejorPaquete, tiempoUpdate,
                    → distanciasAnteriores, dondeVa, distancias);
                    ActualizaArrays(cuantasVentanas, tiempoUpdate);
                    CuentaPaquetes(tipoPaquete, &unos, &ceros);
                    fprintf(ficheroTipo, "%d\t %f\t %f\t\n", tiempo,
                    → unos, ceros);
                }
            }
            for(iVentana = 0, rho = 0;iVentana<nVentanas-2;iVentana
            → ++){
                diferenciaPaquetes = (float)(paquetes[iVentana+1]-
                → paquetes[iVentana]);
                rho = rho +(diferenciaPaquetes/(ventana*p*nNodos));
            }
            rho=rho/nVentanas;
            printf("El parametro de orden para %d creados y h= %f
            → vale %f y hay %d paquetes activos\n", creadosCada,h
            → , rho, nPaquetesActivos);
            fprintf(ficheroH, "%f\t %f\n", p, rho);
            nPaquetesActivos=0;
            nPaquetesActivosAntes=0;
            fclose(ficheroP);
            fclose(ficheroTipo);
            free(paquetes);
        }
        fclose(ficheroH);
}
    //Cosas de nodos (no se liberan cada vez)
    free(primero);
    free(ultimo);
    free(tieneUtiles);
```

```
    free(elMejorPaquete);
    free(primeroLuego);
    free(ultimoLuego);
    free(cuantosEnMiCola);
    free(cuantosEnMiColaLuego);
    free(conCuantosCada);
    free(conQuienesCada);
    free(distancias);
    free(mejoresPayOff);


    //Cosas de paquetes, se van liberando
    free(dondeEsta);
    free(dondeVa);
    free(quienVaDespues);
    free(cuantasVentanas);
    free(payOff);
    free(tipoPaquete);
    free(distanciasAnteriores);
    free(tiempoUpdate);

    printf("\nFin\n");
    return 0;
}
```

A.5.1. Librería utilizada

A la librería mostrada en A.1.1 se añaden las funciones:

```
void IgualaArrays(int *cuantasVentanas, int *cuantasVentanasLuego,
    → float *payOff, float *payOffLuego, int *tipoPaquete, int *
    → tipoPaqueteLuego, int *distanciasAnteriores, int *
    → distanciasAnterioresLuego, int *tiempoUpdate, int *
    → tiempoUpdateLuego);
void CalculaPayOff(int *distanciasActuales, int *distanciasAnteriores
    → , float *payOff, float *mejoresPayOff, int *dondeEsta,int *
    → primero, int *cuantasVentanas, int *tieneUtiles, int *
    → quienVaDespues, int *elMejorPaquete, int *tiempoUpdate);
void CambiaEstrategias(int *tipoPaquete, float *mejoresPayoff, float
    → *payOff, int *dondeEsta, int *tieneDisponibles, int *
    → elMejorPaquete, int *tiempoUpdate, int *distanciasAnteriores,
    → int *dondeVa, int *distancias);
void CuentaPaquetes(int *tipoPaquete, float *unos, float *ceros);
void ActualizaArrays(int *cuantasVentanas, int *tiempoUpdate);

void IgualaArrays(int *cuantasVentanas, int *cuantasVentanasLuego,
    → float *payOff, float *payOffLuego, int *tipoPaquete, int *
    → tipoPaqueteLuego, int *distanciasAnteriores, int *
```

```
    → distanciasAnterioresLuego, int *tiempoUpdate, int *
    → tiempoUpdateLuego){
     int iPaquete;
     for(iPaquete = 0; iPaquete< nPaquetesActivos; iPaquete++){
          cuantasVentanasLuego[iPaquete] = cuantasVentanas[iPaquete];
          payOffLuego[iPaquete] = payOff[iPaquete];
          tipoPaqueteLuego[iPaquete] = tipoPaquete[iPaquete];
          distanciasAnterioresLuego[iPaquete] = distanciasAnteriores[
              → iPaquete];
          tiempoUpdateLuego[iPaquete] = tiempoUpdate[iPaquete];
     }
}

void DistanciasActuales(int *dondeEsta, int *dondeVa, int *distancias
    → , int *distanciasActuales){
     int iPaquete;
     for (iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
          distanciasActuales[iPaquete] = distancias[dondeEsta[iPaquete
              → ]*nNodos+dondeVa[iPaquete]];
     }

}

void CalculaPayOff(int *distanciasActuales, int *distanciasAnteriores
    → , float *payOff, float *mejoresPayOff, int *dondeEsta,int *
    → primero, int *cuantasVentanas, int *tieneUtiles, int *
    → quienVaDespues, int *elMejorPaquete, int *tiempoUpdate){
     int iPaquete, iNodo, visitamos, salimos;
     for(iNodo = 0; iNodo<nNodos; iNodo++, visitamos = 0 ){
          visitamos = primero[iNodo];
          if(visitamos != -1){
          }
          tieneUtiles[iNodo] = 0;
          salimos  = 0;
          for(visitamos = primero[iNodo]; visitamos != -1 && salimos ==
              →  0;){
               if(cuantasVentanas[visitamos] != 2){
                    visitamos = quienVaDespues[visitamos];
               }
               else{
                    salimos = 1;
                    tieneUtiles[iNodo] = 1;
                    elMejorPaquete[iNodo] = visitamos;
                    mejoresPayOff[iNodo] = ((float)(distanciasAnteriores[
                        → visitamos]-distanciasActuales[visitamos]))/((
                        → float)(tiempo-tiempoUpdate[visitamos]));
```

```
                }
            }
        }
        for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
            payOff[iPaquete] = ((float)(distanciasAnteriores[iPaquete]-
                → distanciasActuales[iPaquete]))/((float)(tiempo-
                → tiempoUpdate[iPaquete]));
            if(payOff[iPaquete] > mejoresPayOff[dondeEsta[iPaquete]]&&
                → cuantasVentanas[iPaquete] ==2){
                mejoresPayOff[dondeEsta[iPaquete]] = payOff[iPaquete];
                elMejorPaquete[dondeEsta[iPaquete]] = iPaquete;
            }
        }
}

void CambiaEstrategias(int *tipoPaquete, float *mejoresPayoff, float
    → *payOff, int *dondeEsta, int *tieneDisponibles, int *
    → elMejorPaquete, int *tiempoUpdate, int *distanciasAnteriores,
    → int *dondeVa, int *distancias){
    int iPaquete; float fermi; float alea, siONo;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        siONo = (float) tieneDisponibles[dondeEsta[iPaquete]];
        fermi=(float) (1.0/(1.0+exp(((double)(payOff[iPaquete]-
            → mejoresPayoff[dondeEsta[iPaquete]])))));
        if(generaRand()<fermi*siONo){
            if(tipoPaquete[elMejorPaquete[dondeEsta[iPaquete]]]!=
                → tipoPaquete[iPaquete]){
                if(tipoPaquete[elMejorPaquete[dondeEsta[iPaquete
                    → ]]]==1){
                    tipoPaquete[iPaquete] == 1;
                    tiempoUpdate[iPaquete]= tiempo;
                    distanciasAnteriores[iPaquete]= distancias[
                        → dondeEsta[iPaquete]*nNodos+dondeVa[iPaquete
                        → ]];
                }
                else{
                    tipoPaquete[iPaquete] == 0;
                    tiempoUpdate[iPaquete]= tiempo;
                    distanciasAnteriores[iPaquete]= distancias[
                        → dondeEsta[iPaquete]*nNodos+dondeVa[iPaquete
                        → ]];
                }
            }
        }
    }
}
```

```
void ActualizaArrays(int *cuantasVentanas, int *tiempoUpdate){
    int iPaquete;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        if(cuantasVentanas[iPaquete]==1){
            tiempoUpdate[iPaquete] = tiempo;
        }
        if(cuantasVentanas[iPaquete]<2){
            cuantasVentanas[iPaquete]++;
        }
    }
}

void CuentaPaquetes(int *tipoPaquete, float *unos, float *ceros){
    int nCeros = 0, nUnos = 0;
    int iPaquete;
    for(iPaquete = 0; iPaquete < nPaquetesActivos; iPaquete++){
        if(tipoPaquete[iPaquete]==0){
            nCeros++;
        }
        if(tipoPaquete[iPaquete] == 1){
            nUnos++;
        }
        if(tipoPaquete[iPaquete] != 0&& tipoPaquete[iPaquete]!=1){
            printf("::::::::::::::::");
        }
    }
    *ceros = (float) nCeros/nPaquetesActivos;
    *unos = (float) nUnos/nPaquetesActivos;
}

void RevisaOrden (int *quienVaDespues, int *dondeEsta, int *primero){
    int iNodo, visitamos;
    for(iNodo = 0; iNodo < nNodos; iNodo++){
        for(visitamos = primero[iNodo]; visitamos != -1;visitamos =
            → quienVaDespues[visitamos]){
            if((dondeEsta[visitamos] != dondeEsta[quienVaDespues[
                → visitamos]])&& (quienVaDespues[visitamos]!=-1)){
                printf("::::::::::::::::::VAYA␣VAYA
                    → ::::::::::::::::::\n");
            }
        }
    }
}
```