Universitat de Girona
Escola Politècnica Superior

Projecte/Treball Final de Carrera

Estudi: Eng. Tècn. Informàtica de Sistemes. Pla 2001

Títol:

Incorporació de mesures d'ample de banda disponible a l'algorisme d'encaminament AntNet-Qos

Document: Memòria

Alumne: Xavier Vallejo López

Director/Tutor: Liliana María Carrillo Flórez
Departament: Electrònica, Informàtica i Automàtica
Àrea: Arquitectura i Tecnologia de Computadors

Convocatòria (mes/any): 09/06

Agraïments:

A la meva tutora Liliana María Carrillo Flórez, per la plena dedicació a l'hora d'ajudar-me a realitzar el meu projecte.

A en Carles Guadall Blancafort, pels seus consells que m'han ajudat a entendre i millorar aquest algorisme.

Al grup BCDS (Broadband Communications and Distributed Systems) de la Universitat de Girona per l'excel·lent ambient de treball i la disposició de les seves instal·lacions.

A la meva família, pel seu suport i ànims durant tota la durada del projecte.

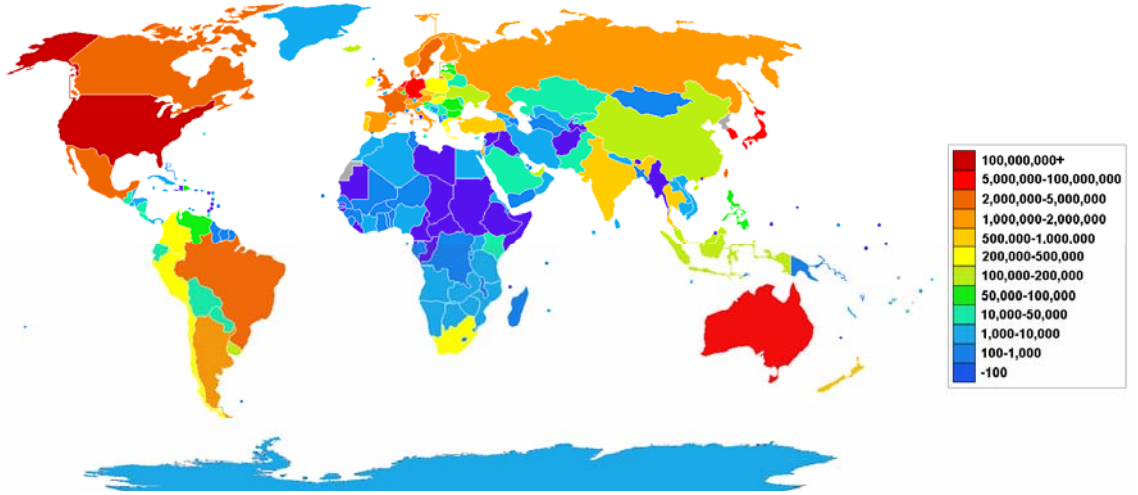
A tots ells, moltes gràcies.

0. ÍNDEX

1. Introducció

La comunicació sempre ha estat present en tota la història de l'home. Gràcies a aquesta, els humans ens podem organitzar en societats ja que ens permet intercanviar informació entre varis individus, i d'aquesta manera organitzar-nos. Les comunicacions en el món informàtic es van revolucionant amb l'arribada d'Internet, ja que permet una gran integració amb la majoria de persones de tot el món. No només en l'ús d'usuari (obtenir informació navegant via web, intercanviar missatges via correu electrònic, i moltes altres coses), sinó que també tot tipus d'empreses han pogut reduir costos d'una manera espectacular. En resum, Internet ha estat una revolució en tots els aspectes del món modern.

Si ens endinsem en aspectes més tècnics, trobem que Internet també l'anomenem "la xarxa de xarxes". És a dir, Internet està formada per milions de xarxes, que aquestes s'agrupen en subxarxes. Una xarxa és un conjunt d'estacions (estació ens referim tant a un ordinador, com un altre dispositiu com telèfons). La informació viatja entre estacions, i aquestes estacions estan connectades a través d'uns enllaços. Ens podem imaginar com si fossin autopistes els enllaços, i estacions les sortides d'aquestes.



Imatge 1: Llocs d'Internet en el món per països

Aquesta informació (dita trànsit), ha de ser gestionada per uns dispositius. En Internet, aquests dispositius es diuen enrutadors (encaminadors, routers). Segons la destinació a la que es vol arribar, l'enrutador decideix per quin enllaç ha d'enviar. El problema resideix en que per Internet en circulen multituds alhora, i per tant no és tant senzill com un es pensa dirigir-ho. Cal observar aspectes com l'estat dels enllaços, i la seva velocitat. A més a més, els paquets es poden perdre, ja que pot haver un

sobrevolum de dades per anar a un destí, i no hi hagi espai per emmagatzemar-les (cues), o que la informació no pugui arribar al destí en un temps raonable (time to live). Es pot donar el cas que un router falli, o un enllaç, i s'hagin de buscar vies alternatives per arribar al destí. Finalment, segons com sigui la nostra informació, s'ha de donar prioritat a diversos aspectes. Un correu electrònic ha d'arribar íntegre al destí, ja que sinó seria inintel·ligible. En canvi, no cal que l'enviament sigui instantani. Una videoconferència no ha de tindre retards ni variació del retard (jitter), ja que llavors no es podria seguir la imatge amb la veu, en canvi, en la videoconferència es pot permetre petites pèrdues de qualitat o veu, ja que no afecta molt al seu ús.

En el grup d'investigació BCDS de la Universitat de Girona, s'ha estat investigant i experimentant un algorisme per gestionar diferents tipus de trànsit a l'estil d'una xarxa d'Internet. A través d'un simulador de xarxes (OMNeT++) el qual realitzem les proves, aquest algorisme ha de respondre ràpid a les possibles variacions de les condicions de la xarxa, i el propi algorisme s'ha d'autoajustar a les necessitats de la xarxa.

1.1. Objectius

Actualment, la xarxa d'Internet està formada per encaminadors que segueixen un patró estàtic. Per estàtic ens referim a que la informació que rep un router sempre segueix un mateix camí de sortida segons les taules d'encaminament. Aquestes taules han estat tradicionalment fixes, i per tant per anar a una determinada estació, el router sempre envia la informació pel mateix enllaç. Aquesta metodologia és fàcil d'implementar, però té el problema de que no es tenen en compte les condicions de la xarxa.

Per això, hi ha la necessitat de desenvolupar una nova metodologia la qual permeti un major dinamisme a l'hora de gestionar, aconseguint-ne una major eficiència. L'algorisme AntNet [1] i altres derivats (el qual es basa aquest projecte) implementen uns agents de control (paquets anomenats formigues) que analitzen la xarxa, i actuen sobre els encaminadors variant les rutes del trànsit si és necessari.

Aquest projecte final de carrera pretén investigar i experimentar una nova línia de desenvolupament d'algorismes dinàmics. A partir de l'algorisme AntNet-QoS [2] hem incorporat noves mesures a l'algorisme (mesura de l'amplada de banda disponible i jitter), les quals combinant amb la mesura de retard ja feta servir, ens permet adaptar-nos millor a les condicions actuals del trànsit en la xarxa i als requeriments específics de qualitat (QoS) per part del trànsit.

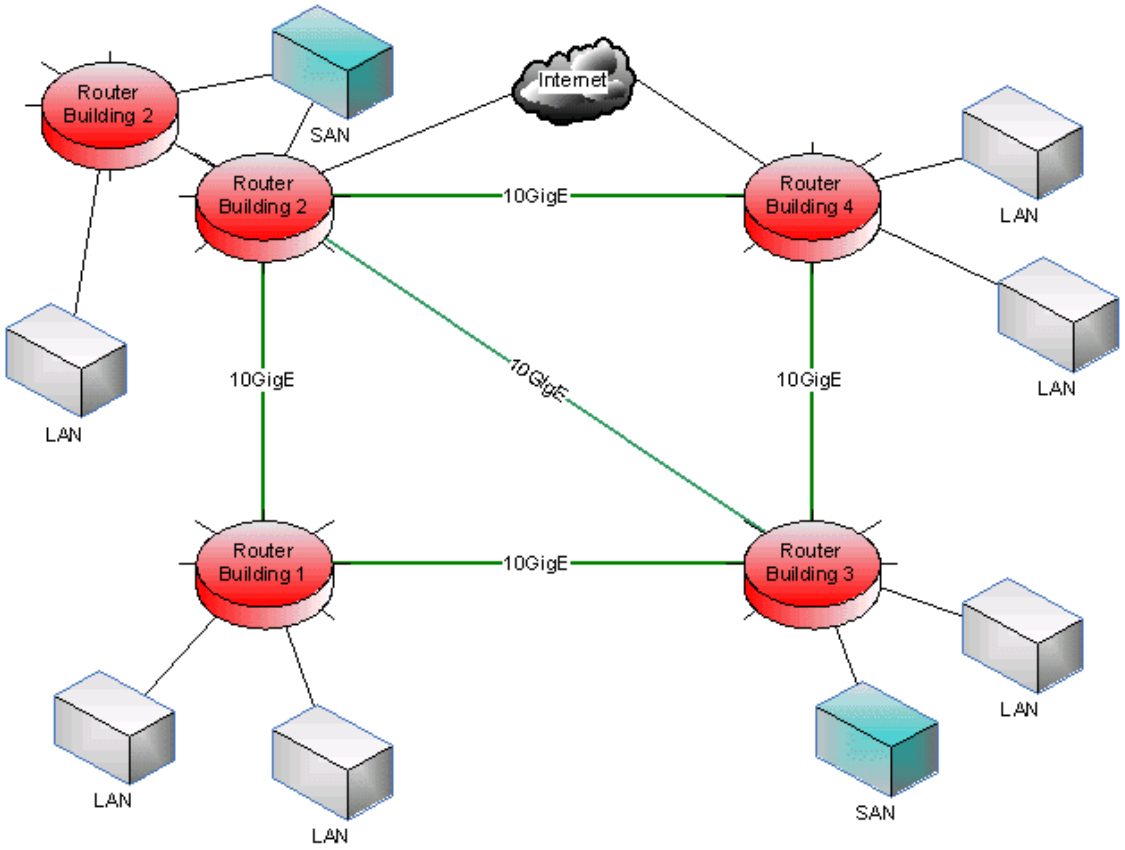
1.2. Abreviatures

AF	Assured Forwarding Enviament Assegurat
BCDS	Broadband Communications and Distributed Systems Comunicacions de Banda Amplada i Sistemes Distribuïts
BE	Best Effort
BW	Bandwidth Amplada de Banda
CoA	Class of Ants Classes de formigues
CoS	Class of Service Classes de Servei
CBR	Constant Bit Rate
CPU	Central Processing Unit Unitat de Processat Central
DiffServ	Differentiated Services Serveis Diferenciats
EF	Expedited Forwarding Enviament Expedit
GUI	Graphical User Interface Interfície d'Usuari Gràfica
IETF	Internet Engineering Task Force
IntServ	Integrated Services Serveis Integrats
Kbps	Kilobit per second Kilobits per segon
LAN	Local Area Network Xarxa d'Àrea Local
MAC	Media Access Control Control d'Accés al Medi
MPI	Message Passing Interface Interfície de Pas de Missatges
Mbps	Megabit per second Megabits per segon
OSI	Open System Interconnection Interconnexió de Sistemes Oberts
QoS	Quality of Service Qualitat de Servei
SWP	Shortest Widest Path
TCP	Transmission Control Protocol Protocol de Control de Transmissió
TTL	Time To Live Temps de Vida
VoIP	Voice over IP Veu a través de IP
WSP	Widest Shortest Path

2. Teoria de Xarxes

Una xarxa és un conjunt d'estacions (ordinadors, telèfons, agendes personals...). Els paquets viatgen entre les estacions connectades a través dels enllaços. Un node el definim com un punt de connexió de la xarxa. Com a exemple, els routers són nodes. Un exemple senzill és el d'una xarxa local en una casa (LAN), en que tots els ordinadors es connecten a un router i aquest dóna un únic accés a Internet (el router seria un node).

Els missatges són un, o varis paquets que en conjunt són la informació que un sol·licita. D'exemples trobem molts: una part d'una pàgina web, d'un fitxer, d'un correu... Els missatges poden viatjar d'una estació origen a una estació destí (connexió end-to-end o punt a punt), o d'una estació origen a vàries estacions destí (broadcast o multipunt).



Imatge 2: Una xarxa simple entre routers (pot ser una xarxa MAN)

2.1. Tipus de Xarxes

Una classificació de les xarxes és entre les xarxes de commutació i les xarxes de difusió.

En les **xarxes de commutació**, el missatge no el reben totes les estacions, sinó només la estació destí. No hi ha un camí únic, i per poder fer l'enviament, cal definir el camí per on s'ha d'enviar. Aquesta decisió es coneix com a encaminament. Un exemple clar és la telefonia fixa, la qual per realitzar una trucada, cal establir un canal entre l'origen i el destí.

Tècniques en les xarxes de commutació n'hi ha dos: commutació de circuits i commutació de paquets.

La commutació de circuits, cal establir un canal entre emissor i receptor. Un cop s'estableixen els recursos, llavors es fa la transmissió i un cop s'acaba la transmissió, s'alliberen els recursos i el canal.

La commutació de paquets, els missatges es divideixen en petits fragments, i a dins d'ells s'afegeix informació de control. Els routers poden millorar l'eficiència al poder combinar varis missatges alhora. Per enviar els paquets, també existeixen dues tècniques.

La tècnica de Datagrama tracta els paquets independentment, i per tant és possible que els paquets arribin al destí desordenadament, degut a lentituds en cues, o que es perdin (en tal cas caldrà demanar retransmissions). La tècnica de Circuit Virtual, abans es reserva una ruta i els paquets viatgen amb un identificador de la ruta. Les rutes, a l'igual que la commutació de circuits, es creen i es destrueixen.

Les **xarxes de difusió** fan servir un medi compartit per intercanviar-se la informació. És a dir, un missatge enviat per una estació el reben totes les estacions. Totes les estacions comproven si els paquets van destinats a elles o no, en el cas que no sigui per ell, no agafa aquell paquet. Al fer servir un únic medi, els missatges han de compartir un camí comú, i s'organitzen amb tècniques d'accés múltiple. En aquest cas no es fa servir ni commutació ni encaminament. Un exemple clar de xarxes de difusió són les xarxes Ethernet.

Tècniques d'accés múltiple hi ha vàries, de més simples a més modernes: Aloha, CSMA i CSMA/CD. Aquesta última és la feta servir en Ethernet, i consisteix en escoltar el canal abans d'enviar, i mentre s'envia, escoltar per si es produeixen

col·lisions (una col·lisió es produeix quan dues estacions envien alhora pel mateix canal).

Altres classificacions de les xarxes corresponen a la seva topologia física (en anell, token ring...) o segons la seva extensió (LAN,MAN,WAN per definir respectivament local, metropolitana o extensa).

2.2. Encaminament

L'encaminament és una de les peces principals d'una xarxa. En gran part, hi depèn el rendiment d'aquesta. Consisteix en que cada node fa servir una taula pròpia d'encaminament, la qual indicarà segons el trànsit que li entra cap a quin enllaç dirigir-lo.

Els protocols d'enrutament fan servir mesures per determinar el camí a seguir. Algunes mesures són l'amplada de banda disponible, el delay (retard) o el jitter (variació del retard). Segons la importància que li donem, aquests valors ajuden a determinar la ruta.

Un algorisme d'encaminament ha d'aconseguir la màxima eficiència de la xarxa, i alhora aconseguir els mínims costs possibles. A continuació expliquem els objectius d'un algorisme d'encaminament:

- **Simplicitat**: Quan més simple sigui, menys recursos caldran per gestionar-lo, i per conseqüent, utilitzarà menys overhead (cicles de CPU en que el router processa les dades).
- **Estabilitat**: L'algorisme ha de poder respondre el millor possible a canvis sobtats en l'estat de la xarxa. Per exemple, a caigudes d'enllaços o cues plenes, sense que la xarxa caigui o s'interrompin tasques.
- **Convergència**: Degut als possibles canvis en la xarxa, l'algorisme no s'ha de quedar estancat, sinó que ha d'actuar ràpidament sobre tots els nodes de la xarxa, avisant dels canvis i que tots es posin d'acord.
- **Optimització**: Tal i com hem comentat anteriorment, l'algorisme ha d'aconseguir utilitzar tots els recursos de la xarxa el millor possible, i intentant que hi hagi equitat entre els possibles routers.

3. QoS (Quality of Service)

A mesura que anem evolucionant en els aspectes d'una xarxa, ens trobem que molts cops, ens interessa donar més prioritat a un tipus de trànsit que un altre perquè els seus requeriments així ens ho demanen. Per aquesta raó, les tècniques per realitzar el control s'anomenen Qualitat de Servei. Inicialment, Internet no estava pensat per gestionar diferents tipus de trànsit. Encara avui en dia, hi ha forces xarxes que no suporten QoS (tenen la mateixa prioritat per tots els usuaris i trànsit) i això pot ocasionar problemes si no es gestiona correctament.

No totes les aplicacions tenen les mateixes necessitats. Per posar un bon exemple, en una videoconferència cal que les dades viatgin ràpidament entre l'origen i el destí. Si la imatge ens arribés amb retràs, la conversa no seria fluida i per tant, no podríem seguir-la. En canvi, poden haver petits errors durant la transmissió ja que no és imprescindible (sorolls,...). Una aplicació de correu electrònic, ha d'enviar els correus sense cap error, ja que llavors hi ha el perill que el missatge sigui inintel·ligible, però, no és necessari que hi arribi immediatament al destí, i pot esperar.

3.1. Definicions

Delay (retard): En general, és la demora que hi ha entre la emissió i recepció. El retard total el formen un subconjunt de retards:

- **Temps de propagació**: És el temps que triga una senyal en viatjar una distància determinada en un medi específic (par telefònic, fibra òptica, ones de ràdio...)
- **Temps de processat**: El temps que tarda en processar un paquet en un node d'una xarxa.
- **Temps de transmissió**: És el temps que tarda en transmetre un paquet en un determinat bitrate.

Bandwidth: En aspecte tècnic, l'amplada de banda és la diferència entre la freqüència més alta i la més baixa disponibles per senyals de xarxes. Nosaltres aplicarem la definició com la quantitat de dades que es pot transmetre en un espai fixat de temps (velocitat d'arribada).

Jitter: Són petites variacions en temps o fase d'un senyal transmès que provoquen errors o pèrdues de sincronització.

Time To Live: Els paquets porten un temps de vida, per si no poden arribar al destí, s'eliminin i no formin congestions a la xarxa.

3.2. Classes de QoS

Com hem vist, les aplicacions necessiten diferents tipus de serveis d'acord a les seves necessitats. El trànsit el classifiquem en diferents classes:

- **EF**: És la classe en que li donem el trànsit més urgent i crític. Cal que tingui baix jitter, baix delay i petites/mitjanes pèrdues de paquets. La seva missió és transportar trànsit de temps real (videoconferències, veu per ip (VoIP), etc).
- **AF**: Dintre de AF en podem distingir vàries classes, de més qualitat a menys (AF1, AF2,...). En AF pot pujar el jitter o delay, però han d'haver poques pèrdues de paquets. Com a exemples serien aplicacions com la web, o el ftp.
- **BE**: És la classe en que s'ajunta el trànsit que no requereix de Qualitat de Servei. El correu electrònic és un bon exemple.

3.3. Arquitectures de QoS

En Qualitat de Servei s'han establert dos models definits per la IETF (és l'organisme més important en definir estàndards).

- **IntServ (Serveis Integrats)**: Aquesta arquitectura consisteix en que els routers donaran els recursos necessaris segons les demandes de les aplicacions. Per cada aplicació que demani recursos, caldrà assignar-los individualment. Aquest model no és eficaç per grans xarxes on hi ha molt de trànsit, ja que suposa un volum excessiu de treball pels routers.
- **DiffServ (Serveis Diferenciats)**: En aquest model tot el trànsit es classifica segons les classes ja explicades. Per cada classe de servei, el trànsit es gestiona de la mateixa manera. És a dir, per exemple: el trànsit del mateix tipus passa per les mateixes cues però si és d'un altre tipus podria experimentar una probabilitat de descart més alta. DiffServ funciona en xarxes com Internet, ja que els routers no s'han de preocupar en reservar canals ni recursos, simplement han de fer la feina d'encaminar els paquets.

4. El simulador OMNeT++

4.1. Introducció

A l'hora de realitzar el projecte, un dels primers passos que he hagut d'aprendre és a manejar el simulador amb el que he treballat. És evident que intentar implementar un algorisme experimental en una xarxa de veritat, té un cost molt alt, i a més seria molt més complicat de posar-ho en pràctica. Per això, els simuladors de xarxes ens permeten dissenyar la nostra xarxa segons les nostres necessitats, i amb un cost pràcticament de zero.

Existeixen varis simuladors en Internet, tant de versions gratuïtes com de pagament. Ens citarem tres simuladors diferents amb les seves avantatges e inconvenients:

- **The Network Simulator – ns2 [3]:** Ns-2 és un simulador d'events discrets orientat a la recerca en xarxes. Dóna suport a la simulació dels protocols TCP, enroutament i multicast tant en xarxes amb cables com sense. Està escrit en llenguatge de programació C++ i fa servir les llibreries gràfiques Tcl. Per funcionar-ho cal disposar d'un entorn UNIX (Linux, FreeBSD, OS X, i fins i tot amb Windows a través de Cygwin). L'aplicació és gratuïta (de codi lliure, llicència GNU GPL), i un dels seus usos més popular és en l'àmbit educacional.
- **QualNet [4]:** QualNet és un altre bon simulador de prestigi, en principi més complert que ns2 i OMNeT++ al incloure molts més protocols i llibreries. El principal problema és que no és gratuït en cap versió. En el nostre projecte, no s'ajustava a les nostres necessitats, ja que el codi de partida en que hem treballat està desenvolupat amb el simulador OMNeT++.
- **OMNeT++ [5]:** OMNeT++ és un simulador basat en components, modular i de codi obert. Es pot utilitzar des de comandes, o amb un entorn gràfic fent servir les llibreries Tcl/Tk. Encara que no disposa de la qualitat de Qualnet en nombre de models de simulació, aquest simulador disposa d'un gran suport amb una forta comunitat d'usuaris. OMNeT++ és gratuït en l'ús acadèmic, però si es vol fer servir en àmbit comercial, llavors s'ha de pagar una llicència A l'igual que Ns-2, cal disposar d'un entorn UNIX pel seu funcionament.

4.2. OMNeT++

El simulador OMNeT++ ens aporta un entorn de simulació **d'events discrets**. Això significa que no només pot executar simulacions de xarxes d'ordinadors, encara que és el seu ús principal. Podem trobar simulacions de xarxes TCP/IP, protocols com Ethernet o Token Ring, dissenys de xarxes ad-hoc e infraestructures, etc.

Com hem esmentat abans, OMNeT++ fa servir una **estructura modular**. Els mòduls es programen en llenguatge C++, mentre que per unir-los entre sí i fer un model, fem servir un llenguatge propi d'alt nivell: NED. L'avantatge de la programació modular és que podem reutilitzar codi per a altres mòduls i/o projectes.

Podem classificar els mòduls en simples i compostos. Els mòduls compostos són format per conjunts de mòduls simples, o d'altres compostos. En els mòduls simples és on programem el nostre codi.

Una punt fort d'aquest simulador resideix en que podem fer servir un potent entorn gràfic per mostrar les nostres simulacions. És bastant intuïtiu i ens pot servir per saber com està la nostra xarxa en qualsevol moment determinat. Per contra, les simulacions fetes amb entorn gràfic són més lentes que si les executem des de comandes.

Al finalitzar les nostres simulacions, si hem definit un mòdul d'estadístiques, OMNeT++ s'haurà encarregar de guardar els valors de la simulació en un o varis fitxers. Amb les utilitats que porta el simulador, ens és capaç de mostrar gràfiques molt completes dels nostres resultats.

Finalment, podem destacar que OMNeT++ suporta execucions amb paral·lel. Per això, és necessari que hi tinguem configurat les llibreries MPI. Els nostres programes no s'han de programar per fer una execució paral·lela. OMNeT++ té tècniques que permeten que es pugui realitzar una paral·lelització.

Com ens indica el seu autor András Varga, les principals característiques del programa són:

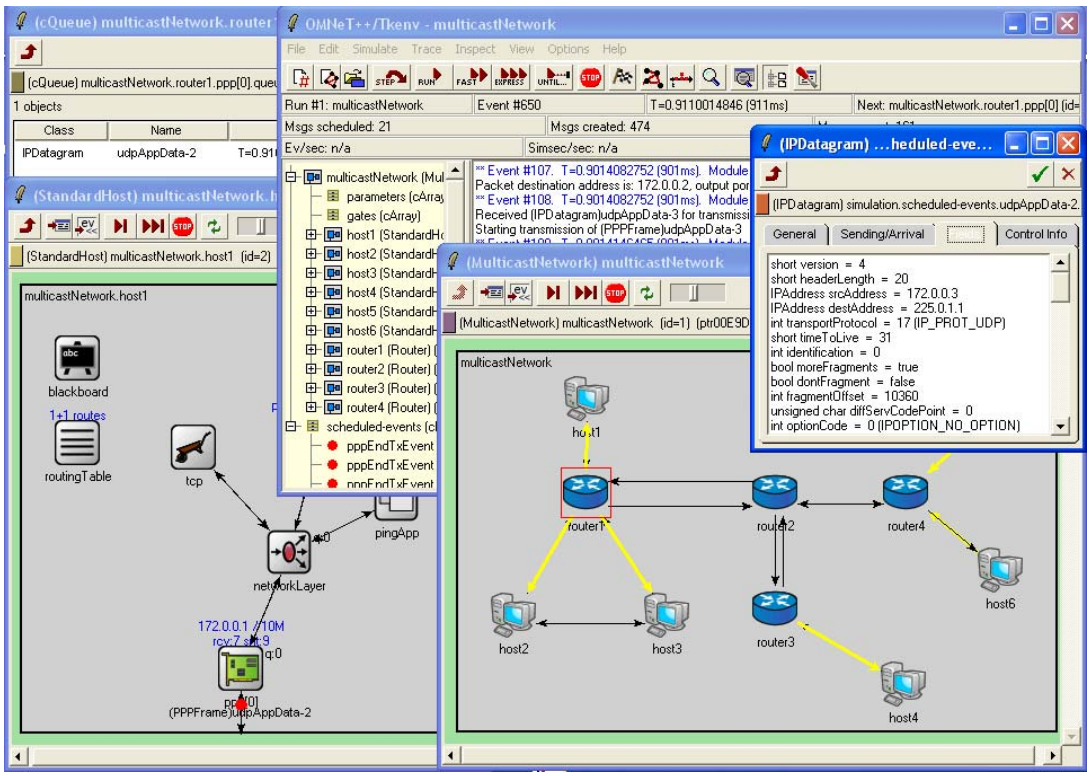
- Llibreria del nucli de simulació
- Compilador pel llenguatge NED (nedc).
- GUI (Interfície gràfica) per l'execució de la simulació, crea un executable (Tkenv)
- Interfície via comandes per l'execució de la simulació (Cmdenv)

- Eina gràfica de vectors de sortida (Plove)
- Eina gràfica per visualitzar resultats escalars (Scalars)
- Utilitats d'ajuda al desenvolupament:
    - Generador de llavors de nombres aleatoris (seedtool)
    - Creació de makefiles automàtic (opp_makemake)
    - Generació de documentació del model (opp_nedtool)
    - Editor gràfic de fitxers NED (gned)
    - Etcètera ...
- Documentació, exemples...

La documentació que s'inclou amb el simulador és molt completa i molt útil, especialment el manual d'usuari [6], ja que repassa tots els aspectes del simulador, i és una gran eina a l'hora d'aprendre a programar amb ell.

A més a més, gràcies al gran suport que rep aquest simulador, existeix una llista de correu en la pàgina oficial en que es pot consultar tot tipus de preguntes i respostes, en les quals hi apareix el propi autor del programa.

Les utilitats del simulador són una gran eina de treball, i d'una qualitat molt alta. Realitzen la seva funció perfectament, integrant-se fàcilment amb l'entorn, i són senzilles de fer servir.



Imatge 3: El simulador OMNeT++ en l'entorn Windows

4.2.1. Instal·lació d'OMNeT++

OMNeT++ és una aplicació que s'ha d'executar en un entorn UNIX. Segons el sistema operatiu en que estem treballant, triarem una versió o una altra:

- Per sistemes UNIX, ens serveix des de versions de GNU/Linux (Fedora, Ubuntu, etc...), Solaris, i també Mac Os X. En aquest cas ens haurem de baixar el codi font de l'aplicació i compilar-lo (és possible que ens demani varis paquets per poder fer la instal·lació.). També es poden trobar versions compilades per alguna versió de Linux, encara que no estan mantenides pel creador
- Per versions Windows, hi ha dues alternatives. La primera és fer servir un entorn UNIX que funcioni sota Windows, en aquest cas es diu Cygwin. D'aquesta manera disposarem de les comandes típiques d'entorns UNIX en Windows. La segona manera és integrant-ho amb Microsoft Visual C++, encara que les ordres per compilar i executar seran un pèl diferents. En tots dos casos, s'ha de baixar la versió en binari del simulador, e instal·lar-la.

A l'hora de realitzar el projecte, hem escollit la versió UNIX per treballar. OMNeT++ té una llicència Acadèmica (i en el nostre cas el podem fer servir lliurement), i a més a més estem desenvolupant en un entorn de codi lliure i per tant no disposem de cap restricció en quant a llicències privatives.

La versió que hem fet servir ha estat la 3.2, i s'ha desenvolupat en un Fedora Core 4 i també en una Kubuntu 6.06, en tots dos casos amb resultats satisfactoris. En el moment del redactat d'aquest document, l'última versió estable del programa és la 3.2p1.

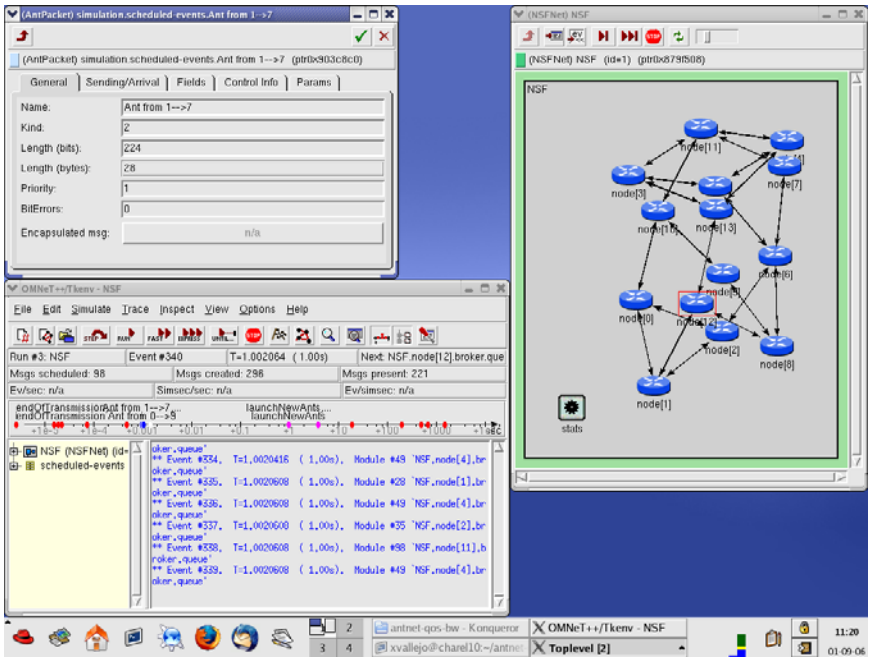
La instal·lació d'OMNeT++ ja s'ha documentat en altres projectes de final de carrera. En aquest cas, citaré una instal·lació en UNIX la qual ja va ser explicada en el projecte en que es va desenvolupar l'algorisme AntNet-QoS [7], però actualitzada.[1]

Instal·lació pas a pas:

1. Descarga de la pàgina del simulador (http://www.omnetpp.org) el codi font de la versió desitjada (en el nostre cas, la versió 3.2).

[1] *És possible que els següents passos no es puguin executar al peu de la lletra. Si més no, les indicacions escrites són una bona guia per poder tindre una instal·lació satisfactòria.*

2. Descompressió de l'arxiu en format tar.gz amb el codi font del simulador.
   Comanda: "tar zxvg omnetpp32.tgz"
3. Moure la carpeta extreta al directori d'inici del nostre usuari (ex:/home/usuari)
4. Configuració de les variables:
   PATH, LD_LIBRARY_PATH i TCL_LIBRARY_PATH.
   Depenent de la versió de UNIX (Linux), ens caldrà modificar el fitxer ocult ".bash_profile" o ".bashrc" en l'intèrpret de comandes bash:
   export PATH=$PATH:~/omnetpp3.2/bin
   export LD_LIBRARY_PATH=~/omnetpp3.2/bin
   export TCL_LIBRARY_PATH=~/omnetpp3.2/lib
5. Ens situem al directori /omnetpp3.2
6. Per comprovar si podem compilar correctament, executem l'ordre: #./configure.
7. Per pantalla anirà comprovant si disposem dels paquets necessaris. Normalment, fan falta els paquets Tcl i Tk en versions binàries i de desenvolupament, i altres paquets per l'entorn gràfic com BLT,... El paquet MPI és opcional si volem executar simulacions paral·leles.
8. Un cop tinguem els paths correctes i no faltin paquets necessaris, el compilem amb: #make all (amb make simple pot donar errors de compilació).
9. Si tot ha funcionat correctament, ja tindrem el programa a punt per utilitzar-lo. Per provar si funciona, dins del directori sample, hi ha un script ./rundemo que ens mostrarà exemples del programa.



Imatge 4: L'algorisme AntNet-QoS en un entorn Linux en funcionament

4.2.2. Configuració i funcionament d'un exemple

En el punt anterior hem parlat de fitxers amb llenguatge NED i amb llenguatge C++. Els fitxers NED defineixen l'estructura de la xarxa (enllaços, portes d'entrades i de sortides, mentre que els fitxers C++ és el codi dels mòduls simples. Els dos tipus els podem escriure amb qualsevol editor de text.

Per poder compilar un programa, un cop disposem dels fitxers codi font, el primer que cal fer és crear un Makefile. Un makefile és un fitxer en que s'inclou totes les comandes necessàries per tal de poder compilar i enllaçar, obtenint l'executable de la nostra aplicació.

En UNIX, creem el nostre makefile amb la comanda:

#opp_makemake

Aquesta commanda admet diversos paràmetres. Els més útils són:

-f: força a generar un nou makefile, encara que ja hi hagi algun.

-h: ens ofereix totes les opcions

-u: ens permet triar si volem executar en un entorn gràfic o comandes (defecte gràfic):

-u Tkenv / -u Cmdenv

-o: podem definir un nom propi en l'executable.

Un cop generat el Makefile, podem començar amb compilar (UNIX):

#make

El primer que realitza és comprovar els fitxers NED amb un compilador propi que porta (ja que és un llenguatge creat per OMNeT++). Un cop compilats, llavors executa el compilador g++ (per defecte, encara que pot ser un altre) i compila els fitxers .cpp. Un cop finalitzat i si ha anat bé, ens haurà generat uns fitxers .o (fitxers objectes, en Windows són .obj). Només queda enllaçar-los i generarà el nostre executable.

Per executar el nostre programa, simplement la cridem pel seu nom. Si hem triat un entorn gràfic, ens engegarà la finestra principal, i altres finestres (dibuix de la xarxa,...)

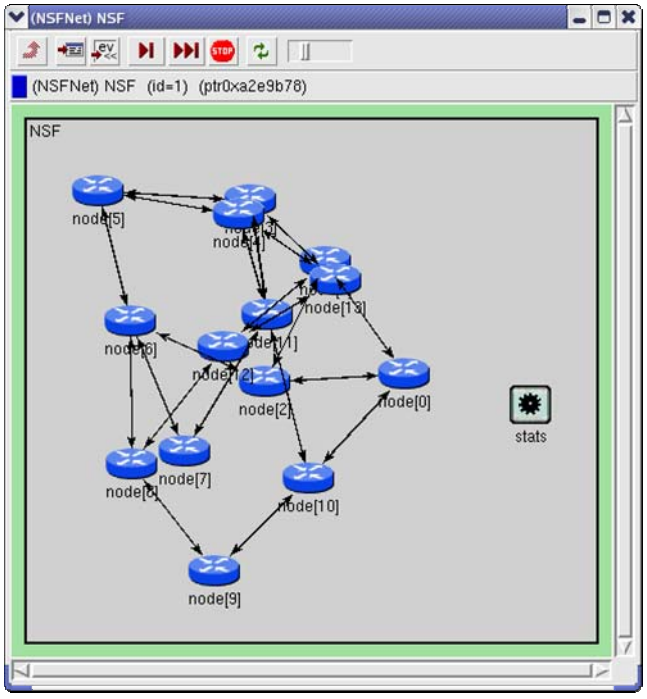
Si ens cal netejar i eliminar l'executable i els fitxers objectes, la comanda *#make clean* realitza aquesta funció.

5. Teoria del protocol d'encaminament AntNet-QoS

5.1. Introducció

L'algorisme AntNet original (i AntNet-QoS), proposa una manera de resoldre l'encaminament amb multi-agents mòbils que actuen com a paquets de control i que coneixem com a paquets formiga per inspirar-se en la intel·ligència de les colònies de formigues al seu ambient natural.

Aquest tipus d'algorismes són una bona opció per donar solucions **ràpides** a l'encaminament en entorns dinàmics per tractar-se d'algorismes **distribuïts** i **adaptatius** que treballen, a l'igual que en les colònies de formigues, deixant un rastre indicant els millors camins a seguir per les altres formigues. En la natura, les formigues busquen menjar i deixen rastres que són les feromones, i en el nostre cas busquen destins i el deixen de rastre és en les taules d'encaminament en els routers.



Imatge 5: Vista general de la xarxa

Les formigues dels algorismes originals calculen els millors camins basant-se en **mesures estimades** del temps que els triguen en arribar al destí. Si es fessin mesures exactes, els algorismes serien molt més senzills però caldrien més recursos i serien més lents a l'hora de trobar solucions. Per això s'ha d'analitzar què es el que es vol aconseguir a l'hora de dissenyar un algorisme.

El comportament tradicional de les formigues consisteix en el següent:

1. Les formigues es creen cada Δt de temps, i s'envien inicialment cap a destins aleatoris per tal d'anar explorant tota la xarxa en busca de bons camins i després s'envien més formigues cap als destins més probables del trànsit.

2. Cada formiga actua al mateix temps, conjuntament, amb les altres. Les formigues es comuniquen entre elles modificant les dades per on passen.
3. A cada salt de la formiga, aquesta calcula les dades i mesures fins a arribar al destí. Per arribar al destí, en cada node es decideix el pròxim salt.
4. Quan arriba al destí, es genera una formiga de tornada, que conté la ruta per on ha anat.
5. Aquesta formiga, **sí,** modifica les taules d'encaminament de cada router per on passa. Es modifiquen segons els resultats calculats sobre la qualitat del camí (les mesures).
6. En arribar al destí, la formiga de tornada (o sigui, en el node d'origen de la formiga d'anada) mor.

5.2. Descripció de l'algorisme

L'algorisme AntNet-QoS es basa en l'algorisme AntNet, però se li ha incorporat Qualitat de Servei. D'aquesta manera, es pot controlar millor el trànsit de la xarxa i adequar-nos millor a les necessitats que sorgeixin. Tal i com hem explicat, per diferenciar trànsit fem servir una arquitectura DiffServ. Tant en les formigues d'anada com de tornada, per cada tipus de trànsit (Type of Service: ToS), hi ha un tipus de formiga per (Class of Ant: CoA) encarregat de prendre les mesures del seu respectiu ToS.

Les formigues de cada ToS es comuniquen indirectament amb les altres formigues del mateix ToS.

- **Taula d'encaminament:** Cada router disposa d'una única taula d'encaminament per cada ToS i destí. Segons on vulgui anar el paquet, s'han definit les probabilitats als enllaços dels veïns.

$$\sum_{n \in N_k} P_{nd} = 1, d \in |1, N|, N_k = \{veins(k)\}$$

  On *k*=node, *d*=destí, *n*=veí. La suma de probabilitats per anar a un destí dels enllaços dels veïns ha de ser 1.

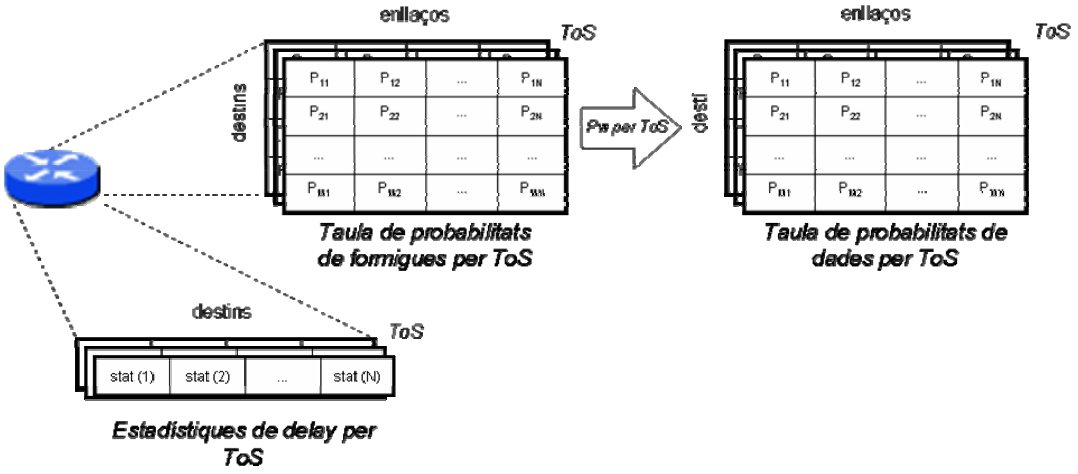
- **Taula d'estadístiques:** No només s'analitza el resultat actual que pot obtenir una formiga, sinó que s'acumulen una sèrie d'estadístiques per saber quins han estat els millors camins. Es fa servir un model estadístic a partir d'una mitjana μ

i una variància σ per cada ToS i destí. Per evitar acumular moltes mostres antigues, està definida una finestra on es fan servir els últims valors obtinguts.

$$\mu_{d,tos} = \mu_{d,tos} + \eta(O_{tos,k->d} - \mu_{d,tos})$$
$$\sigma_{d,tos} = \sigma_{d,tos} + \eta((O_{tos,k->d} - \mu_{d,tos})^2 - \sigma_{d,tos}{}^2)$$

On O representa la mesura actual de la formiga de ToS per anar del node k al destí d.

La principal diferència entre l'algorisme AntNet-QoS i el d'aquest projecte, consisteix en que AntNet-QoS només fa càlculs estimatius sobre el delay (retard) associat als camins, mentre que l'altre a més a més calcula l'amplada de banda disponible i el jitter (el qual implica disposar de més taules d'estadístiques).



Imatge 6: L'estructura interna d'un node de l'AntNet-QoS: per un costat hi ha les taules en que es guarden els valors de les mesures, i amb aquests es calcula el reforçament que actuarà en la taula de probabilitats de formigues. D'aquesta taula es transformarà en una altra semblant però per dades.

6. AntNet-QoS amb mesures d'amplada de banda

6.1. Punt de partida

El primer que vaig haver d'adaptar respecte l'algorisme AntNet-QoS va ser que l'havíem d'actualitzar a la nova versió de l'OMNeT++. L'algorisme es va desenvolupar durant el 2004, amb la versió 3.0. Amb la nova versió 3.2, hi van haver varis canvis que feien que inicialment no funcionés l'algorisme amb el simulador (errors de castings, canvi en la gestió del generador de nombres aleatoris, falta de memòria en les piles,millores en els mòduls d'estadístiques...).

Un cop arreglat els errors i avisos a l'hora de compilar, el següent pas abans de començar a implementar la idea del projecte, va ser la traducció de variables, i comentaris a l'anglès. No només pot servir per si es publica el codi (per exemple, en la pàgina oficial d'OMNeT++ s'hi publiquen projectes de models), sinó que calia tindre a punt un codi més intel·ligible ja que durant els mesos de març a juny, van vindre dos alumnes d'Erasmus que van treballar amb el codi d'AntNet-QoS desenvolupant nous projectes de final de carrera[8, 9].

Finalment, volíem provar l'efecte de la funció squash en l'algorisme AntNet-QoS, i que després també s'ha aplicat amb les mesures d'amplada de banda. La funció squash actua de manera que el reforçament (valor que determina si un camí és bo o no), s'incrementés si el node disposa de molts veïns, per tal de marcar bé el bon camí i es notés respecte als altres. En canvi, si hi ha pocs veïns, el reforçament no s'incrementa tant.

Quan aquests preparatius van estar a punt, llavors es va començar amb el disseny i la implementació de mesures d'amplada de banda, el qual es va iniciar en la data d'inici del projecte.

6.2. Disseny de l'algorisme

Per poder realitzar les estimacions d'amplada de banda i de jitter, ens calen enviar trens de formigues cap a cert destí. Un tren de formigues consisteix en enviar una ràfega de formigues alhora. Observant en quins instants arriben tant en valors absoluts de temps, com en valors relatius entre ells, obtenim les dades necessàries per realitzar els càlculs de l'amplada de banda, el retard i la variació del retard.

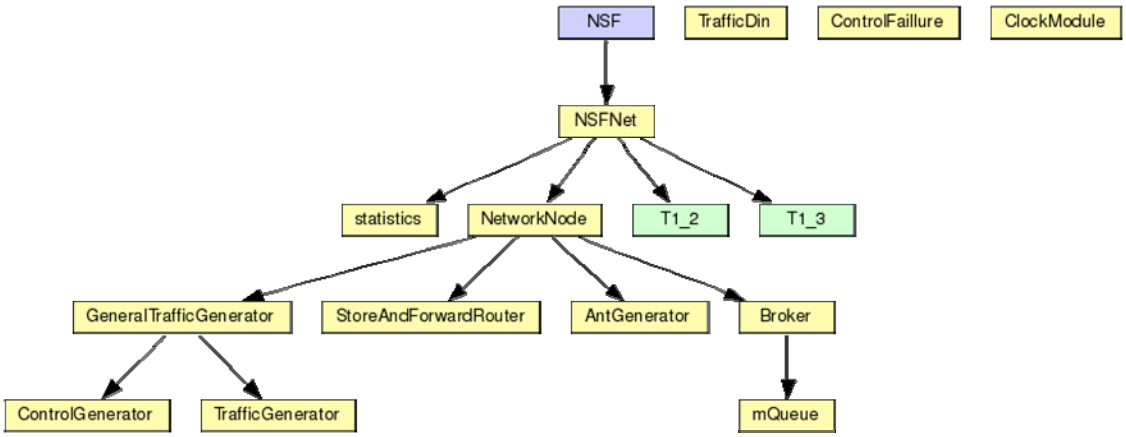
El generador de formigues (Ant_Gen) és l'encarregat de generar tantes *n* CoA com *n* ToS hi hagin. Nosaltres definim prèviament el nombre de formigues que formen un tren, i també el tamany de les formigues del tren.

És important la mida de les formigues del tren, ja que per poder fer els càlculs, aquestes han de ser de tamany fix. El temps de transmissió de la formiga a través del canal és directament proporcional a la mida d'aquesta.

Segons l'article de recerca descrit en [10] en que ens basem en fer els experiments del nostre projecte, el tamany dels paquets ha de ser entre 500 i 700 bytes i el nombre de formigues del tren pot variar entre 8, 16, 24, 32 o 64. En els nostres experiments fem servir un tren de 8 paquets, ja que el cas d'enviar amb més freqüència aquests trens de prova, les mesures d'amplada de banda són semblants (aproximades) tant amb 8 com 32 paquets, i per tant si fem servir menys paquets aconseguim no saturar tant la xarxa i controlar així el seu overhead.

Cada formiga del tren, a cada salt realitza el càlcul d'amplada de banda i jitter particulars. L'última formiga del grup, a més a més, realitza un càlcul de les dues mesures de manera total (mesures respecte al flux d'aquest tren). Per particular entenem que es mesura el bw d'un sol paquet. En canvi, el total mesura el bw de tot el tren (i resulta una mesura més fiable).

Un cop que totes les formigues del tren han fet els seus càlculs i arriben al destí, aquestes moren i llavors el node destí genera una formiga de tornada (BackAnt), que seguirà la mateixa ruta per on han passat les formigues, i contindrà les dades generades pel tren. Aquesta formiga, a cada pas per un router, li donarà les dades (delay, bandwidth, jitter), i el router realitzarà un càlcul determinant la bondat del camí, i ho aplicarà a les seves taules d'encaminament. A cada salt s'aniran actualitzant dades dels routers. Un cop la formiga de tornada hagi arribat al destí (origen per les formigues d'anada), llavors es mor.



Imatge 7: Diagrama d'ús complert de l'AntNet-QoS amb mesures de bw

El diagrama de la imatge ens mostra tota l'estructura dels mòduls de l'algorisme. NSF es refereix al tipus de xarxa que fem servir en la simulació. T1_2 i T1_3 són els noms que definim als enllaços, en els experiments T1_2 és un enllaç de 10 Mbits/s i T1_3 de 3 Mbits/s. El mòdul més important és el NetworkNode, ja que dins del node trobem, d'esquerra a dreta: el generador de tràfic, el router, el generador de formigues, i les cues

6.2.1. Càlculs teòrics

El nombre de formigues en el tren i el tamany d'aquestes no corresponen a nombres a l'atzar. En aquest apartat intentarem explicar alguns d'aquests valors, i també exposarem les maneres de calcular els nous paràmetres que s'afegeixen al ja existents a l'algorisme AntNet-QoS (càlcul de l'amplada de banda i de jitter).

En el simulador OMNeT++, podem definir la capacitat que tenen els diversos enllaços, i també si es produeixen errors, el retard que hi ha i fins i tot programar que es caiguin enllaços en la xarxa. Avui en dia, parlant de tecnologies ADSL i xarxes Ethernet, velocitats normals són en unitats de Megabits per segon (Mbps). En els nostres experiments, definim velocitats de 3 Mbps i 10 Mbps.

Si per exemple triem el següent tren: 500 bytes, 32 paquets:

500 bytes * 8 bits/byte * 32 paquets = 128000 bits = 0,128 Mbits

O 700 bytes, 64 paquets:

700 bytes * 8 bits/byte * 64 paquets = 358400 bits = 0,3584 Mbits

Observem que ocupem molt poc amplada de banda comparat amb la capacitat de l'enllaç (3 Mbps). Això és bo, ja que hem de pensar que això és un tren per un sol ToS, i poden haver més d'un. A més a més, el més important són les dades que han de poder viatjar pels enllaços, i aquestes poden requerir molt més amplada de banda.

Per calcular l'amplada de banda disponible, la fórmula general és la següent:

$$bw = \frac{paq.(bits)}{temps(seg)\ d'arribada\ dels\ bits}$$

Ens serveix tant per fer els càlculs particulars com càlculs totals (en el particular és el tamany d'un sol paquet i en el total és el tamany de tot el tren).

Hi ha altres tècniques per calcular l'amplada de banda disponible i més precises, algunes d'aquestes fan servir tot l'amplada de banda del canal en alguns instants. Però tal i com hem dit, ens interessa fer estimacions i no càlculs exactes ja que s'ha de fer d'una manera senzilla i ràpida.

Per la mesura del jitter, hem d'entendre que entre els paquets del tren, hi ha una petita distància entre ells, ja que és impossible enviar varis paquets alhora per un enllaç. És possible que els paquets variïn aquesta distància ja que passen a través de cues, i poden haver retencions que afectin a l'enviament normal dels paquets. Si l'enviament és perfecte, el millor jitter que es pot obtenir és 0.

La fórmula per calcular el jitter és:

$$\frac{1}{n-2}\sum_{i=2}^{n}\left|(t_i - t_{i-1}) - (t_{i-1} - t_{i-2})\right|$$

On *n* és el nombre de paquets del tren i la *i* és el número de paquet que li correspon dins del tren.

6.2.2. Reforçament

El reforçament es produeix en l'etapa d'actualització de les taules d'encaminament del router. Aquest procés consisteix en aplicar uns càlculs els quals ens indica si el camí que s'avalua és un bon camí o no, i es reflecteix en les taules. En el cas de les formigues simples, només tindrem un reforçament de delay. En canvi, pel tren de formigues es calcula el delay, bandwidth i jitter, i dels tres obtenim un sol reforçament combinat. Cada un dels reforçaments ha de comprendre entre 0 i 1, ja que són probabilitats que les hem d'aplicar en les taules.

El primer pas abans de calcular-lo, és actualitzar els valors de les estadístiques. A partir de les mesures, es calcula una mitjana i una variació de cada paràmetre. També es guarden els millors valors obtinguts pel router, però ens limitem a una finestra d'observació dels últims valors *W*, perquè seria una despesa molt gran de mantenir tots els valors en una simulació llarga.

Un cop fet aquest pas, es calcula el reforçament de cada valor:

Delay: Un camí és més bo quan menys delay (retard) té. És a dir, si les mesures ens donen un delay baix, significa que sortirà un reforçament alt.

La fórmula per poder calcular el delay és:

$$r = c_1 * \left(\frac{W_{best}}{T} \right) + c_2 * \left(\frac{I_{\sup} - I_{\inf}}{(I_{\sup} - I_{\inf}) + (T - I_{\inf})} \right)$$

On *r* és el reforçament entre 0 i 1, *c1* i *c2* són dues constants, *W(best)* és el millor delay per anar al destí de la finestra d'observació, *T* és l'últim delay que s'ha mesurat i *Isup* i *Iinf* són els límits aproximats per un interval d'estimació μ.

Les constants c1 i c2 serveixen per definir la prioritat entre la mesura actual (c1) i un històric de mesures (c2). Valors bons per l'experimentació han resultat ser de c1=0,7 i de c2=0,3.

En *Wbest*, el millor delay significa el valor més baix possible, i per tant, la divisió entre *Wbest* i *T,* mai podrà ser més gran de 1, ja que Wbest s'actualitza abans de fer aquest càlcul. A un valor de T més gran (més retard), el resultat serà menor (menys reforçament).

Per *Isup* i *Iinf* , a *Iinf* se li assigna el valor *Wbest* (el valor més baix possible), i per *Isup* se li aplica la fórmula:

$$I_{\sup} = \mu + z(\sigma / \sqrt{|W|})$$

On *z* és:

$$z = 1 / \sqrt{1 - \beta}$$

I on β defineix el nivell de confiança seleccionat.

El conjunt de la segona divisió en l'equació de la *r*, s'observa que tampoc podrà ser el resultat més gran de 1.

Bandwidth: La manera de calcular l'amplada de banda difereix en la del retard, ja que un camí és més bo, quanta més bw disposa. Per tant, la fórmula queda canviada i cal aplicar-la al revés:

$$r = c_1 * \left(\frac{Bw}{W_{best}} \right) + c_2 * \left(\frac{I_{\sup} - I_{\inf}}{\left(I_{\sup} - I_{\inf}\right) + \left(I_{\sup} - Bw\right)} \right)$$

En aquest cas, *Wbest* és el millor valor d'amplada de banda, i tal i com hem dit, serà sempre un valor més gran o igual que *Bw* (mesura actual).

Isup se li assigna el valor de *Wbest* i a *Iinf* se li aplica la mateixa fórmula feta servir en *Isup* del delay.

Jitter: Pel jitter, es comporta d'una manera semblant al delay, ja que es busca obtenir quanta menys variància del retard, millor. Es fa servir la mateixa estratègia que pel delay, i se li apliquen les mateixes fòrmules.

L'única diferència amb el delay, consisteix en que és possible que el jitter ens doni valors de 0 (no hi ha variància del retard). Com que fer divisions per 0 dóna valors infinits, s'ha aplicat que en el cas que el jitter sigui 0, no es fa la divisió sinó que directament ja li donem a la divisió el resultat de 1, ja que és el màxim valor que pot obtenir.

Si hem realitzat els tres càlculs, llavors cal combinar els tres reforçaments en un de sol (que s'anomena reforçament compost). El calculem a partir de:

$$r_{comp} = p_{delay} * r_{delay} + p_{bw} * r_{bw} + p_{jitter} * r_{jitter}$$

On les *p* les definim prèviament en el fitxer de configuració del router (router_config_file) per cada ToS. Per exemple, es pot proposar pel ToS0 un *pdelay* de 0,5 , p*bw* 0,4 i *pjitter* 0,1[2]. Les probabilitats han de sumar 1, ja que el resultat màxim de *rcomp* no pot passar de 1.

Per les formigues normals (no pertanyen al tren), només es realitza el càlcul del delay.

Un cop s'obté el valor del reforçament, es pot aplicar una funció correctora anomenada *squash*. Aquesta funció funciona millor amb nodes en que tenen molts veïns. Si el valor de r és bo, llavors la funció squash l'incrementa més per tal que es noti respecte als demés veïns. Si pel contrari és dolent, i disminueix la r si aquesta és dolenta.

[2] Això vol dir que és possible dissenyar els tipus de serveis que ofereix la xarxa. Per exemple: per tenir una xarxa que suporti la qualitat de "baix jitter" s'ha de configurar un ToS amb tren de formigues que li doni alta importància al reforçament del jitter, o sigui, al paràmetre *pjitter* de la funció combinada dels reforçaments. El disseny dels tipus de servei podria ser tasca dels administradors de xarxes.

$$s = \frac{\left(1 + \exp\left(\frac{a}{|N_k|}\right)\right)}{\left(1 + \exp\left(\frac{a}{r * |N_k|}\right)\right)}$$

On *a* és un valor que el definim (10 és un bon valor), *r* és el reforçament com a dada d'entrada i *Nk* és el nombre de veïns del node k.

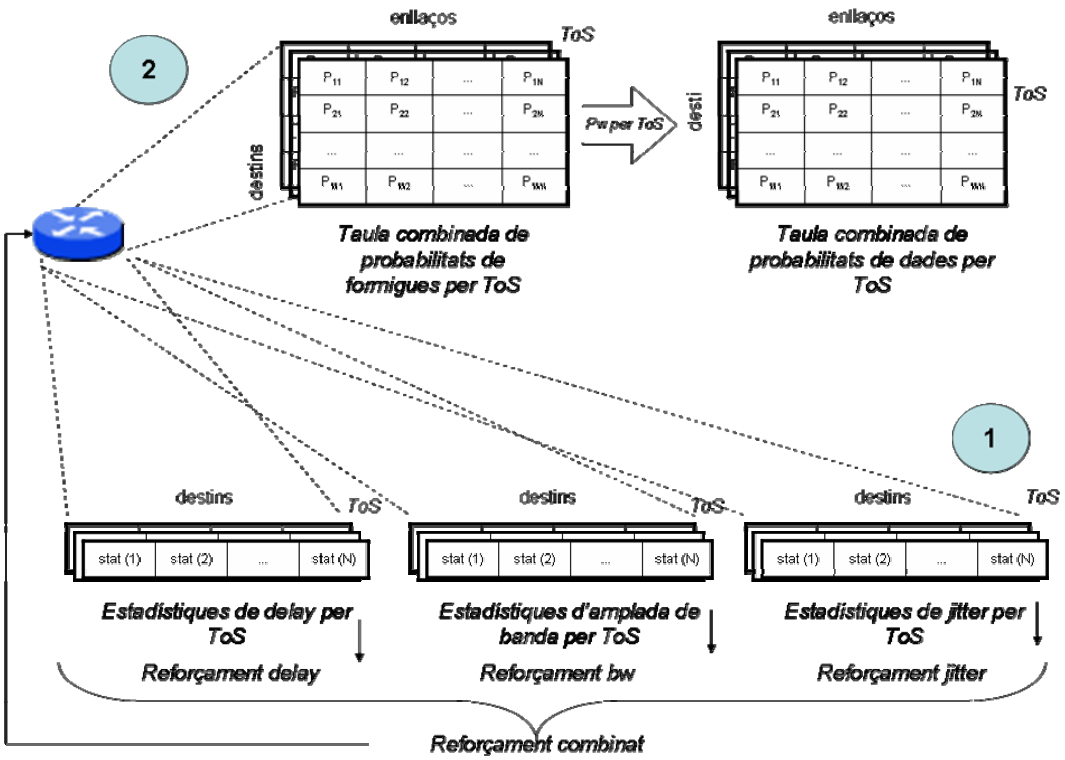
6.2.3. Taules d'encaminament

Un cop obtingut el valor de reforçament amb squash o sense, l'hem d'aplicar per actualitzar la taula d'encaminament del router. Recordem que cada reforçament, és per un sol ToS i destí, i que actualitza la corresponent taula d'encaminament.

La taula d'encaminament funciona amb probabilitats, i la suma de les probabilitats d'un ToS i un destí ha de ser 1. Quan es modifica la taula, un valor augmenta i els altres disminueixen per tal de complir la probabilitat. A diferència d'obtenir tres tipus diferents de reforçament per cada paràmetre, al crear un sol reforçament combinat, només disposarem d'una sola taula d'encaminament per dirigir els paquets.

Hem de fer una distinció entre taules d'encaminament de formigues, i taules d'encaminament per les dades. Els càlculs de reforçament i la seva aplicació es fa en la taula de formigues. La taula de dades s'obté aplicant una fórmula de potència (pow) a la taula de formigues perquè les dades tinguin una tendència més alta a seguir els camins més bons. Aquesta funció augmenta les probabilitats bones i decreix les probabilitats dolentes, així les dades no segueixen camins dolents. Al fitxer de configuració del router trobem el valor del *pow* definit. Hem utilitzat un pow igual a 1,2 continuant amb la filosofia de l'AntNet tradicional.

A la figura 8 trobem l'estructura general del nou AntNet-QoS amb mesures de jitter i amplada de banda que es pot comparar i diferenciar clarament de l'anterior estructura d'AntNet-QoS que es mostra a la figura 6.



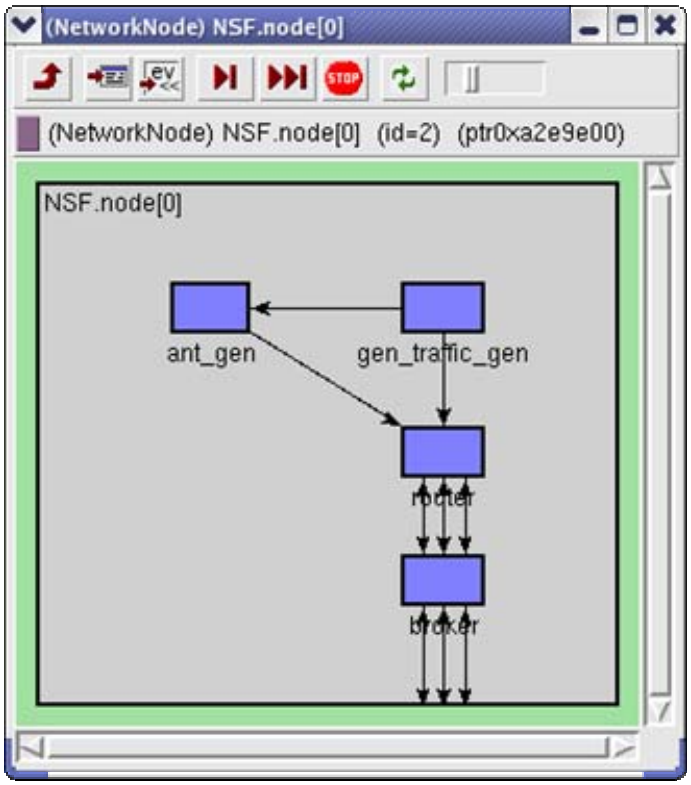
Imatge 8: L'estructura actual de l'AntNet-QoS amb les millores aplicades

6.3. Descripció detallada dels mòduls

Aquest projecte de final de carrera es basa a partir d'un algorisme ja implementat (AntNet-QoS), el qual li hem afegit una sèrie de millores. Moltes parts són comunes entre els dos algorismes, per tant, en la descripció dels mòduls explicarem les diferències i canvis que s'han produït en els mòduls més importants.

Abans, farem un brevíssim repàs a tots els mòduls de l'algorisme, per tal que ens hi situem:

- Ant_gen: S'encarrega de generar les formigues.
- Gen_traffic_gen: Mòdul compost en,
    - Control_gen: Controla tot el trànsit que es genera al node.
    - Traffic_gen: Crea trànsit estàtic cada x temps determinat.
    - Dynamic: Genera trànsit però segons condicions de la xarxa.
- Router: Encarregat d'enroutar els paquets, i fer les taules d'encaminament.
- Broker: Mòdul compost en,
    - Queue: Guarda en cues de dades i de formigues els paquets.
- Stats: Gestiona totes les estadístiques de la xarxa.



Imatge 9: Estructura interna del node 0

6.3.1. Ant_Gen

Aquest mòdul té com a missió la generació de formigues i trens de formigues. És important fer aquesta distinció, ja que per la QoS que faci servir trens, aquesta només enviarà trens de formigues i no formigues normals. A l'enviar trens, les taules d'encaminament faran servir les tres mesures (delay, bandwidth i jitter). En canvi, si se li defineix sense tren, només enviarà formigues soltes, i les taules d'encaminament només es veuran afectades per les mesures de delay (com a l'AntNet-QoS).

Un altre canvi important respecte AntNet-QoS consisteix en els instants de temps en que es generen les formigues. En l'original, per cada Δt de temps, es generava una sola formiga que aquesta escollia un ToS i un destí. En el nostre algorisme, a través de la nostra configuració definim que per cada ToS, en el temps Δt que li definim generarà una formiga a un destí. D'aquesta manera podem controlar millor els intervals que genera formigues per cada ToS.

Al canviar els instants de temps, e incorporar varis instants de temps, la manera per calcular les probabilitats ha canviat. Abans es triava entre 0 i 1 un ToS i un destí. Cada ToS anava en un interval d'entre 1/nºToS.

Exemple: Si tenim 3 ToS, el ToS 0 va entre 0 i 0,33; ToS2 0,33-0,66 i ToS3 0,66-0,99.

Al fer el nostre canvi, quan generem la probabilitat ja sabem el ToS, i per tant només hem de triar el destí. Una solució adoptada és agafar l'interval de 0 a 0,33.

A l'incorporar la possibilitat de generar trens de formigues, sorgeixen nous paràmetres (tamany dels paquets, nombre de formigues del tren...). Concretament, ens centrarem en el temps que s'ha d'esperar entre la creació de formiga i formiga del tren.

Si intentem crear totes les formigues alhora i enviar-les, és impossible que s'enviïn totes alhora, provocant que s'hagin d'encuar i per tant, que hi hagin saturacions. L'òptim és buscar un valor que sigui petit, però permeti un enviament continu dels paquets del tren i segons el trànsit que hi circuli per la xarxa.

Un exemple, calculem per un enllaç de 10 Mbps quin seria un bon delay entre **formigues d'un tren d'un sol ToS**.

Si disposem d'un trànsit de tipus CBR (Connstant Bit Rate), amb un tamany dels paquets de 4352 bits i un retard entre enviament de paquets de 0,04 segons llavors:

$$\frac{4352 bits}{0{,}04 s} = 108800 bits/s \approx 0{,}1 Mbits/s$$

El trànsit té una velocitat de 0,1 Mbits/s dels 10 Mbits/s que disposa l'enllaç (per tant, hi ha capacitat suficient per transportar aquest i més trànsit).

Si el tamany del paquet és de 700 bytes:
700 bytes * 8 bits/bytes = 5600 bits d'un paquet del tren.

$$\frac{5600bits}{X} = \frac{100000bits}{1seg}$$

$$X = 0{,}056seg \approx 0{,}06seg$$

On X és el temps d'espera entre formigues. En aquest cas, enviaríem un tren de formigues amb una velocitat de 0,1 Mbps composat per 8 formigues enviades cada 0,06 segons. Als nostres experiments, aquests trens de formigues amb aquestes característiques s'envien cada 20 segons.
En el cas de formigues que **no formen part d'un tren**, aquestes ocupen aproximadament 24 bytesi als nostres experiments són enviades cada segon cap a un cert destí. Aquestes formigues sense tren gairebé no ocupen espai. S'ha de recordar que poden existir *n* ToS sense tren i llavors el node generaria *n* formigues ToS cada segon i cap a un cert destí per ToS definit per la formula:

$$P_{tos,d} = \frac{\int tos, sd}{\sum_{t=1}^{ntos}\left(\sum_{d=1}^{N}\int t, sd\right)}$$

On *ntos* és el nombre de tipus de trànsit.

La configuració d'aquest mòdul es realitza a través del fitxer **ant_net.txt.** Un exemple de com definir un ToS és el següent:
tos=0 train=1 size_packet_train=700 num_packets_train=8 delay_ant=20.0 time_gen_prob=10 delay_in_train=0.06
Aquesta línia del fitxer indica que pel ToS0 es faran servir trens de formigues, el tamany d'aquestes serà de 700 bytes, el tren estarà forma per 8 formigues, el temps d'espera entre generacions de trens de formigues serà de 20 instants de temps del simulador (que interpretem com a segons), i el temps d'espera a l'hora de generar cada formiga del mateix tren serà de 0,06 segons.

6.3.2. Router

El router és una de les peces clau en la nostra xarxa, ja que ha de gestionar tot l'encaminament dels paquets. Aquest mòdul ha estat un del que més s'han modificat respecte AntNet-QoS. Les funcions completes del router inclouen la recepció i enviament de paquets de tot tipus, la gestió de les seves taules d'encaminament, els càlculs de reforçament per modificar aquestes taules i també de l'amplada de banda i jitter dels trens de formigues.

La gestió en un router dels paquets que pot rebre:

Quan li arriba un missatge, primer comprova si és una formiga d'anada, de tornada, una dada o és un missatge el qual li indica que actualitzi la taula d'encaminament.

- **Formiga d'anada (i no és al destí):**

  El router ha de comprovar no només formigues soltes, sinó també els trens de formigues. En el cas que la formiga sigui d'un tren, llavors ha de mirar a quina posició hi pertany en el tren. Quan la generem, se l'incorpora com a dada el número de formiga dins el tren, per tal que el router ho pugui distingir.

  A la primera formiga del tren, cal guardar el temps d'arribada al router, ja que llavors combinat amb el temps en que hi arriba l'última formiga del mateix tren, obtindrem un interval de temps necessari per calcular l'amplada de banda disponible en l'enllaç d'on ha vingut. Per lògica, en l'última formiga del grup es fan els càlculs d'amplada de banda i jitter.

  A continuació, la primera formiga del grup ha de decidir a quin node farà el següent salt, ja que per anar a un destí, a cada node les formigues decideixen a on saltar. El node escollit es guarda en el router, per tal que les demés formigues del tren segueixin a la primera d'elles. Per evitar possibles problemes d'encaminament, a les formigues (siguin del tren o no), no se'ls hi permet tornar endarrera. En el cas que la formiga no sigui del tren, s'eliminaran els possibles cicles [3] i només es decidirà a quin destí ha de saltar.

[3] En les formigues del tren, no eliminem cicles del ser recorregut ja que els necessitem pels càlculs d'amplada de banda i jitter. En canvi, les formigues normals només calculen el delay, i els hi és més favorable que puguin eliminar cicles.

- **Formiga d'anada (i és al destí):**

  En aquest cas, també es realitzen tots els càlculs anteriors si forma part d'un tren de formigues. La diferència consisteix en que ara no hem de decidir a on saltar, ja que s'ha arribat al destí. En el tren de formigues, l'última formiga és l'encarregada de generar una formiga de tornada, que seguirà exactament la mateixa ruta que ha seguit el tren, i a més a més inclourà els resultats dels càlculs de bw i jitter. Si no és del tren, també genera una formiga de tornada però sense càlculs.

  Totes les formigues que arriben al seu destí, són eliminades.

- **Formiga de tornada:**

  La formiga va recorrent el camí pels mateixos nodes per on ha passat la formiga d'anada. En cada node, li dóna al router les dades que disposa i el router realitza els càlculs (reforçament) i actualitza la taula d'encaminament. Per realitzar aquest pas, la formiga quan passa pel broker (la cua), envia un missatge d'actualització al router. Si arriba al destí, la formiga envia un missatge al broker (cua) per tal que aquest enviï el missatge d'actualització al router, i la formiga mor.

- **Dada:**

  El funcionament de les dades és més simple. Si no ha arribat al destí, llavors decideix quin serà el següent salt que realitzarà i s'envia. Si arriba al destí, llavors mor.

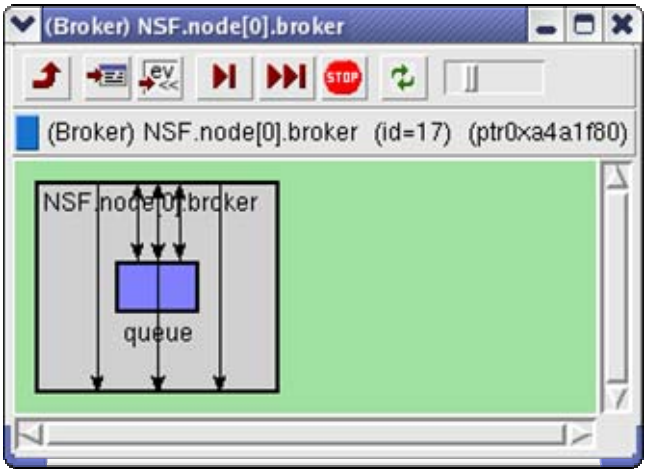
- **Missatge d'actualització:**

  Al rebre el router aquest missatge, inicia el procediment el qual calcula el reforçament (només de delay si el ToS no és de tren de formigues, sinó de delay, bw i jitter), aplica una funció squash i actualitza les taules d'enrutament tant de formigues com de dades (sempre actualitza només el ToS que era la formiga). El reforçament, i les taules d'encaminament s'explicaran més endavant.

En tots els casos, abans de morir les formigues, es guarden estadístiques de les formigues i les dades que han pogut arribar al destí. Si el paquet s'ha de morir, per exemple, perquè ha excedit el temps de vida, també en queda constància.

6.3.3. Broker

El broker és un mòdul compost, i dins seu són les cues. Dins de cada node, les cues es troben a les sortides del routers (és a dir, els paquets que entren van directament al router sense passar per les cues). La seva funció és evitar que es perdin paquets si s'intenta enviar més d'un paquet alhora per l'enllaç.



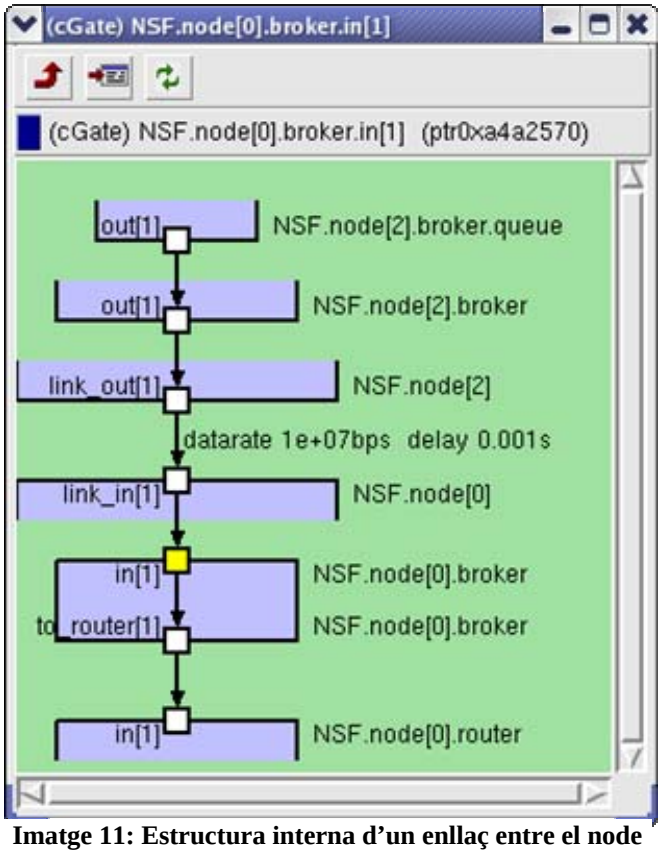
Imatge 10: Les fletxes petites connecten les sortides del router cap a les cues. En canvi, les fletxes grans van del node veí fins al router, sense passar per la

Hi ha tants enllaços com veïns del node, i un enllaç de més per fer la comunicació entre broker i router (per exemple, el broker envia missatges d'actualització al router).

Les cues es divideixen en dos grans grups: cues de formigues i cues de dades. Dins de les cues de formigues, es divideixen en formigues d'anada i formigues de tornada. Les cues de dades, n'hi ha una per cada ToS.

En l'AntNet-QoS, les formigues de tornada tenen més prioritat que les formigues d'anada, però en el nostre algorisme són les formigues d'anada que tenen més prioritat, ja que interessa per tal que els càlculs d'amplada de banda i jitter siguin més exactes.

Un tren de formigues s'encua en la cua de dades i no en la de formigues, perquè llavors el càlcul del delay és una mesura exacte, i no una estimació com a AntNet-QoS.



Imatge 11: Estructura interna d'un enllaç entre el node 2 i el 0 de 10 Mbps. Encara que es mostri d'una sola direcció, és bidireccional (es veu de la perspectiva de que rep el node 0)

Les cues tenen una capacitat, que es defineix en el fitxer de configuració de les cues (*buffers_size.txt*). Teòricament, una cua és una estructura de dades infinita, però en el món real, els buffers tenen una capacitat, i per això les limitem.

Quan es van encuant dades en diferents cues, es dóna sovint el cas que cal triar-ne un per enviar-lo. Es selecciona una cua segons unes probabilitats que les definim en el fitxer de configuració *prob_queue.txt*. Cada ToS en té definida una que li donem segons la prioritat o preferència que ha de tenir aquell trànsit per utilitzar el canal i la suma d'aquestes prioritats ha de ser 1 per fer ús màxim del canal (amplada de banda).

Les cues estan preparades per si s'ha de triar una dada i hi ha cues buides. Per probabilitats, és possible que es pugui triar una cua buida, i en les altres hi ha dades a enviar. Per això, abans d'escollir-ne una, es comprova si hi ha cues buides, i en tal cas es reajusten les probabilitats per no triar la cua buida.

7. Configuració de l'algorisme

Per dur amb èxit una simulació, abans d'executar-la cal que configurem tots els paràmetres que es fan servir. Gràcies a aquesta configuració, es pot adaptar un algorisme a molts estats diferents de la xarxa, i també es pot configurar perquè actuï segons les nostres preferències. Respecte l'algorisme AntNet-QoS, s'han afegit nous paràmetres i fitxers per poder controlar millor els nous trens de formigues.

7.1. Configuració dels arxius NED

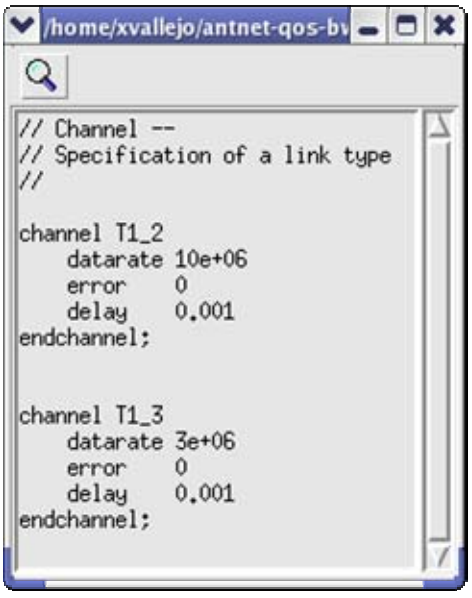
Els fitxers NED defineixen la forma de la xarxa, per exemple: el nombre de nodes, la capacitat dels canals, el temps de processament, etc. També són necessaris per definir les estructures dels mòduls (és a dir, un mòdul com el generador de formigues necessita un fitxer de codi .cpp, un per definir .h i un .ned).

Un fitxer NED fa servir la seva pròpia gramàtica, però no és tant complicada com aprendre un llenguatge de programació. En l'apèndix A del manual de l'OMNeT++ [6] s'explica la notació del llenguatge

En aquest apartat, es mostraran el fitxers que es fan servir per a configurar la xarxa:

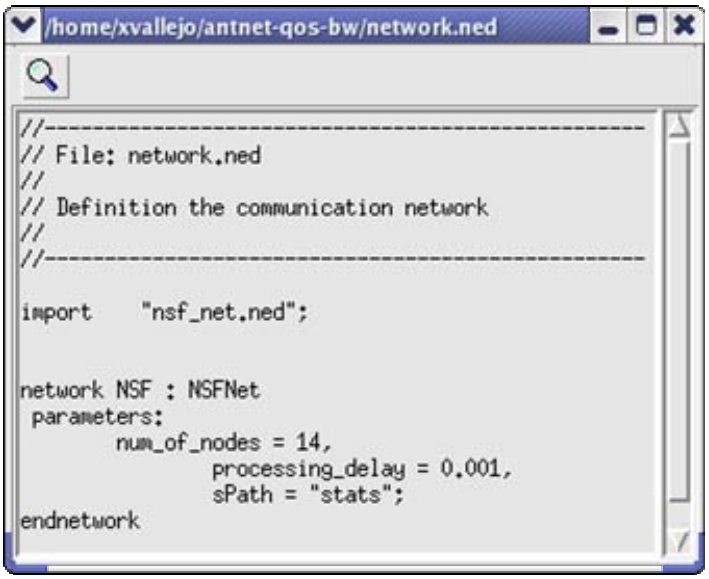
- **channels.ned:**

En aquest fitxer es poden definir tants enllaços com es vulguin, encara que no es facin servir. En l'exemple, T1_2 és el nom de l'enllaç, datarate és la capacitat de l'enllaç en bps, probabilitat d'errors en la comunicació i el delay del temps de transmissió.



Imatge 12: Exemple configuració enllaços

- **network.ned:** Aquí es defineix el nombre de nodes de la xarxa i el temps de processament del routers per la xarxa NSFNet (la que es fa servir).



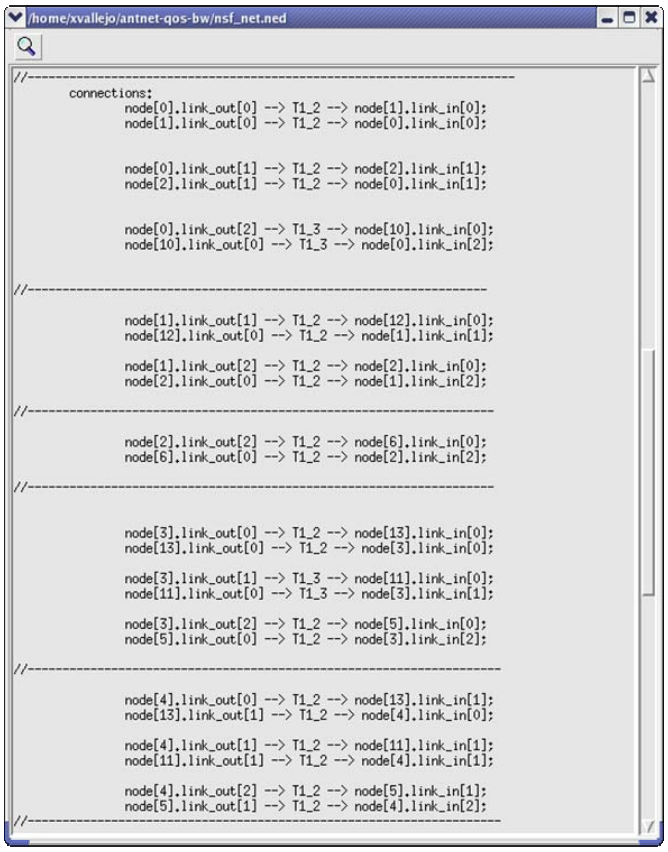
Imatge 13: Paràmetres de la xarxa NSFNet

- **nsf_net.ned:** Si abans s'ha parlat de la xarxa NSFNet, ara toca definir-la en un dels fitxers més importants de configuració.

Es pot subdividir en dues parts: en la primera, es descriuen els paràmetres del mòdul NSFNET. En l'exemple, li hem d'indicar els paràmetres del network.ned. Dins del contenidor NSF, cada node es pot veure com un submòdul. Per això, en *submodules* hem de definir el contingut d'un node (paràmetres i el nombre de veïns). Un cop s'ha definit, cal realitzar les connexions entre els nodes, definint cada enllaç com unidireccional i especificant el canal definit a channels.ned

En la següent pàgina hi ha dues imatges que representen el contingut d'un nsf_net.ned.

Imatge 14: Primera part, defineix el mòdul NSFNet.

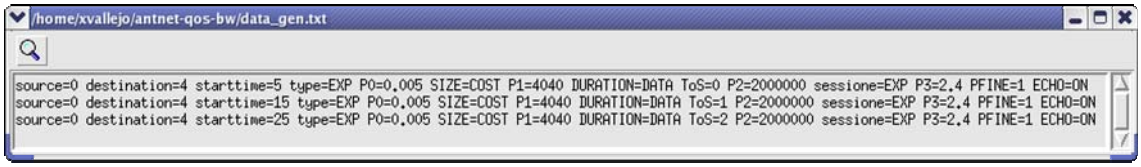


Imatge 15: Segona part, defineix les connexions entre nodes

7.2. Configuració del trànsit

Els arxius que es faran servir ja no són del llenguatge NED, sinó fitxers de text simples, que els llegiran els mòduls quan s'inicialitzi la simulació. Els anteriors fitxers tenien una avantatge ja que es comprovava la sintaxi en l'etapa de compilació. En canvi, en els txt s'ha de comprovar manualment per no equivocar-nos. Però en canvi, si es volen fer vàries proves canviant paràmetres, llavors és una avantatge el fitxer txt ja que no és necessari tornar a compilar[4].

- **data_gen.txt:**



Imatge 16: Configuració de la generació de trànsit

Cada línia representa una classe de servei diferent, indicada pel paràmetre ToS. source i destination significa l'origen i destí per on viatjaran les dades.

starttime: És el temps d'inici de l'enviament de dades.

type representa el tipus de retard entre l'enviament de paquets. Un és CBR (Constant Bit Rate), mentre que el fet servir és EXP (Exponencial).

p0: Paràmetre utilitzar per determinar el temps d'espera entre paquets.

SIZE representa la funció per calcular el tamany del paquet. Cost és constant, sinò és funció exponencial

p1 és el valor agafat pel càlcul del size (en bits).

DURATION li indica si ha de generar dades tota l'estona (ALL), durant un temps (TIME) o una quantitat de dades (DATA).

p2: Indica el temps o quantitat de dades de Duration.

sessione indica si la següent sessió de dades és EXP exponencial o COST constant i P3 indica el temps de generació de la següent sessió.

PFINE: Amb un 0 indica que ha de generar més mòduls en cas que DURATION no sigui ALL, i amb un 1 no en genera més.

[4] Els fitxers NED tenen una opció per poder-se modificar i que no calgui compilar cada cop. S'ha de definir el paràmetre "*preload-ned-files=*.ned*" en omnetpp.ini

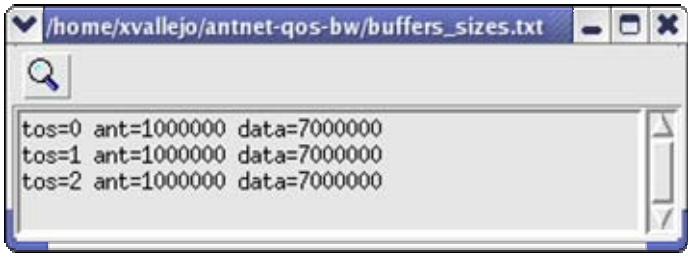
- **broke_link.txt**:

Imatge 17: Definició de links defectuosos

El fitxer té un funcionament molt simple: en cada línia es defineix un enllaç i el temps en que l'enllaç deixarà de funcionar. L'únic que cal fer és posar el mateix enllaç dos cops, perquè és bidireccional. Si es posa un temps superior al de la simulació, llavors l'enllaç no fallarà.

7.3. Configuració de la gestió de cues

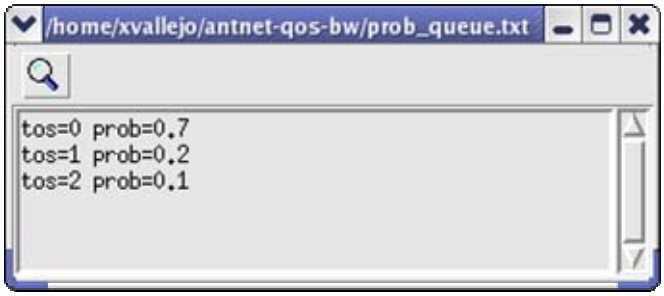
- **buffers_sizes.txt:**



Imatge 18: Definició del tamany de les cues

Cada línia representa un ToS diferent. El tamany de les cues es divideix en el de les formigues i el de les dades. Els valors són en bits i per tant en aquest cas el tamany de la cua de formigues és d'1 Mb i per dades 7 Mb; en total cada ToS ocupa 8 Mb en cues.

- **prob_queue.txt:**

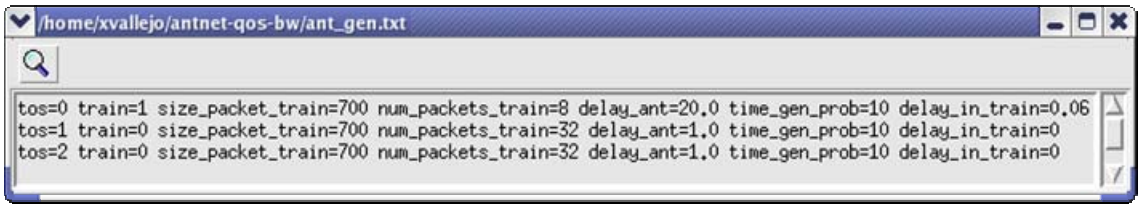


Imatge 19: Probabilitats de les cues

Per cada ToS li definim una probabilitat, i la suma de les probabilitats ha de ser 1. Aquest valor significa que el ToS tindrà més o menys possibilitats de se elegit per enviar paquets que es trobin en la seva cua.

7.4. Configuració dels agents de control

- **ant_gen.txt:**



Imatge 20: Configuració de les formigues i trens de formigues

Aquest arxiu és totalment **nou** respecte AntNet-Qos i incorpora moltes novetats referents al mòdul de generar formigues. Com és habitual, cada línia representa un ToS diferent.

train: Amb el valor de 1, indica que aquella ToS farà servir trens de formigues i tot el que implica, i no farà servir formigues soltes. Amb el 0 enviarà formigues soltes (com AntNet-QoS).

size_packet_train: Indica amb **bytes (no bits)** el tamany de cada formiga del tren. Si el ToS no fa servir tren, el valor s'ignora.

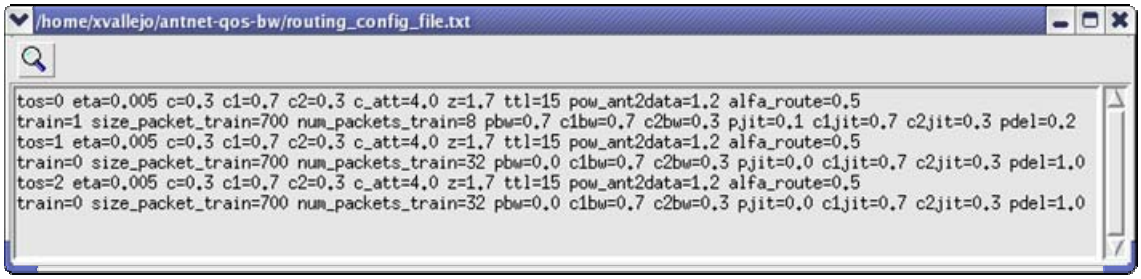
num_packets_train: Representa el nombre de formigues que forma el tren.

delay_ant: Representa el temps d'espera entre generacions de trens de formigues. En l'exemple, el ToS0 generarà cada 20 segons un tren de 8 formigues de tamany 700 bytes. El ToS1 genera cada segon una formiga.

time_gen_prob: Actua a l'hora de la probabilitat de generar formigues, i funciona per tots el ToS.

delay_in_train: Només funciona si es fa sevir trens de formigues. Indica el temps, en segons, que s'ha d'esperar entre generar formigues del tren, per evitar que s'enviïn totes alhora i s'acumulin a les cues. Defineix la velocitat del flux de formigues d'aquest tren.

- **routing_config_file.txt**



Imatge 21: Configuració del router

En aquest cas, cada dues línies representa un ToS. La segona línia de cada ToS ha estat afegida respecte AntNet-QoS.

En la primera línia destaquen els següents paràmetres:

c: Paràmetre pel càlcul de la finestra d'observació.

c1 i c2: Són els coeficients fets servir per calcular el reforçament del delay.

z: Paràmetre pel càlcul dels intervals d'estimació en el moment de fer el reforçament.

ttl: Temps de vida (segons) dels paquets.

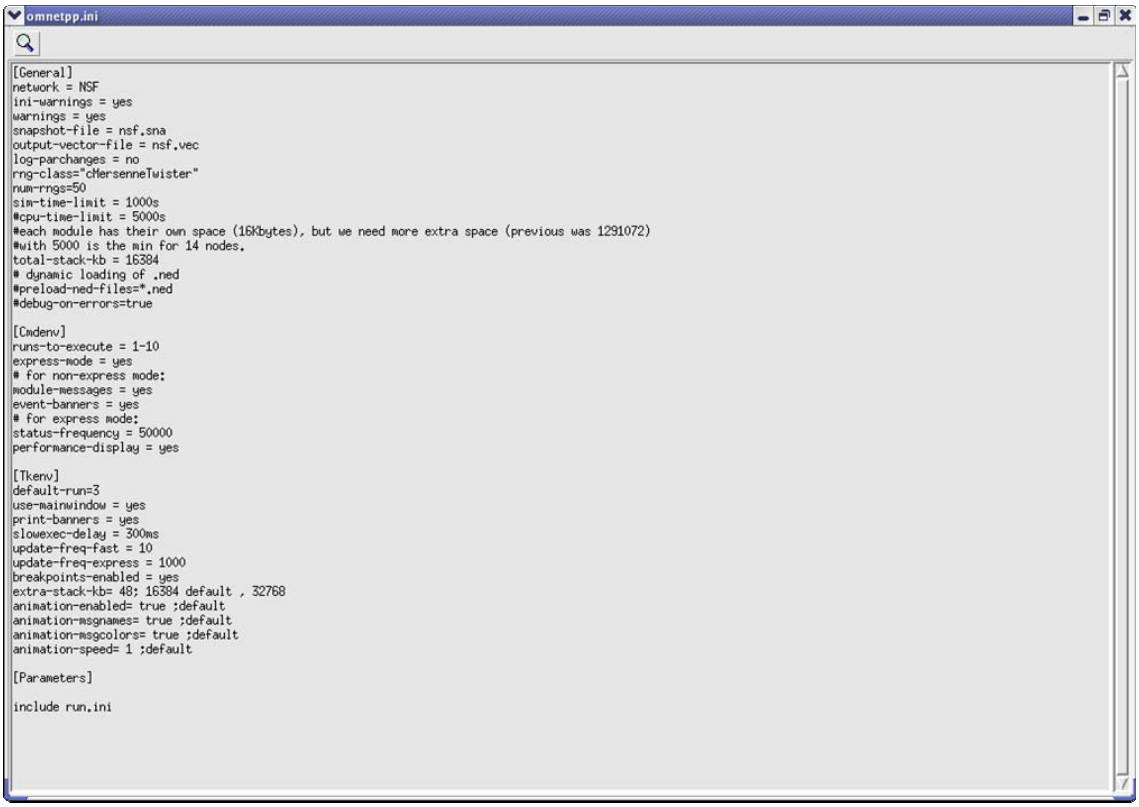
pow_ant2data: Paràmetre fet servit pel càlcul de la taula de formigues a la de dades.

En la segona línia trobem:

1. Els primers quatre paràmetres **han de coincidir** amb els del ant_gen.txt
2. pbw és el pes que li donem al reforçament de bw, i c1bw i c2bw els coeficients.
3. pjit és el pes que li donem al reforçament de jitter, i c1jit i c2jit els coeficients.
4. pdelay és el pes que li donem al reforçament del delay.

7.5. Configuració general del simulador

- **omnetpp.ini:** El principal fitxer de totes les simulacions és aquest. Es troba dividit en quatre seccions que a continuació les detallem:



Imatge 22: Arxiu de configuració de la simulació

[General]: En aquesta part es configuren aspectes que afecten a tota la simulació. Els valors més significatius serien:

Network: defineix quin tipus de xarxa fem servir per simular (es poden tindre més d'una xarxa creada).

Rng-class: ens diu quin generador de nombres aleatoris es fa servir. Per defecte fem servir el cMersenneTwister, però n'hi ha d'altres com cLCG32.

Num-rngs: Valor que diu quants nombres aleatoris es poden fer servir. Variant aquest valor també es varia la llavor del nombre aleatori, i s'aconsegueixen simulacions amb diferents resultats.

Sim-time-limit: Temps en que es para la simulació.

Total-stack-kb: Espai que reservem per guardar en memòria els mòduls.

[Cmdenv]: Aquest mode correspon a l'execució de simulacions sense entorn gràfic.
Runs-to-execute: definim quantes simulacions volem que executi. En pot ser una de sola, vàries (entre comes: 1,2,4), o tot un conjunt finit (1-4). Al Makefile s'ha d'escollir simulacions del tipus comandes.
Express_mode: amb "*yes*", s'executarà el més ràpid possible.

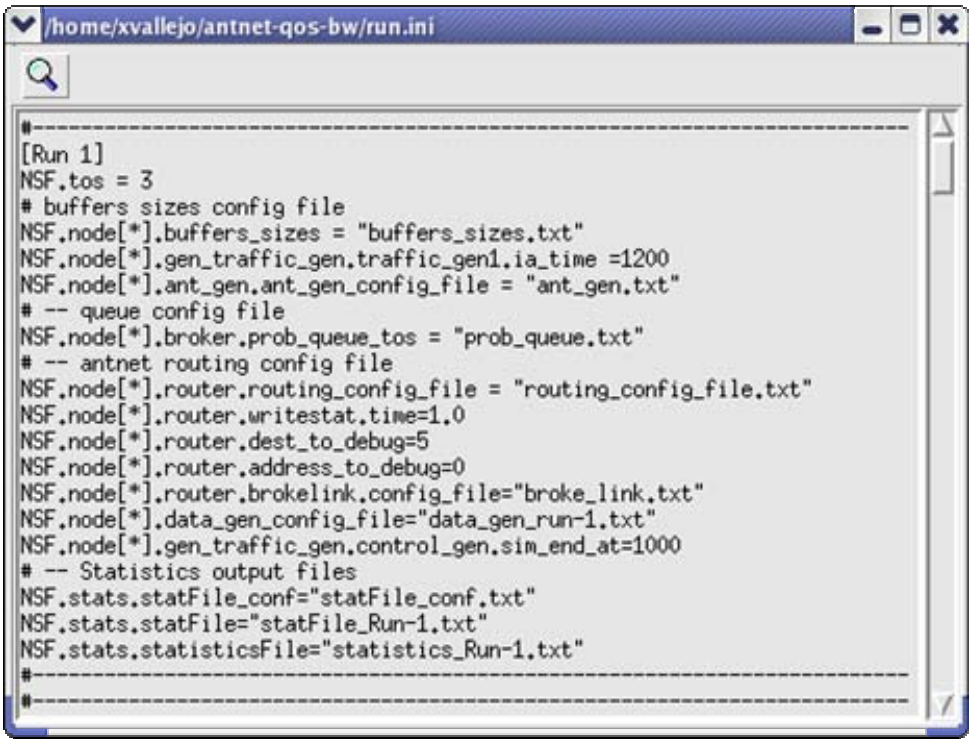
[Tkenv]: Aquest mode correspon a l'execució de simulacions amb entorn gràfic. Al Makefile s'ha d'escollir simulacions del tipus gràfic.
Default-run: En mode gràfic **només** podem executar una sola simulació.
Breakpoints_enabled: Útil si intentem depurar el codi en busca d'errors.

[Parameters]: S'inclou un fitxer .ini, en aquest cas run.ini, on es defineixen més aspectes de la simulació. Aquest apartat és opcional, però en el nostre algorisme en cal quan volem fer més d'una simulació utilitzant el entorn de comandes.

Si només volem fer una simulació amb l'entorn gràfic o una simulació amb l'entorn de comandes llavors podem copiar el contingut de [Run 1] dins dels [Parameters] del fitxer omnetpp.ini.

Existeixen molts més paràmetres i opcions que es poden definir. Un cop més, es recomana si es vol ampliar coneixements que revisi el manual d'usuari de l'OMNeT++.[6]

- **run.ini:**



Finalment, l'últim arxiu de configuració que s'ha definit és el run.ini. El fitxer realitza un pont entre omnetpp.ini i els fitxers de txt, ja que sinó seria impossible que els mòduls poguessin llegir les dades en el cas de fer més d'una simulació.

Com a paràmetres finals, destacar:
NSF.node[*].gen_traffic_gen.traffic_gen1.ia_time: Representa el temps que es genera trànsit dinàmic[5] (no confondre amb l'estàtic, que és el que hem definit anteriorment). Està posat per sobre del temps final de simulació, per això no se'n generarà.
NSF.node[*].gen_traffic_gen.control_gen.sim_end_at: Indica el temps en que s'atura de generar trànsit estàtic. En l'exemple, coincideix amb la fi de la simulació.

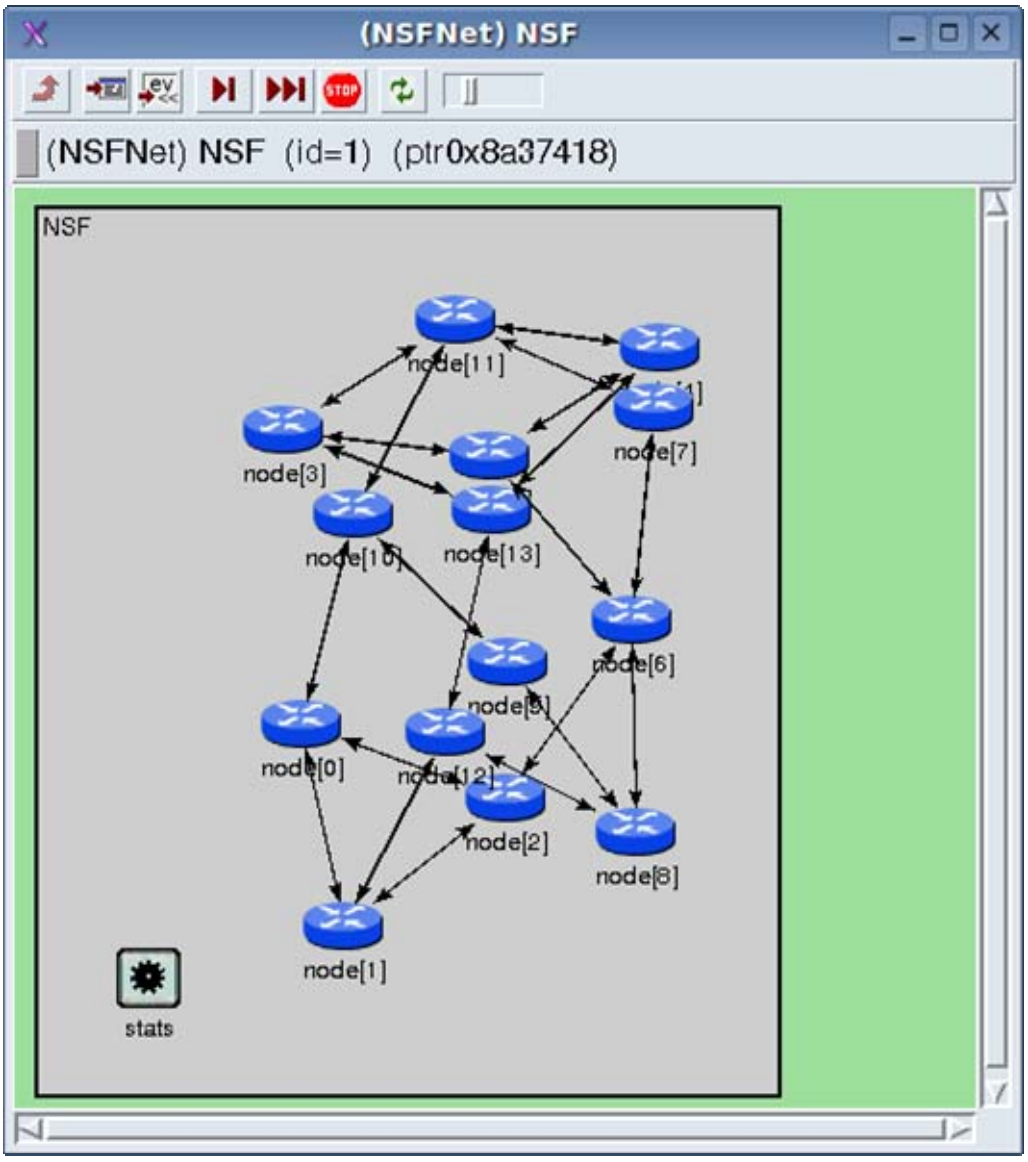
[5] Trànsit estàtic significa que tota l'estona envia un volum de dades més o menys constant. En canvi, el mòdul dinàmic és afectat segons l'estat de la xarxa, i pot augmentar o disminuir el volum generat. Pels nostres experiments, ens és més útil el trànsit estàtic i per això no fem servir el dinàmic.

8. Experiments i resultats

8.1. Canvis en taules d'encaminament

En aquest experiment s'ha intentat mostrar el comportament de les taules d'encaminament de les dades en una simulació.

La simulació té una durada de 1000 instants de temps (equivalen a 1000 segons aproximadament). La xarxa feta servir és la següent:



S'envia trànsit de dades des del node 0 fins al node 3, i en aquesta simulació es centrarà en analitzar el **node 0** que és el que ens repartirà el trànsit entre els tres camins possibles del veí 1, 2 i 10.

El node 0 enviarà trens de formigues, i formigues a tots els nodes, però per l'experiment interessa el que tinguin per destí el node 3.

La capacitat dels enllaços del node 0 són:

- 0->1: 10Mbps
- 0->2: 10Mbps
- 0->10: 3Mbps

Com es pot observar, el camí més curt per anar del node 0 al 3 és el 0->10->11->3.

Si l'analitzem des del punt de vista del retard, ens dóna un bon resultat, però amb la incorporació de la mesura de l'amplada de banda, no resulta un camí tant bo ja que l'enllaç entre el node 0 i el 3 és de 3 Mbps i no de 10 Mbps.

Un camí alternatiu seria el 0->1->12->13->3.

Es realitza un salt més (el delay serà més alt i, per tant, un resultat pitjor en la mesura de temps), però en canvi disposa d'una amplada de banda millor.

Si l'algorisme rep la nova informació de l'amplada de banda, hauria de ser capaç de redirigir una part del tràfic de l'enllaç 0->10 cap a l'enllaç 0->1, aconseguint no saturant-lo tant.

En **l'experiment**, s'ha definit que el ToS 0 enviï trens de formigues cada 20 instants de temps, i el ToS 1 i ToS 2 només envien formigues soltes cada segon.

Les probabilitats dels reforçament en el ToS 0 corresponen a: Bw: 70%, Jitter: 10%, Delay: 20%. Els ToS1 i 2 com que només fan servir delay, aquesta serà del 100%.

El tren de formigues consisteix en 8 formigues, i cada una amb un tamany de 700 bytes.

La càrrega de trànsit en la xarxa és del 30%, la qual cosa ens permet detectar si poden haver saturacions.

Dins dels 1000 instants de temps, generem del trànsit de la següent manera:

- En l'instant T=0, s'inicia la simulació, i per tant les probabilitats són iguals en tots els nodes per tots els destins. No es genera trànsit de dades.
- En T=200, es genera trànsit de dades de tipus CBR, tamany 4352 bits i de ToS=0 i dura fins al final de la simulació.
- En T=500, es genera trànsit CBR, 4352 bits i de ToS=1, i que durarà fins al final.
- En T=700, es genera trànsit CBR, 4352 bits i ToS=2, també dura fins al final.

Finalment, si es vol repetir l'experiment, s'ha realitzat amb la llavor aleatòria rng=45

Resultats (sense fer servir la funció squash):

30% trànsit	No squash 10Mbps	Llavor = 45 10Mbps	0->3 3Mbps	
T=0	**Veí 1**	**Veí 2**	**Veí 10**	
ToS=0	0,333	0,333	0,333	train=yes
ToS=1	0,333	0,333	0,333	train=no
ToS=2	0,333	0,333	0,333	train=no

T=200	**Veí 1**	**Veí 2**	**Veí 10**
ToS=0	0,003	0,99	0,003
ToS=1	0,564	0,434	0,001
ToS=2	2,00E-02	0,53	0,43

T=500	**Veí 1**	**Veí 2**	**Veí 10**
ToS=0	0,0002	0,89	0,101
ToS=1	0,013	0,108	0,877
ToS=2	2,20E-01	0,097	0,678

T=700	**Veí 1**	**Veí 2**	**Veí 10**
ToS=0	0,048	0,692	0,258
ToS=1	0,97	0,028	0,001
ToS=2	7,60E-01	0,183	0,056

T=1000	**Veí 1**	**Veí 2**	**Veí 10**
ToS=0	0,276	6,59E-06	0,723
ToS=1	7,92E-11	0,074	0,925
ToS=2	6,62E-05	0,997	0,002

- En l'instant 200, el trànsit del ToS=0 es dirigeix pel node 2 (una possible ruta que segueixi seria 0->2->6->5->3). En canvi, pel ToS=1 es decanta pel node 1 i 2. I pel ToS=2 es decanta pel node 2 i 10.
- En T=500, el ToS0, la majoria del trànsit continua decantant-se pel node 2, però augmenta lleugerament el veí 10. El ToS1 ha variat la seva ruta, i ara té més probabilitat el node 10. Per últim, el ToS2 ha disminueix el node 2 per augmentar els nodes 1 i 10.
- En T=700, pel ToS0 continua disminuint el veí 2 per augmentar els altres 2. El ToS1 ha hagut un canvi dràstic al concedir la prioritat al node 1. Una possible causa a aquest canvi consisteix en que ja s'ha començat a enviar tràfic del ToS1, i s'ha detectat que pel veí 10 no és suficient, i llavors el veí 1 es pot dirigir millor el trànsit. Finalment, el ToS2 ha seguit augmentant el node 1.

- En T=1000, el ToS0 ha canviat de prioritat i es dirigeix cap al veí 10. També ToS1 ha sofert un altre canvi i es dirigeix pel veí 10. I en ToS2 predomina el node 1.

En resum, es destaca que hi han forces canvis i que les taules varien bastant segons les condicions puntuals de la xarxa i l'algorisme s'adapta a les variacions en el trànsit per aconseguir el millor per cada ToS. Si observem les estadístiques de la simulació, s'observa:

SimTime: 1000
ToS: 0
Sample: 73961
mean delay: 0.0639954
Variance delay : 0.432981
bit/time: 323496
troughtput(tot) mean:247256
troughtput(tot) var: 80115.6
Lose packet for ttl: 2959
Lose packet for queue: 0
Arrived packet : 73961
Lose Ant for ttl: 88
Lose Ant for queue: 0
Arrived Ant (back anf forw) : 6075
% ant/data arrived: 10.7408

ToS: 1
Sample: 72358
mean delay: 0.0111847
Variance delay : 0.006673
bit/time: 316484
troughtput(tot) mean:194484
troughtput(tot) var: 88454.8
Lose packet for ttl: 0
Lose packet for queue: 0
Arrived packet : 72358
Lose Ant for ttl: 0
Lose Ant for queue: 0
Arrived Ant (back anf forw) : 27274
% ant/data arrived: 2.57622

ToS: 2
Sample: 79478
mean delay: 0.0153726
Variance delay : 0.128325
bit/time: 347626
troughtput(tot) mean:194518
troughtput(tot) var: 99485.6
Lose packet for ttl: 97
Lose packet for queue: 0
Arrived packet : 79478
Lose Ant for ttl: 0
Lose Ant for queue: 0
Arrived Ant (back anf forw) : 26826
% ant/data arrived: 2.30558

En el ToS0, hi ha hagut algunes pèrdues de paquets, ja que no han pogut arribar al seu destí. Les pèrdues de paquets s'han produït perquè en un moment de la simulació, hi ha hagut una saturació en les cues, la qual cosa deguda amb la constant generació de trànsit, ha provocat que es morin per temps de vida.

També hi ha hagut pèrdues de formigues, però és un valor insignificant comparat amb el total de formigues arribades.

8.2. Comparació del rendiment dels diferents algorismes AntNet-QoS

Per comprovar les millores que ha pogut obtenir el nou algorisme, s'ha comparat amb la versió anterior AntNet-QoS. Les simulacions s'han realitzat en un dels servidors del laboratori BCDS, ja que sense aquesta ajuda, la simulació hauria estat molt més lenta i difícilment realitzable.

Els experiments (i cada punt de les següents gràfiques) s'han realitzat amb 10 llavors de nombres aleatoris diferents (per evitar possibles resultats falsos). A més a més, els dos algorismes s'han executat amb 10 tipus diferents de càrrega, des de 0% fins al 100%, en intervals de 10%. Per tant, per cada algorisme s'han executat 100 simulacions diferents (cadascuna d'elles de 1000 instants de temps).

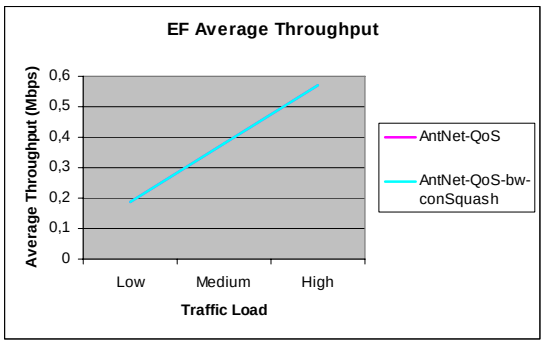
Tota la càrrega s'ha classificat en tres tipus: **càrrega baixa, mitja i alta**. En canvi, el trànsit, al ser una xarxa DiffServ, s'ha classificat en EF, AF i BE.

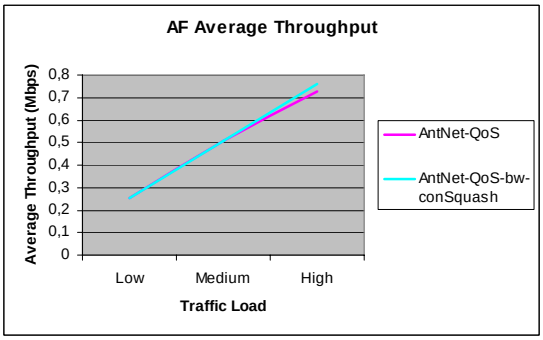
Es fan servir 3 tipus de servei: el ToS 0 enviï trens de formigues cada 20 instants de temps, i el ToS 1 i ToS 2 només envien formigues soltes cada segon.

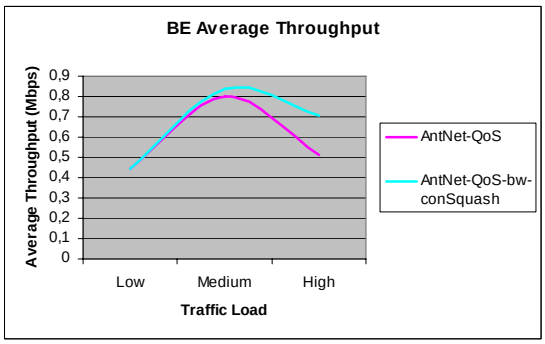
Les probabilitats dels reforçament en el ToS 0 corresponen a: Bw: 70%, Jitter: 10%, Delay: 20%. Els ToS1 i 2 com que només fan servir delay, aquesta serà del 100%.

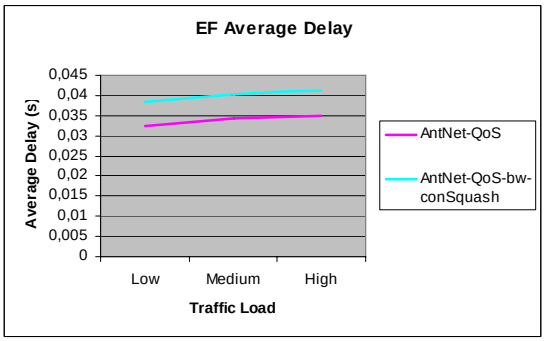
El tren de formigues consisteix en 8 formigues, i cada una amb un tamany de 700 bytes.

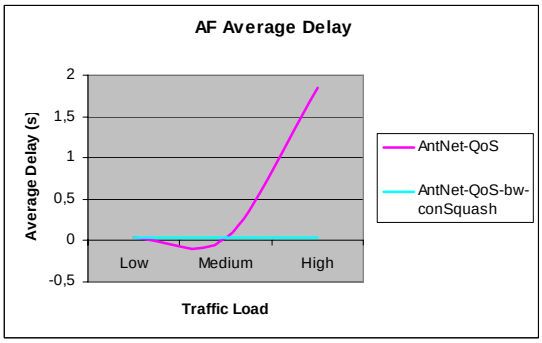
L'estructura de la xarxa és la mateixa explicada que en el primer experiment.

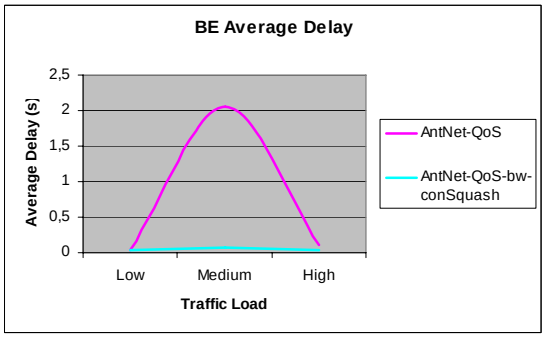




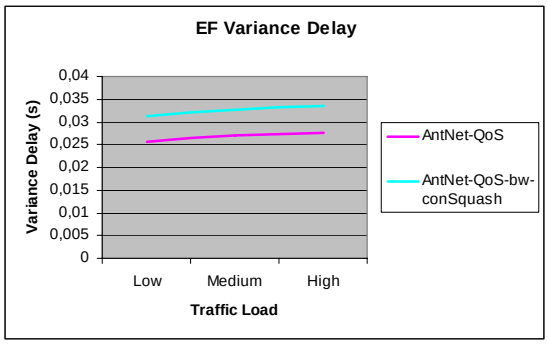


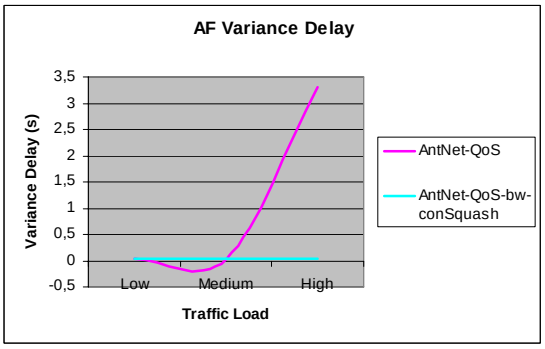


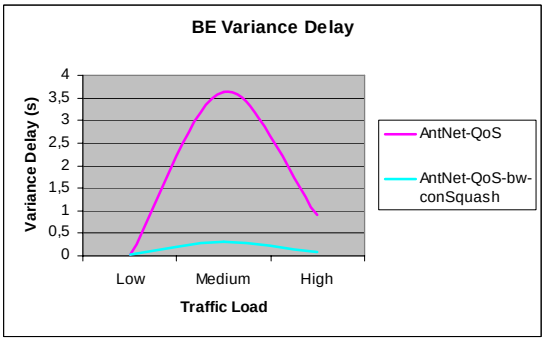


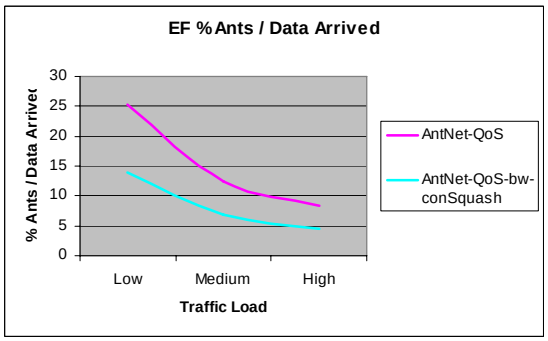


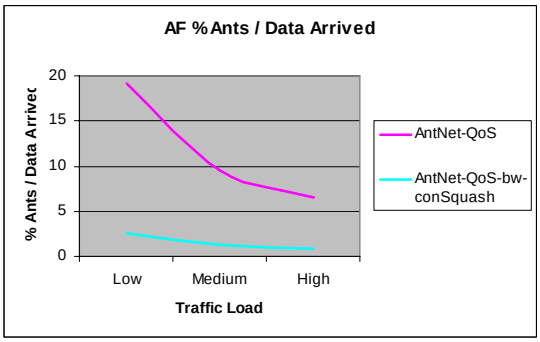
En aquesta pàgina mostrem les gràfiques de la mitjana de la velocitat (throughput) i del retard.

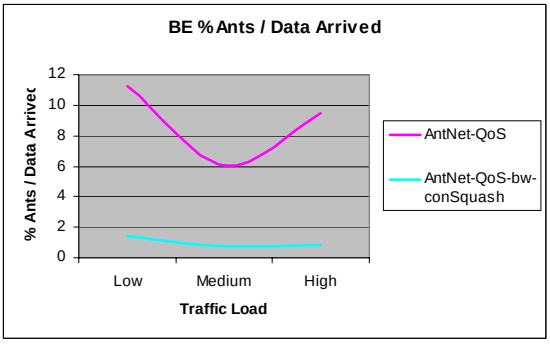












I aquestes gràfiques representen la variància del retard i el % entre formigues i dades rebudes en els nodes.

Comentaris sobre els resultats:

- En general, hi han millores en el throughput respecte de l'AntNet-QoS anterior, el qual vol dir que en el nou AntNet-QoS hi han menys pèrdues.
- Hi han millores quant al retard i la variació del retard, i totes aquestes millores amb la gran avantatge de que estem generant menys overhead (temps d'execució que no forma part de l'usuari). És a dir, ens calen generar menys quantitat de formigues amb respecte del total de dades... i menys que a l'AntNet-QoS original.
- El retard de la classe BE baixa quan hi ha molta carrega a la xarxa, perquè si es mira també la gràfica de throughput, ens podem donar que hi han pèrdues o menys throughput... Significa que, amb càrrega alta, hi ha una disminució en el retard que no es verdadera, sinó que és el reflex de pèrdues de paquets.
- Respecte a les gràfiques de retard i variació del retard de la classe EF, l'AntNet-QoS sense anàlisis de bw es comporta millor... En canvi, cal comptar que la classe EF està buscant el camí amb mes throughput o velocitat i això no el pot aconseguir amb el camí mes curt, és a dir, el de menys delay. Al final s'aconsegueix que aquest transit viatgi per camins de menys throughput i és possible que es trobi amb que té una mica més de retard.
- En resum, observem que els resultats obtinguts amb les noves millores són bons respecte l'algorisme original.

9. Conclusions i Treballs Futurs

Aquest projecte final de carrera m'ha permès desenvolupar una nova línia en la investigació de nous algorismes en la gestió de trànsit en xarxes DiffServ. A partir de l'algorisme (AntNet-QoS), i amb la teoria que m'han ofert els diversos documents citats en la referència, s'ha intentat implementar una nova filosofia, si més no, fer-ne una primera aproximació.

El treball ha estat difícil i complicat per diverses raons. En primer lloc, he hagut de familiaritzar-me amb la filosofia de les formigues de l'AntNet, i després comprendre el codi font de l'algorisme. A més a més, he hagut de dominar el simulador de xarxes OMNeT++. Un cop adquirit els coneixements previs, llavors ha calgut de fer una recerca en busca de millores aplicables a l'algorisme AntNet-QoS que permetessin una millora en la gestió del trànsit. Finalment, m'ha calgut implementar les noves idees i realitzar vàries proves per obtenir uns resultats.

Entre les millores obtingudes, cal destacar, que a partir de l'algorisme AntNet-QoS s'han incorporat noves mesures (mesura de l'amplada de banda disponible i jitter), les quals combinant-les amb la mesura de delay (retard) ja feta servir, ens permet adaptar-nos millor a les condicions actuals del trànsit en la xarxa i als requeriments específics de qualitat (QoS) per part del trànsit.

Cal recordar, que és un projecte que forma part d'un treball de recerca més ampli, i per tant hi han moltes coses a fer i millorar....

Com possibles treballs futurs, en destaco:

1. *Si per problemes o variacions en la xarxa, l'última formiga d'un tren no hi arriba, llavors no es realitzen els càlculs i tot el tren de formigues no servirà per actualitzar les taules d'encaminament. Hi ha altres propostes per intentar solucionar aquest problema. Una d'elles consisteix en intentar aprofitar la resta de paquets i fer mitjanes. A més a més, també es pot donar el cas que l' última formiga no s'hagi perdut, i que pugui arribar al destí però molt més tard (a causa de retencions en cues...), llavors una altra idea seria incorporar algun sistema de control, com poden ser temporitzadors... En tot cas, són propostes de millores.*
2. *Una altra millora que es podria realitzar consistiria en facilitar el procés de configuració de l'algorisme. Actualment es troba dispersa en varis fitxers de*

text, i encara que es pot manipular amb una certa facilitat, no deixa de ser complicat al principi. Una altra possibilitat consisteix en crear un programa en un entorn gràfic, que permetés modificar aquests fitxers, i a més a més, podria comprovar si les dades que entra l'usuari poden ser errònies.

3. *Les taules d'encaminament són massa sensibles a les variacions de trànsit, i poden rebre massa reforçament. Caldria controlar els valors dels reforçaments, i la quantitat de cops que han de realitzar les actualitzacions de les taules. En canvi, si es fa servir la funció squash, els valors de reforçament són massa petits, i per tant també caldria optimitzar-los.*
4. *Podem comentar que es podria fer un estudi més detallat sobre la quantitat de formigues que han de ser part del tren i el tamany que s'ha d'utilitzar per veure què tan aproximades son les mesures al variar aquests paràmetres. I així es comprova millor el que està escrit a l'article [10].*

Finalment, l'esforç que he fet ha estat molt positiu perquè m'ha permès ampliar els meus coneixements en xarxes.

A nivell de programació, també he aprés molt, ja que he hagut de treballar el codi font amb altres companys, col·laborant en equip, implementant noves funcionalitats, revisant-lo amb depuradors de codi, etc. Alhora, els meus coneixements en Linux han augmentat significativament treballant-hi amb ell (configuració, instal·lació del simulador, eines de compilació, etc), i en simuladors basats en events discrets.

A més a més, treballar en el projecte m'ha servit per explorar la meva faceta com a investigador, ja que aquest és un projecte molt lligat a la investigació.

Per tot això, el projecte m'ha estat un complement molt important dels estudis efectuats durant els meus anys de la carrera d'Enginyeria Tècnica en Informàtica de Sistemes.

10. Referències

[1] DI CARO, G. i DORIGO M.: *AntNet: Distributed Stigmergetic Control for Communications Networks*, IRIDIA Université Libre de Bruxelles, Desembre 1998, Journal of Artificial Intelligence Research 9, pp. 317-365

[2] CARRILLO, L.; GUADALL, C.; MARZO, J. L.; DI CARO, G.; DUCATELLE, F.; GAMBARDELLA, L.M. : *AntNet-QoS: a Quality of Service Scheme Based on the Use of Multiple Classes of Ant-like Mobile Agents,* Institut d'Informàtica i Aplicacions, Universitat de Girona; Istituto Dalle Molledi Studi sull'Intelligenza Artificiale , Switzerland

[3] The Network Simulator ns-2. En http://www.isi.edu/nsnam/ns/

[4] QualNet. En http://www.scalable-networks.com/

[5] OMNeT++. Programa creat per András Varga en http://www.omnetpp.org/

[6] VARGAS, A.: *OMNeT++ Discrete Event Simulation System Version 3.2 User Manual*, 29 Març 2005

[7] GUADALL, C.: *Incorporació de Qualitat de Servei a l'algorisme d'encaminament AntNet*, Setembre 2004

[8] DE TROYER, T.: *Cooperation Strategies between Ant Agents in AntNet-QoS*, Juny 2006

[9] COLMAN, L: *Strategies to build an autonomic routing system with AntNet-QoS*, Juny 2006

[10] HU, N. i STEENKISTE, P.:*Evaluation and Characterization of Available Bandwidth Probing Techniques,* IEEE Journal on selected areas in Communications, Vol 21, nº6, Agost 2003, pp. 879-894

[11] WANG, J. i NAHRSTEDT, K.: *Hop-by-Hop Routing Algorithms For Premium-class Traffic In DiffServ Networks*, Department of Computer Science, University of Illinois at Urbana-Champaign, Juny 2002

[12] WANG,J.: *Load Balancing In Hop-by-Hop Routing With and Without Traffic Splitting,* University of Illinois at Urbana-Champaign, 2003

[13] TADRUS,S i BAI,L.: *Qcolony: a Multi-pheromone Best-fit Qos Routing Algorithm as an Alternative to Shortest-path Routing Algorithms*, School of Computer Science and IT, University of Nottingham, International Journal of Computational Intelligence and Applications Vol. 5, No. 2 (2005) pp. 141-167

[14] S. BLAKE; D. BLACK; M. CARLSON; E. DAVIES; Z. WANG; W. WEISS.: *An Architecture for Differentiated Services,* Network Working Group, RFC 2475, Desembre 1998

[15] K. NICHOLS; S. BLAKE; F. BAKER; D. BLACK.: *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers,* Network Working Group, RFC 2474, Desembre 1998

[16] OKUMUS, I; MANTAR, H; HWANG, J; CHAPIN, S.: *Inter-Domain QoS Routing on Diffserv Networks: A Region Based Approach,* Mugla University, Syracuse University, Agost 2002

[17] MANTAR, H; HWANG, J; OKUMUS, I; CHAPIN, S.: *A Scalable Intra-Domain Resource Management Scheme for Diffserv Networks*, Harran University, Syracuse University, Seoul National University.

[18] ZHANG, P i KANTOLA, R.: *Mechanisms for Inter-Domain QoS Routing in Differentiated Service Networks,* Laboratory of Telecommunication Technology, Helsinki University of Technology, 2000

[19] Apunts de Xarxes, Arquitectures Avançades de Computadors i de Serveis Públics de Transmissió de Dades.