Escola Politècnica Superior
de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: ESTUDI I SIMULACIÓ DE NETWORK CODING

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Pau Pericay Vendrell

DIRECTOR: Jesús Alcober Segura

CO-DIRECTOR: Alberto José González Cela

DATA: 23 de setembre del 2010

Títol: Estudi i simulació de Network Coding

Autor: Pau Pericay Vendrell

Director: Jesús Alcober Segura

Co-Director: Alberto José González Cela

Data: 23 de setembre del 2010

Resum

Actualment, el sistema de transmissió de les xarxes que formen les nostres comunicacions està basat en què els dispositius intermediaris reben la informació per un canal d'entrada i la tornen a enviar per les interfícies de sortida que tinguin disponible, així com a màxim copien la informació per enviar-la per totes les interfícies que tingui. Dit d'una altre forma, la xarxa només transporta sense cap intel·ligència, la intel·ligència està als nodes finals.

En aquest treball hem estudiat la tècnica anomenada “Network Coding” proposada per primer cop l'any 2000 per Alshwede et. al. que es basa en trencar amb aquest paradigma utilitzant els nodes intermediaris de la xarxa per fer operacions de codificació amb la informació. Això fa possible que la informació, els paquets en el cas de Internet, aconsegueixin millores en el “throughput” i en el retard respecte al mètode de encaminament tradicional.

El “Network coding” es pot aplicar en diferents entorns, cablejats, fixes, mòbils o sense fils, aconseguint beneficis com és la millora del retard, però també millores en seguretat o per exemple en l'eficiència dels recursos en l'entorn sense fils. Un dels aspectes més interessants que proporciona és la capacitat “multicast”.

En aquest treball em fet primer la part teòrica, tant les seves característiques com els diferents modes de funcionament depenent de les operacions que es realitzin per fer la codificació. Aprofundim amb els conceptes matemàtics que hi han darrera tot explicant les funcions o aplicacions en que pot ser beneficiós. Després de la part teòrica provem el seu funcionament amb la simulació d'una xarxa que implementa aquesta tècnica amb el programa OPNET fent una simulació d'una transmissió d'informació en una xarxa de papallona (“Butterfly network”) fent servir la codificació XOR, i fem el mateix però més teòricament amb una simulació de la codificació LNC (“Linear Network Coding”) amb el MATLAB.

Title: Simulation and Study of Network Coding

Author: Pau Pericay Vendrell

Director: Jesús Alcober Segura

Co-Director: Alberto José González Cela

Date: September 23th 2010

Overview

Currently, the transmission system of networks that make up our communications devices, is based on that the information intermediaries receive an input channel and resubmit to the output interfaces that are available, as well as the maximum copy information to send it to all interfaces that you have. Is like to say that the network only carries with no intelligence, intelligence is in the final nodes.

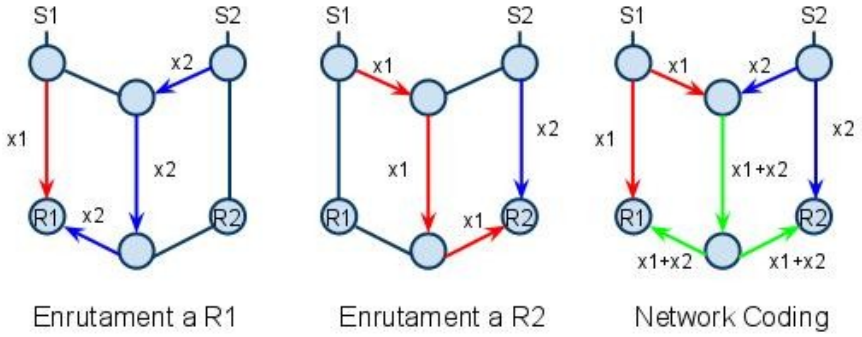
In this work we studied the technique called "Network Coding" first proposed in 2000 by Alshwede et. al. , based on breaking with this paradigm using the intermediaries nodes of the network for operations with the encoding information. This allows information packets, in the case of the Internet, to achieve improvements in the "throughput" and the delay with respect to traditional routing methods.

The "Network coding" can be applied in different environments, wired, fixed, mobile or wireless, achieving benefits such as improvement of the delay, throughput, but also improvements in security or for example in resource efficiency in the wireless environment . One of the most interesting aspects is that provides multicast capability.

In this work we have done first the theoretical part, both their characteristics and the different operating modes depending on the operations carried out for coding. Deepen the mathematical concepts behind explaining that there are functions or applications that can be beneficial. After the theoretical part we test its performance with the simulation of a network that implements this technique with the program using OPNET simulation of a data transmission network in a butterfly (Butterfly network) using XOR encryption and do the same but with a theoretical simulation of the encoding LNC ("Linear Network Coding") with MATLAB.

Índex

CAPÍTOL 1. INTRODUCCIÓ



Les xarxes d'avui en dia permeten la transmissió de tot tipus de dades o media (textos, fotos, imatges o vídeos) i és un fet de que cada cop el contingut P2P i el vídeo representen un percentatge major (veure [6]).

La transmissió d'aquesta informació és pot realitzar de forma “unicast” (1:1), “multicast” (1:N), “broadcast” o “anycast”. Com també altres mètodes menys populars com el “geocast”. No obstant, un problema actual són les comunicacions “multicast” ja que existeixen encaminadors que no el suporten, tant sols implementats en alguns dispositius dintre les xarxes dels operadors o els problemes de seguretat intrínsecs, entre altres complicacions. Per això podem fer una implementació al nivell d'aplicació, com és el “Bittorrent” o “l'Emule” en fitxers P2P o el “Zattoo” en vídeo, tot i que resultarà no tant efectiva com fer la implementació a nivells inferiors.

Però una premissa fonamental de les xarxes és la forma en que es tracta la informació, ja siguin paquets de Internet o senyals en la xarxa de telefonia, es transporten de la mateixa forma que els cotxes en una autopista o els líquids a través d'una xarxa de canonades. És a dir, els fluxos de dades comparteixen els recursos, però la informació en si és independent. Avui en dia, l'encaminament, l'emmagatzematge de dades, el control d'errors, i en general totes les funcions de xarxa tenen en comú aquest principi.

Durant aquests últims anys el processament computacional s'ha anat abaratint d'acord amb la llei de “Moore” i el coll d'ampolla ha canviat a l'ample de banda de la xarxa al tenir de suportar la creixent demanda per part de les aplicacions que inunden les xarxes.

Això va portar a l'observació de que els nodes no només poden tornar a enviar la informació, sinó que també poden processar els fluxos d'entrada independents, representant una ruptura del model anterior de la independència dels fluxos de dades.

El “Network Coding” es basa en trencar aquest paradigma on els paquets de sortida d'un node són recombinats per estalviar-se'n l'enviament de varis a

només un o pocs. Així amb la realització d'una operació aconseguim una gran millora de rendiment.

Aquesta tècnica suposa un canvi de mentalitat que pot proporcionar grans beneficis com poden ser una millora del rendiment de les xarxes,més robustesa en la pèrdua de paquets o seguretat en el medi de forma natural. Tot i així, actualment hi han bastants dubtes sobre si mai s'arribarà a aplicar de manera massiva a les xarxes a causa de la dificultat de aplicar-ho en l'estructura actual.

Això és a causa de que es pot aplicar la codificació al nivell d'aplicació, cosa que tornarà la xarxa ineficient, o als nivells més baixos, on ja serà més eficient però serà molt costós implementar.

A més, estudis demostren que en els esquemes “multicast” el “Network Coding” obté gran avantatges, ja que amb la implementació a nivell d'aplicació obté millors resultats que les altres tecnologies de encaminament i la implementació a nivell de “hardware” té la problemàtica de que actualment casi cap equip ho suporta i per tant representa un cost molt elevat.

A la pràctica sembla que el principal competidor en aquest àmbit, la tecnologia P2P, continua essent preferida, tot i que existeix un gran esforç per part de la comunitat científica per tal de millorar la tècnica i per tant s'obre la porta a un nou horitzó en les comunicacions entre les màquines.

1.1 Objectius

Aquest treball es centra en comprendre i estudiar la tècnica anomenada “Network Coding” amb el nou paradigma que aporta. Farem una síntesi dels avantatges i inconvenients que presenta amb les possibles aplicacions d'ús que pugui tenir. Explicarem el seu funcionament, amb les característiques matemàtiques necessàries per la comprensió dels algoritmes, i anomenarem els diferents modes de operació que existeixen. Farem un disseny d'una xarxa amb Network Coding usant primer el mode de operació de XOR i després usant el que s'anomena “Linear Network Coding”, amb la posterior simulació de les ambdues xarxes. També explicarem detalladament el procés que hem seguit per tal de realitzar les simulacions, tant per el procés de instal·lació i configuració del software necessari, com per fer el disseny amb les eines proporcionades. I finalment anunciarem els resultats obtinguts i els contrastarem.

CAPÍTOL 2. Fonaments de Network Coding

2.1 Introducció a Network Coding?

El “Network coding”, la traducció en català seria: codificació en la xarxa, és un mecanisme de transmissió de dades del qual la principal característica és que modifica la informació de transport en els nodes intermedis que formen la xarxa. Aquest fet suposa trencar amb el paradigma de la forma de tractar els fluxos de dades que Internet ha conservat des de la seva creació. Aquesta modificació que es realitza en els nodes intermedis de la xarxa es basa en la fusió de les dades gràcies a operacions per tal d'aconseguir beneficis en la transmissió. Va ser proposat per primer cop l'any 2000 en l'àmbit de teoria de la informació (veure [1]) per Ahlswede et. al. en el document “Network information flow” i posteriorment s'han anat fent varis estudis per tal de millorar-ne el funcionament i adaptant-ne l'ús en diferents medis. En el fons, el que es va proposar en aquest document, és deixar de tractar la informació com un fluid i tractar-la de manera individual.

Els avantatges que suposa aquest mètode es veuen repercutits directament en el rendiment de la xarxa quant es realitza “multicast” com és l'estalvi de bateria, ample de banda, retard i en seguretat. Resumint, l'objectiu del “Network Coding” és augmentar el rendiment de la xarxa i reduir el nombre de transmissions necessàries.

2.2 Avantatges i desafiaments

Com hem esmentat abans hi han varis avantatges a l'hora de utilitzar aquesta tècnica però també hi han desafiaments que de moment no s'han pogut solucionar. Cal dir que els avantatges i desavantatges que existeixen van molt lligats a la topologia de la xarxa que ens trobem.

A la següent taula hi trobarem un resum dels avantatges i els desavantatges, ho sigui els desafiaments actuals, en la utilització del “Network Coding”:

Taula 2.1 Avantatges i desafiaments principals

AVANTATGES	DESAFIAMENTS
Millora “throughput” de la xarxa	Complexitat computacional
Eficàcia dels recursos Wireless	Seguretat no suficient
Seguretat	Integració amb la infraestructura
Robustesa en pèrdua de paquets	Cost econòmic
Resistència a falles d'enllaços	
Millora el retard de les transmissions	
Eficiència energètica	

Tot seguit comencem explicant millor els avantatges que aporta la tècnica:

2.2.1 Avantatges

2.2.1.1 El Rendiment

El benefici més demostrat en aquesta tècnica és el rendiment que es treu a l'hora de fer "multicast". En la següent Fig. 2.1 podem veure un exemple dels beneficis de usar el "Network Coding":

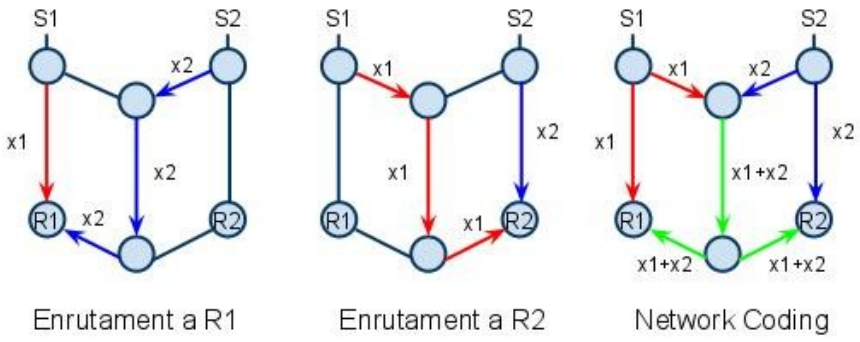


Fig. 2.1 Comparació encaminament amb Network Coding

En aquesta figura mostra una xarxa representada com un graph directe on els vèrtexs corresponen a terminals i les arestes corresponen a enllaços. Aquesta xarxa es coneix amb el nom de xarxa de papallona ("Butterfly Network", en anglès). A més assumim que el temps és ranurat i que per cada enllaç només podem enviar un bit per cada ranura de temps. Hi han dos fonts (S1 i S2) i dos receptors (R1 i R2). Cada font produeix un bit a cada ranura de temps que representem amb x1 i x2 respectivament.

Els dos receptors volen rebre la informació de les dues fonts al mateix moment, o sigui estem en un model de "multicast". Com es pot observar a l'enllaç del mig tenim un coll d'ampolla, ja que només podem enviar un bit a l'hora; el que van proposar Ahlswede et. al. és al node intermedi fer una operació de suma dels dos bits que es tenen d'enviar així podem transmetre x1 + x2 en una sola ranura temporal. Crearem un tercer bit, l'anomenarem x3, que serà la suma de x1+x2, que viatjarà per l'enllaç del mig i després es tornarà a enviar a R1 i R2. El que hauran de fer els receptors és una operació amb aquest bit i el rebut directament de les fonts per tal d'aconseguir l'altre bit que no tenen.

Explicarem aquest exemple més en detriment en l'apartat Funcionament del "Network Coding", però podem observar les millores que en aquest cas de "multicast" dona en qüestió de "throughput" i de retard.

2.2.1.2 Els Recursos en les xarxes sense fils

En un escenari on la transmissió es realitzi en un medi sense fils, el “Network Coding” proporciona un estalvi de bateries, d'ample de banda i de retard.

A la següent figura considerem una xarxa sense fils ad-hoc on podem apreciar una topologia de tres nodes on A i B han d'aconseguir tenir els arxius binaris de l'altre amb l'inconvenient que no es veuen entre ells, l'únic que veuen és el node S. El nodes tenen la particularitat de que només tenen una sola interfície i per tant només poden acceptar informació d'una sola font al mateix temps. Com és d'esperar amb el model tradicional, el node S representarà un coll d'ampolla ja que l'hi arribaria els bits “a” i els “b” i només podria retransmetre un dels dos al mateix temps. Usant el “Network Coding” podrem transmetre els dos arxius binaris al mateix temps fent una simple suma xor dels bits de “a” i “b”,com es pot apreciar a la Fig. 2.2 :

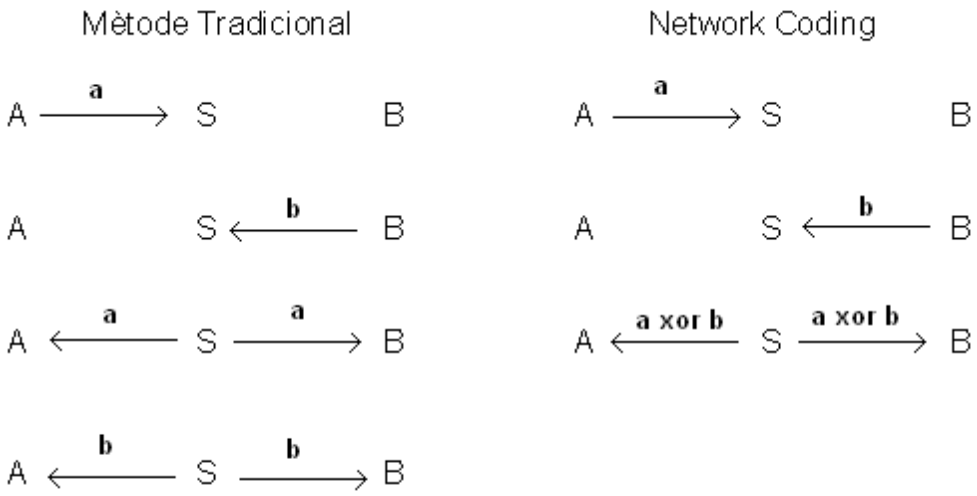


Fig. 2.2 A l'esquerra la transmissió tradicional sense fils i a la dreta el model Network Coding amb el medi inalàmbric.

L'únic que hauran de fer els nodes A i B és una altre operació xor entre els bits rebuts del node intermedi S i els seus propis de l'arxiu binari, per aconseguir l'arxiu de l'altre node.
Els beneficis de aplicar el “Network Coding” es veuen en l'estalvi d'energia, el node S només transmet un cop en lloc de dos, en el retard, hi ha una transmissió menys que en el mètode tradicional, en l'ample de banda wireless, el canal sense fils és ocupat menys temps, i hi haurà menys possibilitats d'interferència ja que ocupem menys temps un canal.

2.2.1.3 La Seguretat

El fet d'enviar combinacions lineals de paquets enlloc de dades no codificades ofereix una forma natural avantatjosa de la diversitat multi camí contra els atacs

de escolta “wiretapping”. Per tant és una forma de protecció per els sistemes que requereixin un baix nivell de seguretat que ve de forma implícita en el mecanisme i per tant només és vàlid contra atacs simples com aquest.
Per posar un exemple tenim de considerar quatre nodes, A,B,C i D on A és la font que envia informació al node receptor D. Hi han dos possibles camins ABD o ACD com en la següent Fig. 2.3:

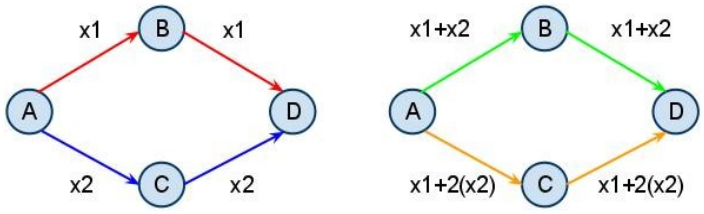


Fig. 2.3 En el graf de l'esquerra no enviem codificat i l'obtenció de x1 o x2 és fàcil si escoltem el medi, a la dreta al codificar les dades l'usuari malintencionat no sabrà reconèixer cap bit.

Ara suposem que hi ha un usuari malintencionat, en Carles, que amb l'atac de “wiretap” vol esbrinar quina informació enviem, però únicament pot escoltar un camí i no tindrà accés a l'altre complementari. Si els símbols son enviats no codificats com és en el primer graf de l'esquerra en Carles podrà llegir un dels dos símbols. En canvi amb les combinacions lineals dels símbols de la dreta enviades per camins diferents, en Carles no podrà llegir cap part de la informació. Si per exemple arribés a llegir x1+x2 les seves probabilitats de endevinar un dels dos símbols és del 50%, la mateixa probabilitat de endevinar-ho aleatòriament.

2.2.1.4 Robustesa a la pèrdua de paquets

La pèrdua de paquets sol ser un gran problema, sobretot en les xarxes sense fils, que pot venir des de el tall d'un enllaç, la sobrecàrrega d'un buffer o col·lisions en el medi.
Hi han bastants solucions per aquest problema, el més directe potser és l'ús del mecanisme que té el protocol TCP, amb un sistema de reconeixement de paquets coneguts com a ACK, on quant els paquets de informació son rebuts per el receptor tot seguit son reconeguts amb un missatge enviat enrere a la font i, si la font no rep el missatge de reconeixement per un paquet particular, torna a retransmetre el paquet.

Una altre solució és la codificació del canal, també conegut per “erasure coding”. Aquesta tècnica aplicada al node font introdueix un grau de redundància als paquets així els missatges poden ser recuperats encara que només siguin un subconjunt els paquets enviats per la font que hagin arribat bé.

Erasure coding s'aplica a la font, però si ho apliquem en els nodes intermedis com és el cas del Network Coding també servirà com a protecció de la informació. Per entendre aquesta situació ho farem amb l'exemple de la figura següent Fig. 2.4:

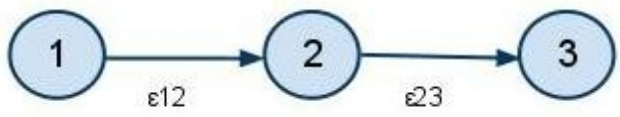


Fig. 2.4 Tres nodes units per dos enllaços, amb probabilitat de pèrdua de cada enllaç de ε_{12} i ε_{23}.

Considerarem la següent xarxa on hi han tres nodes i estan connectats per dos enllaços de probabilitat de pèrdua ε_{12} per l'enllaç entre el node 1 i el 2, i de ε_{23} entre el 2 i el 3.

Aplicant el "erasure coding" correcte al node font aconseguim una taxa de $(1-\varepsilon_{12})(1-\varepsilon_{23})$ paquets per unitat de temps. Però si apliquem dos codificacions per cada enllaç amb la descodificació i codificació al node 2, aconseguirem uns millors resultats, ho sigui aconseguirem una taxa de comunicació de $(1-\varepsilon_{12}, 1-\varepsilon_{23})$ que resulta casi sempre major que $(1-\varepsilon_{12})(1-\varepsilon_{23})$.
El problema d'aquesta tècnica rau en el retard, ja que ens tenim d'esperar un cert temps a rebre el número de paquets necessaris per poder descodificar, així si ho apliquem per cada enllaç el retard creixerà de manera molt significativa.

El fet és que "erasure coding" podríem dir que és un estil de "network coding", que aplicat en sectors de la xarxa determinats proporciona robustesa contra la pèrdua de paquets cosa que es tradueix en una millora de la eficiència de la xarxa. Segons estudis el "Random Linear Network Coding" (RLNC) presenta unes característiques bones per aconseguir aquesta robustesa.

2.2.1.5 Resistència a falles d'enllaços

Amb "Network Coding" fem la xarxa mes robusta contra les falles dels enllaços gràcies a la redundància dels camins que enviem les dades. Un exemple seria la següent xarxa Fig. 2.5:

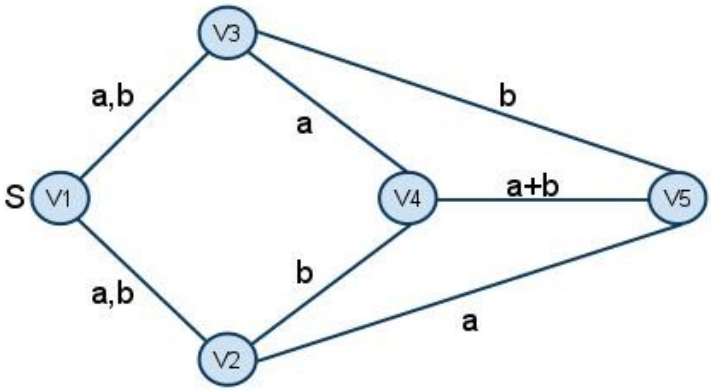


Fig. 2.5 Exemple de xarxa robusta usant la codificació.

Podem observar que hi ha un conjunt de nodes V on el V1 és la font que vol enviar dos bits a V5. Al receptor li arribarà informació per tres enllaços, el de V3 a V5 i el de V2 a V5 enviaran un sol bit per unitat de temps mentre que el V4 a V5 enviarà sumats els dos bits. Això fa que si cau un sol enllaç, sigui quin sigui, podrem arribar a aconseguir els dos bits, amb la única condició de que V5 ha d'estar preparat per fer la descodificació del paquet de V4.

2.2.2 Desafiaments

Els desafiaments més grans que es te d'enfrontar en el futur el son:

2.2.2.1 La complexitat

Usar “Network Coding” vol dir que els nodes que amb el mètode tradicional només havien de re-enviar la informació ara tindran d'adquirir funcions addicionals com son la codificació i descodificació de les dades. Com que els nodes hauran de realitzar operacions aritmètiques sobre camps finits en temps real , per exemple una suma xor, tant per examinar els paquets com per codificar, descodificar i enviar, necessitaran certa quantitat de memòria que si no en tenen suficient la velocitat del node es veurà limitada i per tant el rendiment de la xarxa baixarà.
Per tant el desafiament que hi ha és trobar un punt intermedi entre la complexitat de les operacions i el rendiment de la xarxa.

2.2.2.2 La seguretat

Les xarxes on la seguretat és la característica més important, estil les xarxes de bancs, han de garantir protecció per els atacs més sofisticats. Els mecanismes actuals estan pensats que només la font i el receptor son els únics elements tracten les dades. Amb el “Network Coding” els elements del mig han de realitzar les operacions necessàries per tal de fusionar la informació i això fa

que els nodes intermediaris se sumin a la font i al receptor als elements que tracten les dades.

Així que serà necessari afegir mecanismes d'autenticació de les dades en tots els nodes que modifiquin les dades, per assegurar la seguretat de la informació.

Els atacs que es fomenten en aquesta situació s'anomenen Bizantins* i es basen en trobar una vulnerabilitat en algun dels components de la xarxa. En el cas de la codificació en la xarxa com que tenim més components modificant la informació, la xarxa es veu més vulnerable.

*El nom de Bizantí ve donat per els problemes que tenien els generals de l'imperi Bizantí que per decidir un pla comú d'atac s'havien de comunicar amb missatgers i algun d'aquests podria ser un traïdor diguen que donarien suport a un pla i després feien lo que els hi semblava.

I en els atacs de interferència ("jamming atacks"), el nombre de paquets que es poden corrompre serà molt major que sense usar "network coding", ja que si enviem un paquet que és la suma de dos afectarem a tots dos paquets. Tot i així un mètode ha estat proposat per posar-hi remeï en (veure [12]).

2.2.2.3 La integració amb la infraestructura actual

Un dels problemes més grans que es planteja és com aplicar la tècnica en l'escenari real de les xarxes de comunicació d'avui en dia. Per tal d'integrar la tècnica s'haurien de fer canvis físics en els equips de encaminament i això representa un cost econòmic molt gran. A nivell de aplicació si que és possible sense un gran cost, però la tècnica resulta menys eficient que la implementació a nivell físic.
També es planteja com es pot integrar el "Network Coding" amb els protocols de xarxa que hi ha actualment, ja conviuen tots en un mateix sistema en el que es tenen d'entendre per poder fer les transmissions.

2.3 El teorema del Màxim Flux i el Mínim tall

Sigui G = (V,E,C) una xarxa (graph) amb un conjunt de V vèrtex i de E arestes, on C és la capacitat de cada enllaç E. C només suporta una unitat al mateix temps i hi poden haver varies arestes E paral·leles.

Considerem un node S Є (que pertany a) V vol enviar informació a un altre node R Є V.

Un tall entre S i R és un conjunt d'arestes on la eliminació d'una de elles representa la desconnexió entre S i R.

El valor del tall és la suma de les capacitats de les arestes que formen el tall. El mínim tall és el camí entre S i R amb el menor valor possible. Per tant, si suposem que la capacitat de les arestes és una unitat, podrem dir que el valor del tall és igual a número de arestes que te.

Així en l'exemple del mínim tall podem observar com només pot existir un únic valor de tall, però poden existir diversos talls diferents, ja que poden existir camins diferents.

En el cas del model “unicast” (quant només hi ha un receptor a la xarxa) , la màxima tassa de transmissió és igual al valor del mínim tall. Aquesta afirmació és la que constitueix el teorema del màxim flux i el mínim tall de Karl Menger, demostrant que la màxima quantitat de flux de la font S al receptor R és igual a la mínima capacitat necessària per eliminar de la xarxa de forma que cap flux pugui arribar de S a R.

Menger va demostrar aquest teorema al 1927. Casi trenta anys més tard es va tornar a provar per part de Ford i Fulkerson, i per separat al mateix any per Elias et. al..

Més tard, Edmonds va demostrar al treball “Edge-Disjoint Branchings” (veure [7]), que el màxim rendiment en l'escenari broadcast també era possible, i va proposar un algoritme de temps polinòmic.

2.4 El teorema del Network Coding

En l'escenari “multicast” les coses són diferents i ja no podem arribar al límit superior usant les tècniques tradicionals de encaminament.

Al juliol del 2000, Rudolf Ahlswede, Ning Cai, Shuo-Yen Robert Li i Raymond W. Yeung van proposar un nou model per aconseguir aquest límit en l'escenari “multicast”.

Com diuen en el “Network Information Flow” (veure [1]): “Al contrari a la intuïció d'un, el nostre treball demostra que en general no és òptim que la informació sigui enviada en multicast com un fluid, on simplement és encaminada o copiada.” En canvi, proposen Ahlswede et. al., si en el nodes intermedis es realitzen tasques computacionals, com combinar paquets entrants en un o varis paquets sortints, es pot aconseguir arribar al límit superior.

L'origen de la idea del “Network Coding” es pot remuntar en un treball anterior de R.W. Yeung al 1995 (veure [2]), que l'acabaria de formar en el document de Ahlswede et. al..

Per tal de demostrar que no podem arribar al nivell superior en multicast considerarem un escenari de la xarxa G = (V, E), on hi han S1,...,Sh fluxos dels nodes font S que simultàniament transmeten informació a N receptors R1,...,Rn.

Assumim que G és un graf acíclic dirigit, ho sigui que no té cicles i que per cada vèrtex V hi ha cap camí que comenci i acabi un altre cop a V. La següent Fig. 2.6 podem observar un exemple de xarxa on en tots els vèrtex no tenen un camí de tornada.

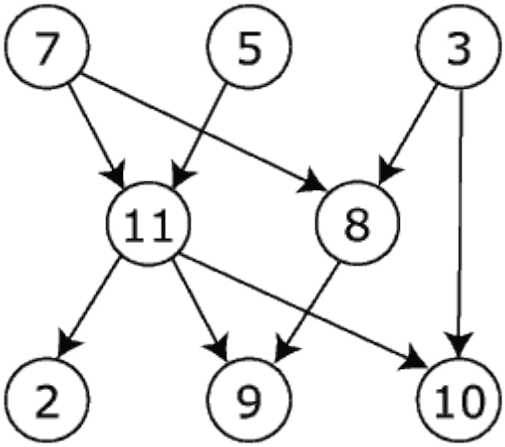


Fig. 2.6 Xarxa on cap dels vèrtex te un camí de retorn a ell mateix.

A més els vèrtex de G tenen la capacitat d'una unitat al mateix temps i el mínim tall entre la font i cada receptor és h. Per ara també no considerarem que hi hagi retard en les transmissions.

Amb una xarxa d'aquestes característiques si un dels receptors R qualsevol fa servir la xarxa, la informació de h fonts pot ésser enviada per h camins. Però quant varis receptors R fan servir simultàniament la xarxa perquè han de rebre informació pot ser que els camins es superposin i això suposi una disminució de les taxes de transmissió.

En canvi si fem servir el teorema del “Network Coding” aconseguirem les mateixes taxes com si cada un dels receptors tingués la xarxa disponible per a ell sol.

Al 2003 Shuo-Yen Robert Li, Raymond W. Yeung i Ning Cai van provar satisfactòriament la validesa de les teories presentades per Ahlswede et. al. , en el document “Linear Network Coding”. Van demostrar que es podia arribar al límit superior fins i tot amb un alfabet de camp finit.

Aquestes operacions lineals son sumes i multiplicacions dintre d'un camp de Galois finit F_q , on la mida d'aquest camp influirà directament a la complexitat computacional, és per això que com més petit sigui millor. En l'apartat de “Diferents tipus de network coding” explicarem detalladament el funcionament d'aquesta tècnica.

Koetter i Médard al mateix any, en el document (veure [2]), van proposar associar una variable aleatòria als coeficients no coneguts quant realitzem la codificació lineal.

T. Ho juntament amb en Koetter i en Médard van introduir a (veure [3]) un algoritme distribuït per dissenyar codis de xarxa lineals i a més van proposar usar codis aleatoris en els nodes, més tard aquesta tècnica es coneixeria per “Random Linear Network Coding”.

L'any següent, al 2004,Fraguoli et al, a (veure [5]) va desenvolupar un mètode distribuït i determinista per el disseny del “Network coding” traslladant el problema com un graph de colors.

Al 2005 Randall Dougherty, Chris Freiling i Ken Zeger demostren que la tècnica del Network Coding lineal no és suficient en xarxes de multifont i multireceptors amb demandes arbitraries, al document anomenat: “Insufficiency of Linear Coding in Network Information Flow” (veure [8]). A més demostren que en versions més generals de linealitat tampoc arriba a donar els resultats esperats.

Al mateix any Jaggi et. al. a (veure [4]), va introduir uns algoritmes centralitzats de temps polinomi. En va presentar dos: el DLIF (Deterministic linear information flow), un esquema completament determinista i RLIF (Random linear information flow), esquema semi-aleatori.

CAPÍTOL 3. Funcionament del Network Coding

3.1 Implementació a les xarxes de informació

En aquest apartat donarem una visió més pràctica de com funciona la tècnica en l'àmbit que ens pertany de les telecomunicacions. Ara en lloc de vèrtex o nodes, seran elements de nivell 3 del model OSI i les arestes seran cables de xarxa, ja sigui “ethernet”, fibra o “wireless”.
El que no deixem serà l'escenari “multicast” ja que el Network Coding obté beneficis en aquest mode de transmissió.
Una aplicació en la que es pot treure un gran profit és per “Streaming”,entès com la transmissió d'un flux continu de uns nodes a uns altres, en un grup d'usuaris, com es veu al treball “Streaming P2P robusto en redes Ad-hoc utilitzando informacion social” (veure [13]), es molt costós actualment.

El paradigma global de la transmissió en "streaming" son un grup de nodes S, que son els nodes font, volen transmetre informació a uns nodes receptors D.

Per tal d'entendre la situació usarem la topologia de la Fig. 3.1, ho sigui la xarxa de papallona, explicada anteriorment en l'apartat de Rendiment dins Avantatges i desafiaments, amb el mateix paradigma de que per els enllaços només podem enviar un paquet o bit en una ranura de temps:

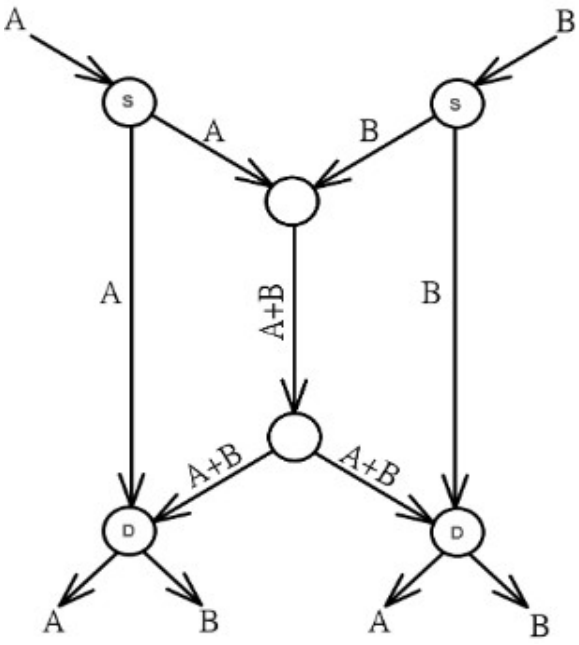


Fig 3.1 Exemple de xarxa de papallona

Si fem servir el mecanisme d'encaminament típic ens trobarem que per l'enllaç del mig només podem transmetre A o B al mateix temps, i això provocarà un coll d'ampolla en la xarxa, referint que mentres A arribarà als node repector D esquerra en una sola ranura de temps haurà de fer servir l'enllaç del mig per arribar al receptor D dret. Al compartir l'enllaç del mig amb les dos fonts i

només que puguin enviar un sol bit o paquet fa que la transmissió no sigui eficient.

Per solucionar aquest problema el que fa el “network coding” és una operació que generarà un sol paquet o bit de sortida i anirà per l'enllaç del mig. Aquesta operació pot ser una simple suma xor com altres solucions més laborioses usant camps finits. Així aquest bit arribarà a un altre node i serà re-enviat finalment a cada node D. Aquests nodes hauran de realitzar una operació necessària per tal de obtenir el bit o paquet resultant que volen. Per exemple el node D de l'esquerra rebrà el bit A sense codificar i de l'enllaç del mig rebrà la suma de A i B. Aquest node haurà de fer la operació necessària per aconseguir B a partir del resultat de la suma de A+B i de A.
Així ambdós nodes destí rebran simultàniament tots dos paquets.

Si ens fixem amb el temps que tarden els nodes en aconseguir els dos paquets podrem veure fàcilment la millora que proporciona la tècnica, ho exemplifiquem en les següents Fig 3.2 i Fig 3.3:

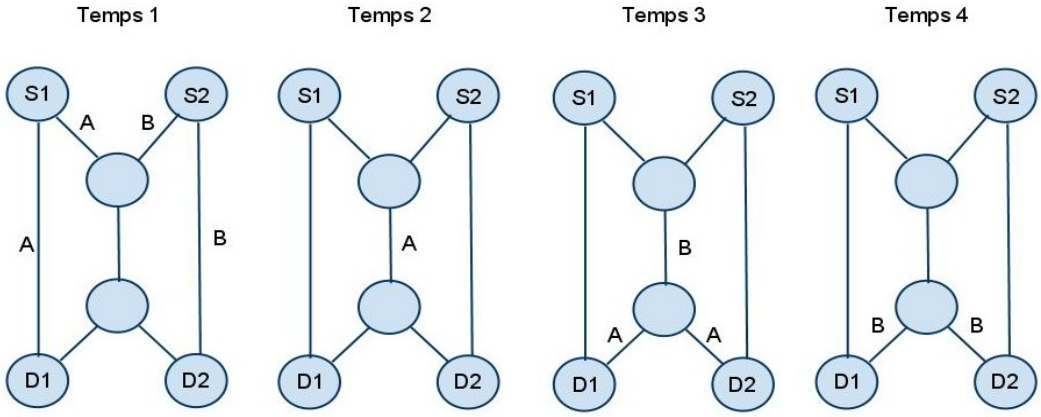


Fig. 3.2 Transmissió sense fer ús del Network Coding.

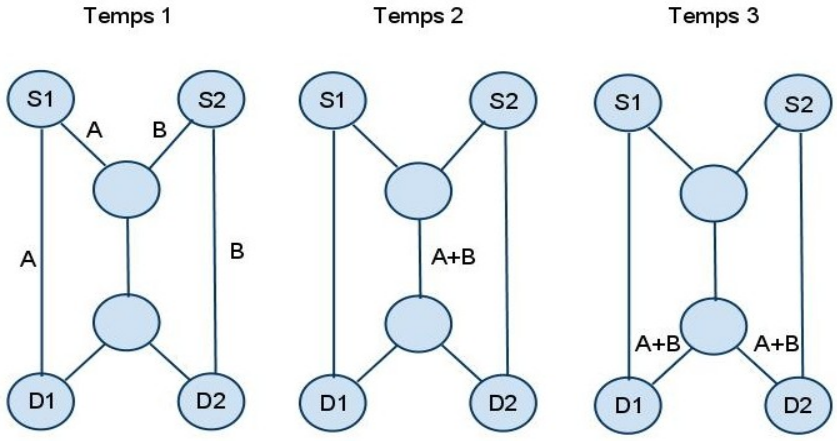


Fig. 3.3 Transmissió amb Network Coding.

S1 i S2 creen A i B respectivament i ambdós bits han de arribar a D1 i D2. No considerarem el temps de propagació i, com abans, la capacitat dels enllaços és d'una unitat, ho sigui un bit. Com es pot observar en la Fig. 3.2 tardem quatre temps per aconseguir els dos bits en D1 i D2, en canvi amb el Network Coding només tres. Això representa una millora del 25% del temps.

3.2 Steiner Tree Packing

El problema de Steiner Tree Packing es basa en trobar el camí més curt de un conjunt de vèrtex que formen una xarxa. És molt semblant al problema de "Minimum Spanning Tree" a diferencia que en el "Steiner" podem afegir vèrtexs i arestes extres, anomenats simplement "Steiner vertices" o "Steiner points". A més hi sol haver varies solucions en un mateix conjunt de vèrtex o sigui diferents "Steiner Trees" a construïr.

A la següent Fig. 3.3 podem diferenciar el "Spanning Tree" del "Steiner Tree":

Fig. 3.3 Exemplificació de xarxa amb Spanning Tree, a la esquerra, i la de Steiner Tree a la dreta.

La diferència dels dos mètodes radica en la utilització dels nodes extres en el "Steiner Tree", de color morat.

Abans hem dit que el "Network coding" permetia aconseguir el rendiment òptim quant és realitzava "multicast" fent servir algoritmes de temps polinomials.

Per la contra aconseguir el rendiment òptim en el mètode tradicional de "routing" és un problema NP-Complet, aquest és el problema dels "Steiner trees", ja que els problemes NP-Complets son característics per la seva dificultat a l'hora de trobar una solució. Així, tot i que els beneficis esperats usant network coding no siguin molt grans, sempre podrem arribar al rendiment esperat usant algoritmes més simples.

3.3 Escenari sense fils

Els recursos en l'escenari sense fils son molt valuosos, tal com hem explicat a la part de avantatges i desafiaments, i les xarxes sense fils mallades solen tenir un rendiment baix i no s'escalen gaire en topologies denses. És per això que l'any 2006, en el document (veure [10]), es va implementar el network coding en aquest tipus d'escenari, li van posar el nom de COPE.

COPE és un nova arquitectura per les xarxes sense fils mallades que insereix una nova capa de codificació entre les capes de IP i MAC. Aquesta nova capa s'encarrega de detectar les possibilitats de codificació i de ajuntar diversos paquets en una sola transmissió. Dit d'una altre forma, COPE implementa el network coding en un entern sense fils mallat.
Aquest seria un dels dos principis en que es basa aquesta nova arquitectura, l'altre és que COPE utilitza la naturalesa broadcast de l'enllaç sense fils, a diferència de l'encaminament on només s'adapta l'entorn com si sigués un enllaç punt a punt.

COPE incorpora tres tècniques de funcionament:

- Escoltes oportunistes

El medi sense fils té la particularitat de que les transmissions son en broadcast i per tant els nodes del voltant d'un emissor poden arribar a veure informació que no va dirigida a ells. El que fa COPE és posar els nodes en mode promiscu per tal de escoltar el medi i emmagatzemar els paquets per un temps limitat T (normalment T=0,5s).
A més cada node envia informes de recepció de forma broadcast per tal d'informar als altres nodes quins paquets te emmagatzemats. Un node que no te paquets per enviar envia igualment els informes de recepció periòdicament.

II Codificació oportunista

Una de les preguntes claus que s'han de realitzar per tal d'aconseguir el màxim rendiment és quins paquets tenim de codificar per tal de que amb el màxim de paquets originals amb una sola transmissió, tots els nodes tinguin suficient informació per poder descodificar les dades necessàries. La següent Fig. 3.4 és un exemple de codificació oportunista:

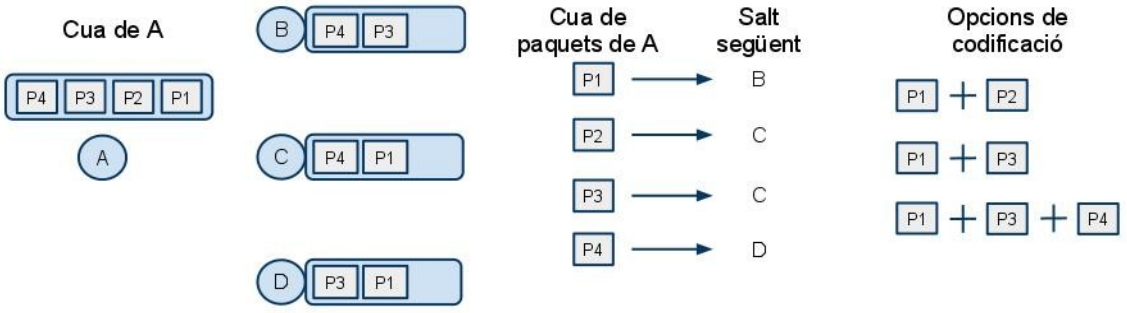


Fig. 3.4 A l'esquerra veiem com el node A te un conjunt de paquets que vol enviar a B,C i D, representats dins del rectangle. Al mig podem observar els salts que faran cada un dels paquets i a la dreta les opcions de codificació que existeixen.

Com es pot observar a la figura tenim un node A que vol enviar un conjunt de paquets a B,C i D, i permetem fer suma de paquets. A més ja estan decidits quins seran els següents salts que volen fer cada paquet, però cal recordar que estem en un entorn sense fils i per tant la informació s'envia broadcast a tots els nodes que arribi a veure la transmissió. Així que tenim tres opcions de codificació: la primera és la de sumar P1 + P2, resulta ser la pitjor perquè el node C sí que podrà descodificar el paquet però el B no. El node C podrà descodificar perquè te P1 i per tant adquirirà P2 que ja li interessa.
La segona opció és una mica millor, ja que B i C podran descodificar i rebre els paquets necessaris. La tercera opció de codificació és la més òptima perquè tots els nodes poden descodificar paquets que son del seu interès, únicament C es queda sense rebre P2.

1. Aprendre els estats dels veïns

En l'apartat anterior suposàvem que els nodes sabien quins paquets tenien els seus veïns amb els informes de recepció. Però amb un escenari congestionat hi poden haver col·lisions que poden fer que un informe arribi massa tard o no arribi, i per tant que es prengui una decisió de codificació no òptima. Per tal de solucionar aquest problema el que fa COPE és per cada veí estableix una probabilitat de tenir un paquet com de la probabilitat de que hi hagi una bona recepció.

3.4 Diferents tipus de Network Coding

La idea del Network Coding és senzilla, fusionar les dades en un punt intermedi de la xarxa. Amb aquesta premissa podem fer servir varies modalitats de codificació segons les seves característiques operacionals.

Em podem diferenciar de tres bàsiques: la que usa un simple operador booleà i que per tant serà la de menys carrega computacional i la més senzilla, la de més cost computacional, la Lineal introduint el Camps de Galois finit i la lineal

aleatòria (RLNC), de menys cost computacional però que necessita introduir informació extra als paquets .

3.4.1 XOR

3.4.1.1 Codificació amb booleà

La tècnica de codificació més simple és realitzar una suma booleana, la més usada és la OR-exclusiva (XOR) bit a bit, però també s'en poden utilitzar d'altres com per exemple la AND. Els nodes intermedis com a màxim hauran de realitzar aquesta operació que no hauria de representar una càrrega molt gran.
En un escenari "multicast" on hi hagi una topologia de papallona el node del mig rebrà dos bits, igual que els receptors finals. Per cada dos bits que el node rebi haurà de seguir la següent taula 3.1 per fer la operació OR exclusiva:

Taula 3.1 XOR

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Posem un exemple que X és el bit resultat de la suma de A xor B. On A i B son els bits que ha rebut un node de dos interfícies d'entrada diferents i X és el bit de l'única interfície de sortida disponible per el node.
Si A és 1 i B és 0, X serà 1. Així enviarem 1 per la interfície de sortida representant als dos bits que ha rebut.
Aquesta operació resulta molt interessant per la seva senzillesa a l'hora d'aplicar i perquè minimitza la latència i el consum d'energia de la xarxa, quant al mateix temps maximitza la taxa de bit.
El problema d'aquesta tècnica és que només es pot fer servir en determinats casos, com és la xarxa de papallona. Això es degut a que necessita que els receptors el l'hi arribin només dos bits al mateix temps.

3.4.1.2 Descodificació lineal amb booleà

El procés de descodificació també és bastant senzill i requereix la mateixa càrrega computacional que la codificació ja que només tenim de tornar a fer una suma booleana.
En l'exemple anterior hauríem rebut un dels dos bits de les fonts, posem que hem rebut A per un enllaç i rebríem el bit X per l'altre. Al node receptor li falta saber B, per tant si tornem a fer la xor amb el bit A, que és 1 i el bit X que també és 1, el resultat de l'operació ens surt 0 que és el valor del bit B.
En la següent Fig. 3.5 podrem apreciar el procés de la transmissió dels bits per l'escenari:

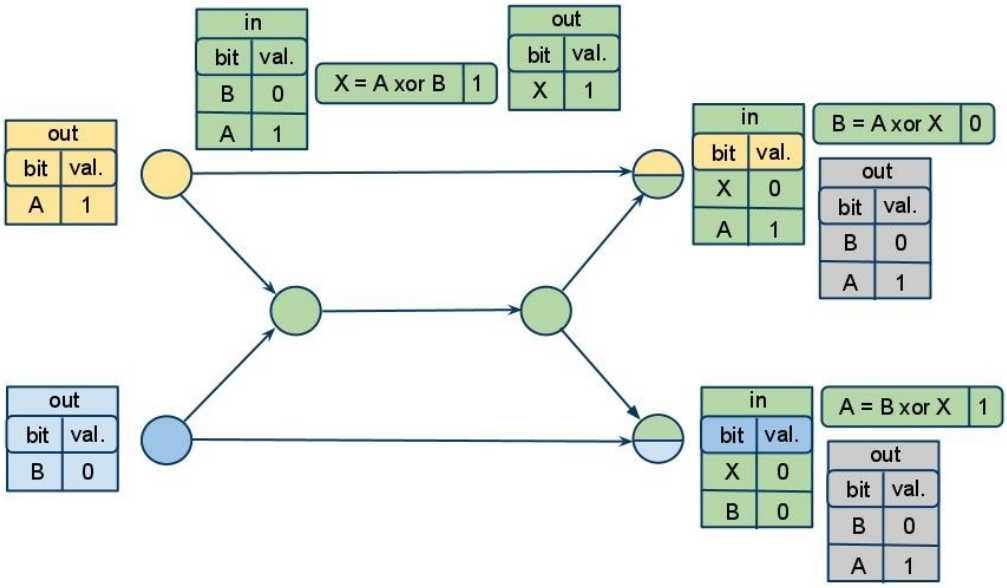


Fig. 3.5 Transmissió dels bits amb codificació XOR en una xarxa de papallona. Els colors dels nodes son els diferents bits que rep o crea.

Com podem observar amb una simple operació booleana com és el XOR podem codificar i descodificar la informació fent servir aquesta mateixa operació a nivell de bit.

3.4.2 LNC

3.4.2.1 Codificació lineal

El següent mètode es basa en operar en sistemes lineals per tal de aconseguir una major flexibilitat en la forma de que els paquets son combinats. Representa una millora significativa comparat amb la codificació booleana, ja que fa agrupacions dels bits que formen els paquets de mida L i els combina usant un camp finit, això fa aquesta tècnica més adaptable a les diferents topologies de les xarxes.

La major característica és que tots els nodes han de posar-se d'acord en la utilització dels coeficients, per tant és un mètode centralista que requereix saber la topologia de la xarxa.

Cal tenir en compte que la combinació lineal no és com fer una concatenació, si combinem linealment paquets de mida L, el paquet codificat resultant també tindrà la mida de L, mentre que amb la concatenació la mida augmenta.

Continuant amb les diferències amb el mètode booleà, ja no usem la suma or-exclusiva si no que realitzem una multiplicació d'un coeficient amb un valor d'informació útil. Més concretament la codificació es basa en la combinació lineal dels paquets rebuts en un node per els coeficients d'un camp de Galois.

Si un paquet ha d'ésser combinat amb un altre i no tenen la mateixa mida el més petit s'haurà de emplenar amb 0 fins a tenir la mateixa mida.
Podrem interpretar s bits consecutius d'un paquet com un símbol del camp finit, representat amb F_{2^s}, on cada paquet consisteix d'un vector de L/s símbols. Un exemple senzill per entendre els camps és el camp finit F_2 ={0,1}, on s és igual a 1 i per tant els símbols son els bits 0 i 1.

En el següent exemple podrem veure una visió pràctica de la tècnica:

Assumim que hi han varis paquets a combinar i cada un d'aquests consisteix en L bits.
Un paquet codificat generalment porta informació dels altres paquets originals, però al contrari de la concatenació, no pot recuperar qualsevol part dels paquets originals per ell mateix.

El procés de codificació pròpiament dit comença assumint que tenim uns paquets originals M1,...,Mn generats per un o varis nodes on cada paquet te associat un coeficient g1,..,gn generat dins d'un camp finit F_{2^s}.
Per tal de codificar el que fem es una operació de ajuntar els coeficients amb els paquets originals amb la següent operació:

$$X = \sum_{i=1}^{n} g_i M^i \quad \textbf{(3.1)}$$

L'operació es basa en la multiplicació de cada símbol dels paquets originals per un coeficient predeterminat i després fent el sumatori de tots els símbols combinats. Així que obtindrem els valors on hi hauran els coeficients utilitzats per fer la descodificació.

Cal remarcar la propietat del LNC de que els coeficients son predeterminats mitjançant un mecanisme anterior a la transmissió dins d'un camp finit.

Un cop s'ha codificat un paquet es poden codificar recursivament en paquets ja codificats. Simplement apliquem un conjunt de coeficients h_1 ,...,h_m que no seran iguals als g_1 ,...,g_n perquè seran coeficients respecte als paquets originals i no a X. La codificació dels paquets ja codificats es pot fer varis cops seguits en una transmissió.

3.4.2.2 Descodificació lineal

La descodificació lineal no dista molt de la codificació, de fet es basa en el principi de solucionar la operació que hem realitzat per codificar però ara sabent la X, ho sigui el valor combinat, i la g, els coeficients, per tant no sabem la M, el valor inicial. Per tal de trobar la M tenim de solucionar la següent fórmula:

$$X^j = \sum_{i=1}^{n} g_i^j M^i \quad \textbf{(3.2)}$$

Assumim que rebem els coeficients “g”, amb les dades codificades “X” junts, on el primer paquet serà (g_1,X_1),...,(g_m ,X_m). És un sistema lineal de m equacions i n incògnites, on necessitem $m \geq n$ per tal de tenir l'oportunitat de recuperar la informació, per exemple el número de paquets rebuts a d'ésser com a mínim igual que el nombre de paquets originals.

Cada paquet que rebem podrà ser innovador o no innovador. Els innovadors són els que incrementen el rang de la matriu i els no innovadors, quant realitzem l'eliminació de gauss, la matriu es veurà reduïda a una fila de zeros i serà ignorada.

Per solucionar la funció haurem de aïllar M i passar g a la mateixa banda de X.
Haurem rebut n blocs d'informació codificada , el que farem serà una matriu, que li direm A on en cada fila correspondrà als n coeficients que s'han usat per codificar cada valor X. A la següent Fig. 3.6 quedarà mes clar amb un exemple inventant-nos els valors:

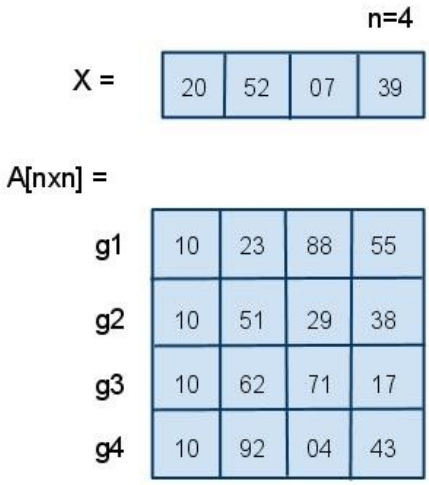


Fig. 3.6 Exemple del vector de valors codificats X de mida n i la Matriu nxn de coeficients A.

Com que hem rebut quatre valors codificats hem format una matriu de quatre per quatre i cada fila és un conjunt de coeficients corresponents als usats en cada valor.
Per tant ara només queda aïllar la M de la següent forma:

$$M = A^{-1} x^T \quad \textbf{(3.3)}$$

Tenim de realitzar la transposició del vector de les dades codificades i la inversa de la matriu de coeficients utilitzant l'eliminació de Gauss, ja que amb l'eliminació de gauss aconseguirem transformar la matriu en la forma de fila "echelon". Un cop realitzada aquesta operació ja podrem fer la multiplicació de la inversa A per la trasposta de x i obtindrem els valors originals de la font.

3.4.3 RLNC

3.4.3.1 Codificació lineal aleatòria

Tal com indica el seu nom, el "Random Linear Network Coding" es basa en escollir linealment els coeficients que farem servir en cada node de la xarxa de manera independent i aleatòria dins d'un camp finit.

A diferència del LNC, no necessitem cap coordinació per part dels nodes font, ja que els coeficients els escull el mateix node sense necessitat de saber tota la topologia de la xarxa ni dels nodes veïns, això fa que aquest mètode sigui de menys complexitat computacional que el LNC. Per altra banda és te d'afegir un "overhead" perquè els nodes receptors sàpiguen com recuperar les dades.

Per explicar millor els mecanisme usarem el següent exemple:

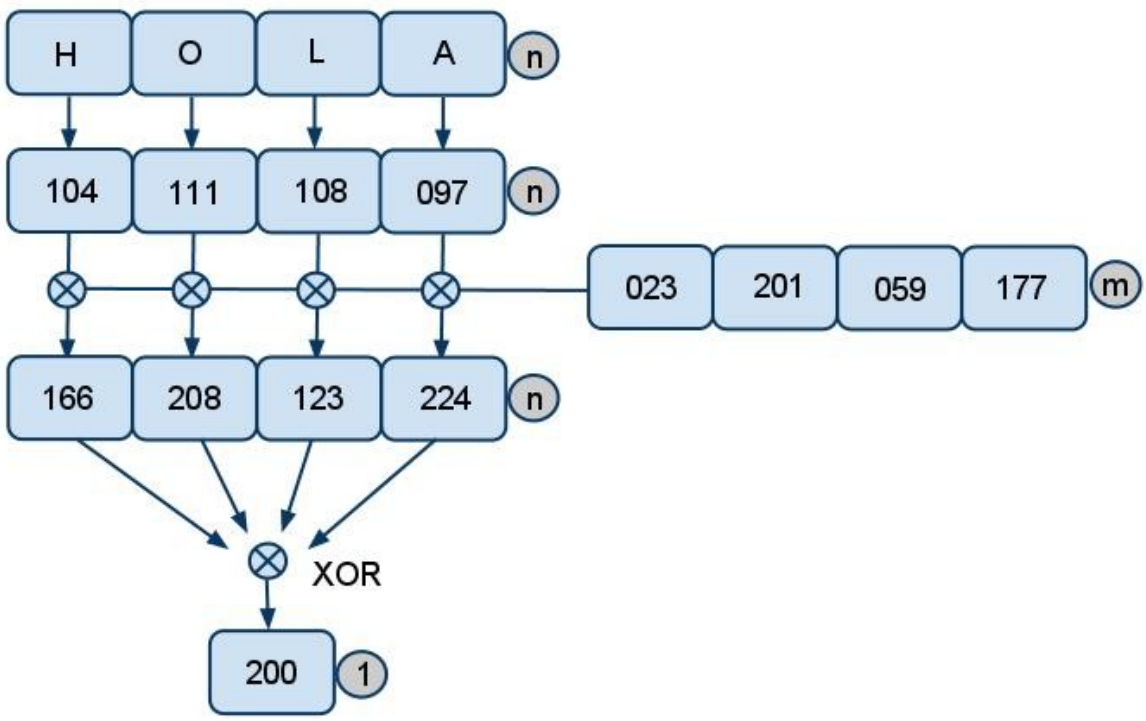
Tenim unes dades a enviar, un número de bits determinat situat a un node font F. El que es fa primer es agafar aquestes dades i dividir-les en en *n* blocs, $[b_1,b_2,...,b_N]$ on cada bloc te un número fix de bits *k* .

Després un node, incloent el node font, escull independentment i aleatòriament un conjunt de coeficients codificadors $[c^p_1 ,c^p_2 ,...,c^p_m]$ dins de un camp de galois* GF(256), on $m \leqslant n$, per tal de codificar la informació i enviar-a a un destí P. Ho sigui el número de blocs haurà de ser igual o major que els coeficients. Després aleatòriament s'escullen m blocs $[b_1,b_2,...,b_m]$ de tots els blocs que ha rebut o dels originals si és la font.

Cal tenir en compte el ràtio m/n anomenat densitat, ja que una baixa relació porta a la descodificació de les matrius disperses. Així el bloc codificat format per x de k bytes és produït per:

$$x = \sum_{i=1}^{m} c_i^p . b_i^p \qquad \textbf{(3.4)}$$

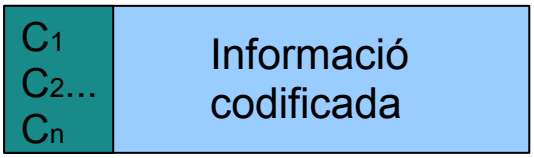
Amb el següent exemple es podrà entendre millor el procés de codificació:



Podem veure que hi han quatre blocs diferents que formen la paraula, cada bloc son exactament 1 byte. El procés de codificació primer converteix les lletres en valors ASCII del rang [0,255], després cada byte es multiplicat per un coeficient de codificació generat aleatòriament dins del camps de Galois, GF(256). La suma d'aquests productes és un bloc codificat.

Els coeficients usats per codificar els blocs originals normalment son integrats a la capçalera del bloc codificat per a la seva transmissió. Necessitem un total de n coeficients que s'incorporaran com una capçalera "overhead" de n bytes per bloc codificat.

Aquesta capçalera que estem introduint en cada bloc que transmetem representarà una retallada del rendiment de la transmissió i podria arribar a representar un problema a tenir en compte.



Per tant un “overhead” de n bytes és important quant n és gran (p.e.: 128) i la mida del bloc petita (p.e.:1KB).

Per tal de minimitzar l'efecte negatiu al rendiment que pot produir la capçalera existeix el següent mecanisme:
Si m / n = 1 i la llavor te tots els blocs en elaborar un bloc codificat, tan sols hem d'integrar la llavor aleatòria usada per produir les series de coeficients aleatoris amb un generador de números pseudo-aleatori. Això redueix eficaçment el "overhead" a només 4 bytes, independentment dels valors de n i k.

A mesura que la sessió avança, un node o "peer", va acumulant blocs codificats "x" al seu "buffer" i crea nous blocs codificats per transmetre'ls a altres "peers" de sota seu. Per tal de reduir el retard introduït per l'espera de nous blocs codificats, el "peer" produeix un nou bloc codificat en rebre $\alpha \cdot n$ blocs codificats, on $(0<\alpha<1)$, en la qual el paràmetre ajustable α es coneix com l'agressivitat.

Una α petita implica que els "peers" de sota seu rebran els blocs més ràpidament.

En el moment que un "peer" ha rebut un total de n blocs codificats, $x = [x_1,x_2,...,x_n]$, comença el procés de descodificació. Per descodificar primer forma una matriu A de n x n, usant els n coeficients codificats integrats en els n blocs codificats que ha rebut. Si m / n = 1, els coeficients aleatoris poden ser reproduïts usant la llavor integrada en cada bloc.

Cada fila de A correspon als n coeficients de cada bloc codificat. Després recuperem el bloc original $b = [b_1,b_2,...,b_n]$ amb:

$$b = A^{-1}x^T \quad \textbf{(3.5)}$$

En aquesta equació primer es necessari operar l'invers de A, usant l'eliminació Gaussiana, igual que en el mètode lineal.

Després multiplica A^{-1} per X^T , que te $n^2 \cdot k$ multiplicadors de dos bytes en el camp de Galois GF(256).

3.5 Aplicacions

La utilització actual del "Network Coding" en programes comercials o software especialitzat no es gaire comú. S'han demostrat beneficis pràctics en diferents models de xarxa però la tècnica es veu limitada per la complexitat d'introduir uns nodes amb capacitat per realitzar les operacions necessàries en les xarxes actuals. Creiem que aquesta raó i el fet de que el principal competidor en P2P, el programari "BitTorrent", està gaudint d'una popularitat molt gran, fan que aquesta tècnica no acabi de trobar una excusa per usar-se i pugui ser atractiva per a més funcions.

Tot i així el "Network Coding" és una tècnica bastant nova i podria arribar a ser útil en:

- Distribució de contingut digital o de xarxa P2P

Microsoft ha fet una implementació molt coneguda en aquest àmbit, usant-la en un programa de distribució P2P anomenat Microsoft Secure Content Downloader anteriorment conegut com Avalanche, que va desaparèixer al 2007.

La companyia de Seattle també ha tret un altre programa que usava Network Coding que el van utilitzar per la distribució del Visual Studio 2008 Beta-2 anomenat Nimbus.

Hi ha una discussió oberta de si el Network Coding, i més concretament Avalanche, millora la tecnologia P2P, ens centrarem en els avantatges que pot proporcionar aquesta tècnica:

2. Minimitza el temps de baixada. En sistemes “peer-to-peer” distribuïts de gran mida és molt complex aconseguir trobar la forma d'optimitzar correctament la forma de distribució dels paquets, particularment si els nodes tenen una informació molt limitada sobre la topologia de xarxa subjacent. Amb el Network Coding l'eficiència del sistema depèn molt menys de la topologia de la xarxa i de la programació de la distribució dels paquets. Simplement usant els sistemes que per defecte realitza la tècnica podem aconseguir aquestes millores. Segons els creadors de Avalanche aquest millora les tècniques d'encaminament tradicional i d'un sistema “peer-to-peer” basat en FEC per un ampli marge.
3. Donada la diversitat dels blocs codificats, la solució basada en el “network coding” és més robusta en el cas de que el servidor principal marxi abans d'hora, ho sigui abans que els peers hagin acabat de baixar totes les parts, o en escenaris d'altes taxes de churn (els nodes només s'uneixen per un curt període de temps o surten ràpidament just acabada la descàrrega).
4. A diferència de amb els protocols de encaminament, el “network coding” sofreix només una petita pèrdua de rendiment quant implementem els mecanismes per la cooperació (p.e., ull per ull(“tit-for-tat”) per prevenir el parasitisme (“free-riding”)).

Les contres son:

1. Un peer pot arribar a necessitar molt de temps i recursos per tal de descodificar les dades rebudes
2. Pot ser difícil assegurar l'autenticitat dels coeficients quant hi han moltes peces que formen les dades transmeses.
3. La topologia d'una xarxa “peer-to-peer” esta amb canviant constantment mitjançant l'addició i l'eliminació dels iguals a la xarxa. Aquest canvi constant pot provocar una baixada de l'eficiència perquè el mecanisme ha de adaptar-se.

- Alternativa al ARQ i al FEC

Com s'ha comentat al principi del treball pot ser aplicat com un substitut al FEC (Correcció d'errors endavant) com també del ARQ (Automatic repeat request) al fer les operacions en nodes intermedis.

- Útil contra atacs a la xarxa

Una altre aplicació és fer servir la codificació per evitar atacs maliciosos a la xarxa, com poder ser “spoofing”, “eavesdropping” o “replay atack”.

- Millora rendiment a les xarxes sense fils

S'ha demostrat que aplicar la codificació en les xarxes sense fils millora el rendiment en una topologia mallada.

- Xarxes ad-hoc de sensors

Les xarxes de sensors son d'unes característiques concretes d'estalvi d'energia i aprofitament dels recursos, la codificació encaixa perfectament amb la singularitat d'aquestes xarxes.

3.5.1 Programes comercials

Aquests són els pocs programes que han arribat al món comercial, com és pot veure sobretot gràcies a l'empresa informàtica de Redmond, Microsoft

- Avalanche

També conegut com a Microsoft Secure Content Downloader
Desaparegut des del 2007. Suposadament havia de competir amb els sistemes P2P, com el “Bittorrent”, però és va quedar sense aconseguir el suport necessari fins i tot per part de l'empresa de Redmond.

- Nimbus

Amb el lema “Secure Peer Accelerated Downloads” pretenien com amb Avalanche fer un sistema basat en “Network Coding” i el sistema P2P.
També creat per Microsoft, usat per distribuir el Visual Studio 2008 Beta-2.

CAPÍTOL 4. SIMULADORS

4.1 Entorns de simulació escollits

Per tal de simular el comportament del Network Coding en una xarxa de dades, hem decidit escollir dos programes, el OPNET Modeler i el Matlab.

Amb aquests dos simuladors provarem dues tècniques de Network Coding que em considerat importants per veure el funcionament de la tècnica, que son fer les operacions amb una simple suma XOR i amb el Lineal Network Coding. També ens aprofitarem de part d'un codi per OPNET del grup d'investigació “FUNLAB” de la Universitat de Washington, que em trobat per tal de fer les modificacions necessàries per modelar pels nostres propòsits.

Les raons d'haver escollit el OPNET Modeler son varies:

-La companyia darrera del programa. OPNET Technologies és una gran empresa dedicada exclusivament a la simulació de xarxes, és per això que era d'esperar que el software sigués molt complet i amb molta informació com manuals d'ús.

-Seguint a la raó anterior, que tenen un equip de suport per qualsevol dubte o problema que els usuaris tinguin. Vam poder comprovar com aquest servei és excepcional, amb un bon tracte i ràpid.

-És àmpliament utilitzat en la comunitat científica amb els models molt revisats. Per exemple la capa de TCP és de les més ben implementades de tots els simuladors.

-Tot i ser un software privatiu a la UPC tenim l'opció de baixar-lo gratis per estudiants o investigadors si la universitat disposa d'una llicència.

-Vam trobar un projecte fet en OPNET que s'apropava molt a les nostres necessitats del PFC. El no tenir de començar de zero ens facilitaria la feina de l'estudi del software, que al ser tant complet l'aprenentatge de totes les funcions ens costaria molt de temps que no disposem. Així únicament ens centraríem en estudiar les parts del codi i lo necessari del programari per poder modificar correctament la xarxa.

L'altre programa és el Matlab, l'entorn de computació numèrica més conegut i més usat sobretot en les carreres tècniques i feines d'enginyers.

El Matlab el farem servir per el model més matemàtic del Network Coding, així podrem demostrar el funcionament de les propietats de les operacions.

Les raons per escollir-lo son que ja disposàvem del programa a causa de que l'havíem usat en altres treballs i exercicis de la carrera i perquè teníem coneixements de com anava. Primer ens centrarem en aconseguir fer la simulació d'una xarxa amb el mecanisme de Network Coding més senzill, el xor. I després simularem una xarxa amb Linear Network Coding amb el Matlab.

4.1.1 Entorns de simulació descartats

Tots aquests simuladors han estat descartats a causa de que el codi que vam trobar en Internet estava en OPNET i per les raons que hem explicat avans.

- Omnet++

És un software de simulació d'esdeveniments discrets basat en C++, obert per la comunitat acadèmica i basat en l'entorn del Eclipse. És usat normalment per modelar el tràfic de les xarxes de telecomunicacions així com avaluar el rendiment de la mateixa.

- Ns2/3

“Network Simulator” 2 i 3 son eines de simulació d'esdeveniments discrets, àmpliament usada sobretot per simular xarxes ad-hoc. Resulta molt popular en la comunitat acadèmica per la seva extensibilitat i per la gran quantitat de documentació que hi ha al darrera.

- NECO

NECO és un simulador exclusiu de “Network Coding”, tal i com indica el seu nom format per les dos primeres lletres de cada paraula. Esta escrit en “Python” i resulta gratis ja que resulta ser una versió en proves.

4.1.2 Resum dels simuladors

Simulador	A favor	En contra	Escollit
OPNET	Empresa gran al darrera Extens tipus de models Codi de FUNLAB	Difícil de dominar en totes les capes	Sí
MATLAB	Utilitzat àmpliament a la carrera Molt versàtil	Implementació molt teòrica	Sí
Omnet++	Lliure i obert	Aprenentatge costós	No
Ns2/3	Molt usat per la simulació de xarxes	No te molts models predefinits	No
NECO	Exclusiu sobre Network Coding	Anàlisis de resultats difícil de realitzar	No

CAPÍTOL 5. IMPLEMENTACIÓ FUNLAB

5.1 Codi de Funlab (Universitat de Washington)

UW Funlab és un grup d'investigació de la Universitat de Washington, dels Estats Units d'Amèrica, enfocat a estudiar el disseny i l'anàlisi de la infraestructura de les xarxes digitals futures.
Un dels projectes que estan portant a terme és l'estudi del Network Coding; en la seva web (veure [9]) podem veure com el projecte que estan realitzant es basa en la monitorització de les falles dels enllaços mitjançant el “network coding”. Per tal de posar a la pràctica la seva teoria van realitzar un escenari amb OPNET i han deixat el codi font de l'aplicació en un enllaç de la web.
Cal dir que vam buscar altres alternatives per el OPNET, però tal i com diuen en la web de Funlab és l'únic codi, que coneixen, dedicat a la simulació del “network coding” en aquest simulador. De fet, van haver de modificar els models predeterminats dels encaminadors per tal d'adaptar-los a les funcions extres del “network coding” a més de que les operacions no es fan al mateix OPNET si no que es fan per MATLAB.
L'escenari del Funlab ens servirà per començar, però haurem de modificar la topologia i les diferents parts per tal de adaptar-lo a les necessitats d'aquests treball, a més de configurar el sistema i els programes per tal de adjuntar OPNET i MATLAB.

5.2 Comprovació del funcionament de Funlab

El que hem fet primer és estudiar i provar el funcionament del codi de FUNLAB. Es basa en una topologia de cinc encaminadors situats amb forma de creu que interconnecta a dos terminals situats cada un en un extrem. En la següent Fig. 5.1 podem veure la xarxa en una captura de el OPNET:

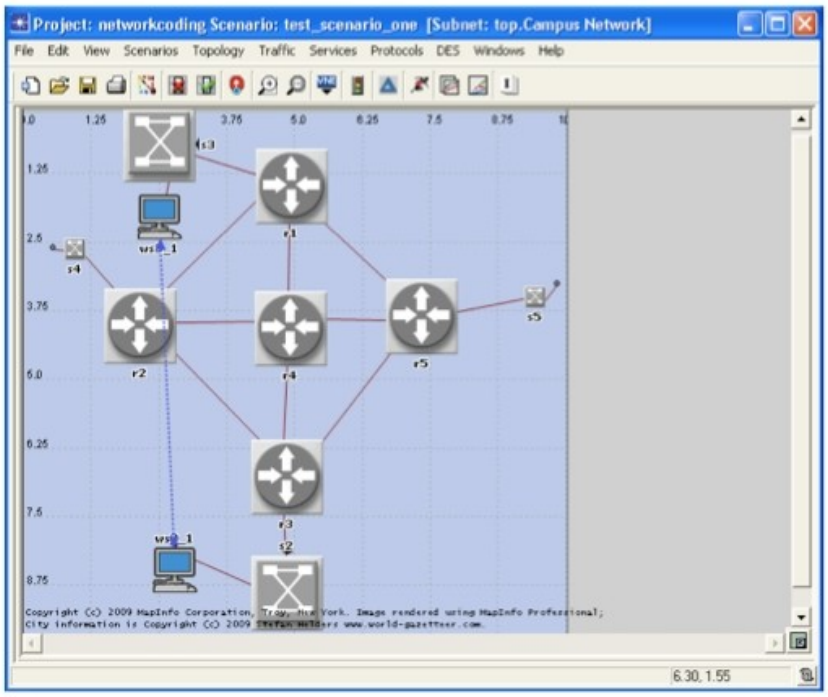


Fig. 5.1 Finestra principal de l'escenari de OPNET.

Podem observar els routers, representats com els elements circulars a les fletxes cap a dins, les estacions de treball amb la forma de un ordinador de sobretaula, els hubs, representats com un quadrat amb una forma de ics i els enllaços de cables ethernet de color vermell unint els diversos elements.
Les dos estacions de treball estan connectades per una fletxa de color blau, aquesta representa el flux de tràfic que hi haurà entre l'estació ws2_1 i la ws3_1, representa un model de “application demand”, un model basat en la inundació de paquets.
Si fem doble click en els encaminadors podrem veure que els han modificat tal i com diuen en la web, han afegit processos com nc_proc i sink, i han dirigit el tràfic entre processos representat amb una línia blava.

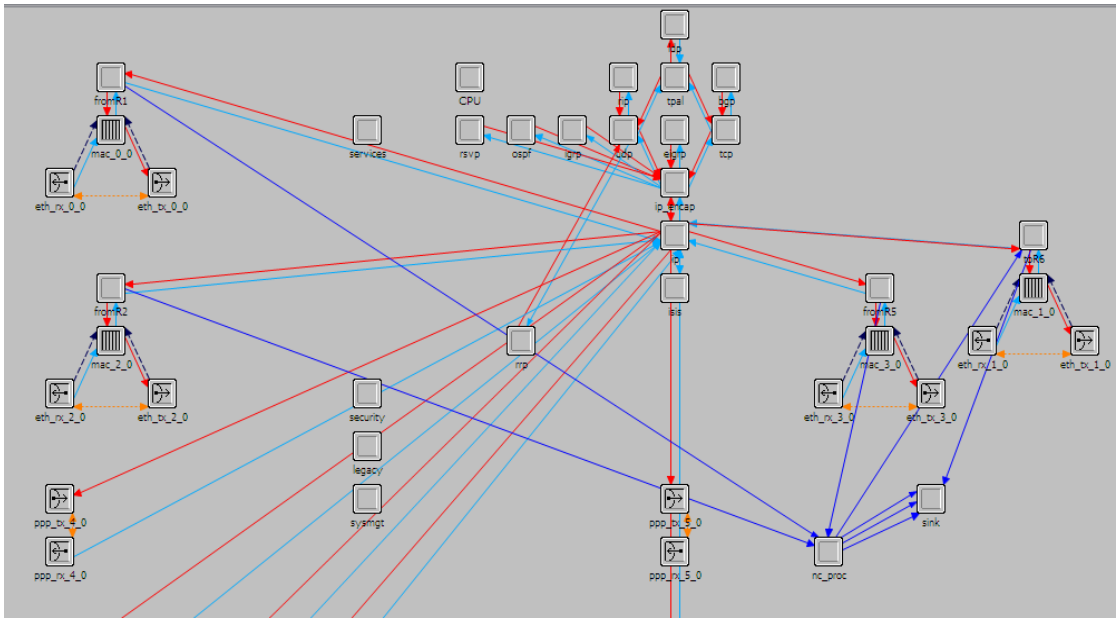


Fig. 5.2 Escenari de processos dins d'un encaminador.

En les interfícies d'entrada dels encaminadors, les línies blaves uneixen el procés ARP amb el procés nc_proc, això es fa per copiar els paquets rebuts a les altres interfícies. I per cada interfície d'entrada que vagi al nc_proc haurà de sortir una línia d'aquest mateix procés que vagi al sink, per tal de llançar els paquets no necessaris.

En les interfícies de sortida hi ha una línia que va del procés ARP al sink, ja que al ser una interfície per enviar tots els paquets que entrin els haurem de tirar. I també per hi haurà una línia que anirà del nc_proc al procés ARP de la interfície de sortida per tal de enviar els paquets que vulguem.

En els atributs de les interfícies hem observat que tenen uns valors de "timeout", ho sigui de espera, per tal de que els paquets arribin al encaminador destí al mateix temps i així es puguin combinar correctament. També hi ha un altre valor anomenat "nc_value" que és el número a combinar amb el valor rebut per la interfície d'entrada, on amb el 0 és un número aleatori, i "bitsize of nc value" que és la mida del codi que farem servir per els càlculs dels camps finits.

Es una mica enganyós però el procés nc_proc no és qui realitza les operacions de network coding, si no que és a les interfícies de sortida és on hi ha implementada aquesta funció. Si entrem a una interfície de sortida, ho sigui en un procés ARP, que tenen el nom com "toR1", ens trobarem amb que el procés realment es diu "nc_ip_arp_v4" i podrem veure el diagrama d'estats.

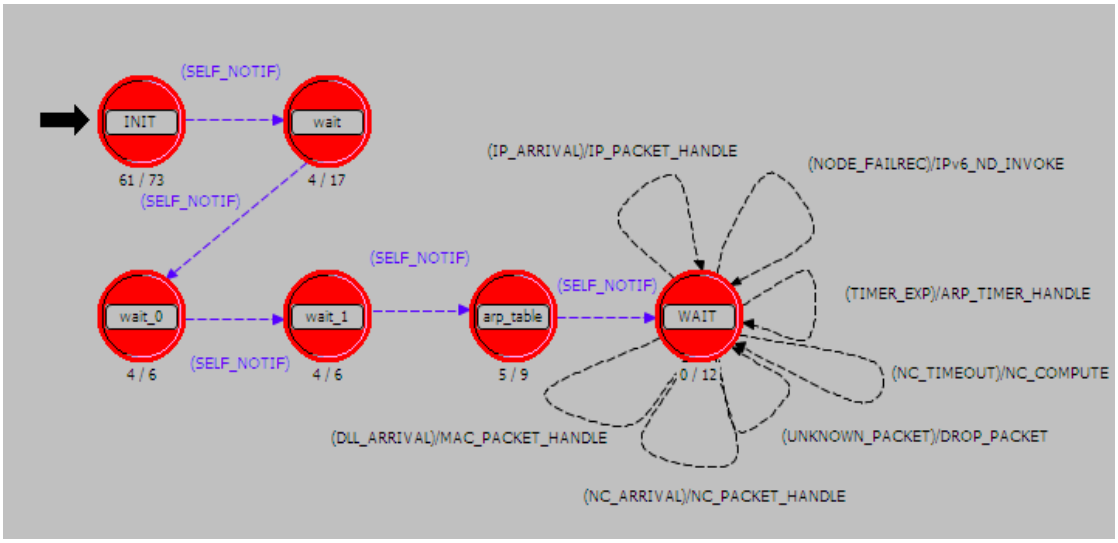


Fig. 5.3 Diagrama d'estats del procés nc_ip_arp_v4.

En el diagrama podem observar diferents estats, representats amb rodones i les accions que canvien els estats que son línies que connecten als estats. Dins de cada estat hi ha el codi escrit en llenguatge C, ho sigui el nivell més baix del OPNET.

En aquesta interfície, com hem anunciat abans, és on es realitzen les operacions del network coding. Primer la interfície s'encarrega de guardar en una cua els paquets que s'han de operar i de comprovar el timeout s'ha establert en cada paquet

Un cop s'ha esperat el temps de “timeout”, si es necessari, s'agafen tots els paquets de la cua i es combinen fent servir el MATLAB amb la funció gf() per els camps finits. Després es multiplica el valor de les interfícies amb la dels paquets, aquest valor és el “nc_value” que hem anunciat abans.

Un cop combinats tots els paquets de la cua ens surt un paquet nou que serà el que enviarem per la interfície de sortida. En la següent imatge podem veure l'esquema de la xarxa amb els nc_value de cada interfície i els valors que es creen un cop combinats.

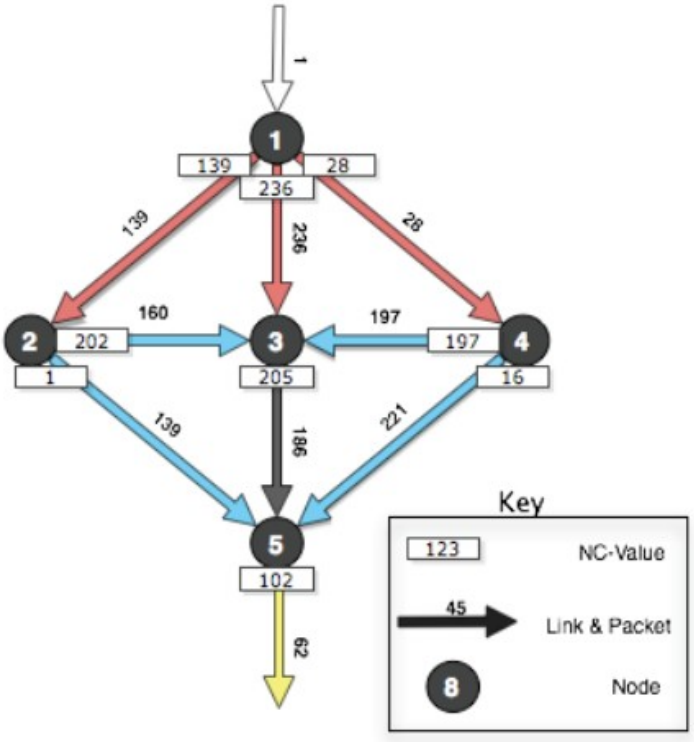


Fig. 5.4 Escenari xarxa de FUNLAB amb els valors de codificació anomenats "NC-Value". Els valors sobre les línies son els resultats de la codificació.

Els diferents colors representa el temps en que s'envia el paquet, ho sigui totes les línies de color vermell s'envien a l'instant 1, i les blaves al 2. Això vol dir que el node 3 haurà de posar un "timeout" als paquets que vinguin del node 1 per esperar a tenir el "buffer" i així poder fer les operacions.

5.3 Resultats

Hem muntat la xarxa tal i com deia en el manual i no hem obtingut els mateixos resultats que anunciava el document. A més els valors de nc_value eren diferents ja al primer moment. Pensem que la versió que hi havia a Internet estava més actualitzada que la del document.

També ens hem adonat que en el node 3 no fa la suma amb els tres paquets, només agafa dos paquets i els envia,i en l'altre instant dos més, ho sigui es queda sempre un dels tres sense combinar.

Com podem veure el valor que surt de l'encaminador 5 és el 62, producte de la suma dels tres valors que ha rebut, per tant en aquesta xarxa hi han dos operacions de codificació al node 3 i al 5.
Cal tenir en compte que en el codi de FUNLAB no hi ha cap implementació de la descodificació, ja que no tornen a aconseguir el valor 1 en cap moment.

Així la nostre valoració sobre la implementació la donem com a negativa i amb el contingut del codi imcomplet.

CAPÍTOL 6. IMPLEMENTACIÓ OPNET

6.1 OPNET Modeler

Opnet Modeler és un software de monitorització i simulació de xarxes de la companyia nort-americana OPNET Technologies, Inc. fundada al 1986. Aquest no és l'únic software que ofereixen, per exemple tenen una versió del Modeler enfocada només a dispositius mòbils, però també tenen programes encarregats de detectar falles o possibles errors de configuració d'una xarxa.

Per tal d'utilitzar els productes de OPNET és necessari pagar la llicència del software, però també ofereixen l'opció de que sigui gratis si es fa servir en centres educatius com universitats.

En aquest projecte usarem el Opnet Modeler en un sistema operatiu Windows XP SP3 per les raons que ja em explicat abans. A l'annex 1 mostrarem un tutorial de com instal·lar-lo en un entorn Windows ja que no es una feina senzilla.

OPNET esta distribuït per capes. Hi ha la capa més alta, la dels components com poden ser els encaminadors o el cablejat, la dels processos de cada element, per exemple dintre l'encaminador ens trobarem el procés de la interfície d'entrada , el del nivell de ARP, etc. i la dels estats de cada procés representada com diagrames d'estats que seguirà cada element.
Per moure's entre capes només tenim de fer doble click sobre un element i ens sortirà una finestra amb la capa inferior que forma l'element.

El codi es troba dins de cada estat o en la finestra de processos. A la barra de a dalt, just abans del botó de compilar hi han quatre botons més, aquests son els blocs de les funcions i capçaleres, en blau, i abans dels blocs hi han les variables, en vermell.

Fig 6.1 Botons dels Blocs de variables i funcions

El codi principal de tots els estats i de altres funcions es troba en el boto anomenat “Edit Function Block”, el codi de les capçaleres es troba al costat anomenat “Edit Header Block”. En el Bloc de les capçaleres hi trobarem els “includes”, la definició de les estructures que es fan servir al bloc de les funcions, i les definicions de les transicions dels estats.
Als botons de les variables hi podem trobar les variables globals del proces com les temporals.

6.2 Simulació de network coding amb XOR

En aquest apartat ens centrarem en simular una xarxa on s'apliqui el network coding mitjançant l'ús de operadors booleans, en aquest cas implementarem el or exclusiu, també conegut com a XOR. Donat que aquest mètode funciona en determinades topologies, les xarxes que podem realitzar es veu reduïda, i hem pensat d'escollir la xarxa que sempre s'utiltiza per demostrar l'ús del network coding, la xarxa de papallona.
La xarxa de papallona es veu formada per sis nodes on dos son les fonts de informació, normalment els dos nodes superiors, i dos son els receptors, situats a baix. Al mig hi han dos nodes més, el que es situa al mig entre les fonts i serà l'encargat de combinar els paquets i el de a baix encarregat de re-enviar el paquet combinat als receptors. Els enllaços que uniran els nodes seran cablejats amb ethernet de 100Mbps i els nodes seran elements de capa tres.

Els nodes seran encaminadors i a més hi hauran estacions de treball que generaran els nodes i els rebran. Com observarem més tard la xarxa no és exactament igual tal com hem dit ara i haurem d'afegir un encaminador entre els nodes que faran de font i l'estació de treball, això és degut a la naturalesa de la simulació, que en el següent apartat justificarem.

La estructuració de la xarxa serà que els nodes fonts tinguin cadascú dues lletres de la paraula HOLA, per exemple que el node 1 envii H i L i el node 2 envii O i A. I que els dos nodes receptors hagin de rebre la paraula HOLA completa. L'objectiu és enviar H i O al mateix moment i, L i A, després, per tal de que en el node intermediari combini les dues lletres. Les lletres les representarem amb numeració ASCII.

6.3 Modificacions del codi de Funlab

Com que vam decidir partir del codi de FUNLAB hem tingut de adaptar-lo en les exigències descrites a l'apartat 6.2.
El primer pas que hem fet es canviar la topologia de la xarxa, ho sigui hem desfet la forma de creu per muntar la de forma de papallona. Per això unirem els encaminadors amb cables ethernet, cal tenir en compte les interfícies que fem servir en cada dispositiu, ja que els encaminadors tenen interfícies d'entrada i de sortida. Hem eliminat els repetidors i s'han implementat tres estacions de treball. També s'han afegit dos encaminadors un per completar la xarxa de papallona i l'altre que unirà l'estació de treball generadora de paquets amb les dos fonts.
Això ho fem perquè ens vam adonar que mantenint el tipus de tràfic “application demand” només enviava paquets de sol·licitud al node destí i les estacions de treball només tenen una interfície de sortida.
Així l'encaminador el que farà cada cop que rebi un paquet de sol·licitud serà enviar per cada interfície les lletres que s'hauran de re-enviar als altres encaminadors. La topologia té la forma de la següent Fig. 6.2:

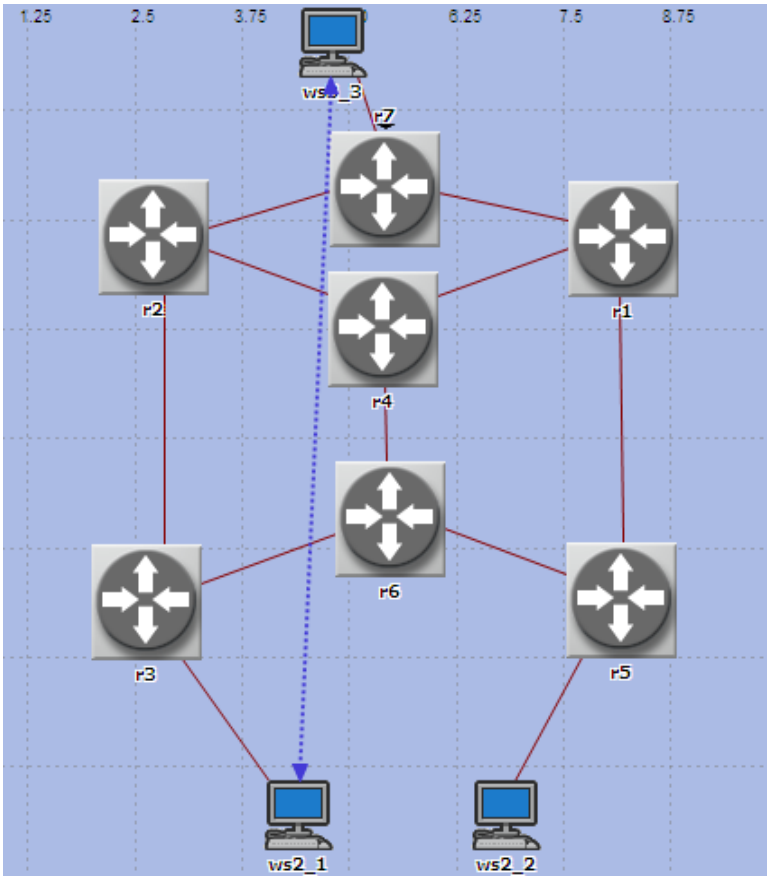


Fig. 6.2 Xarxa de papallona implementada en OPNET.

Com es pot observar hi han dos estacions receptores tot i que només dirigim el tràfic a una, això es degut a que els encaminadors ja re-enviaran tots els paquets per tota la xarxa i arribaran tant a R3 com a R5.
En el model de tràfic si fem un click dret a sobre de la línia discontinua podrem accedir al menú dels atributs. Dels atributs ens tenim de fixar amb la duració, que l'hem fixat en un “Start Time” de 100 i “End Time” de 151, en el protocol de transport, que farem servir UDP i en el model del tràfic que és “Application Demand “ com ja hem dit. En aquesta capa només afegir que els models que hem usat per els encaminadors son “nc_router_59_upgrade”, la de les estacions de treball son “ethernet_wkstn” i els enllaços son 100BaseT, ho sigui un cable de ethernet de 100Mbps.

Si entrem en un encaminador ens trobarem a la capa inferior on hi han els processos units amb fletxes de varis colors, com hem dit abans, les de blau fosc son les que indiquen el tràfic dels paquets. Podem distingir els processos d'entrada i de sortida per el nom ARP i un número, nosaltres canviarem aquests noms i posarem “fromR” i el número del encaminador del que entri el flux, i “toR” amb el número del encaminador de destinació. Aquests canvis faran que sigui molt mes fàcil determinar si son interfícies d'entrada o sortida i a més en quin encaminador estan dirigits.

En aquest nivell ens assegurarem que en totes les interfícies d'entrada i sortida tinguin l'atribut de “Source nc value to use” al valor 1, això és perquè tots usin el mateix flux de dades. També haurem de revisar com estan posades les fletxes blau fosc, per una interfície d'entrada hi ha de haver: una fletxa del procés “fromRX” dirigida al “nc_proc” i del “nc_proc” en té de sortir una al procés “sink”. Aquest procediment per cada interfície d'entrada que hi hagi.
Per les interfícies de sortida serà el següent, una fletxa blau fosc del “nc_proc” al procés de sortida “toRX” i d'aquest procés una fletxa al “sink”, això ho farem per cada interfície de sortida que tinguem.

Com hem dit anteriorment la combinació de paquets és realitza en els processos de sortida de cada encaminador que sigui necessari. En el nostre cas ens interessa que el node del mig R4 per la interfície “toR6” realitzi les funcions de network coding, així que mirarem que conté el procés.

La funció “nc_process_queues(int flg)” és la que conté el mecanisme de network coding, primer ens trobem que declara les variables on el punt més important és “UdpT_Dgram_Fields *udpfields” ho sigui el punter els camps del paquet UDP que estem enviant.Com que ens interessa guardar els valors de nc_value creem un vector per emmagatzemar-los anomenat valo i li posem la mida de fins a tres posicions amb valo[2], ja que com a màxim tindrem dos paquets a la cua.

Després comprova que hi hagi una interrupció de “timeout”, dit d'una altre forma, s'espera fins que el temps de timeout hagi passat per tenir tots els paquets a la cua. Un cop hagi saltat la interrupció fa una crida al sistema perquè obri el MATLAB.
Això ho fa perquè les operacions en el codi de FUNLAB les realitza el MATLAB, nosaltres hem decidit que era millor que el mateix OPNET fes les operacions, ja que en el nostre cas son simplement sumes xor. I després el que fa és recórrer tots els paquets que hi han a la cua fent les operacions en el camp de galois de la següent forma:

```
for (i = 0; i <= op_prg_list_size(nc_list); i++)
{
        pair = (Nc_Queue_Pair * ) op_prg_list_remove (nc_list,OPC_LISTPOS_HEAD);
        iciptr = (Ici *) op_prg_list_remove (ici_list,OPC_LISTPOS_HEAD);
        op_ici_install(iciptr);

        data = ip_dgram_data_pkt_get(pair->pkt);
        op_pk_nfd_access(data,"fields",&udpfields);
        /* Matlab functions go here */
        /* For all enqueued packets, add the T. */
        sprintf(&matlab_str, "if (find(T==%d)) else T=[ T %d ]; end;", udpfields-
>nc_value,udpfields->nc_value);
        sprintf(&matlab_str, "T=[T %d];",udpfields->nc_value,udpfields->nc_value);
        sprintf(&matlab_str, "T=[T %d];",udpfields->nc_value);
        engEvalString(matlab_Engine, &matlab_str);
}
```

Així tot el mecanisme de combinar els paquets que es feia al MATLAB l'hem eliminat i s'ha aplicat de la següent forma:
En el bloc de les funcions hem modificat just quant acabem de agafar el paquet amb la funció “ip_dgram_data_pkt_get” i el guardem amb un paquet anomenat data, seleccionem la informació dels camps del paquet UDP amb “op_pk_nfd_access(data,"fields",&udpfields)” i el que hem afegit nosaltres és que guardem el valor de nc_value en el vector “valo[i]” i mostrem per la consola quin és.

```
data = ip_dgram_data_pkt_get(pair->pkt);
op_pk_nfd_access(data,"fields",&udpfields);

valo[i] = udpfields->nc_value;
printf(" (%d):[val=%d]",i,udpfields->nc_value);
```

Amb aquesta modificació tenim els valors dels paquets guardats en el vector anomenat valo[2], això ens interessa per poder fer les operacions.
Tot seguit hem afegit:

```
if(i > 1)
        {
         printf("\n valor 0: %d",valo[0]);
         printf(" valor 1: %d",valo[1]);
         res = valo[0] ^ valo[1];
         printf(" Resultat xor:: %d\n",res);
         udpfields->nc_value = res;
        }

if (i > 0)
        {
         printf("Enviem el valor: %d\n",udpfields->nc_value);
         ip_dgram_data_pkt_set (pair->pkt, data);
         op_pk_send (pair->pkt, pair->stream_index);
         op_prg_mem_free(pair);
        }
```

Son dos condicions sobre la variable i, la i és un enter que l'hem usat per recórrer els paquets de la cua, ho sigui ens indica el número de paquets que hi ha. Si i és major que 1, ho sigui que hi han mínim dos paquets, ens mostrarà per consola els valors que porten i farà la suma xor. Amb el llenguatge C l'operació xor es representa amb l'operador “^”. Després mostrem el valor resultant i el guardem al nc_value del paquet que enviarem.
L'altre condició és per enviar el paquet, en aquest cas és major que 0 perquè encara que tinguem un sol paquet i no hagen necessitat fer la combinació l'enviarem per la interfície.
En aquest procés no haurem de modificar res del “header block”, així que amb el que hem fet fins ara tenim uns encaminadors que realitzen la suma xor amb els valors que els hi arriba, ara falta canviar els valors que s'envien per poder enviar la paraula HOLA.

Com hem dit abans qui crea les lletres HOLA és el R7, el que passa és que ho fa en cada iteració, ho sigui cada cop que rep un missatge de l'estació ws3_3.
Aquest cop si que modificarem el bloc de header, al posar les variables fora del bloc de les funcions el que fem es equivalent a que siguin variables globals, així en cada interfície posarem en un vector les lletres que volem enviar, per exemple en la interfície “toR2” afegim:

```
int data_nc[3] = {0,72,76};
int data_inx = 0;
```

La primera línia de codi és un vector de enters anomenat data_nc que conté, primer un zero, per qüestions de que el simulador no realitzava correctament les operacions a la primera iteració i després les lletres en codi ASCII, el 72 és la lletra “H” i el 76 és la “L”. La funció d'aquest vector és la de guardar les lletres que enviarem més tard.

En l'altre interfície, “toR1”, ens trobarem la mateixa estructura però amb els valor 79, que representa la “O” i el 65 que és la “A” codificada. Després hi trobem un enter anomenat data_inx que l'hem inicialitzat a 0, que fa la funció de contar en quin valor ens trobem del vector data_nc.

Per tal de crear i enviar aquestes lletres hem fet els següents canvis en el bloc de les funcions de les dues interfícies de sortida.

Tornem a anar a la funció “nc_process_queues(int flg)” que és la que en aquest encaminador s'encarregarà de enviar les enviar les lletres codificades.
Afegirem el següent codi just sota de “op_pk_nfd_access”.

```
udpfields->nc_value = data_nc[data_inx];
data_inx++;

printf(" (%d):[val=%d]",i,udpfields->nc_value);

if(udpfields->nc_value==72)
printf("\\Envio la 'H'");

if(udpfields->nc_value==76)
printf("\\Envio la 'L'");
```

En les dues primeres línies el que fem és dir que el primer valor del vector data_nc el guardi al nc_value del paquet per enviar, després fem que data_inx augmenti el valor per passar al següent número a enviar, que serà la primera lletra.

Després mostrem el valor per la consola que enviem amb el “printf”, aquest pas i els següents son simplement perquè quedi més clar del procediment a la consola. Ja que després el que fem és que si hem enviat un dels valors que coincideixen amb les lletres ens mostri per la consola la traducció del valor que enviem en lletra.

I l'únic que ens queda en aquest encaminador és la mateixa funció que en el R4 de enviar el paquet, amb la mateixa condició de que hi han de haver per lo menys un paquet per enviar, això simplement ho comprovarem amb el condicional “if (i > 0)”.

Únicament ens falta modificar els encaminadors R3 i R5 que seran els que rebran les lletres i per tant també hauran de fer la descodificació.
La idea és la mateixa que en el codificador, únicament que aquí com que haurem de fer la traducció de nombres a lletres també necessitem posar les variables en el bloc del header. En aquest cas posarem:

```
int resul[4];
int sul=0;
```

Bàsicament és el mateix que en l'altre header que hem modificat, únicament que enlloc d'omplir el vector “resul”, el deixem buit per omplir-lo amb les lletres que rebem.
En el bloc de funcions afegim el següent codi en la funció “nc_process_queues(int flg)” just després de la comprovació dels paquets que hem rebut:

```
if(i > 1 && valo[0]!=0)
 {
        /*CALCUL XOR*/

        printf("\n valor 0: %d",valo[0]);
        printf(" valor 1: %d",valo[1]);
        res = valo[0] ^ valo[1];
        printf(" Resultat xor:: %d\n",res);

        resul[sul]= valo[0];
        sul++;
        resul[sul]= res;
        sul++;
}


if(sul==4)
 {
        printf("\n Paraula final : %d %d %d %d",resul[0],resul[1],resul[2],resul[3]);
        printf(" ==> ASCII: HOLA\n");
}
```

El primer condicional comprova que tenim més de un paquet rebut i que el valor del primer nc_value no és 0, ja que com hem dit abans el 0 l'enviem per qüestions de la configuració de la xarxa i no ens interessa que s'operi amb els altres valors. Dins del condicional fem les operacions de descodificació, com que hem anat guardant els valors del nc_value en les posicions del vector valo[], ara només haurem de tornar a fer la suma xor per aconseguir la lletra que ens falta. A la tercera línia és quant aconseguim el valor resultant i el mostrem per consola amb el “printf”.

En les següents quatre línies el que fem és guardar els valors rebuts en el vector que hem creat anomenat resul[sul] i augmentar l'enter “sul” cada cop que afegim un número al vector.

Per finalitzar hem posat la condició de que si tenim tots els quatre valors de HOLA, ho comprovem amb si “sul” és igual a 4, i els mostrem per pantalla per saber que ja hem acabat de rebre la paraula.

Amb aquestes modificacions obtindrem que amb dues iteracions, de veritat tres però no la contem al ser de configuració, passarem quatre valors a dos receptors diferents.

6.4 Resultats

Tal i com hem dit enviem els caràcters H O L A codificats de la següent manera:

H O L A

72 79 76 65

D'aquests “H” i “L” van cap a R2, i “O” i “A” cap a R1. A la primera iteració transmetrem “H” a R2 i “O” a R1, la suma xor del encaminador R4 donarà 7, en la Fig. 6.3 es veu el recorregut dels valors:

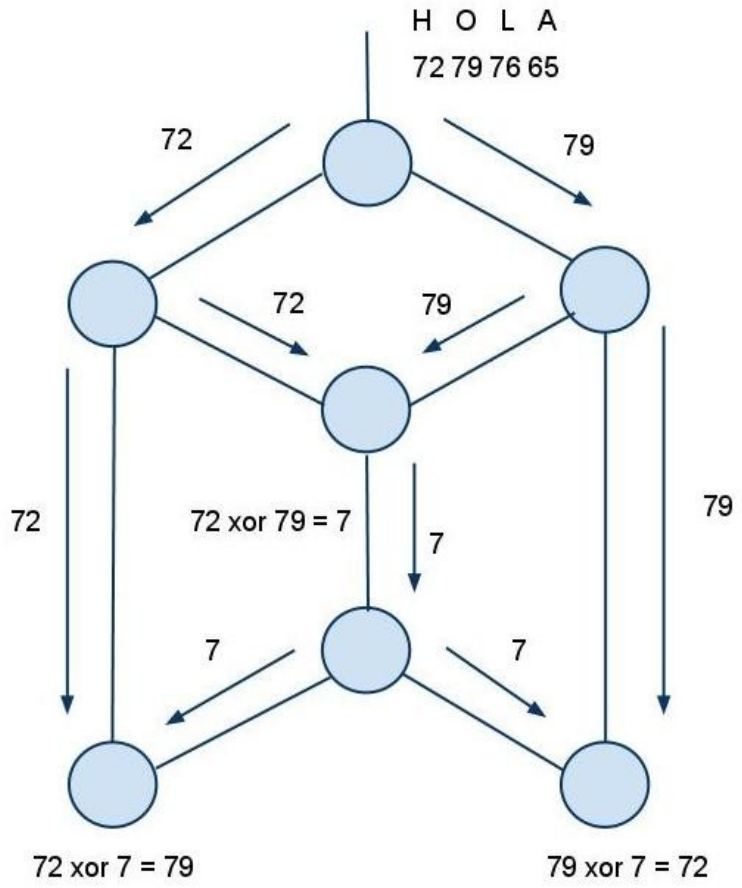


Fig. 6.3 Esquema de transmissió dels valors de la paraula HOLA.

La consola que ens surt és aquesta:

```
(110.500|19196)(0):[val=79]\Envio la 'O' 79
(110.500|19193)(0):[val=72]\Envio la 'H' 72
(111.000|2422) (0):[val=72] 72
(111.000|2428) (0):[val=72] 72
```

```
(111.000|7536) (0):[val=79] 79
(111.000|7545) (0):[val=79] 79

(112.500|10686) (0):[val=72] (1):[val=79]
 valor 0: 72 valor 1: 79 Resultat xor:: 7
 Enviem el valor: 7

(113.000|15910) (0):[val=7] 7
(113.000|15907) (0):[val=7] 7

(116.000|13243) (0):[val=79] (1):[val=7] 7
 valor 0: 79 valor 1: 7 Resultat xor:: 72

(116.000|4985) (0):[val=72] (1):[val=7] 7
 valor 0: 72 valor 1: 7 Resultat xor:: 79
Final de la iteracio 1

(120.500|19196)(0):[val=65]\Envio la 'A' 65
(120.500|19193)(0):[val=76]\Envio la 'L' 76

(121.000|2422) (0):[val=76] 76
(121.000|2428) (0):[val=76] 76

(121.000|7536) (0):[val=65] 65
(121.000|7545) (0):[val=65] 65

(122.500|10686) (0):[val=76] (1):[val=65]
 valor 0: 76 valor 1: 65 Resultat xor:: 13
 Enviem el valor: 13

(123.000|15910) (0):[val=13] 13
(123.000|15907) (0):[val=13] 13

(126.000|13243) (0):[val=65] (1):[val=13] 13
 valor 0: 65 valor 1: 13 Resultat xor:: 76

 Paraula final : 72 79 76 65==> ASCII: HOLA

(126.000|4985) (0):[val=76] (1):[val=13] 13
 valor 0: 76 valor 1: 13 Resultat xor:: 65

 Paraula final : 72 79 76 65==> ASCII: HOLA
```

Podem observar les dos iteracions i com al final de la segona tenim tots els quatre valors que formen la paraula.

Els primers valors que estan dins del principi de cada línia signifiquen (“temps de la simulació”|”valor que es correspon a una interfície”). Seguidament entre parèntesis ens indica la posició de la cua en que es troba el paquet determinat. El “:” separa la posició del valor que transporta el paquet, que va entre les claus “[ ]”. En algunes interfícies només tindrem un paquet a la cua i per tant simplement haurem de fer l'enviament, en els que veiem que tenim varis paquets en cua realitzarem l'operació de xor, indicada just sota de aquesta línia amb els valors a operar i el resultat. La valoració global d'aquesta prova és positiva ja que hem obtingut els resultats esperats.

CAPÍTOL 7. IMPLEMENTACIÓ MATLAB

7.1 Simulació del Network Coding lineal (LNC)

En aquest apartat ens centrarem en estudiar el funcionament del mecanisme de la codificació lineal i les diferències que hi podem trobar amb el mètode booleà.

L'objectiu que ens hem marcat és realitzar la xarxa que van plantejar en el document de FUNLAB però només fent servir el programa MATLAB. El que feia el codi de FUNLAB era establir els components amb el OPNET i les operacions per MATLAB, nosaltres ens centrarem en aquesta part i no realitzarem l'estructura amb els elements de la xarxa.

La xarxa que plantejaven els de FUNLAB era la següent, Fig. 7.1:

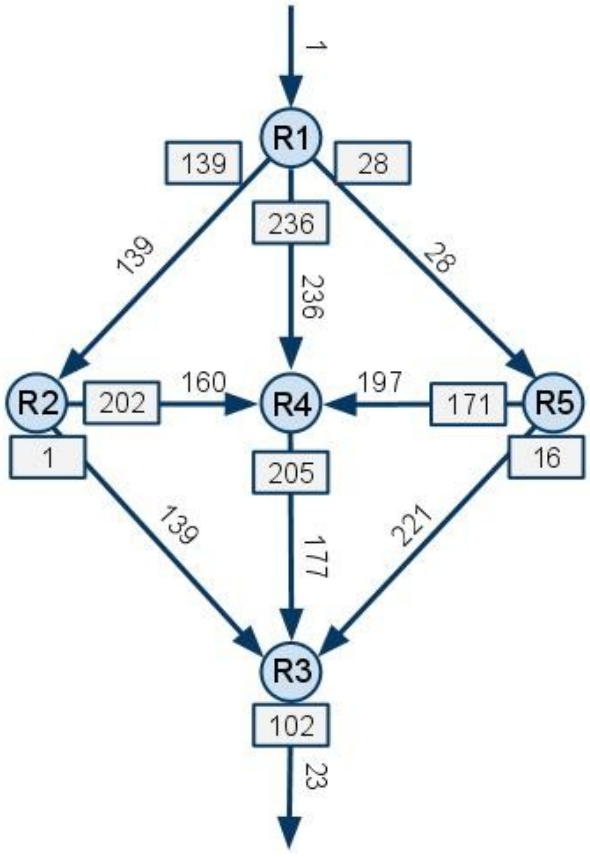


Fig. 7.1 Esquema de la xarxa amb els resultats de les codificacions i els coeficients a fer servir.

On els cercles representen els encaminadors, valors dins els rectangles son els coeficients lineals que s'usen en cada interfície i els números sobre les fletxes son els valors que enviem, equivalent al “nc_value” de l'altre exercici.

7.2 MATLAB

El MATLAB és un programari de simulació matemàtica molt potent que s'usa pràcticament en qualsevol carrera tècnica i amb múltiples implementacions en els treballs actuals. Creat per l'empresa “The MathWorks”, agafa el seu nom de l'anglès “Matrix laboratory” ja que permet modificar i crear fàcilment matrius, a més de dibuixar funcions i dades, implementar algoritmes, etc. Però és un programari privatiu i ha rebut crítiques per no obrir el seu codi. Tot i així existeixen alternatives lliures, com és el Scilab o el GNU Octave.

7.3 Implementació del codi en MATLAB

La implementació de la codificació lineal l'hem fet de la següent forma:

Per cada interfície d'un encaminador farà la operació del valor que l'hi arriba per el coeficient propi de la interfície, per exemple una interfície fa les següents funcions:

```
%%%%%%%%%%R1%%139%%%%%%%%%

t=1;
y=0;
coe=139;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);
end

z_double11 = double(y.x);
```

La variable “t” és el valor que s'envia com a valor original, l'hem determinat com a 1. La “y” és la variable que emmagatzemarà el resultat de l'operació entre el valor i el coeficient i “coe” és el valor del coeficient per la interfície que es troba.

Per fer les operacions el que tenim de fer és recórrer els valors que hagem rebut amb el “for i=1:lenght(t)” amb la fórmula “`y = gf(coe(i),8) * gf(t(i),8)`” per a cada valor que hem rebut. Podem veure que es basa en una multiplicació d'uns valors dins d'uns camps de Galois (gf) on primer hi ha el valor del coeficient o del valor original i després hi ha el polinomi que es 8. Això significa que el camp finit tindrà una longitud de fins a 256 (2^8) i per tant el resultat de “`gf(coe(i),8)`” és 139.

Per últim la variable “z_double11” és el resultat després de convertir a double el valor resultant de l'operació del coeficient. El número 11 és una forma per distingir els valors de les interfícies que en aquest cas significa el router 1 i la interfície 1.

En les interfícies que hi arribin més de un paquet s'hauran d'afegir algunes línies, per exemple:

```
%%%%%%%%%%R4%%205%%%%%%%%%%

t =[z_double51 z_double12 z_double22];
coe=[205 205 205];
z=0;
for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);
z = gf(y,8) + gf(z,8);
end
z_double4 = double(z.x);
```

La interfície del router 4 rebrà 3 paquets i per tant tindrem un vector a “t”, al lloc dels valors útils, podem observar que son els resultats de les altres interfícies que enganxen amb l'encaminador 4.

El “coe” és el coeficient que ara és un vector perquè l'haurem d'aplicar en els tres valors útils i la “z” fa la mateixa funció que la “y” però amb l'operació de suma de després.

Recorreguem els tres valors i farem les operacions dins del camps de Galois, i per cada operació fem la suma “`z = gf(y,8) + gf(z,8)`” de el valor resultant amb els dels altres paquets. Z és 0 perquè el primer valor no es sumarà amb res.

7.4 Resultats

El codi complet el podreu trobar a l'annex 2, on hi han tots els arxius utilitzats a més dels resultats de la consola.

Com que hem aconseguit simular el que ens teníem previst, i pensant que no ens havien de sortir els mateixos resultats del FUNLAB, considerem que és un resultat positiu. La implementació ha estat de la codificació, la descodificació s'ha de revisar.

CAPÍTOL 8. IMPACTE AMBIENTAL

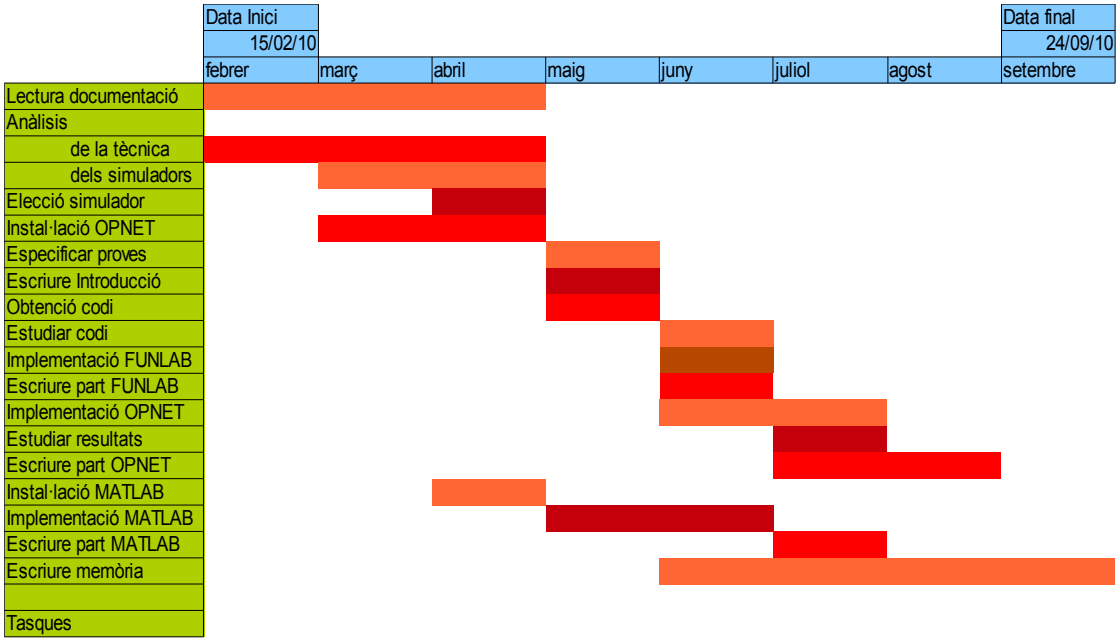
Avui en dia vivim en un món on els recursos ja no son il·limitats, els tenim de compartir entre països o fer guerres per aconseguir-los. L'egoisme intrínsec de l'humanitat no ha sabut veure els beneficis de tractar la naturalesa a llarg termini, i potser ara és massa tard, perquè la terra casi segur realitzarà un canvi auto-regulador climàtic que podria afectar i molt a l'espècie humana tan adaptada ja no al medi, si no a una societat consumista.

És per evitar que aquest canvi sigui més gran del que molta gent podria suportar que s'estan fent esforços a última hora implementant energies renovables i minimitzant l'impacte ambiental de les nostres accions o dels objectes que usem.

En aquest treball hem analitzat una tècnica que el que fa és estalviar energia al realitzar dues transmissions amb una sola, això optimitza el temps de transmissió d'un bit. Si mai s'arriba a implementar massivament a les xarxes de Internet pot significar un estalvi molt gran de transmissions i per tant d'energia que utilitzem per la transmissió de les dades.

CAPÍTOL 9. VALORACIÓ ECONÒMICA

Per de establir una valoració econòmica d'aquest treball farem el diagrama de “Gantt”.



El diagrama està implementat a mida major a l'annex 3.

Podem apreciar que aquest treball ha costat vuit mesos de treball per part de l'investigador on en el cas de l'autor correspon a un estudiant apunt de ser enginyer tècnic. Aproximadament fixem el cost de l'enginyer en 7€ l'hora, contant que aquest treball ha estat realitzat de mitja jornada i que per tant s'han treballat 175 dies, quatre hores cada dia, ens surt un cost total per part del treballador de 4900€. Gràcies a la utilització de programari que no ha significat cap cost addicional podem dir que els 4900€ és la valoració econòmica total.

CAPÍTOL 10. CONCLUSIONS

Després d'haver realitzat l'estudi del “Network Coding” en podem treure les següents conclusions:

El “Network Coding” és una tècnica que realment ofereix una millora en el desplegament d'un esquema “multicast”, oferint millors retards, fent més eficient les transmissions en els medis sense fils i oferint una seguretat que ve intrínseca amb la tècnica. El problema més gran que es troba és en l'aplicació pràctica ja que si es fa en nivells OSI baixos aconseguirem un rendiment bo però s'hauran de canviar tots els dispositius intermediaris que usin la tècnica, i això resulta molt costós. Entre els diferents modes de codificació creiem que el que té més futur és el lineal aleatori (RLNC), ja que no necessita la coordinació de tota la xarxa, cosa molt útil en entorns com són Internet.

El codi que vam trobar de FUNLAB ha resultat no ser tant útil com en un principi ens pensàvem, ja que a part de trobar errors en el document hem pogut observar que només estava implementada la codificació, sense la descodificació.

En quant al les nostres implementacions podem dir que han sigut satisfactòries. A la simulació en OPNET on havíem d'enviar la paraula HOLA i rebre-la en els dos receptors, usant la codificació “XOR” en els nodes intermedis, ha estat un èxit, ja que ho hem simulat sense cap problema. Es pot observar el codi que ens surt a la consola que rebem en els dos dispositius la paraula.

I en la simulació en MATLAB hem aconseguit aplicar els valors de FUNLAB i ens han sortit uns valors que no eren iguals als del departament però creiem que estan bé al trobar errors en el document de FUNLAB.

10.1 Línies futures

Hem pogut observar que és un tema que actualment atrau l'interès comunitat científica ja que han sortit molts treballs dedicats en aquesta tècnica.

Per aquest treball s'ha de comprovar més exhaustivament el funcionament del “Network Coding” s'hauria de fer un estudi comparatiu entre el mètode tradicional de encaminament i el “Network Coding”, tot comparant els retards i els rendiments de la transmissió. També s'hauria de simular un escenari fent servir la codificació de RLNC.

Per tal posar a prova el mecanisme en un entorn real seria interessant fer un estudi sobre el comportament en un entorn sense fils amb dispositius ad-hoc.

CAPÍTOL 11. BIBLIOGRAFIA

- Documents

[1]: R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung, “Network Information Flow,” in IEEE Transactions on Information Theory, July 2000, vol. 46, pp. 1204–1216.

[2]:R. W. Yeung, “Multilevel diversity coding with distortion,” IEEE Transaction on Information Theory, vol. 41, pp. 412–422, 1995

[3]:T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, “The benefits of coding over routing in a randomized setting,” *International Symposium on Information Theory (ISIT)*, 2003.

[4]: S. Jaggi, P. Sanders, P. A. Chou, M. Effros, S. Egner, and L. Tolhuizen, “Polynomial time algorithms for multicast network code construction,” *IEEE Trans. Inf. Theory*, Vol. 51, No.6, 2005

[5]:C. Fragouli, E. Soljanin, and A. Shokrollahi, "Network Coding as a Coloring Problem," *Proceeding of CISS*, 2004.

[6]:http://www.wired.com/magazine/2010/08/ff_webrip/all/1

[7]:Fulkerson,D. R. ; Harding,Gary, “On Edge-Disjoint Branchings”, 2 mar 2007.

[8]:Dougherty, Freiling, and Zeger. Insufficiency of Linear Coding in Network Information Flow.

[9]:http://www.ee.washington.edu/research/funlab/network_coding/index.html

[10]:Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, Senior Member, IEEE, and Jon Crowcroft, Fellow, IEEE, "XORs in the Air: Practical Wireless Network Coding".

[12]:C. Gkantsidis and P. R. Rodriguez. Cooperative security for network coding file distribution. Technical Report accepted at Infocom, 2006.

[13]:Alberto José González, Francesc Rillo, Jesus Alcober, Daniel Rodríguez, Javier López. “Streaming P2P robusto en redes Ad-hoc utilitzando informacion social”, JITEL 2009, Cartagena, Spain. September 2009.

[14]:C.Fragouli and E.Soljanin. Network Coding Fundamentals. *Foundations and Trends in Networking*, Vol. 2, Issue 1:1-133, 2007.

[15]:C. Fragouli and E. Soljanin. Network Coding Applications. *Foundations and Trends in Networking*, Vol. 2, Issue 2:135-269, 2007.

- Enllaços

Avalanche : research.microsoft.com/en-us/projects/avalanche/default.aspx

Nimbus : research.microsoft.com/en-us/projects/nimbus/default.aspx

OPNET : www.opnet.com

MATLAB : www.mathworks.com

FUNLAB : www.ee.washington.edu/research/funlab/network_coding/index.html

Escola Politècnica Superior de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

ANNEXOS

SIMULACIÓ DE NETWORK CODING

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Telemàtica

AUTOR: Pau Pericay Vendrell

DIRECTOR: Jesús Alcober Segura

CO-DIRECTOR: Alberto José González Cela

DATA: 23 de setembre del 2010

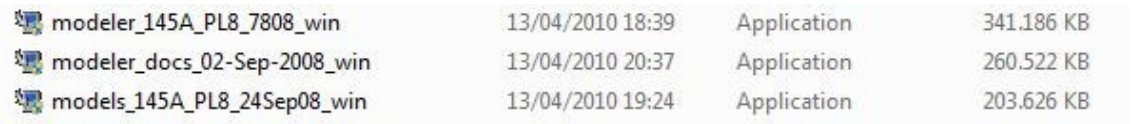
ANNEX 1
INSTALACIÓ DE OPNET MODELER 14.5

Aquest annex és una guia de la instal·lació del software OPNET MODELER de la versió 14.5A, que és la que hem usat en aquest treball. Cal dir que hem afegit aquesta guia perquè l'obtenció i la conseqüent instal·lació no ha estat ràpida ni fàcil, ja que no existeix un manual d'usuari que t'expliqui clarament els passos a seguir. Així que amb la prova i error, i mirant algunes webs de Internet hem elaborat la següent guia d'instal·lació per un entorn Windows. El nostre software ha estat instal·lat en un entorn XP SP 3, però no hi hauria de haver diferències en un Vista o 7, simplement canviaran la localització d'algunes carpetes o noms de variables.

El que tenim de fer primer de tot és baixar o instal·lar un compilador de c++, per exemple si tenim un "Microsoft Visual Studio" ja el tenim. En el cas que ja el tinguem instal·lat podrem passar al següent pas, si no ens el tenim de baixar el "Visual Studio" o l'aconseguim d'alguna institució com pot ser la universitat. En el nostre cas ja el teníem i només el vam instalar.
A la següent captura es pot apreciar la carpeta de destinació a que instal·larem el programari.

Aquesta localització serà important per més tard, quant hagem de modificar les variables del sistema.

El següent pas és baixar-se el programari OPNET MODELER.
El que tindrem de fer primer es registrar-nos a la web posant les nostres dades personals i les del centre educatiu o empresa de que formem part. En el nostra cas em escollit baixar-nos una versió acadèmica ja que la UPC te llicència d'aquest programa i per tant no ens representa cap cost extra. Així que vam demanar a la nostra universitat la llicència que és un nom d'usuari i un password. Ara ja podrem baixar-nos el programa, simplement busquem la versió que ens interessa a la seva pàgina, hem dit que faríem servir la versió 14.5A i posem per baixar les tres parts que hi han disponibles: el “Modeler”, que és el programa en sí, el “Documents” que tal com indica el seu nom son les ajudes i documents del programa i el “Models” que hi podem trobar els models de dispositius que farem servir per crear les nostres xarxes.
Un cop baixats tindrem els tres arxius:

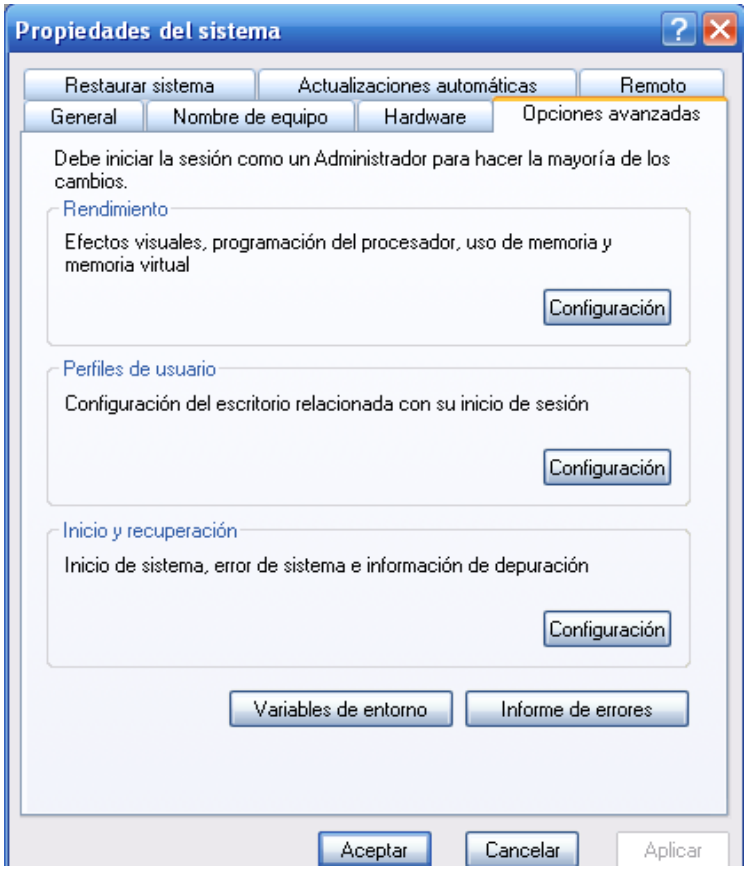


Seguidament ja podem continuar amb la pròpiament dita instal·lació del programa a l'ordinador. Cal tenir en compte l'ordre de instal·lació dels diferents components, així primer instal·larem el “modeler”, segon el “models” i per últim el “modeler_docs”. El procés d'instal·lació no comporta dificultats i no hauria de donar cap problema, simplement tenir en compte a quin directori s'instal·la.

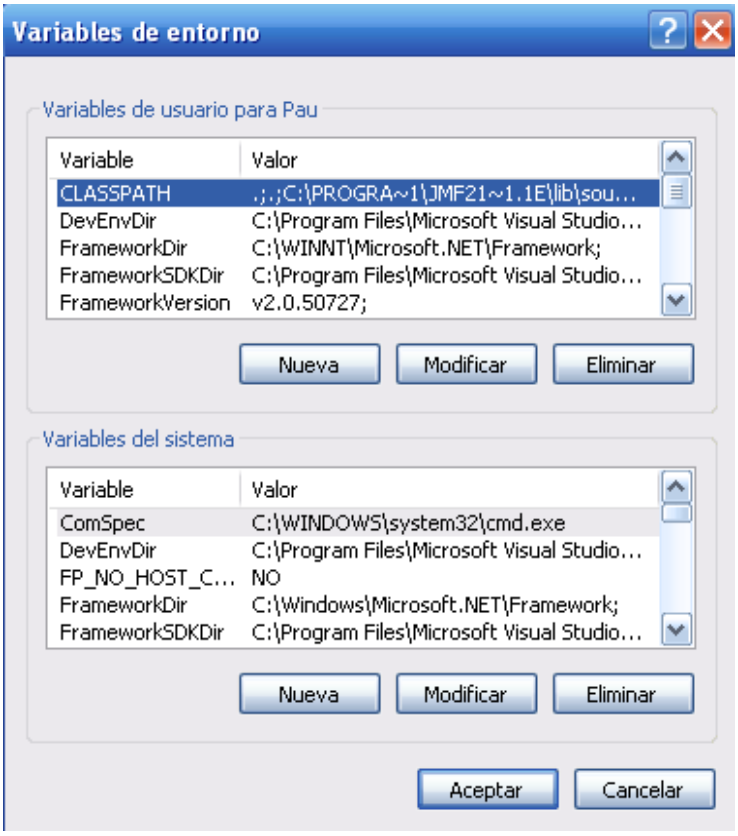
Ara vam tenir de fer un pas extra que és exclusiu en el nostre cas, ja que el codi de “funlab” feia les operacions en el MATLAB, de forma que vam tenir de instal·lar el MATLAB i ajuntar-lo amb el OPNET.

La instal·lació del MATLAB no hi hauria d'haver cap problema, per aquest treball vam fer servir la versió R2008b, com en els altres casos és bo saber el directori on s'ha instal·lat el programa.

Ara ja tenim tots els programes que necessitem a l'ordinador, el següent pas és anar a les variables del sistema del Windows a INICIO->MI PC->(fer click dret)->PROPIEDADES->(click a la pestanya) OPCIONES AVANZADAS-> VARIABLES DE ENTORNO.
A la següent imatge es pot apreciar la finestra on trobarem les variables del sistema:



Cliquem al botó de “variables de entorno” i ens sortirà la finestra següent:



Ens trobem que hi han dues classes de variables, les de usuari i les del sistema. A nosaltres ens interessa afegir a les del sistema les següents variables:

PATH = C:\Program Files\MATLAB\R2008b\bin\win32;

INCLUDE = C:\Program Files\MATLAB\R2008b\extern\include;

LIB = C:\Program Files\MATLAB\R2008b\extern\lib\win32\microsoft;

Aquestes variables serveixen per tal de configurar el MATLAB, és important tenir en compte que depenent del sistema operatiu no agafarà bé l'adreça si no posem “PROGRA~1” en lloc de “Program Files” .

Es van haver d'afegir algunes variables més, per tal de que quedi clar quines son les necessàries posarem les que tenim configurades en el ordinador en que es van fer les implementacions, sense les anteriors que les haurem d'afegir:

PATH =C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;C:\Program Files\Microsoft Visual Studio 8\VC\BIN;C:\Program Files\Microsoft Visual Studio 8\Common7\Tools;C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\bin;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\bin;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\bin;C:\WINNT\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual Studio 8\VC\VCPackages;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\Bin;C:\WINNT\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual Studio 8\VC\bin;C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;C:\Program Files\Microsoft Visual Studio

8\VC\vcpackages;C:\Program Files\MATLAB\R2008b\extern\include;C:\Program Files\MATLAB\R2008b\bin\win32;

INCLUDE =C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\INCLUDE;C:\Program Files\Microsoft Visual Studio 8\VC\INCLUDE;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\include;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\include;

LIB= C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\LIB;C:\Program Files\Microsoft Visual Studio 8\LIB;C:\Program Files\Microsoft Visual Studio 8\VC\PlatformSDK\lib;C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0\lib;

LIBPATH= C:\Windows\Microsoft.NET\Framework\v2.0.50727;C:\Program Files\Microsoft Visual Studio 8\VC\ATLMFC\LIB;

VCINSTALLDIR= C:\Program Files\Microsoft Visual Studio 8\VC;

VSINSTALLDIR= C:\Program Files\Microsoft Visual Studio 8;

VS80COMNTOOLS= C:\Program Files\Microsoft Visual Studio 8\Common7\Tools\;

DevEnvDir= C:\Program Files\Microsoft Visual Studio 8\Common7\IDE;

FrameworkDir = C:\Windows\Microsoft.NET\Framework;

FrameworkSDKDir= C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0;

FrameworkVersion= v2.0.50727;

MSVCDir= C:\Program Files\Microsoft Visual Studio 8\Vc;

NetSamplePath= C:\Program Files\Microsoft Visual Studio 8\SDK\v2.0;

Un cop hem posat totes les variables del sistema ja podrem executar el OPNET per primer cop. Acceptem els termes d'ús i ens trobarem amb la finestra inicial, haurem de anar a Edit ->Preferences.
S'ens obrirà una finestra on a dalt de tot hi ha una entrada per buscar, escriurem “bind_shobj_flags” i suposant que tenim un sistema de 32-bits com era en el nostre cas, anirem a la part de “value” del costat de on posa “32-bits Network Reprositories flags” i afegim el següent:

/LIBPATH:C:\PROGRA~1\MATLAB\R2008b\extern\lib\win32\microsoft
/LIBPATH:C:\PROGRA~1\MATLAB\R2008b\extern\include

Les dues línies han d'estar separades per un espai.
Després farem el mateix procediment de buscar amb “bind_shobj_libs” i a l'apartat de “value” afegirem les següents llibreries: “libmat.lib libeng.lib libmex.lib libmx.lib“.

Ara ja tenim el MATLAB totalment enllaçat amb el OPNET.

ANNEX 2
IMPLEMENTACIÓ LNC EN MATLAB

En aquest annex hi ha el codi complet de la simulació realitzada amb el MATLAB del LNC.
El codi en la codificació esta dividit entre les interfícies dels nodes, on hem afegit un comentari de l'encaminador que representen i el coeficient que fan servir.

El següent arxiu “lnc.m” hi trobarem el codi de la codificació:

```
%%CODIFICACIÓ I ENVIAMENT%%

%%%%%%%%%%R1%%139%%%%%%%%%

t=1;
y=0;
coe=139;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);

end
z_double11 = double(y.x);

%%%%%%%%%%R1%%236%%%%%%%%%

t=1;
y=0;
coe=236;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);

end
z_double12 = double(y.x);

%%%%%%%%%%R1%%28%%%%%%%%%

t=1;
y=0;
coe=28;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);

end
z_double13 = double(y.x);

%%%%%%%%%%R2%%1%%%%%%%%%

t =z_double11;
```

```
y=0;
coe=1;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);

end

z_double21 = double(y.x);

%%%%%%%%%%R2%%202%%%%%%%%%%

t =z_double11;
y=0;
coe=202;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);

end

z_double22 = double(y.x);

%%%%%%%%%%R5%%171%%%%%%%%%%

t =z_double13;
y=0;
coe=171;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);

end
z_double51 = double(y.x);

%%%%%%%%%%R5%%16%%%%%%%%%%

t =z_double13;
y=0;
coe=16;

for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);

end
z_double52 = double(y.x);

%%%%%%%%%%R4%%205%%%%%%%%%%

t =[z_double51 z_double12 z_double22];
coe=[205 205 205];
z=0;
for i=1:length(t),
y = gf(coe(i),8) * gf(t(i),8);
z = gf(y,8) + gf(z,8);
end
z_double4 = double(z.x);
```

En el mateix arxiu vam provar de realitzar la descodificació , per això vam utilitzar una funció “gauss_elim”, que realitza l'eliminació gaussiana que vam posar a l'arxiu del mateix nom.

```
%%%DECODIFICACIO%%%

%Matriu coeficients
%C=[0 139 202;0 0 236;0 28 171];
C=[139 202 205;236 205 0;28 171 205];

%Matriu de valors codificats
%x = [160,236,197];
x=[160 236 197];

r=gauss_elim(C,x);

%El resultat de l'eliminació de gauss és:
re=(-0.9652)+(2.2623)+(-0.7943);

b=gf(re,8);
b_double = double(b.x);
```

Els resultats tot i sortir el valor esperat, vam considerar que no podíem donar com a vàlid el codi.

I a l'arxiu “gauss_elim.m” com hem dit hi trobarem el codi de l'eliminació de Gauss:

```
function x = gauss_elim(A,b)

% inputs: A, an n x n matriu
%         b, an n x 1 vector
%
% output: x, an n x 1 vector que resol Ax = b


% aconsegueix la mida de la matriu A
n = length(A);


% forward elimination
for i = 1:n-1
  for j = i+1:n

    m = A(j,i)/A(i,i);

    A(j,:) = A(j,:) - m*A(i,:);
    b(j) = b(j) - m*b(i);
  end
end
```

```
% vector de solucions
x = zeros(n,1);

% backward substitution
x(n) = b(n)/A(n,n);

for i = n-1:-1:1
  x(i) = (b(i) - A(i,i+1:n)*x(i+1:n))/A(i,i);
end
```

ANNEX 3
DIAGRAMA DE GANTT