Escola Tècnica Superior d'Enginyeria
Electrònica i Informàtica La Salle

Treball Final de Màster

Màster Universitari en Enginyeria de Telecomunicació

Disseny, implementació i simulació de mecanismes per a protocols de transport en les *Long Fat Networks* heterogènies

Alumne
Adrià Mallorquí Campà

Professor Ponent
Alan Briones Delgado

ACTA DE L'EXAMEN
DEL TREBALL FI DE MASTER

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Adrià Mallorquí Campà

va exposar el seu Treball de Fi de Màster, el qual va tractar sobre el tema següent:

Disseny, implementació i simulació de mecanismes per a protocols de transport en les *Long Fat Networks* heterogènies

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

laSalle
Ramon Llull University

Disseny, implementació i simulació de mecanismes per a protocols de transport en les *Long Fat Networks* heterogènies

Treball Final de Màster

16/9/2019

La Salle Enginyeria – Universitat Ramon Llull
Departament de Telemàtica

Autor: Adrià Mallorquí
Ponent: Alan Briones

laSalle

Ramon Llull University

Abstract

CAT: TCP és el protocol de transport orientat a connexió més utilitzat avui en dia. Aquest no aconsegueix uns bons nivells d'utilització de l'ample de banda en enllaços de gran capacitat i alta latència (Long Fat Networks o LFN) per culpa de les mancances del seu control de congestió. Al llarg dels anys han anat sorgint diverses modificacions del control de congestió de TCP per tal de millorar-ne el seu rendiment, així com també han aparegut altres protocols alternatius no basats en TCP. Cap d'ells però, està orientant a tenir un comportament *unfriendly* per tal d'ocupar el màxim de l'ample de banda possible.

El protocol MBTAP es va pensar per poder transferir fitxers de gran mida en enllaços d'alta velocitat, amb l'objectiu d'obtenir el màxim d'ample de banda possible amb un control de congestió agressiu. Aquest protocol però, no pot obtenir un rendiment òptim quan la LFN és heterogènia (amb un enllaç sense fils), ja que no és capaç de diferenciar les pèrdues degudes a la congestió de les pèrdues degudes a erros de canal. Aquesta mancança també es dóna amb TCP, motiu pel qual han anat sorgint modificacions del protocol que intenten solucionar el problema. Cap d'ells, però, està pensat per tenir un comportament agressiu en LFNs, com pretén MBTAP.

D'aquest fet sorgeix l'evolució del protocol, l'OMBTAP, que tracta d'adaptar l'anterior versió del protocol a escenaris de xarxes LFN heterogènies. En aquest treball es presenta el disseny i la implementació del protocol OMBTAP en un simulador de xarxa, Riverbed Modeler, i es realitzen diverses proves. Els resultats obtinguts són força satisfactoris, ja que es pot comprovar que OMBTAP presenta una millora de rendiment notable respecte MBTAP en escenaris de xarxes heterogènies. Tot i això, el funcionament del protocol encara és inestable quan diversos fluxos OMBTAP comparteixen el mateix enllaç, motiu pel qual es presenten les futures línies de desenvolupament.

Paraules clau: Protocol de transport, estimació de la capacitat de la xarxa, velocitat d'enviament, simulació, Long Fat Networks, xarxes heterogènies, xarxes sense fils.

CAST: TCP es el protocolo de transporte orientado a conexión más utilizado hoy en día. Este no consigue unos buenos niveles de utilización del ancho de banda en enlaces de gran capacidad y alta latencia (Long Fat Networks o LFN) por culpa de las carencias de su control de congestión. A lo largo de los años han ido surgiendo diversas modificaciones del control de congestión de TCP para mejorar su rendimiento, así como también han aparecido otros protocolos alternativos no basados en TCP. Ninguno de ellos, pero, está orientando a tener un comportamiento *unfriendly* para ocupar el máximo del ancho de banda posible.

El protocolo MBTAP se pensó para poder transferir archivos de gran tamaño en enlaces de alta velocidad, con el objetivo de obtener el máximo de ancho de banda posible con un control de congestión agresivo. Este protocolo, sin embargo, no puede obtener un rendimiento óptimo cuando la LFN es heterogénea (con un enlace inalámbrico), ya que no es capaz de diferenciar las pérdidas debidas a la congestión de las pérdidas debidas a errores de canal. Esta carencia también se da con TCP, por lo que han ido surgiendo modificaciones del protocolo que intentan solucionar el problema. Ninguno de ellos, sin embargo, está pensado para tener un comportamiento agresivo en LFNs, como pretende MBTAP.

De este hecho surge la evolución del protocolo, el OMBTAP, que trata de adaptar la anterior versión del protocolo a escenarios de redes LFN heterogéneas. En este trabajo se presenta el diseño y la implementación del protocolo OMBTAP en un simulador de red, Riverbed Modeler, y se realizan varias pruebas. Los resultados obtenidos son bastante satisfactorios, ya que se puede comprobar que OMBTAP presenta una mejora notable en el rendimiento respecto MBTAP en escenarios de redes heterogéneas. Sin embargo, el funcionamiento del protocolo aún es inestable cuando varios flujos OMBTAP comparten el mismo enlace, por lo que se presentan las futuras líneas de desarrollo.

Palabras clave: Protocolo de transporte, estimación de la capacidad de la red, velocidad de envío, simulación, Long Fat Networks, redes heterogéneas, redes inalámbricas.

ENG: TCP is the most commonly used connection-oriented transport protocol. Even though, it does not achieve good levels of bandwidth utilization on high capacity and high latency links (Long Fat Networks or LFN) due to the characteristics of its congestion control. Over the years, various modifications to the TCP congestion control have been emerging to improve its performance, as well as other non-TCP-based alternative protocols have appeared. Despite that, none of them is aiming to have an unfriendly behavior to occupy the maximum possible bandwidth.

The MBTAP protocol was designed to be able to transfer large files on high-speed links, in order to obtain the maximum possible bandwidth with an aggressive congestion control. This protocol, however, cannot obtain an optimal performance when the LFN is heterogeneous (with a wireless link), since it is not able to differentiate losses due to congestion of losses due to channel errors. This deficiency also occurs with TCP, so modifications of the protocol have been emerging that try to solve the problem. None of them, however, is intended to have aggressive behavior in LFNs, as MBTAP claims.

From this fact, the evolution of the protocol arises, the OMBTAP, which tries to adapt the previous version of the protocol to heterogeneous LFN scenarios. This paper presents the design and implementation of the OMBTAP protocol in a network simulator, Riverbed Modeler, and several tests are carried out. The results obtained are quite satisfactory, since it can be seen that OMBTAP has a notable improvement in performance compared to MBTAP in heterogeneous network scenarios. However, the operation of the protocol is still unstable when several OMBTAP flows share the same link, so future development lines are presented.

Keywords: Transport protocol, network capacity estimation, throughput, simulation, Long Fat Networks, heterogeneous networks, wireless networks.

laSalle
Ramon Llull University

ÍNDEX

1 Introducció

La Internet que coneixem avui en dia ha canviat molt respecte la que va ser concebuda als seus inicis. Al llarg dels anys els usu aris han anat incrementant les seves demandes de consum, tant en termes de volum de dades com de rapidesa. Això ha provocat un canvi en el paradigma de les xarxes d'interconnexió de dades.

Una de les causes d'aquesta evolució és l'increment de consum de serveis multimèdia en línia. Actualment, existeix un gran volum d'intercanvi de dades d'aquest tipus, ja sigui per l'enorme diversitat de continguts oferts com per la gran mida d'aquests. Aquest fet ha evidenciat la necessitat de disposar de xarxes amb un ample de banda major per poder realitzar connexions *end-to-end* de bona qualitat, i en molts casos els nodes finals es troben separats per llargues distàncies.

Dins les xarxes d'alta capacitat s'hi troben les *Long Fat Networks* (LFN). Aquestes es caracteritzen per tenir un gran ample de banda i valors de *Round Trip Time* (RTT) [1] elevats. Una xarxa es considera que és LFN si el seu *Bandwidth-Delay Product* (BDP) és superior als 100.000 bits [1]. Per exemple, un enllaç WAN SONET-3 (148,6 Mbps) que tingui un RTT de 700 μs obté un BDP de 104.020 bits, de manera que ja es pot considerar una LFN.

Les característiques de les LFNs provoquen que el protocol de transport més utilitzat a la xarxa, *Transmission Control Protocol* (TCP), baixi el seu rendiment i no aprofiti tot l'ample de banda de l'enllaç.

Aquest fet ha originat la definició de diverses extensions del protocol per tal de mitigar-ne les seves mancances [2], així com múltiples variacions en el seu control de congestió. A més, també han sorgit nous protocols que permetin aprofitar al màxim les capacitats de les xarxes LFN. Un d'aquests protocols és l'anomenat MBTAP [3], dissenyat pel grup de recerca GRITS[2] de La Salle-URL. Aquest protocol està orientat a l'enviament d'arxius de gran mida a través d'una LFN. A més, aquest protocol està concebut per tenir un comportament agressiu i "*unfriendly*", és a dir, que aprofiti la major part de l'ample de banda de l'enllaç fins i tot quan aquest es comparteix amb fluxos de dades d'altres protocols. Per aquest motiu, és necessari que el protocol sigui capaç d'estimar de forma precisa quina és la capacitat màxima de l'enllaç i així poder establir una velocitat d'enviament que concordi amb aquesta estimació.

Des del grup de recerca, es va realitzar una primera implementació física (pre-prototipat) del protocol, amb la qual es va poder veure que s'havien de realitzar alguns canvis en el seu disseny [4]. Un cop programada aquesta primera versió del protocol per entorns físics, es va creure convenient realitzar una implementació del protocol en un simulador de xarxa (Riverbed Modeler[3]). El fet de disposar del protocol en un entorn de simulació permet que es realitzin proves que no es podrien executar en entorns físics a causa de

[1] : EL *Round Trip Time* es defineix com el temps que passa des que s'envia un paquet fins que se'n rep la seva resposta.

[2] : Grup de Recerca en Internet Technologies and Storage.

[3] : Riverbed Modeler és un software simulador de xarxa. Tota la informació a: https://www.riverbed.com/es/products/steelcentral/steelcentral-riverbed-modeler.html

la manca de recursos (com per exemple simular escenaris amb enllaços dedicats de 20 Km i d'alta capacitat o fer proves amb un nombre elevat de nodes), obrint així un nou ampli ventall de possibilitats. Gràcies a les simulacions, es va observar que el mètode d'estimació de la capacitat d'un enllaç no assolia el seu objectiu, ja que el que realment s'estimava era l'ample de banda lliure (disponible) del canal. En una posterior iteració, es va redissenyar el mecanisme d'estimació de l'ample de banda i es va modificar lleugerament el control de congestió del protocol per assolir l'objectiu d'utilitzar la màxima capacitat de l'enllaç. Es van realitzar diverses proves dins del simulador Riverved Modeler, i els resultats van ser satisfactoris tot i observar certes limitacions:

- El protocol no era eficient quan dos fluxos MBTAP es disputaven l'ample de banda de l'enllaç. Per tant, era necessari implementar un mecanismes que controlés la disputa (*fairness*) entre aquests fluxos.
- El protocol no estava concebut per a xarxes heterogènies, de forma que només pot aconseguir el rendiment màxim en xarxes completament cablejades.

Avui en dia, les comunicacions tenen lloc generalment sobre xarxes heterogènies de les quals es desconeix la seva composició. El fet que aquestes puguin estar formades per trams cablejats i altres trams sense fils implica un desconeixement, a priori, de les característiques exactes de la xarxa.

Concretament, això afecta de manera directa sobre els protocols de transport (nivell 4), ja que la concepció d'aquests protocols, majoritàriament, sol ser extrem a extrem. Això vol dir que es desconeixen les condicions i característiques dels trams que componen el camí que comunica els dispositius finals.

En el disseny de protocols de transport (excepte aquells creats especialment per a tal efecte) s'han assumit generalment un seguit de premisses que no són certes en molts entorns actuals amb xarxes heterogènies. L'exemple més clar es troba en el fet que la majoria de protocols de transport assumeixen que la pèrdua de paquets sempre serà deguda a la congestió de l'enllaç, sense tenir en compte altres possibilitats que són molt probables en entorns sense fils i que poden provocar pèrdues aleatòries (interferències o *fading*, entre d'altres). D'aquesta manera, el protocol reacciona a una "falsa congestió" reduint la tassa d'enviament i fent més ineficient la comunicació [5]. A dia d'avui, l'aparició i ús estès de les xarxes sense fils implica que algunes de les assumpcions que es van fer en el moment de dissenyar aquests protocols de comunicacions ara no es compleixin i el rendiment d'aquests es vegi afectat de manera negativa, requerint una adaptació dels protocols de transport a aquest tipus d'entorns.

Les xarxes LFN d'avui en dia poden contenir trams de l'enllaç que siguin formats per medis sense fils i, per tant, el protocol MBTAP no és una excepció a la problemàtica esmentada. La seva definició va ser enfocada a xarxes d'ample de banda i latència elevats, però sense proveir de cap mecanisme específic per a la seva adequació en entorns sense fils. És per això que és l'objectiu d'aquest treball analitzar les característiques concretes d'aquests entorns i la seva implicació a nivell de transport, recopilant possibles mecanismes d'altres protocols que puguin ser adaptats i incorporats al control de congestió del protocol MBTAP per millorar el seu rendiment, confeccionant

així el protocol Optimized MBTAP (OMBTAP). El focus principal d'aquest protocol és, doncs, incorporar un mecanisme que permeti detectar quan una pèrdua és generada per congestió o quan és generada per altres condicions aleatòries del canal, de manera que la reducció del Sending Rate (SR) només es doni en el primer cas, aconseguint així una transferència de dades més ràpida i eficient.

L'organització d'aquest treball queda de la següent manera. A l'apartat 2 es repassa l'Estat de l'Art dels protocols de transport més rellevants, així com els mecanismes de d'estimació de l'ample de banda que es poden trobar a la literatura. També s'expliquen els mecanismes de detecció de congestió en xarxes sense fils que existeixen. A l'apartat 3 es presenten els mecanismes implementats del protocol MBTAP des d'un punt de vista teòric, així com els resultats que aquest va obtenir en les simulacions. A continuació, també es defineixen quins mecanismes s'incorporen i quines modificacions es fan en el disseny de l'OMBTAP. A l'apartat 4 es mostra el procés d'implementació del protocol en el simulador Riverbed Modeler. A l'apartat 5 es mostren i s'analitzen els resultats de les proves realitzades, comparant el rendiment assolit pel protocol OMBTAP en front del seu predecessor MBTAP. Finalment, a l'apartat 6 s'extreuen les conclusions finals de treball i es presenten les línies de futur.

1.1 Motivació i objectius.

La motivació principal per realitzar aquest Treball Final de Màster és la de complementar la feina feta al grup de recerca GRITS durant un llarg període de temps, aportant un valor addicional al projecte MBTAP amb la implementació i simulació del protocol OMBTAP. L'altra element que ha motivat el desenvolupament del treball ha estat el d'aconseguir uns coneixements elevats i poder aprofundir en el funcionament d'un simulador de xarxa com Riverbed Modeler, ja que és un tipus de coneixement que no s'ha vist durant els estudis de grau i màster i que des d'un punt de vista personal resulta força interessant.

Els objectius del treball són:

- Aprendre i aprofundir en el funcionament del simulador Riverbed Modeler, tant a nivell d'usuari (configuració d'escenaris, simulacions i recol·lecció de resultats) com a nivell de desenvolupador (programació de processos dins del simulador).
- Ser capaç d'implementar l'especificació del protocol OMBTAP dins del simulador Riverbed Modeler.
- Analitzar el funcionament i el rendiment del protocol OMBTAP a partir de la realització de diverses simulacions i l'anàlisi dels seus resultats.

2 Estat de l'art

Aquest estat de l'art se centra en repassar (1) els diversos controls de congestió utilitzats en protocols de transport concebuts per xarxes cablejades i els (2) mecanismes d'estimació de l'ample de banda de l'enllaç, utilitzats com a estudi i referència pel disseny del protocol MBTAP; i (3) presentar els mecanismes de detecció de congestió utilitzats pels protocols de transport per a xarxes heterogènies (xarxes sense fils).

2.1 Controls de congestió per a xarxes cablejades

L'objectiu del control de congestió és ser capaç de detectar les situacions de congestió abans que el canal es col·lapsi, actuant en conseqüència. Existeixen dues causes principals que generen congestió a la xarxa:

- El receptor no és capaç de tractar tota la informació rebuda i descarta part dels paquets.
- El coll d'ampolla no és capaç d'assumir tot el tràfic generat de manera que en els nodes intermitjos es comencen a col·locar paquets en cua. Si la saturació supera la capacitat dels *buffers* dels nodes intermitjos, aquests comencen a descartar els paquets rebuts, generant pèrdues.

Per tal d'evitar la saturació de la xarxa, existeixen dues estratègies principals:

- **Preventives:** S'utilitzen tècniques com el control d'admissió, és a dir, es limita el nombre d'usuaris que poden transmetre així com la seva velocitat màxima d'enviament. Per poder aplicar aquesta estratègia és necessari fer un bon dimensionament de la xarxa i tenir-ne tot el control.
- **Reactives:** Aquestes tècniques pretenen resoldre la situació de congestió un cop aquesta s'ha detectat. Existeixen dues classes de detecció de congestió:
    - Realimentació directa: Els nodes intermitjos interpreten si hi ha congestió o risc que aquesta es produeixi a partir de la càrrega que pateixen els seus *buffers*, notificant-ho als extrems mitjançant el marcatge de paquets o l'enviament de missatges especials.
    - Realimentació indirecta: Els nodes finals detecten les situacions de congestió a partir de la pèrdua de paquets o a partir de mesurar el *jitter* (variació del retard) de la comunicació.

A continuació s'expliquen els diferents mecanismes de control de congestió que hi ha a la literatura.

2.1.1 Control de congestió TCP

TCP és el protocol de transport orientat a connexió més utilitzat avui en dia. Les seves principals característiques són la confiabilitat i la integritat de la informació, oferint un control de paquets perduts [6]. Per aquest motiu, és necessari que el receptor informi de la correcta transmissió dels paquets al receptor amb missatges de confirmació ACK (*acknowledgement)*. El *throughput* o velocitat d'enviament varia segons les capacitats

del receptor i de la xarxa. Originalment, aquesta velocitat es regulava amb els següents mecanismes:

- **Slow Start (SS):** Es duplica la mida de la finestra de congestió per cada ACK rebut. Aquesta creix exponencialment fins que es perd algun paquet o s'arriba a un límit màxim fixat (*ssthresh*). S'utilitza al principi de l'enviament per saber quina és la capacitat màxima d'enviament.
- **Congestion Avoidance (CA):** Un cop s'ha acabat el procés SS i s'ha arribat al límit *ssthresh*, el creixement de la finestra de congestió és lineal fins que es torna a detectar congestió, moment en que la finestra es redueix i es torna a començar amb el procés SS.

TCP defineix diferents paràmetres que indiquen el nombre de bytes que es poden transmetre:

- ***Congestion Window (Cwnd):*** És la finestra de congestió, que correspon al nombre de bytes que es poden transmetre en una finestra. El seu valor va augmentant o disminuint durant el transcurs de la transferència.
- ***Slow Start Threshhold (Ssthresh):*** És el límit superior de la finestra de congestió en el procés SS. Quan s'arriba a aquest valor, es passa a utilitzar el mecanisme CA en lloc de SS. A l'inici de la comunicació aquest valor és elevat i en el moment que es detecta congestió es redueix a la meitat.
- ***Advertised Window (Awnd):*** És la finestra de congestió anunciada pel receptor. Correspon al màxim nombre de bytes que aquest pot processar sense descartar paquets. No és una limitació de la xarxa ni dels nodes intermitjos, sinó que és exclusivament del receptor.
- ***Transfer Window*****:** És la finestra d'emissió actual, que correspon al mínim entre *cwnd* i *awnd*.

Els mètodes SS i CA permeten augmentar la finestra de congestió i enviar cada vegada més informació sense rebre un ACK. Malgrat això, quan l'emissor no rebi cap ACK durant un temps de guarda, es produirà un *timeout* i s'interpretarà que el receptor no ha rebut els paquets enviats. En aquest moment, el valor de la finestra de congestió es restaura a l'inicial (igual a 1) i *ssthresh* passa a ser la meitat del valor de *cwnd* actual. Un cop actualitzats aquests valors, es retransmeten els paquets que s'han interpretat com a perduts i torna a començar l'enviament des del procés SS.

Aquest forma de procedir fa que la connexió sempre sigui irregular, provocant una dent de serra que mai podrà aprofitar tot l'ample de banda de l'enllaç, tal i com es mostra a la Fig. 1.

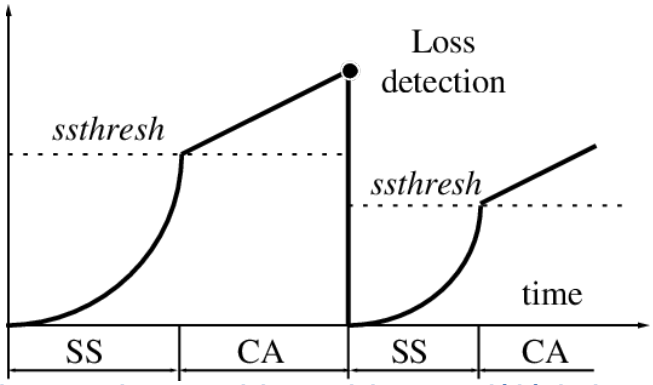

Fig. 1: Funcionament del control de congestió bàsic de TCP.

Per millorar el rendiment de TCP, van sorgir dos mecanismes més:

- **Fast Retransmit:** Originalment, si el receptor rebia un paquet desordenat, es generava un ACK que podia ser interpretat de dues maneres: o bé el paquet encara no ha arribat al receptor i es retransmetrà quan expiri el *timeout*, o bé el paquet ja s'ha perdut i s'ha de tornar a enviar. Fast Retransmit consisteix en enviar dos ACK duplicats per indicar que el paquet ha estat perdut i que, per tant, no cal esperar a que expiri el *timeout* per tornar a enviar-lo.
- **Fast Recovery:** Aquest mecanisme evita restaurar el valor de *cwnd* a 1 i començar de nou amb el procés SS. Quan es reben tres ACKs del mateix segment s'interpreta que aquest s'ha perdut i s'aplica el Fast Recovery, reduint *ssthresh* i canviant el valor de *cwnd* a la meitat de *ssthresh*. A més, en lloc de començar de nou amb SS, es continua la transmissió directament amb CA.

Una de les parts crítiques del protocol TCP és el de donar un valor al *timeout* de la connexió. Aquest valor es calcula a partir del RTT de la transmissió, entès com el temps que passa des de que s'envia un segment fins que se'n rep l'ACK corresponent. A partir d'aquest valor s'estima quin ha de ser el *timeout* de la connexió, però en moltes ocasions de congestió aquests valors no reflecteixen la realitat.

Segons la utilització d'aquets 4 mecanismes definits es conformen les 5 modificacions de TCP explicades a continuació [6].

2.1.1.1 TCP Tahoe

És la versió de TCP que a més d'utilitzar Slow Start i Congestion Avoidance, incorpora el mecanisme Fast Retransmit. A més, també aporta una nova manera de calcular el RTT que millora els resultats de l'original. El principal inconvenient és que després de la pèrdua d'un paquet es torna a començar amb el procés SS.

2.1.1.2 TCP Reno

És una de les versions més utilitzades actualment. Aquesta versió incorpora el mecanisme de Fast Recovery, de manera que no es penalitza tant el rendiment de la connexió. A més també implementa dos mecanismes addicionals menors anomenats Header Prediction i Delayed Acknowledgements [7].

2.1.1.3 TCP New Reno

Pretén resoldre les mancances dels mecanismes SS i CA. A l'inici de la comunicació, el protocol intenta buscar el valor òptim de *ssthresh* amb la tècnica *Packet pair*. Aquesta tècnica consisteix en enviar dos paquets de forma consecutiva i mesurar la diferència de recepció dels respectius ACKs, amb la qual es pot estimar el nivell de congestió de l'enllaç [8].

2.1.1.4 TCP SACK

Aquesta versió implementa tots els mecanismes anteriors i, a més, afegeix mecanismes per adaptar la connexió a xarxes de tràfic elevat i pèrdues constants de paquets. Quan el receptor detecta que se n'ha perdut un, envia un ACK duplicat (DUACK) per indicar que la resta de paquets s'han rebut correctament i un triple ACK per indicar quin és el segment que s'ha perdut. D'aquesta manera el transmissor sap quin és el paquet o els paquets que ha de reenviar exactament. També incorpora l'ús de missatge SACK [1].

2.1.1.5 TCP Vegas

La última modificació del TCP base consisteix en intentar que l'emissor s'anticipi a la situació de congestió. El protocol va comprovant a cada instant la diferència entre la velocitat de transferència potencial i la real (la que s'ha arribat a transmetre correctament), de manera que es pot assolir un transmissió de dades amb una velocitat força constant. El problema d'aquesta versió és que pot afectar de manera dràstica el rendiment d'altres fluxos de la xarxa *best-effort*, que no és l'objectiu de TCP (es vol que sigui *fair* i *friendly*) [9].

2.1.2 Controls de congestió TCP per a comunicacions de llarga distància

Actualment també existeixen diverses modificacions de TCP que estan orientades a transmissió de dades a llarga distància.

2.1.2.1 Parallel TCP Reno (P-TCP)

Aquesta versió de TCP Reno pretén augmentar el *throughput* aconseguit a partir de transmetre diversos *streams* (fluxos) de dades en paral·lel. El número de *streams* màxim que es poden transmetre en paral·lel depèn de diversos factors com la xarxa, el tipus d'informació, etc. S'aproxima que normalment es poden transmetre uns 16 *streams* en paral·lel. El principal inconvenient que presenta és que no permet la diferenciació ni la priorització del tràfic de cada *stream* [2].

2.1.2.2 Scalable TCP (S-TCP)

Aquesta modificació també parteix de TCP Reno. En aquest cas, modifica el control de congestió, utilitzant un creixement exponencial en lloc de l'*Additive Increase* del protocol original. A més, utilitza un decrement multiplicatiu amb un factor de 0,125, aconseguint uns nivells de *throughput* força més elevats [10].

2.1.2.3 FAST TCP

És una modificació del TCP Vegas. Està concebut per a xarxes amb latències grans, de manera que intenta no penalitzar la finestra de congestió quan es detecten retards a la xarxa, cosa que sí fan altres protocols com TCP Reno que interpreten aquests retards com a congestió i redueixen el valor de *cwnd*. Tot i això, no arriba a assolir uns nivells de *throughput* tant bons com HS-TCP o S-TCP [11].

2.1.2.4 High Speed TCP (HS-TCP)

Aquest protocol es basa en el funcionament de TCP Reno fins que el valor de *cwnd* supera els 38 paquets. A partir d'aquest moment, el creixement de la finestra de congestió ve predefinit per una taula del propi protocol, i la reducció de la *cwnd* després d'una pèrdua és menor (menys de la meitat). El principal inconvenient d'aquest protocol és que el control de congestió es converteix un una eina poc dinàmica i inflexible [12].

2.1.2.5 HSTCP-LP

Aquesta modificació es basa en HS-TCP i TCP-LP. Aquest protocol és molt poc intrusiu, ja que només utilitza l'ample de banda disponible de l'enllaç. Per tal de detectar la congestió, es basa en calcular les variacions del RTT. El principal inconvenient d'aquest protocol és que no aconsegueix assolir nivells de *throughput* elevats quan l'enllaç es troba congestionat [13].

2.1.2.6 H-TCP

Aquest protocol es basa en HS-TCP. La principal diferència és que en lloc d'utilitzar les taules predefinides, utilitza un algoritme AIMD heterogeni que permet més flexibilitat a l'hora d'adaptar-se a la capacitat del canal, basant el seu comportament en les pèrdues produïdes i el RTT. El seu control de congestió és força complex pel que fa a la implementació dels càlculs d'aquest [14].

2.1.2.7 Binary Increase Control TCP (BIC-TCP)

Aquesta versió de TCP combina un creixement additiu de *cwnd* quan la finestra de congestió és mitjana o alta i un creixement binari quan la finestra és petita. El protocol inicia la transmissió amb un Additive Increase, augmentant la finestra de manera més lenta que amb el Slow Start. A continuació, s'utilitza el mecanisme Binary Search, que actualitza el valor de *cwnd* al punt mig entre W_{max} (valor de *cwnd* on s'han produït les darreres pèrdues) i W_{min} (últim valor de *cwnd* on no s'han perdut pèrdues). Finalment, s'aplica el mecanisme Max Probing que provoca un creixement exponencial de la finestra. A més, quan es detecten pèrdues, la finestra de congestió es redueix en un factor β. El principal inconvenient d'aquest protocol és que es tarda molt en arribar a uns nivells de *throughput* elevats, tot i que quan s'hi arriba, s'aprofita al màxim l'ample de banda de l'enllaç i té una gran estabilitat [15].

2.1.2.8 TCP Westwood+

Aquest protocol utilitza el mètode d'estimació d'ample de banda de TCP Westwood a partir de la recepció dels missatges ACK. Aquesta informació s'utilitza per calcular *cwnd* i el límit on cal aplicar Slow Start. A més, corregeix el problema que presenta TCP Westwood amb els ACKs comprimits. De totes maneres, només utilitza els mecanismes Slow Start i Conegestion Avoidance, de manera que manté algunes de les mancances del TCP original [16].

2.1.2.9 CUBIC

CUBIC és una millora de BIC-TCP que destaca per la seva gran estabilitat en transferències a alta velocitat. Gràcies a aquest fet, s'ha convertit en la versió de TCP utilitzada per defecte en els sistemes Linux des de la versió 2.6.18 [15].

Aquest protocol substitueix la Binary Search de BIC-TCP per una funció de creixement cúbica, de manera que quan el valor de *cwnd* és molt més baix que W_{max}, l'increment és molt accentuat. En canvi, quan el valor de *cwnd* és proper al de W_{max}, es van fent increments petits, intentant superar el valor de W_{max} de mica en mica (veure Fig. 2).

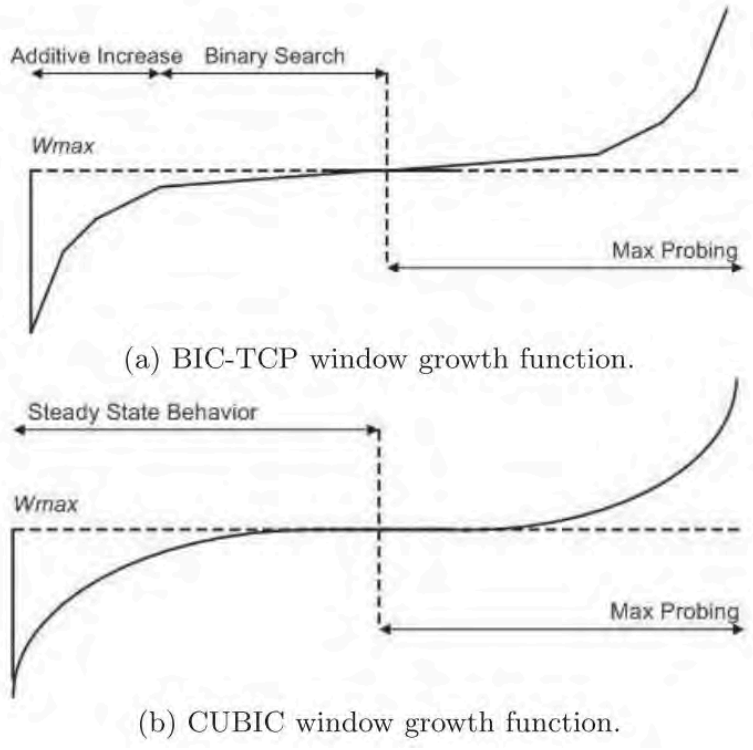

Fig. 2: Funció de creixement de la finestra de congestió de BIC-TCP (a) i de CUBIC (b).

Pel que fa a la reducció de la finestra de congestió després de pèrdues, manté el Multiplicative Decrease de BIC-TCP amb un factor $\beta = 0{,}2$. Addicionalment, incorpora un mecanisme de Fast Recovey que BIC-TCP no contempla. El principal inconvenient de CUBIC segueix sent el mateix que el del seu predecessor, i és el temps de convergència que es necessita per arribar a l'enviament estable d'alta velocitat.

2.1.2.10 Cascaded TCP

Cascaded TCP intenta aconseguir majors nivells de *throughput* a partir de dividir una connexió TCP lògica en diverses connexions TCP independents utilitzant *relay agents* de nivell 4, tal i com es mostra a la Fig. 3. D'aquesta manera, les latències de les connexions són menors, els controls de congestió de cada connexió actuen independentment i el *feedback* es rep de manera més ràpida, de manera que s'aconsegueixen majors nivells d'utilització de l'enllaç. L'inconvenient d'aquest tipus d'enfocament trenca amb el principi *end-to-end* del propi TCP [17].

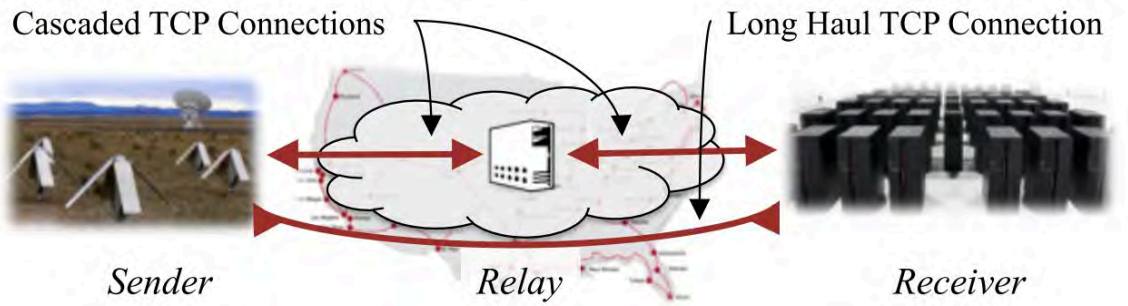

Fig. 3: Esquema d'una connexió Cascaded TCP amb l'ús d'un *relay agent* de nivell 4.

2.1.3 Comparativa

Una comparativa de les diverses versions de TCP disponibles al *kernel* de Linux mostra que S-TCP, BIC-TCP i CUBIC són els protocols que obtenen nivells de *throughput* més elevats (veure Fig. 4 i Fig. 5), tot i que això provoca que els seus nivells de *friendliness* respecte altres versions de TCP siguin inferiors.

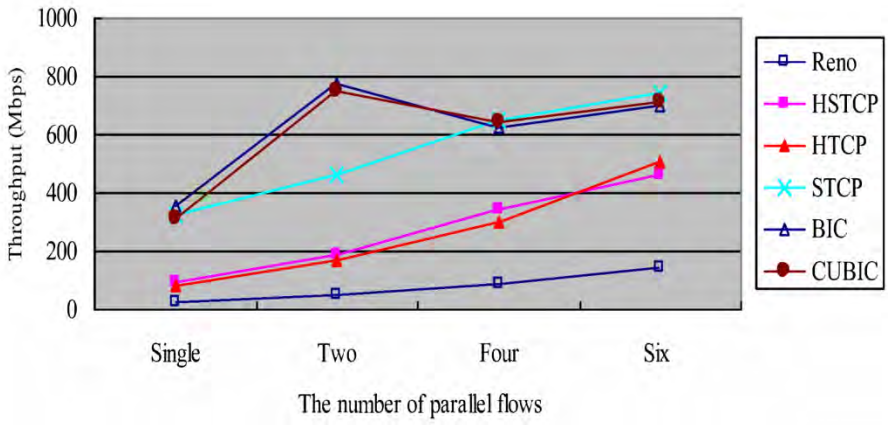

Fig. 4: *Throughput* agregat de diverses versions de TCP en funció del nombre de fluxos concurrents.

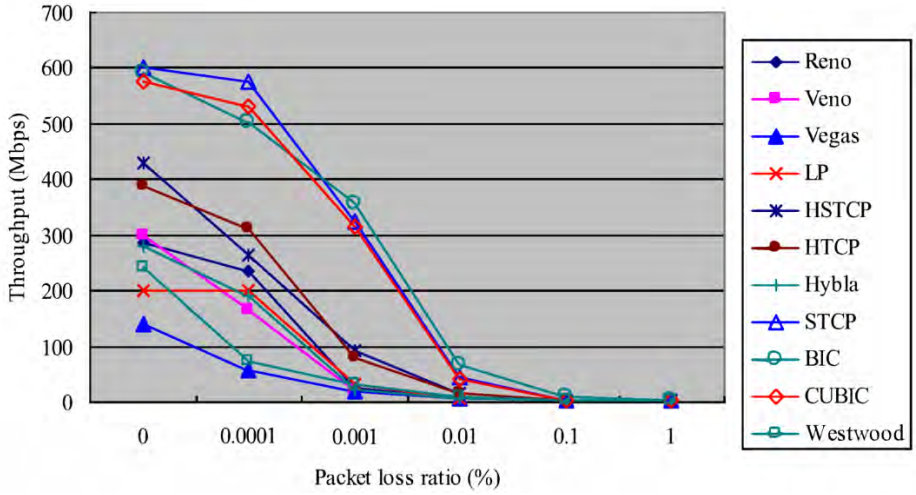

Fig. 5: *Throughput* assolit per diverses versions de TCP en funció del percentatge de pèrdues.

2.1.4 Altres protocol per a llarga distància

2.1.4.1 Stream Control Transmission Protocol (SCTP)

SCTP [18][19] aporta una sèrie de mecanismes i funcionalitats addicionals de les que ofereix TCP, augmentant de forma notable el *throughput* assolit i aconseguint un comportament més òptim. En primer lloc, utilitza un *4-way handshake* en lloc del *3-way handshake* de TCP, oferint així una protecció en front d'atacs de denegació de servei (DoS).

Aquest protocol utilitza les mateixes variables de TCP: *cwnd, awnd, sshthresh*. En el cas de SCTP, però, sempre que el valor de *cwnd* sigui inferior a *sshthresh* s'utilitza el mecanisme Slow Start. Aquest mecanisme està modificat de manera que només s'incrementa la finestra de congestió si s'ha rebut l'ACK correctament i en la iteració anterior s'ha utilitzat la capacitat de tota la finestra.

També s'introdueix un canvi en el mecanisme Congestion Avoidance. Es defineix una nova variable anomenada *pba* (*partial bytes acked*). Al començament, el valor de *pba* és 0 i s'incrementa a mesura que es van confirmant els bytes enviats mitjançant els ACKs. Durant la transmissió, el valor de la finestra de congestió només s'incrementa si *pba* és igual o superior a *cwnd* i en l'anterior iteració s'ha utilitzat la capacitat de tota la finestra.

Per detectar els paquets perduts també utilitza un *timeout* i un mecanisme Fast Retransmit similar al de TCP SACK. La diferència respecte aquest recau en el fet que en lloc de transmetre 3 ACKs per sol·licitar la retransmissió, se n'envien 4.

Posteriorment, va sorgir una millora del protocol anomenada New-Reno SCTP [20]. Aquest protocol incorpora el mecanisme Fast Recovey. A més, es defineix una política per no incrementar el valor de *cwnd* quan aquest mecanisme s'està utilitzant.

Finalment, també han sorgit diversos mecanisme per millorar el comportament de SCTP en entorns de xarxes sense fils [21][22].

2.1.4.2 Protocols basats en UDP

Existeixen un conjunt de protocols basats en UDP que tenen com a objectiu proporcionar un control de congestió eficaç i altres mecanismes de confiabilitat sense ser protocols orientats a connexió per definició.

Un dels més destacats és UDP-based Data Transfer (UDT) [23], que presenta una millor utilització de l'ample de banda i un bon rendiment en transferències d'arxius de gran mida en xarxes de llarga distància. Tot i això, també se n'han detectat algunes deficiències.

Un altre protocol sota aquesta classificació és High-performance and Flexible Protocol (HpFP) [24]. Aquest protocol envia missatges ACK de forma asíncrona als paquets rebuts, amb una periodicitat de 200ms. D'aquesta manera se soluciona el problema del retard a les LFN. L'enviament de segments de dades també es fa de forma independent a la recepció dels ACKs. També compte amb un sofisticat i complet control de congestió, tot i que els autors prefereixen no desvelar el seu algoritme. El que sí que es diu és que el protocol intenta adaptar-se a l'ample de banda disponible de l'enllaç, de manera que la intenció del protocol és ser *friendly*.

2.2 Mecanismes d'estimació de l'ample de banda de l'enllaç

Existeixen dues tècniques principals d'estimació d'ample de banda: la dispersió de *packet pairs* i la dispersió de *packet trains* [25].

La dispersió de *packet pair* consisteix en enviar un parell de paquets de mida L [bits] de forma consecutiva. Pressuposant que no hi ha tràfic en tot el canal, quan aquest parell de paquets travessa el coll d'ampolla de l'enllaç amb una capacitat C [bps], pateix una dispersió δ [s]. Aquesta dispersió es pot calcular amb la divisió δ=L/C. D'aquesta manera, si es pot mesurar la dispersió que pateix el *packet pair*, és possible estimar la capacitat de l'enllaç amb la fórmula C=L/δ. A la Fig. 6 es mostra aquest fet de forma il·lustrativa.

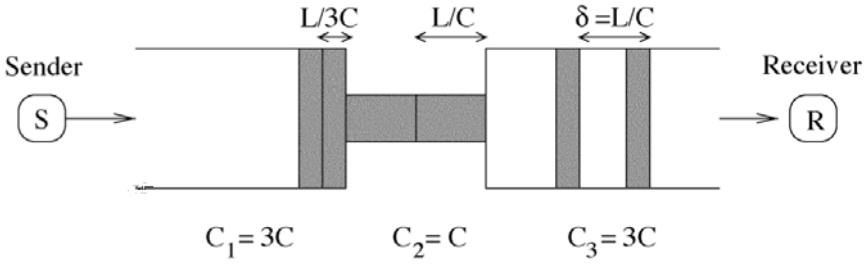

Fig. 6: Dispersió d'un *packet pair*.

Sense la presència de tràfic creuat, totes les estimacions d'ample de banda calculades obtindran el mateix valor, ja que la dispersió patida sempre serà la mateixa independentment de la mida dels paquets. Això sí, és necessari que els dos paquets del mateix parell siguin de la mateixa mida, ja que en cas contrari patiran *delays* diferents.

En canvi, davant la presència de tràfic creuat, és possible que un dels dos paquets del *packet pair* (o els dos) siguin col·locats en cua, i així provocar un canvi en la seva

dispersió, tal i com es postra a la Fig. 7. Si el primer paquet es posa en cua més temps que el segon, la dispersió queda comprimida. En canvi, quan és el segon paquet el que queda més temps en cua, la dispersió s'expandeix.

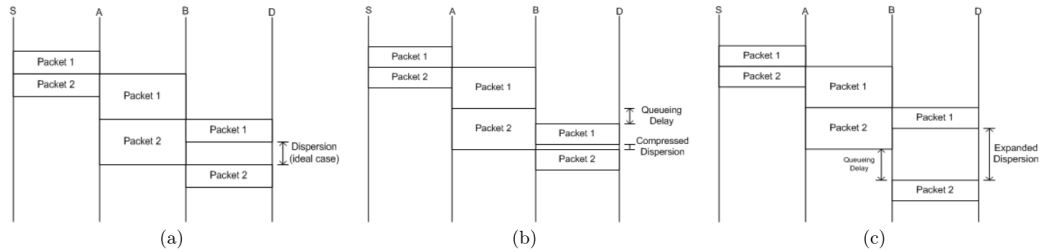

Fig. 7: Compressió i expansió de la dispersió

Per aquest motiu, és necessari enviar diversos *packet pairs* per poder calcular la capacitat de l'enllaç. Si es fa una distribució amb totes les mostres de capacitat calculades a partir de la dispersió, s'observa que s'obté una distribució multimodal, és a dir, amb més d'una moda. Les modes locals que estimen un valor de capacitat superior s'anomenen Post-Narrow Capacity Modes (PNCM) i les modes locals que estimen una capacitat inferior a la real són anomenades Sub-Capacity Dispersion Range (SCDR). A la Fig. 8 s'observa que si l'enllaç està poc congestionat, la moda més freqüent coincideix amb la capacitat de l'enllaç. Aquesta moda local s'anomena Capacity Mode (CM). En canvi, si l'enllaç està considerablement congestionat, les modes SCDR seran més freqüents que la CM.

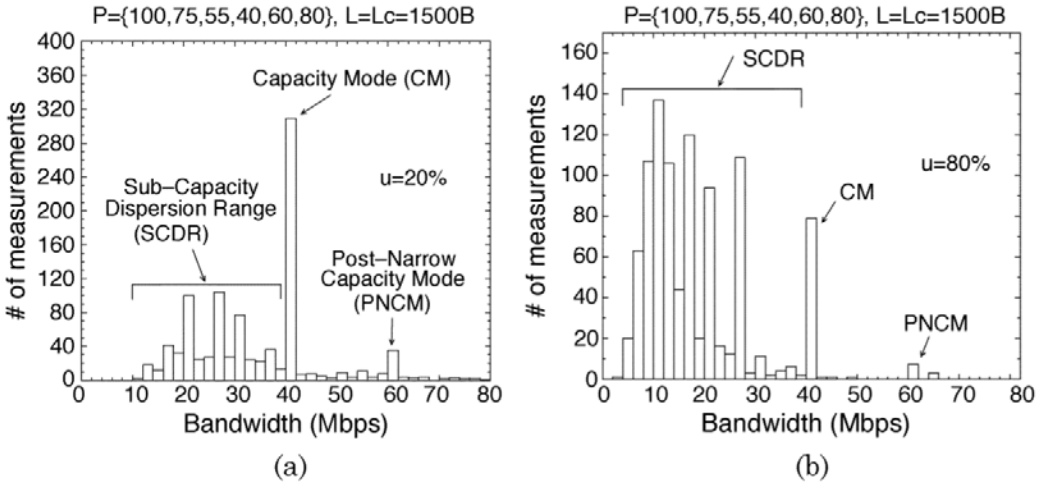

Fig. 8: Distribució multimodal de la capacitat estimada amb un volum de tràfic creuat igual al 20% (a) i al 80% (b) de la capacitat de l'enllaç.

Pel que fa a la mida L dels paquets, es pot veure a la Fig. 9 que si L és massa petit apareix una PNCM com a moda més freqüent. En canvi, si L és massa gran, hi haurà més probabilitat que els paquets es vegin afectats pel tràfic creuat (siguin posats en cua) i, per tant, apareixeran SCDR amb més força.

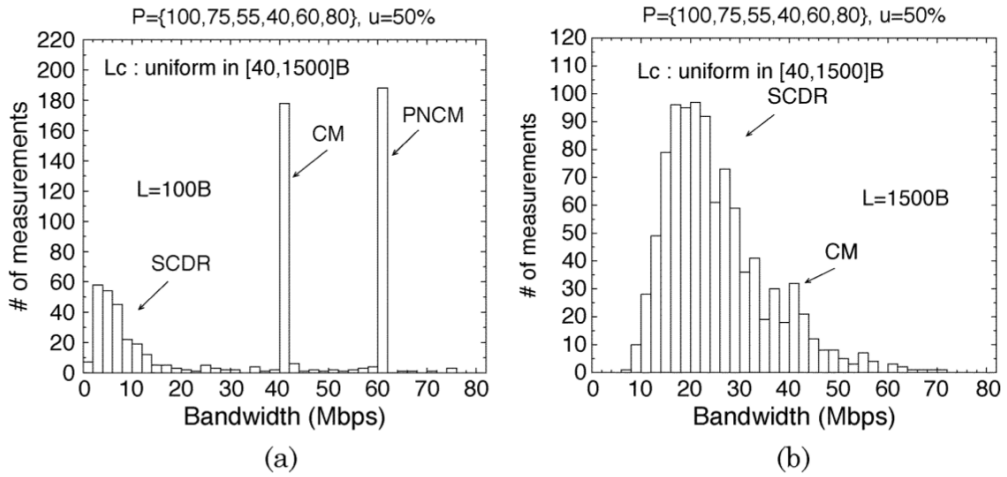

Fig. 9: Distribució multimodal de la capacitat estimada amb un volum de tràfic creuat del 50% i amb mida de paquets de 100 bytes (a) i 1500 bytes (b).

També es demostra que si s'utilitzen *packet pairs* amb mida L variable, les SCDR s'eixamplen i perden força, de manera que la CM guanya protagonisme (veure Fig. 10). Una bona praxis és utilitzar un rang de mida de paquets entre L_{min} i L_{max}, on aquests no siguin ni massa grans ni massa petits respecte la mida del tràfic creuat.

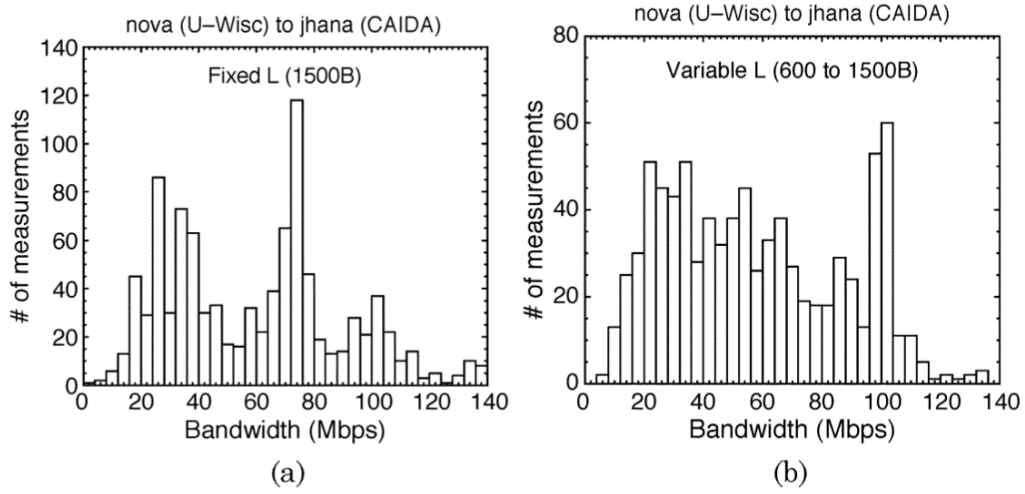

Fig. 10: Distribució multimodal de la capacitat estimada amb una mida de paquets fixe (a) i variable (b).

Pel que fa a la dispersió *packet train*, aquesta consisteix en enviar una ràfega de N paquets de forma consecutiva. Per extensió de la dispersió del *packet pair*, és senzill interpretar que la capacitat del canal C(N) sense presència de tràfic creuat es pot calcular a partir de la dispersió del *packet train* Δ(N):

$$C(N) = \frac{(N-1)L}{\Delta(N)}$$

A la Fig. 11 s'observa que, igual que passa amb els *packet pairs*, quan es fan diverses mesures de la capacitat estimada amb la presència de tràfic creuat s'obté una distribució multimodal. Tot i això, com més gran sigui la longitud del tren N, més probabilitats hi ha que els paquets es vegin afectats pel tràfic creuat i siguin posats en cua. Quan això

succeeix, la CM i les PNCM van perdent força fins a desaparèixer, mentre que les SCDR agafen força. Si N és suficientment gran, la distribució dels valors C(N) perd la variabilitat i és independent de N, quedant una sola moda (distribució monomodal). El problema recau en que aquesta moda no és la capacitat real de l'enllaç, sinó el valor d'una mètrica anomenada Average Dispersion Rate (ADR), que està relacionada amb l'esperança de la dispersió temporal del *packet-train*. Està demostrat matemàticament que aquesta mètrica és un llindar inferior de la capacitat real del canal i un llindar superior de l'ample de banda disponible (residual) de l'enllaç. És a dir, que la capacitat màxima serà igual o superior a l'ADR i l'ample de banda lliure serà igual o inferior a l'ADR [25].

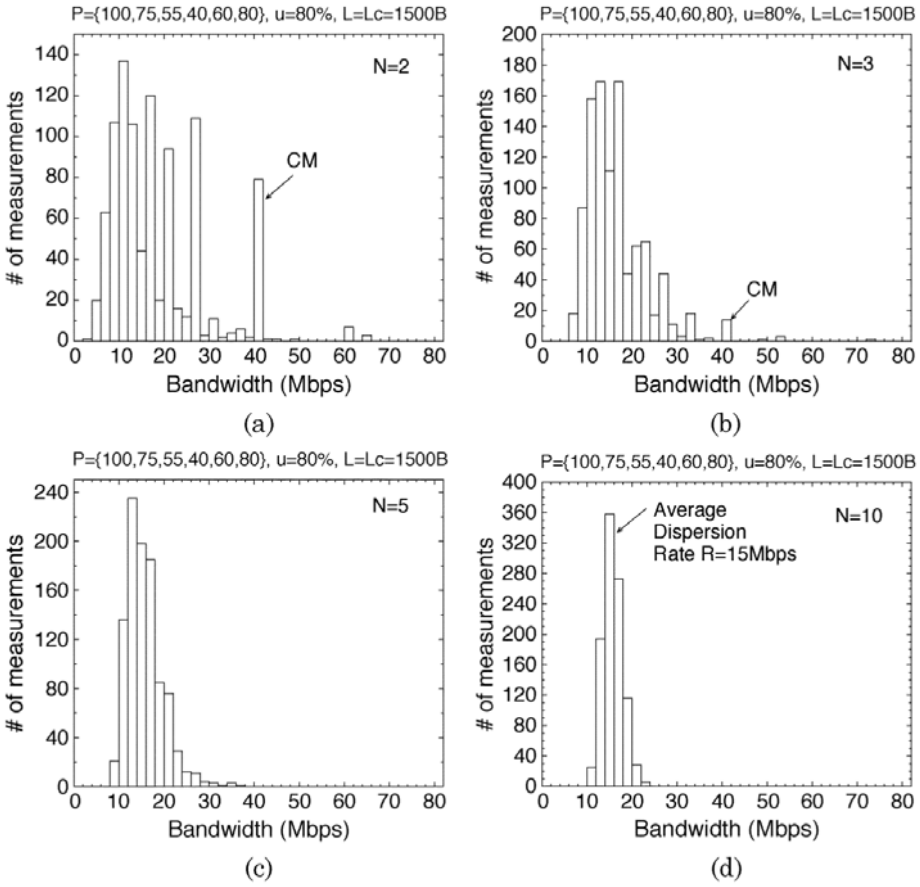

Fig. 11: Distribució multimodal de la capacitat estimada amb una longitud de ràfega de 2 paquets (a), 3 paquets (b), 5 paquets (c) i 10 paquets (d). En aquesta última s'observa com la dispersió es converteix en monomodal i apareix l'ADR.

Un cop identificades les dues tècniques de dispersió de paquets i per a què serveixen, cal analitzar els mecanismes d'estimació d'ample de banda que existeixen a la literatura. Abans de res, però, cal diferenciar entre dos tipus de mecanismes [26][27][28]: els que estimen l'ample de banda disponible de l'enllaç i els que estimen la capacitat total del canal. En el primer cas, es troben nombrosos mecanismes com *cprobe*[29], *TOPP*[30], *Delphi*[31], *pathload* [32], *abing* [33], *IGI* [34], *Spruce* [35], *pathchirp* [36], *probegab* [37] i *Assolo* [38]. Aquests mètodes, però, no apliquen en el cas que s'està treballant, ja que l'objectiu del protocol MBTAP és estimar la capacitat tota de l'enllaç. Per aquest motiu, els mecanismes esmentats no s'analitzen en profunditat. Pel que fa a l'estimació de la capacitat total, existeixen 5 mètodes diferents a la literatura, que es descriuen a continuació.

2.2.1 Bprove

El primer mecanisme és *bprove* [29], que utilitza la dispersió d'un *packet pair* per estimar la capacitat de l'enllaç. Aquest mecanisme utilitza un processat de les mostres calculades on aquestes es filtren a partir d'interseccions i unions amb l'objectiu de descartar les que s'hagin pogut veure afectades pel tràfic creuat. A més, *bprobe* utilitza mides de paquet variables per millorar la precisió del càlcul quan el tràfic creuat és d'una mida de paquet fixa (i.e. 40, 576, o 1500 bytes). *Bprobe* només necessita tenir accés a l'emissor, ja que els paquets enviats són missatges "ICMP Echo" que el *peer* remot pot respondre amb missatges "ICMP Reply", els quals s'utilitzen per fer els càlculs.

2.2.2 Nettimer

Un altre mètode és *Nettimer* [39][40], que també té com a base la mesura de la dispersió dels *packet pairs*. *Nettimer* utilitza una tècnica estadística sofisticada, la qual anomenen "Kernel Density Estimation" per processar les mostres de dispersió preses. Aquesta tècnica calcula la moda dominant de la distribució de dispersions mesurades. A causa que els càlculs es fan en la recepció de paquets, aquest mètode necessita tenir accés tant en l'emissor com en el receptor.

2.2.3 Sprobe

El següent mètode que va sorgir és *sprobe* [41], el qual es caracteritza per la seva rapidesa i els pocs recursos que consumeix. *Sprobe* consisteix en enviar *packet pairs* (paquets TCP SYN) que són contestats amb paquets TCP RST. De la mateixa manera que passa amb *bprove*, aquest mètode només necessita accés a l'emissor, ja que les mesures es prenen amb les respostes TCP RST. A causa de la simplicitat d'aquest mètode, la seva precisió en les estimacions és pitjor, especialment davant la presència de tràfic creuat.

2.2.4 Pathrate

A continuació va aparèixer *pathrate* [25], el primer en assolir uns resultats força precisos. Aquest mètode és més complex i consisteix en quatre fases:

- Fase 1: s'envien *packet trains* de longitud N incremental fins que s'arriba a un valor N_{max} (màxima longitud del tren sense que es generin pèrdues).
- Fase 2: s'envien 60 *packet trains* de longitud N incremental, des de N = 2 fins a N = min(10,N_{max}). Amb els valors de la capacitat calculada C(N)[4], s'obté una resolució d'ample de banda W per poder agrupar les mostres en les distribucions computades en les fases posteriors. Si els valors C(N) són tots molt similars entre ells, s'interpreta que no hi ha presència de tràfic creuat i es dóna com a vàlida la mitjana aritmètica d'aquests valors, finalitzant aquí el procés d'estimació.

[4] Recordatori de la fórmula: C(N) = (N-1)*L / Δ(N)

- Fase 3: s'envien 1000 *packet pairs* de mida variable entre L_{min} = 550 bytes i L_{max} = 1500 bytes. Es computa la distribució multimodal i se seleccionen les modes locals.
- Fase 4: s'envien 500 *packet trains* de longitud N_{max} i mida de paquets L_{max}. Es computa la distribució monomodal i s'escull la moda com el valor de l'ADR. Totes les modes locals calculades a la fase 3 que tinguin un valor inferior a l'ADR es descarten. Per cada una de les modes locals restants, es calcula la seva figura de mèrit (té a veure amb la semblança i quantitat dels valors C dins de cada moda local) i s'escull la moda amb més mèrit com a valor de la capacitat màxima de l'enllaç.

Com es pot observar, aquest mètode és força llarg i lent (tarda diversos minuts en obtenir el resultat final), amb l'afegit que la gran quantitat de càlculs que s'hi fan requereixen un cost computacional elevat. A causa que els càlculs es realitzen en recepció, es necessita l'accés tant en l'emissor com en el receptor. A la Fig. 12 es mostra un dibuix esquemàtic sobre el funcionament d'aquest mètode.

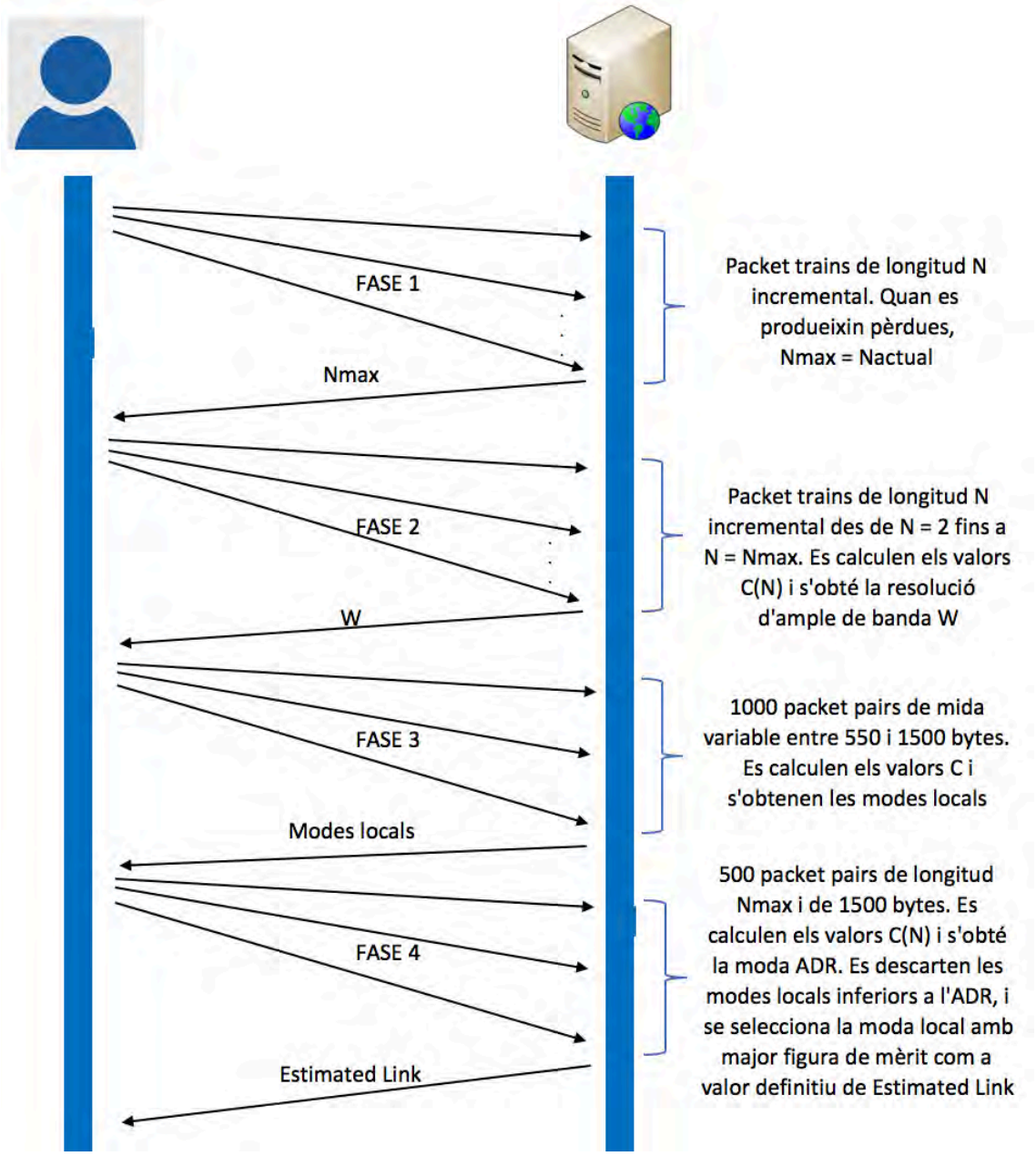

Fig. 12: Diagrama d'intercanvi de paquets del mètode *pathrate*.

2.2.5 CapProbe

Un altre mètode que presenta uns resultats molt bons és *CapProbe* [42]. En aquest cas, en lloc de computar distribucions i extreure'n modes, la filosofia recau en calcular el mínim *One Way Delay* (OWD) dels *packet pairs*. Per donar una mostra com a vàlida, s'ha de complir la següent condició:

$$\mathrm{OWD}_{min}\{\text{paquet } 1 + \text{paquet } 2\} = \mathrm{OWD}_{min}\{\text{paquet } 1\} + \mathrm{OWD}_{min}\{\text{paquet } 2\}.$$

Després de complir aquesta condició, els tres valors mínims obtinguts no poden canviar durant les següents N mostres preses per tal d'assegurar que aquests mínims són valors estables i la mostra és vàlida. D'aquesta forma, és molt probable que la mostra donada

com a vàlida no hagi estat posada en cua en els nodes intermitjos i que, per tant, sigui el cas ideal en que el temps de dispersió entre els dos paquets es pugui utilitzar directament per calcular la capacitat de l'enllaç. Després de realitzar diverses proves, els autors asseguren que un màxim de 100 mostres són suficients per aconseguir una mostra vàlida que permeti calcular la capacitat màxima del canal, així com un valor de N = 40 mostres també és suficient per dona validesa als mínims obtinguts. D'aquesta manera, el mètode *CapProbe* treballa amb dues mides diferents de paquets (p_1 = 700 bytes i p_2 = 900 bytes) i consisteix en dues fases:

- Fase 1: s'envien *packet pairs* de mida p_1 fins que es troba una mostra vàlida. En cas de no trobar-ne cap després d'haver enviat 100 *packet pairs*, hi ha dues opcions:
    - Si la capacitat màxima i la capacitat mínima calculades durant les 100 mostres són molt dispars (rati superior a 50), s'interpreta que no existeix suficient resolució temporal i que, per tant, s'ha d'augmentar la mida dels paquets. En aquest cas s'augmenten els valors de p_1 i p_2 un 20%, amb un límit màxim de 1500 bytes (ja que aquesta és la MTU dels paquets perquè es fragmentin), i es torna a repetir la prova.
    - Si la capacitat màxima i mínima calculades no són molt dispars (rati igual o inferior a 50), significa que totes les mostres han sigut posades en cua, de manera que cal reduir la mida dels paquets per reduir la seva probabilitat de ser col·locats en cua. En aquest cas es redueixen els valors de p_1 i p_2 un 20% i es torna a repetir la prova.
- Fase 2: s'envien *packet pairs* de mida p_2 fins que es troba una mostra vàlida. En cas de no trobar-ne cap després d'haver enviat 100 *packet pairs*, s'augmenten o es redueixen els valors de p_1 i p_2 seguint el mateix criteri que abans, i es torna a repetir la prova des de la fase 1. En el cas que sí es trobi una mostra vàlida, també hi ha dues opcions:
    - Si les capacitats calculades a la fase 1 i a la fase 2 són més d'un 5% dispars, es torna a repetir la prova des de la fase 1.
    - Si les dues capacitats són menys suficientment similar (igual o inferior al 5%), s'interpreta que la mitjana aritmètica entre els dos valors és la capacitat de l'enllaç.

A la Fig. 13 es mostra l'esquema de funcionament del mètode *Capprobe*.

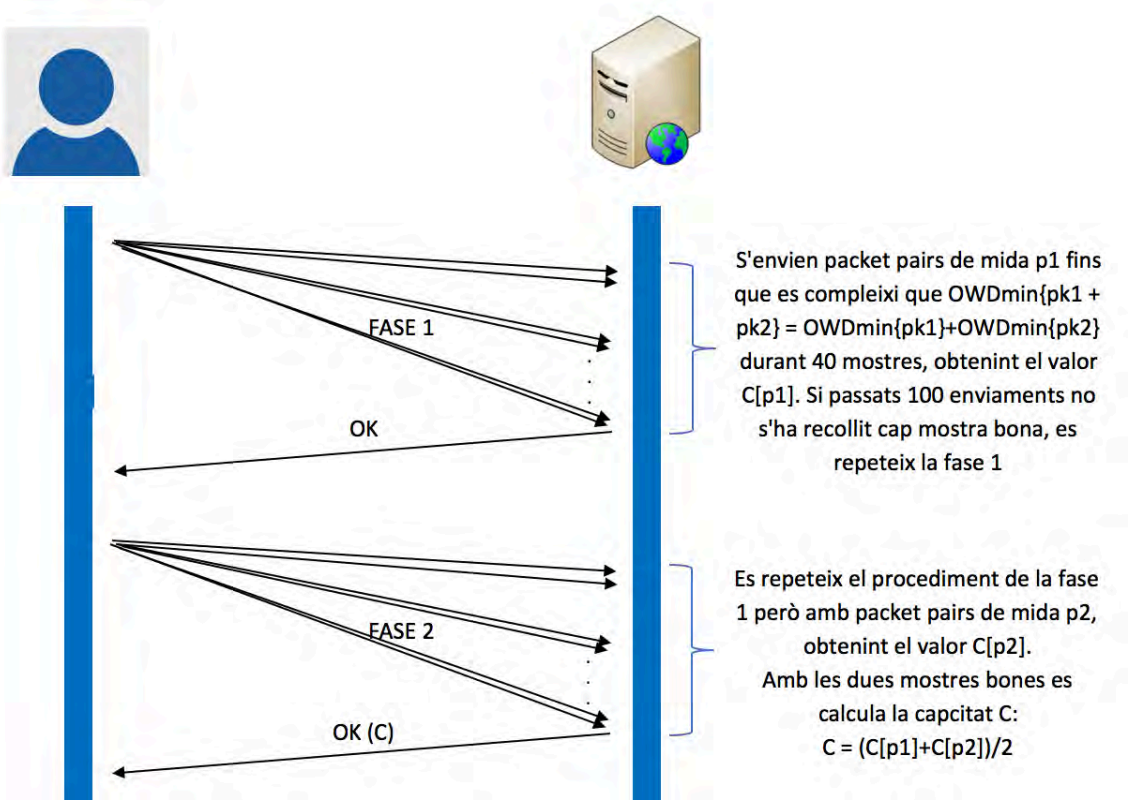

Fig. 13: Diagrama d'intercanvi de paquets del mètode *CapProbe*.

Aquest mètode requereix un cost computacional menor i el seu temps de càlcul també és inferior, de manera que és força més eficient. Com que en aquest cas també es fan les mesures en recepció, es necessita tenir accés a l'emissor i al receptor. A través de proves realitzades en diversos escenaris i sota condicions de tràfic creuat diferents [42], es demostra que *CapProbe* i *pathrate* tenen una capacitat d'estimació i precisió molt similars, però que en canvi el temps necessari per realitzar aquesta estimació és notablement menor amb *CapProbe* (veure Fig. 14). Posteriorment, van sorgir dues variacions d'aquest mètode: *TCP Probe* [43], que és un adaptació per integrar el mecanisme dins del protocol TCP; i *AsymProbe* [44], que realitza el procés en els dos sentits de la comunicació per poder calcular la capacitat màxima en enllaços asimètrics.

	UCLA-2 131.179.33.171		UCLA-3 131.179.136.151		UA 130.160.47.35		NTNU 140.122.77.6	
	time	C	time	C	time	C	time	C
CapProbe	0'03	5.5	0'01	96	0'02	98	0'07	97
	0'03	5.6	0'01	97	0'04	79	0'07	97
	0'03	5.5	0'02	97	0'17	83	0'22	97
	0'07	5.6	0'01	98	0'09	98	0'04	99
	0'03	5.6	0'02	99	0'09	95	0'04	96
pathrate	6'10	5.6	0'16	98	5'19	86	0'29	97
	6'14	5.4	0'16	98	5'20	88	0'25	97
	6'5	5.7	0'16	98	5'18	133	0'25	97
	6'14	6.8	0'16	98	5'19	88	0'26	97
	6'20	5.8	0'16	98	5'19	132	0'25	97

Fig. 14: Comparativa de la capacitat estimada (en Mbps) i el temps de duració (en minuts) entre els mètodes *CapProbe* i *pathrate* amb presència de tràfic creuat. Les capacitats reals dels enllaços són de 100 Mbps excepte l'enllaç UCLA-2 que és de 5,5 Mbps.

2.2.6 Compound probe

Finalment, també va aparèixer un mètode anomenat *compound probe* [45]. Aquest és un mètode iteratiu que va calculant la capacitat màxima en cada salt de l'enllaç amb l'enviament de *packet pairs*. Un cop obtinguts els valors màxims de tots els salts, s'interpreta que la capacitat de tot el canal en conjunt és el mínim d'aquests valors. Aquesta metodologia fa que el procés d'estimació sigui molt llarg quan entre l'emissor i el receptor hi ha diversos nodes, i a més no aporta un nivell de precisió i robustesa al tràfic creuat superiors que els mètodes anteriors, motiu pel qual és considerat un mecanisme menys interessant.

2.3 Mecanismes de detecció de congestió per a xarxes sense fils

El problema dels controls de congestió repassats a l'apartat 2.1 és que sempre assumeixen que els paquets es perden a causa de la congestió de la xarxa. Això és cert en entorns cablejats, però en el moment que hi ha algun enllaç sense fils entre l'emissor i el receptor pot ser que es produeixin pèrdues per altres causes (i.e. *fading*). Si això succeeix, no cal disminuir la velocitat de l'enviament, ja que realment l'enllaç no està congestionat i les pèrdues són aleatòries.

Per aquest motiu, han sorgit diverses versions de TCP i altres protocols que intenten discernir les pèrdues per congestió de les pèrdues aleatòries del canal, de manera que augmenti l'eficiència de l'enviament de dades en xarxes sense fils.

2.3.1 Mecanismes de Congestió Explícita

La sèrie de mecanismes exposats a continuació trenquen amb el paradigma d'extrem a extrem del nivell de transport i requereixen la implementació de processos en nodes intermedis de la xarxa, motiu pel qual la seva aplicació es veu clarament limitada a casos de control total de la xarxa entre emissor i receptor, a més de certa capacitat de

configuració dels nodes de la xarxa. Malgrat això, algunes de les propostes que presenten aquests mecanismes són cada vegada més adoptades pels fabricants de dispositius i administradors de xarxes, de manera que poden ser vàlides en alguns casos, facilitant la detecció de congestió i millorant les prestacions de les xarxes en general, i de les xarxes sense fils en particular.

2.3.1.1 Explicit Congestion Notification (ECN)

Explicit Congestion Notification (ECN) [46] és una notificació activada pels dispositius de xarxa a la capçalera del protocol de nivell de transport quan s'estima perill imminent de congestió en un punt de la xarxa. Quan el receptor rep un paquet amb una notificació ECN activada, informa a l'emissor mitjançant el corresponent ACK que estigui utilitzant. És un mecanisme que requereix la seva implementació en els extrems, però també en els routers intermedis, sent útil per a protocols que són molt sensibles a pèrdues (p.ex. TCP), ja que intenta evitar la pèrdua abans que aquesta succeixi. Es requereixen canvis en el comportament i en el processat en els hosts finals, ja que especifica la utilització del bit 8 i bit 9 de la capçalera TCP. El bit 9 es designa al flag d'ECN-Echo (ECE), activat pel receptor per indicar que ha trobat congestió a la xarxa, i el bit 8 com a Congestion Window Reduced (CWR), activat per l'emissor per indicar que s'ha reaccionat a la notificació de congestió. En xarxes sense fils es proposa un mecanisme d'extensió d'ECN per determinar, a més, la causa de pèrdua de paquets, observant la freqüència de paquets ECE rebuts en els ACKs rebuts.

El mecanisme proposat, anomenat RTT ECN Loss Differentiation (RELD) està basat en el control de congestió TCPlike de DCCP, i defineix el següent. [46]

Una pèrdua és considerada per congestió, si i només si:

$$1. \quad ECN > 0$$
$$o$$
$$2. \quad n > 0 \; i \; RTT < avg + 0.6dev$$

- *ECN*: és el nombre de paquets marcats amb ECE.
- *n*: el nombre de paquets marcats com perduts en l'ACK.
- *RTT*: Actual round-trip time.
- avg: Valor mitjà de RTT.
- dev: Desviació de RTT.

2.3.1.2 eXplicit Congestion Protocol (XCP)

Explicit Congestion Protocol (XCP) [47] proposa un mecanisme de feedback a l'emissor per part dels dispositius intermedis (routers o switches). Defineix l'ample de banda útil que ha de tenir cada enviament, a partir del càlcul de la capacitat del canal, la taxa d'enviament fins a aquest moment i la tendència de cues del dispositiu de xarxa. Utilitza una capçalera (Fig. 15) que han de processar i omplir tant els elements finals com els dispositius intermedis.

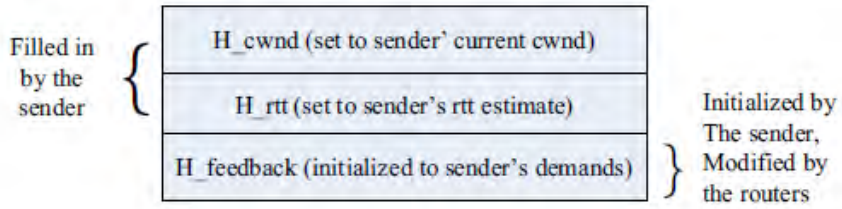

Fig. 15: Capçalera de congestió XCP.

Una variant d'XCP que es va proposar és XCP-b (*blind*), centrada en millorar el rendiment de XCP en xarxes sense fils [48], [49]. El càlcul de la capacitat del canal és complex en certs entorns, com ara 802.11, dependent de les taxes d'enviament de cada estació, el nombre d'estacions, el nombre de col·lisions, etc. XCP-b proposa un càlcul de l'ample de banda residual d'acord amb la informació de cues. El major inconvenient d'aquest enfocament és que un controlador de cua només mesura de manera fiable les variacions de la cua quan el medi està sent plenament utilitzat.

2.3.1.3 Adaptive Congestion Protocol (ACP)

ACP [50] és un altre protocol que utilitza un esquema multi-bit de senyalització explícita de congestió. De forma similar a XCP, un paquet ACP porta una capçalera de congestió que consisteix en 3 camps (Fig. 16).

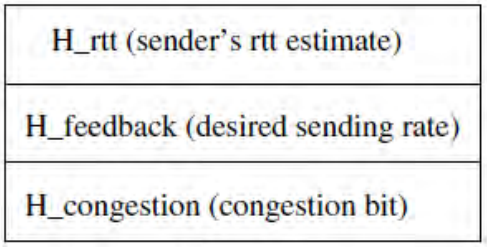

Fig. 16: Capçalera de congestió ACP.

- H_rtt: Transporta el valor de RTT actual.
- H_feedback: Indica la taxa d'enviament calculada per la xarxa.
    - Aquest camp comença amb el valor desitjat per l'emissor i es va actualitzant en cada enllaç que el paquet es va trobant pel camí fins al receptor.
    - En cada punt la capacitat instantània és comparada amb el valor guardat i el camp és actualitzat amb el menor valor.
- H_congestion: Un únic bit que inicialitza l'usuari a zero i és activat si la taxa de dades d'entrada en algun enllaç supera el 95% de la capacitat d'aquest enllaç, informant d'aquesta manera que s'acosta el moment de congestió en un punt de la xarxa i intentant evitar la pèrdua massiva de paquets.

Els resultats utilitzant aquest protocol milloren les prestacions de TCP o XCP tant per fluxos de curta com llarga durada [50]. No cal mantenir estats en els nodes intermedis, però si requereix que aquests realitzin funcions específiques per omplir aquesta capçalera.

En el cas específic de xarxes sense fils, recentment s'han proposat algunes variacions per millorar les prestacions d'ACP sota aquestes circumstàncies específiques [51], definint una variant anomenada iACP. Amb aquestes variacions plasmades en l'algoritme iACP s'aconsegueixen prestacions d'utilització de *thoughput* similars a les aconseguides per XCP-b, però amb un millor resultat de *fairness*.

2.3.1.4 Explicit Wireless Congestion Control Protocol (EWCCP)

Un altre dels protocols definits amb un control de congestió explícit és EWCCP [52]. Aquest protocol utilitza una capçalera (Fig. 17) semblant a les utilitzades per XCP i ACP, intentant tenir un control més exhaustiu de les dinàmiques d'una xarxa sense fils.

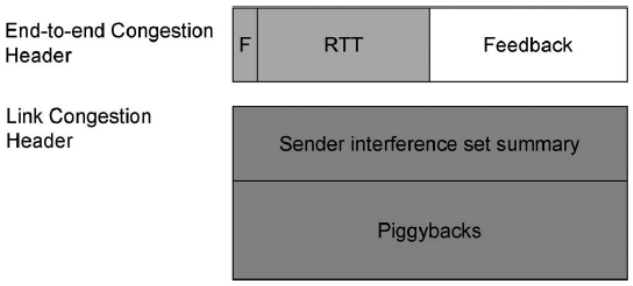

Fig. 17: Capçalera de congestió EWCCP.

La primera part de la capçalera és tractada pels extrems amb informació sobre la congestió percebuda extrem a extrem, mentre que la resta de la capçalera és emplenada i processada pels nodes intermedis per acomodar la taxa d'enviament a la capacitat instantània de l'enllaç.

Utilitzant una sèrie de mecanismes per coordinar l'ús del canal compartit entre diferents enllaços sense fils, EWCCP estabilitza una finestra de congestió menor però molt més òptima en comparació a TCP, que resulta en una ocupació dels buffers i un retard menors. És per això que és un protocol menys agressiu que intenta no superar mai el límit de congestió, però tenint un ús més eficient del canal i una major constància en l'enviament.

2.3.2 Mecanismes de Congestió Extrem a Extrem

Tal i com s'ha comentat prèviament, una de les millores principals que s'intenten proporcionar en el disseny de protocols de transport en entorns sense fils és la possibilitat de diferenciar entre les pèrdues de congestió i pèrdues de canal, per poder tractar els dos casos de manera diferent. Un cop explicades aquelles solucions amb *feedback* explícit per part dels nodes de xarxa, aquest apartat presenta alguns exemples de mecanismes extrem a extrem, així com protocols de transport que els apliquen.

2.3.2.1 Loss Differentiation Algortihms (LDAs)

Aquests mètodes estan generalment classificats en dues categories principals: aquells que es basen en el *Relative One-way Trip Time* (ROTT), o temps que tarda una comunicació d'anada i tornada, tenint en compte l'enviament i *l'acknowledge* rebut; i aquells que depenen del temps d'arribada entre paquets, també conegut com *Inter Arrival Time* (IAT) o *Jitter*.

En el cas del ROTT, algunes propostes [53] realitzen una anàlisi de la variació de ROTT i la relació amb les pèrdues detectades. En cas de percebre un augment en el ROTT quan s'ha detectat la pèrdua, aquesta és considerada que es produeix per congestió. Si no és així, es considera una pèrdua per culpa del canal. Depenent de la forma de realitzar el seguiment del ROTT i de la selecció del llindar per prendre la decisió de quin tipus de pèrdua ha succeit, trobem diferents solucions (ZigZag [54], ZBS [54], Spike [55], *Trend and Loss Density based* (TD) [56], etc.). Per exemple, TD observa si les pèrdues s'han percebut després d'un interval de temps després d'un pic en la corba de ROTT i, a més, mira la densitat de les pèrdues, ja que en poques ocasions es produeix la pèrdua d'un únic paquet per congestió. Quan s'observa una pèrdua d'aquest tipus, automàticament es associada a pèrdua per canal.

D'altra banda, existeixen algoritmes basats en *Jitter* o IAT com Biaz o mBiaz [57] que basen el seu càlcul en l'històric del menor temps entre paquets rebuts i el temps entre els últims paquets, aconseguint detectar en la majoria dels casos el tipus de pèrdua si es dóna el cas, ja que el coll d'ampolla de la xarxa es produeix generalment en l'últim salt i és un segment sense fil.

Altres algoritmes com *Statistical Packet Loss Discrimination* (SPLD) [58] funcionen també d'una manera similar. SPLD té un mòdul dedicat a recol·lectar i analitzar la informació sobre l'arribada dels paquets. Si durant un cert temps no es detecten pèrdues, el mòdul actualitza el temps mínim i mig entre paquets. Llavors, quan es produeix una pèrdua, aquest mòdul s'encarrega de discriminar quin tipus de pèrdues s'han produït amb aquests valors.

2.3.2.2 Protocols i variants basades en TCP pensades per a xarxes sense fils

2.3.2.2.1 Jitter Based TCP (JTCP)

El protocol *Jitter Based TCP* (JTCP) es basa en l'aplicació del *Jitter Ratio* (Jr) en el funcionament del control de congestió de TCP [59].

Per treballar amb el concepte de *jitter*, s'estableix la següent nomenclatura:

- **Temps d'enviament d'un paquet *i*** (t_s (i)). Indica en quin moment temporal s'ha enviat el paquet *i*.
- **Temps de recepció d'un paquet *i*** (t_r (i)). Indica en quin moment temporal s'ha rebut el paquet *i*.

Un cop definits els temps d'enviament i recepció, es relacionen els dos valors per establir el concepte d'*Interarrival Jitter*.

- ***Interarrival Jitter*** (D (i, j)). L'*Interarrival Jitter* compara l'espai temporal entre la recepció de dos paquets (t_r (i), t_r (j)) respecte l'espai temporal entre aquests dos mateixos paquets en l'enviament (t_s (i), t_s (j)). [Adimensional]

$$D\ (i,j) = (t_r(j) - t_r(i)) - (t_s(j) - t_s(i)) = (t_r(j) - t_s(j)) - (t_r(i) - t_s(i))$$

  - Si D (i, j) = 0, no existeix delay entre els paquets *i* i *j*.
  - Si D (i, j) > 0, vol dir que $t_j > t_i$ i que el paquet *j* ha estat encuat.

Definit el concepte d'*Interarrival Jitter*, es relaciona aquest valor per conèixer si les pèrdues produïdes són per problemes aleatoris del canal, com podria succeir en xarxes sense fils, o si són produïdes per la congestió del canal.

- ***Jitter ratio*** (***Jr***). El *Jitter Ratio* estima el rati de paquets en cua (*queued packets*) basant-se en l'*Interarrival Jitter*. Aquest càlcul permet relacionar l'efecte dels paquets en cua en un router intermedi (*bottleneck*) i el *delay* entre paquets a la recepció.

$$Jr = \frac{D(i-1,i)}{t_r(i) - t_r(i-1)} = \frac{\big(t_r(i) - t_s(i)\big) - \big(t_r(i-1) - t_s(i-1)\big)}{t_r(i) - t_r(i-1)}$$

A mesura que la càrrega del trànsit augmenta, comença a produir-se una cua de paquets en els routers intermedis, fet que es veu reflectit en el valor del *Jitter Ratio*. Un cop se supera el nombre de paquets que pot encuar un router intermedi, es produeixen pèrdues per congestió ja que aquest comença a descartar paquets.

Quan es tracten un conjunt de paquets (*congestion window* (w)), l'esquema de distribució (Fig. 18) d'aquests paquets dins la finestra de congestió queda de la següent manera:

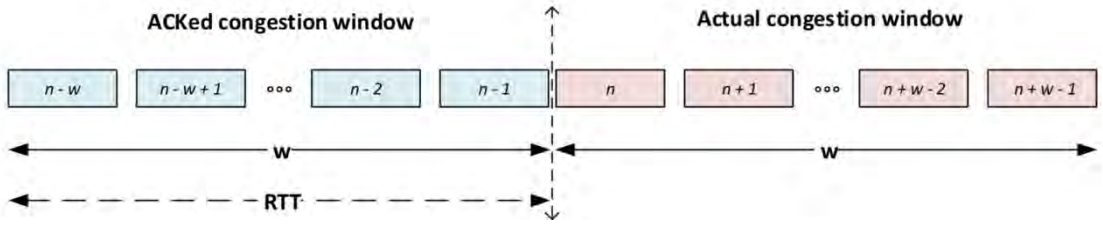

Fig. 18: Distribució dels paquets en dues finestres de congestió

El càlcul del *Jitter Ratio* queda de la següent forma:

$$Jr = \frac{D(n-w,n-1)}{t_r(n-1) - t_r(n-w)} = \frac{\big(t_r(n-1) - t_s(n-1)\big) - \big(t_r(n-w) - t_s(n-w)\big)}{t_r(n-1) - t_r(n-w)}$$

Un cop obtingut el *Jr*, si s'han produït pèrdues, aquest permet diferenciar quin tipus de pèrdues són comparant amb la longitud de la cua del router que actua com a coll d'ampolla.

- ***Queue Decisor*** (***k*/*w***). Aquest factor decisor defineix el nombre de paquets encuats (k) que es considera en una comunicació com a indici de congestió a l'injectar tots els paquets de la finestra de congestió (w).

En cas de produir-se pèrdues, es compara el valor de *Jr* amb el del *Queue Decisor* (k/w):

$$\begin{cases} Jr > \dfrac{k}{w} \rightarrow Pèrdues\ per\ congestió \\ Jr < \dfrac{k}{w} \rightarrow Pèrdues\ per\ canal \end{cases}$$

  - Quan el rati de paquets encuats, *Jr*, supera l'indicat pel *Queue Decisor*, implica que la xarxa està en una situació de congestió.

- Si pel contrari, el valor de *Jr* no supera el *Queue Decisor*, això implica que no hi ha congestió i que les pèrdues s'han produït per algun factor advers al canal.

En TCP amb esquema AIMD, basant-se en proves dutes a terme, el nombre de paquets considerats és k = 1 pel fet que TCP afegeix un segment per cada RTT [59], [60].

El procés que segueix JTCP per activar el control de congestió (Fig. 19) en rebre tres vegades ACKs duplicats (pèrdua de paquet) és el següent:

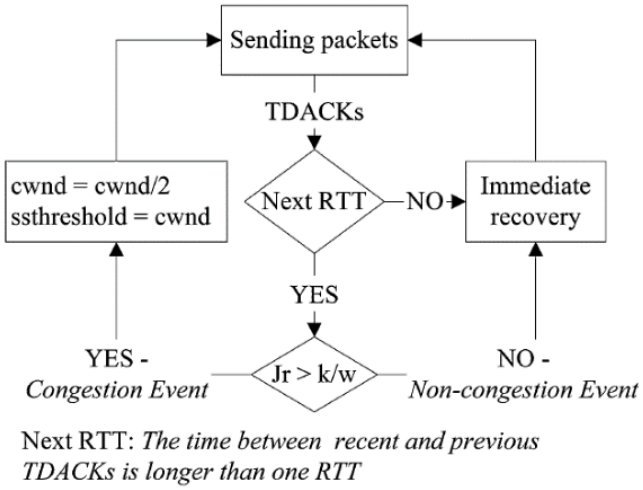

Fig. 19: Diagrama de JTCP en rebre un TDACKs.

2.3.2.2.2 TCP Veno

El protocol TCP Veno (Veno) [61] es basa en els protocols TCP Reno (Reno) i TCP Vegas (Vegas) [9], [62]. Vegas introdueix el concepte de *BaseRTT* que s'explica més endavant en aquest apartat, així com el concepte *ActualRTT*. Aquests dos paràmetres permeten discernir si la connexió entrarà o no en un estat de congestió.

El principal problema de Vegas és el *fairness* degut que no té en compte la resta de connexions en el càlcul del *BaseRTT*. A més, Vegas no està pensat per distingir esdeveniments aleatoris de pèrdues.

Veno està enfocat a solucionar el problema de la detecció de les pèrdues aleatòries que no són degudes a la congestió. Mitjançant el control de l'estat de la connexió, pot distingir entre un esdeveniment de pèrdues per congestió o un esdeveniment de pèrdues aleatori.

Per conèixer el funcionament de Veno, s'introdueixen dos conceptes sobre el *round- trip time.*

- ***BaseRTT***: És el *round-trip* mínim mesurat en la comunicació. També conegut com *BestRTT*. [ms]
- ***ActualRTT***: És l'últim *round-trip* mesurat en la comunicació. [ms]

Partint d'aquests dos conceptes de *round-trip times*, es calculen els següents ratis:

- ***Expected***: Relaciona la finestra de congestió (cwnd) amb el *BaseRTT*.

$$Expected = \frac{cwnd}{BaseRTT}$$

- ***Actual***: Relaciona la finestra de congestió (cwnd) amb l'*ActualRTT*.

$$Expected = \frac{cwnd}{BaseRTT}$$

- La **diferència** (***Diff)*** entre aquests dos ratis permet extreure informació sobre l'estat de la connexió.

$$Diff = Expected - Actual$$

Quan *l'ActualRTT* supera el *BaseRTT* (*ActualRTT* > *BaseRTT*) vol dir que comença a generar-se congestió a la xarxa, atribuint el *delay* al router o element intermedi que comença a encuar els paquets a causa que no pot processar-los tots. Per tant, l'*ActualRTT* es pot definir de la següent manera.

$$ActualRTT = BaseRTT + N/Actual$$

- ***Router queue (N)***: és la cua del router que genera el coll d'ampolla (*backlog queue*). De la fórmula de l'*ActualRTT* es pot extreure el valor actual de la cua *N*.

$$N = Actual \cdot (ActualRTT - BaseRTT) = Diff \cdot BaseRTT$$

Aquest valor *N* permet senyalitzar en quin moment la comunicació comença a entrar en un estat de congestió. Concretament, a Veno, el valor de *N* permet discernir quin tipus de pèrdues s'han produït.

- **Pèrdues:** Quan es produeix una pèrdua, es consulta el valor de la cua *N*.
  - Si $N < \beta$
    - Quan hi ha pèrdua, fa referència a pèrdues aleatòries a causa del canal. El router intermedi no està saturat en no haver-se superat el llindar, de manera que no està congestionat i no ha deuen haver-hi pèrdues per congestió.
  - Si $N > \beta$
    - Quan hi ha pèrdua, fa referència a pèrdues per congestió. Aquesta comparativa indica que ha superat el llindar, indicant que el router intermedi està saturat.
  - Empíricament [61], el valor òptim de β és , ja que a partir d'aquest valor es desestabilitza el sistema.

2.3.2.2.3 TCP Westwood/Westwood+

El protocol TCP Westwood / Westwood + (Westwood) [63], [64], [65] realitza una estimació de l'ample de banda mitjançant el filtrat dels missatges d'ACK rebuts en l'emissor. Després d'una pèrdua, modifica la *congestion window* (*cwnd*) i el *slow start threshhold* (*ssthresh*). Manté la semàntica end-to-end.

TCP Westwood + neix perquè el seu predecessor no realitza mesuraments correctes si es produeix *ACK Compression*. Té un comportament diferent pel que fa a la pèrdua de paquets a causa de congestió (*coarse timeout*) que per canal (3 DUPACKs).

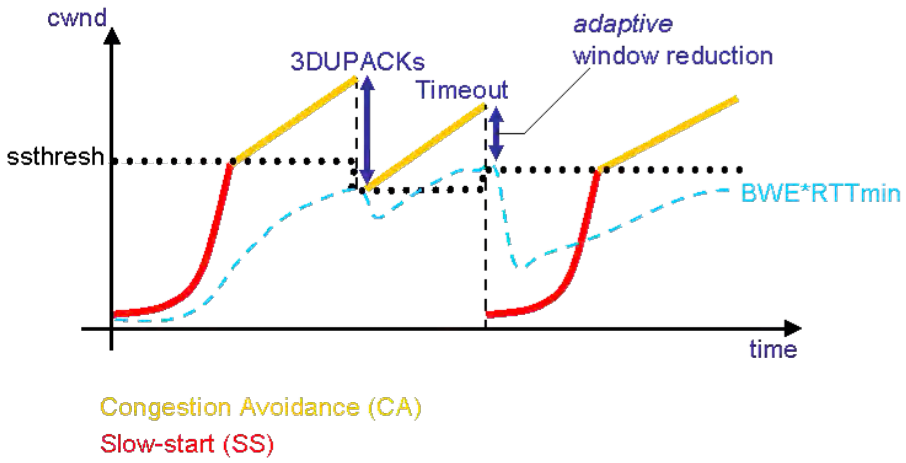

Fig. 20: Control de congestió de TCP Westwood+.

El comportament de Westwood varia segons l'esdeveniment que es produeixi (Fig. 20).

- Recepció d'**ACK**
  - Augmenta la finestra de congestió (*cwnd* = *cwnd* + *1*)
  - Calcula l'ample de banda estimat (BWE)
- Recepció **3 DUPACKs** (principal indicador de pèrdua per canal)
  - $ssthresh = \frac{BWE \cdot RTTmin}{\#segment\ size}$;
    - Si *ssthresh* < *2*, s'estableix que *ssthresh* = *2*.
  - $cwnd = ssthresh$
- *Coarse timeout* (principal indicador de pèrdua per congestió)
  - $ssthresh = \frac{BWE \cdot RTTmin}{\#segment\ size}$;
    - Si *ssthresh* < *2*, s'estableix que *ssthresh* = *2*.
  - $cwnd = 1$

2.3.2.2.4 TCP Jersey/New Jersey

El protocol TCP Jersey (Jersey) [66] neix de la necessitat de diferenciar pèrdues per congestió o per canal. A causa que l'estimació de l'ample de banda la realitza mitjançant el temps de recepció dels paquets ACKs (com més gran és la cadència i velocitat de recepció de ACKs, més gran és la velocitat d'enviament), assumint que els canals d'enviament i recepció són iguals i ideals, comporta al fet que es falsegi la mesura, ja que l'assumpció que els canals són iguals i ideals no és real. El protocol TCP New Jersey (New Jersey) es desenvolupa per introduir un timestamp en els paquets rebuts, amb l'objectiu d'utilitzar aquesta informació en recepció i tornar-la al emissor mitjançant els

paquets ACKs perquè tingui informació real de la cadena d'enviament. Basa el seu funcionament en dos components clau:

- ***Timestamp-base Available Congestion Estimation* ($TABE$).** Estimació de l'ample de banda en l'emissor, processant la informació dels ACKs en funció dels timestamps dels paquets rebuts pel receptor. [Bps]

$$R_n = \frac{T_w \cdot R_{n-1} + L_n}{(t_n - t_{n-1}) + T_w} = \frac{R_{TT} \cdot R_{n-1} + L_n}{(t_n - t_{n-1}) + R_{TT}}$$

  - t_n: Temps d'arribada del paquet **n**. [s]
  - t_{n-1}: Temps darribada del paquet anterior **n-1**. [s]
  - L_n: Mida del paquet **n**. [b]
  - T_w: *Constant Time Window* → RTT. [s]

  - Monitorització dels ACKs reebuts pel càlcul de la finestra de congestió òptima
  - **Congestion window òptima (ownd).** [segments]

$$ownd = \frac{R_{TT} \cdot R_n}{seg\_size}$$

    - seg_size: Mida del segment (payload).
    - No depèn de la configuració de paràmetres.

- ***Congestion Warning (CW).*** Els routers intermitjos detecten que s'està produint congestió i marquen els paquets a la seva capçalera, tal i com s'explica explicat a l'apartat 2.3.1.
  - **CE** a IP.
  - **ECE** i **CWR** a TCP.

L'esquema ECN marca els paquets de manera probabilística sobre la base de l'***average queue length*** (longitud de la cua del buffer del router), mentre que el router informa l'emissor i influencia a TCP.

RED i ECN són sensibles als paràmetres de configuració. Per això, es proposa CW, que només depèn d'alguns paràmetres. En l'esquema proposat de CW, el router ha de marcar tots els paquets quan l'*average queue length* sobrepassa el *threshold* (*thresh*), permetent a TCP que modifiqui el seu comportament. La diferència de comportament es mostra en la següent figura (Fig. 21):

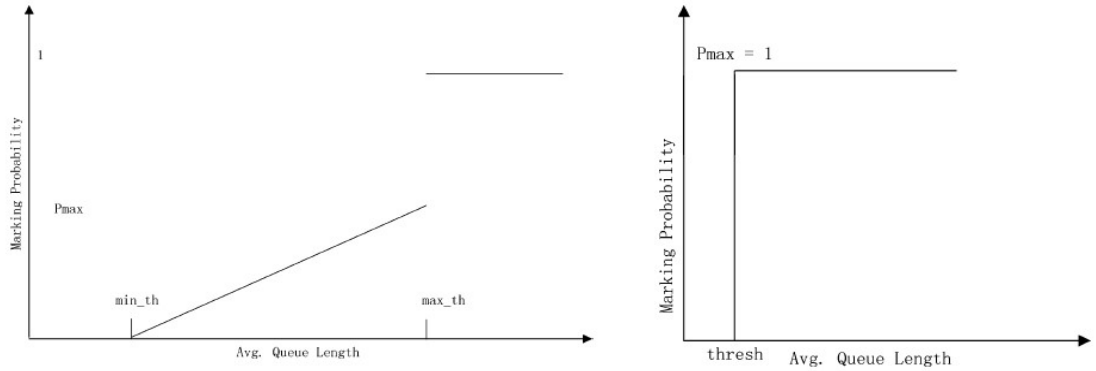

Fig. 21: Funció d'ECN vs Funció de CW [66]**.**

Depenent del valor del pes de la cua, ***queue weight***, l'*average queue length* tindrà un comportament més o menys fidel a l'estat del buffer, assumint que es poden modificar els paràmetres dels routers. A la Fig. 22s'observa com varia el *queue length* depenent del valor del *queue weight*.

- ***Queue weight = 0,002***: Valor original d'ECN. Realitza un *long-term averaging*, suavitzant la funció respecte el comportament real del canal.
- ***Queue weight = 0,2***: Valor fixat per TCP-Jersey. Realitza un seguiment de l'estat de la cua, així com dels pics que es produeixen, donant una informació més precisa del buffer.

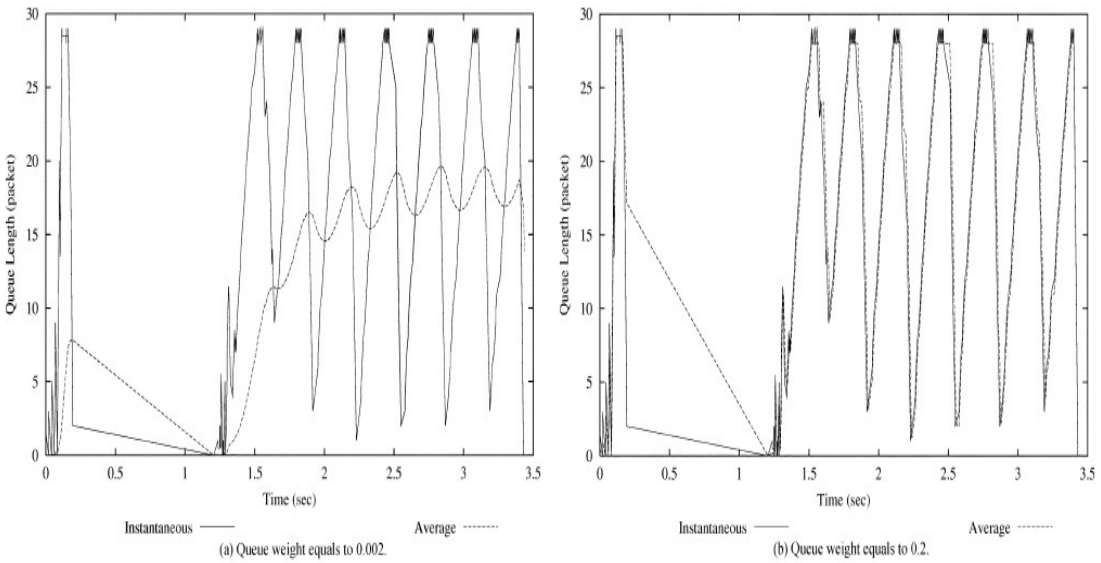

Fig. 22: Comportament del *queue lenght* respecte el *queue weight* amb un valor de 0,002 (a) i amb un valor de 0,2 (b) [66]**.**

A partir d'aquest valor, es requereix al router que indiqui si s'ha produït o no congestió. El comportament de TCP Jersey, per tant, depèn del paquet que rebi, tot i que sempre farà el càlcul d'ample de banda estimat (TABE).

- Recepció d'**ACK** i **CW = 0** (No hi ha congestió)
  - Aplica el procés normal de Reno, *Slow Start* o *Congestion Avoidance*, depenent d'on es situï la *cwnd*.

- Recepció d'**ACK** i **CW = 1** (Sí hi ha congestió)
    - Aplica el procés de *rate control.*
        - *ssthresh = ownd.* La *congestion window* òptima calculada.
        - *cwnd = ssthresh.* Com si estigués a l'estat de *congestion avoidance*.
    - Aplica el procés normal de Reno, *slow start* o *congestion avoidance*, depenent d'on es situï la *cwnd*.
- Recepció de **DUPACK** i **CW = 0** (Pèrdues aleatòries del canal)
    - Aplica el procés normal de Reno, *explicit retransmit* i *fast recovery*.
- Recepció de **DUPACK** i **CW = 1** (Pèrdues per congestió)
    - Aplica el procés de *rate control.*
        - *ssthresh = ownd.* La *congestion window* òptima calculada.
        - *cwnd = ssthresh.* Com si estigués a l'estat de *congestion avoidance*.
    - Aplica el procés normal de Reno, *explicit retransmit* i *fast recovery*.

2.3.2.2.5 Jitter Stream Control Transmission Protocol (JSCTP)

El protocol *Jitter Stream Control Transmission Protocol* (JSCTP) es basa en la semàntica i funcionament de SCTP, afegint el càlcul del jitter per a la diferenciació entre pèrdues per congestió i pèrdues pròpies del canal. Concretament, l'objectiu és poder adaptar i aplicar SCTP a xarxes sense fils [60].

Les característiques principals de SCTP són les següents:

- Ús de SACK (*slow start* i *congestion avoidance* extret de TCP).
- Semàntica *end-to-end.*
- *4-way handshake* per a l'establiment de connexió client-servidor, a més de l'ús d'una *cookie*.
- *Multihoming* i *multistreaming*
    - No s'utilitza per *load balance* ni *load-sharing*. S'utilitza per retransmetre i realitzar *backup*.

Es proposa el mesurament del *jitter* i el càlcul del *jitter ratio* per a la diferenciació entre pèrdues per congestió i pèrdues a causa del canal. Utilitza els mateixos conceptes de JTCP. Per això és necessari afegir un *timestamp* a la capçalera de 12 Bytes de mida.

El càlcul del ***Jitter Ratio***, com a recordatori, es realitza de la següent manera:

$$Jr = \frac{D(n-w, n-1)}{t_r(n-1) - t_r(n-w)} = \frac{\left(t_r(n-1) - t_s(n-1)\right) - \left(t_r(n-w) - t_s(n-w)\right)}{t_r(n-1) - t_r(n-w)}$$

En el cas de JSCTP, el valor de **n** indica l'últim *Transmission Sequence Number* (TSN) *acked* i el valor de **w** indica la mida de la finestra de congestió (segments).

El factor decisori, **Queue Decisor**, defineix el nombre de paquets en cua (***k***) que es considera en una comunicació com a indici de congestió en injectar tots els paquets de la finestra de congestió (***w***).

En cas de produir-se pèrdues, es compara el valor de *Jr* amb el del *Queue Decisor*:

$$\begin{cases} J_r > \dfrac{k}{w} \rightarrow Pèrdues\ per\ congestió \\ J_r \leq \dfrac{k}{w} \rightarrow Pèrdues\ del\ canal \end{cases}$$

La problemàtica es dóna en el cas de no produir-se variacions a la cua, sent 0 l'*average* total, i obtenint un valor de **Jr = 0**. Això porta al fet que el sistema es confongui, atribuint les pèrdues al canal, quan s'han produït a causa a la congestió, i comportant un mal funcionament del sistema.

- ***Smooth Jitter Ratio* ($smoothjr$)**. Aquest filtre es proposa per evitar l'afectació de casos puntuals en què *Jr* = 0, realitzant-se la mateixa comparació amb el *Queue Decisor*. [Adimensional]

$$smooth_{jr} = (1 - \alpha) \cdot\ smooth_{jr} + \alpha \cdot J_r$$

  - El valor de α fixa en quin tant per cent (%) es té en compte el valor de *Jr* obtingut. Empíricament, es proposa α = 0,05.

Per al control de congestió, es proposa utilitzar el sistema utilitzat en TCP New Jersey, *timestamp-based available bandwidth estimation* (TABE).

- ***Timestamp-base Available Congestion Estimation* ($TABE$).** Utilitza els timestamps per registrar la informació de l'enviament i retornar-la al emissor per al càlcul de l'ample de banda, fent una estimació de l'ample de banda en l'emissor i processant el rate de recepció dels ACKs rebuts. [Bps]

$$R_n = \frac{T_w \cdot\ R_{n-1} + L_n}{(t_n - t_{n-1}) + T_w} = \frac{R_{TT} \cdot\ R_{n-1} + L_n}{(t_n - t_{n-1}) + R_{TT}}$$

    - t_n: Temps d'arribada del paquet **n**. [s]
    - t_{n-1}: Temps darribada del paquet anterior **n-1**. [s]
    - L_n: Mida del paquet **n**. [b]
    - T_w: *Constant Time Window* → RTT. [s]
  - Monitorització dels ACKs reebuts pel càlcul de la finestra de congestió òptima
  - **Congestion window SCTP (cwnd).** [segments]

$$cwnd = \frac{R_{TT} \cdot\ R_n}{seg\_size}$$

    - seg_size: Mida del segment (payload).

El procés que segueix JSCTP per activar el control de congestió (Fig. 23) en rebre 4 DupSACK (pèrdua de paquet) és el següent:

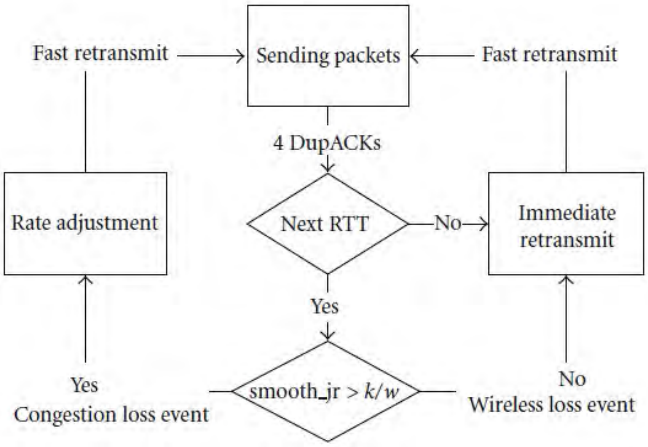

Fig. 23: *Flow Chart* de JSCTP en rebre quatre DupSACKs [60].

- Pèrdues per congestió (*Smooth Jitter Ratio* > *Queue Decidor*): El procés *Rate adjustment* es duu a terme.
  - $ssthresh = RTT \cdot \frac{R_n}{seg\_size}$
  - $cwnd = ssthresh$
  - Reenviament del paquet perdut.
- Pèrdues degudes al canal (*Smooth Jitter Ratio* < *Queue Decidor*): només es realitza el reenviament del paquet perdut, sense modificar la finestra de congestió.
  - Reenviament del paquet perdut.

Aquest procés permet discernir entre els diferents tipus de pèrdues que es produeixen, definint així el comportament del protocol segons la situació donada.

2.3.2.2.6 Altres solucions cross-layer basades en TCP

Es poden trobar una sèrie de variants de TCP que se centren en millorar el rendiment de TCP sobre xarxes ad-hoc, les quals pateixen variacions sobtades de la topologia que poden trencar la ruta entre emissor i receptor. Alguns d'aquests protocols són TCP Feedback (TCP-F) [67], Explicit Link Failure Notification (TCP-ELFN) [68], o TCP-BuS [69]. Els tres protocols presenten mecanismes per detectar un canvi en la topologia que hagi provocat una pèrdua de ruta, sent el node intermedi el que ha detectat la pèrdua de l'enllaç i el que envia a l'emissor un missatge Route Failure Notification (RFN), d'una forma similar a la que es realitza amb la notificació de la congestió en protocols explícits, com XCP. Es disposa de solucions per a entorns sense fils molt dinàmics però basades en mecanismes que requereixen un processat específic per part dels nodes intermedis, la qual cosa descartaria el seu ús excepte en entorns de xarxes sense fils poc complexes, en les quals es disposi de dispositius completament configurables.

2.3.2.3 Protocols de transport basats en UDP sobre xarxes sense fils

2.3.2.3.1 Reliable Blast UDP (RBUDP)

Reliable Blast UDP (RBUDP) [70] va ser un dels primers protocols que va intentar oferir fiabilitat i integritat de dades sobre UDP. Les seves prestacions quant a ample de banda aconseguit van ser superades més endavant per Tsunami i UDT i la seva adaptació a xarxes sense fils és inexistent, ja que parteix amb una metodologia d'enviament de finestra fixa i no disposa de mecanismes per acomodar condicions variables del canal [71] que ho facin més adaptable per a entorns sense fil.

2.3.2.3.2 Tsunami

Tsunami [72] és un protocol que presenta una arquitectura de comunicació utilitzant UDP per a l'enviament de dades i TCP per al control de la transferència (Fig. 24).

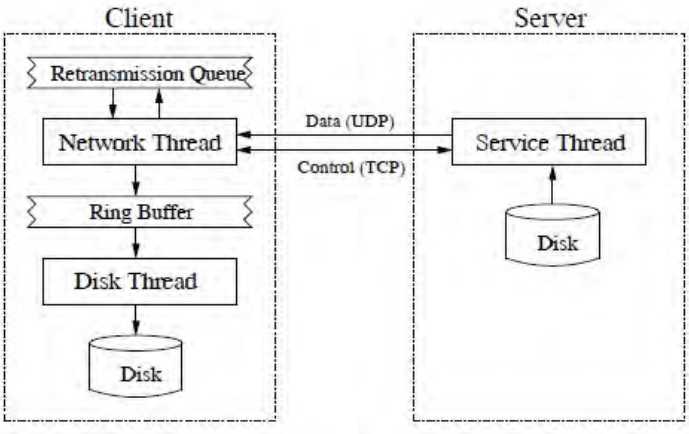

Fig. 24: Arquitectura del protocol tsunami [72]**.**

L'establiment de la sessió, les retransmissions i altres operacions de control són realitzades mitjançant TCP, mentre que la transmissió de les dades útils es realitza mitjançant una altra connexió UDP. Tot i documentar *throughputs* propers al 90% en enllaços dedicats d'1 Gbps, no es tenen dades del seu rendiment en entorns sense fils i no s'han presentat modificacions del protocol específiques per a aquests entorns.

2.3.2.3.3 UDT

UDT [23] utilitza un esquema similar a l'utilitzat per Tsunami, realitzant una connexió UDP i una altra TCP per al control de la transferència. Està especialment dissenyat per a la transferència massiva de dades a alta velocitat per tal d'eliminar i reduir la sobrecàrrega en el processament d'informació rebuda i pèrdues pel desbordament dels buffers, i té un mecanisme de control de congestió que manté l'eficiència, el *fairness* i l'estabilitat de l'enviament. Utilitza un *positive acknowledge* (ACK) que s'envia a cada interval constant, mentre que es genera un *negative acknowledge* (NAK) tan aviat com es detecta la pèrdua de paquets. Igual que altres protocols d'aquesta mateixa família, ha centrat el seu disseny i desenvolupament en xarxes cablejades i especialment en xarxes dedicades que disposen de gran ample de banda, però no es tenen detalls del seu comportament sobre xarxes sense fils, ni tampoc s'han trobat variants específiques per a transmissions d'aquest tipus.

2.3.2.3.4 Reliable UDP with Flow Control (RUFC)

Reliable UDP with Flow Control (RUFC) [73] presenta una solució múltiple amb l'aplicació de diferents polítiques de control de flux sobre un enviament UDP. Mitjançant una capçalera específica (SEG) (Fig. 25) s'encarrega d'afegir la informació necessària en cada segment per processar el control de flux i la integritat de dades.

Field	Size	Field	Size
Sequence No.	4 bytes	Control	2 bytes
Magic No.	1 byte	Version	1 byte
Acknolwdge No.	4 bytes	Signal	2 bytes
Payload Size	2 bytes		

Fig. 25: Capçalera SEG de RUFC [73].

Partint de diferents components disponibles pel protocol (p.ex. etiquetatge i seqüenciació, fragmentació i desfragmentació, retransmissions, *negative acknowledgements*, finestra lliscant, etc.) el protocol utilitza alguns d'ells depenent de la política que vulgui aplicar en cada cas. Les polítiques definides són les següents:

- *Selective Rate Adaptive UDP* (SRAU): Esquema simple on el client ajusta la seva taxa de transmissió basat en el nombre de retransmissions que s'han hagut de realitzar en l'interval de temps anterior. Molt poc flexible ja que està limitat a una sèrie de taxes d'enviament ja prefixades i va variant entre elles en funció de les pèrdues trobades. SRAU no disminueix de manera agressiva la velocitat de transmissió si es produeix un pic de trànsit. És per això que és adequat per als casos en què els pics de trànsit no són molt freqüents.
- *Fine grained Rate Adaptive UDP* (FRAU): Sistema semblant a l'anterior, però en aquest cas sí que augmenta la taxa d'enviament cada vegada que no es produeix cap error, o les retransmissions han estat menors a un llindar determinat. És per això una política molt més agressiva que l'anterior.
- *Window Based UDP* (WUDP): Utilitza un mecanisme de finestra lliscant AIMD que és molt poc òptim per a casos en els quals hi ha molts pics d'enviament, en canals variables com en les xarxes sense fils. Per tant, és una opció clarament descartable per a aquest tipus de mitjans.
- *XOR over UDP* (XUDP): Mecanisme de finestra lliscant com WUDP però enviant a la vegada paquets de forma redundada per evitar gestionar un gran nombre de retransmissions. D'altra banda, aquest sistema disminueix molt l'eficiència de l'enviament ja que s'envia molta més quantitat de dades no útils en cas que no hi hagi errors.

Analitzant les diferents alternatives que ofereix RUFC és possible observar com FRAU pot aconseguir millors prestacions en xarxes sense fils gràcies als mecanismes per adaptar-se ràpidament a variacions en el canal. A la Fig. 26 es mostra un gràfic comparatiu del funcionament de RUFC, i de la millora del *throughput* aconseguit per FRAU respecte a Tsunami o UDT.

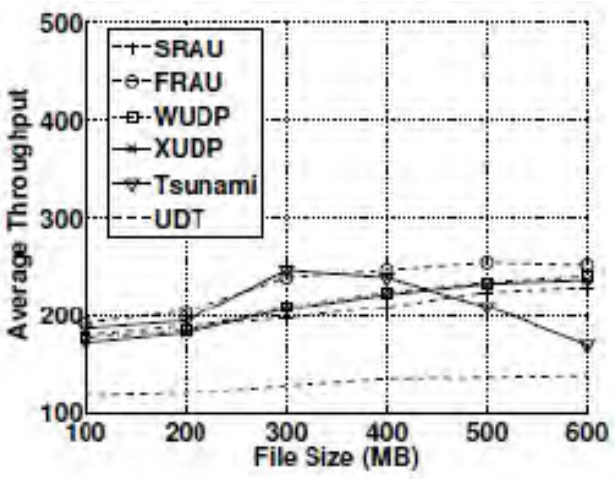

Fig. 26: Gràfica comparativa del *throughput* entres les diferents polítiques de RUFC vs Tsunami vs UDT [73]**.**

Aquests mecanismes poden ser, a més, complementats per altres específics per millorar la transmissió en xarxes sense fils, com la diferenciació de pèrdues de canal i pèrdues per congestió, mitjançant l'anàlisi del *jitter* rebut i/o altres mecanismes explícits de control de congestió [73].

2.3.2.3.5 FOBS

A més dels protocols anteriorment esmentats, cal destacar també l'aportació del protocol FOBS [74], el qual fa una anàlisi estadística de la dinàmica de paquets perduts i realitza una distinció d'aquells paquets que han estat perduts per la xarxa, d'aquells que no s'han pogut processar per limitacions de la CPU dels dispositius finals. Aquest fet apareix especialment de forma més comuna en transmissions de grans volums de dades en què els nodes finals són incapaços de processar la quantitat de paquets rebuts. Tot i no ser una variant específica per a xarxes sense fils, és rellevant tenir en compte l'algoritme proposat per evitar que errors de congestió de la xarxa es vegin emmascarats per aquest tipus d'errors i seguir diferenciant els errors detectats com a errors del canal o de cogestió .

2.3.2.4 Comparativa entre Loss Threshhold Decissors (LTD)

Després d'estudiar les diferents alternatives per diferenciar les pèrdues per congestió de les pèrdues del canal que ofereixen els diversos protocols analitzats, s'observa que els mecanismes que obtenen millors resultats són aquells basats en el *jitter*, concretament en el càlcul del *Jitter Ratio* (JTCP, JSCTP). Per aquest motiu, es s'opta per adaptar aquest mecanisme al protocol OMBTAP.

Protocol	**Veno**	**Westwood+**		**New Jersey**	**JTCP**	**SCTP**	
Factor de decisió	β	α_k	τ	*threshold*	*k*	*K*	α
Descripció	Cua del router	Bandwidth Estimat	Freqüència de tall	Capacitat del buffer	Queue Decisor (k/w)	Queue Decisor (k/w)	Smooth Jr
Valor	3	0,9	0,5	1/3 link buffer	1	1	0,05
Mecanisme	RTT	BWE, RTT		ECN, RTT	Jitter	Jitter	

Taula 1: Comparativa protocols – *Loss Threshold Decisor (LTD).*

3 El protocol OMBTAP

En l'anterior apartat s'han vist nombrosos protocols de transport que intenten resoldre les mancances del TCP original i millorar-ne el seu rendiment, tant per a xarxes cablejades com heterogènies. Tot i això, aquests no tenen un comportament agressiu, ja que intenten adaptar la seva velocitat a l'ample de banda lliure del canal (comportament *friendly*), i no pas acaparar la màxima capacitat. D'aquí va sorgir la necessitat de definir un nou protocol pensat per a la transferència de fitxers d'alta prioritat i que, per tant, ocupéssin el màxim d'ample de banda possible. Aquest protocol és MBTAP, dissenyat des d'un primer moment per tenir un comportament *unfriendly* amb la resta de protocols [3].

En la seva especificació inicial, el protocol MBTAP es divideix en 6 mecanismes:

- **Connexió:** S'inicia la comunicació entre client i servidor evitant atacs DoS.
- **Seguretat:** En aquest procés es prenen mesures d'encriptació i autenticació.
- **Estimació inicial de l'ample de banda:** Mitjançant tràfic de prova es realitza el càlcul de la capacitat màxima de l'enllaç.
- **Intercanvi de dades:** Es realitza la transferència de dades útils amb un control de paquets perduts i un control de congestió.
- ***Heartbeat*:** Client i servidor intercanvien missatges per corroborar que la connexió segueix activa.
- **Desconnexió:** Es finalitza la comunicació.

Durant la realització del pre-prototipat del protocol en físic es van realitzar lleugeres modificacions respecte l'especificació inicial [4] i es va considerar que durant la fase de proves només calien implementar els mecanismes de connexió, estimació, enviament i desconnexió, ja que són els que realment són necessaris per assolir l'objectiu del protocol. El mecanisme de seguretat és un extra que aporta autenticitat al protocol, però que en la fase de proves en que es troba el projecte només afegiria més complexitat a l'hora d'avaluar el seu funcionament. A més, aquest mecanisme seria impossible d'implementar al simulador Riverbed Modeler, ja que aquest no permet incorporar les llibreries de criptografia necessàries per a tal propòsit. De fet, el simulador només permet simular aplicacions en text pla (HTTP, Telnet...) però no els seus homònims que compten amb encriptació de les dades (HTTPS, SSH...).

D'altra banda, el mecanisme de *heartbeat* per comprovar l'estat de la connexió es considera que no és necessari ja que, durant l'intercanvi de dades, els missatges de confirmació entre client i servidor s'envien a una cadència més elevada que els propis missatges de *heartbeat*. Per tant, amb aquests missatges de confirmació ja n'hi ha prou per determinar si la connexió segueix activa o no.

Després de realitzar diverses bateries de proves amb el protocol MBTAP dins del simulador, es van detectar certes mancances que van ser millorades en la següent iteració de disseny del protocol (MBTAPv2), on es va definir un nou sistema inicial d'estimació de la capacitat màxima del canal i es va modificar lleugerament el control de congestió.

Tot i les millores, el protocol MBTAPv2 encara presentava dos inconvenients [75]:

- Quan més d'un fluxe MBTAP competia per l'ample de banda d'un enllaç, el comportament era massa agressiu i es generaven moltes pèrdues. Era necessari, doncs, implementar un mecanisme que controli el *fairness* entre fluxos MBTAP.
- MBTAP estava concebut només per a entorns completament cablejats, de manera que sempre assumia que les pèrdues eren causades per la congestió de l'enllaç. Per tant, es necessitava incorporar un mecanisme que diferenciés entre pèrdues per canal i pèrdues per congestió, adaptant així el protocol a xarxes heterogènies.

Per solucionar aquesta darrera qüestió, sorgeix el protocol Optimized MBTAP (OMBTAP), que incorpora nous mecanismes per acomodar el protocol i el seu control de congestió a aquest tipus de xarxes.

En aquest apartat, primerament es farà un repàs dels mecanismes i el funcionament del protocol MBTAP (versió 2), i a continuació es definiran els mecanismes que s'incorporan a l'OMBTAP. Els formats dels missatges OMBTAP es poden consultar a l'Annex II: Format dels missatges MBTAP.

3.1 Mecanismes MBTAP

3.1.1 Connexió

En aquest procés entre client i servidor sempre és el primer qui realitza la petició de connexió. És important que durant aquest primer intercanvi de missatges es puguin prevenir atacs de denegació de servei (DoS). El procés d'intercanvi de missatges d'aquest mecanisme es pot observar a la Fig. 27.

Primerament, el client envia un missatge d'inicialització (`INIT`) per poder establir la connexió amb el servidor, el qual es troba escoltant peticions de connexió per donar servei a més d'un usuari. En rebre el missatge, el servidor no reserva recursos per evitar així un atac DoS. Aquest respon amb un missatge (`INIT-ACK`) que inclou una *Cookie* amb la informació que associa la connexió client-servidor, juntament amb un *hash* d'aquesta *Cookie* que permetrà, posteriorment, comprovar-ne la seva integritat.

Un cop el client ho rep, respon amb un missatge (`COOKIE-ECHO`) on copia la mateixa Cookie sense realitzar cap modificació. El servidor llavors extreu la *Cookie* rebuda i comprova que el *hash* d'aquesta sigui el mateix que el de l'enviada. Si la verificació és correcta, el servidor reserva els recursos necessaris per establir la connexió amb el client i respon amb un missatge (`COOKIE-ACK`), donant per acabada la fase de

connexió. En cas d'error, el servidor també ho notifica (ABORT) i es dóna per acabada la comunicació.

L'avantatge d'utilitzar un *Four-way Handshake* recau en el fet de no haver de reservar recursos un cop rebuda la petició de connexió per part del client, ja que aquests no es reservaran fins a obtenir la resposta amb la *Cookie* i que aquesta sigui correcta, assegurant així un nivell de prevenció enfront d'atacs DoS i integritat de la connexió.

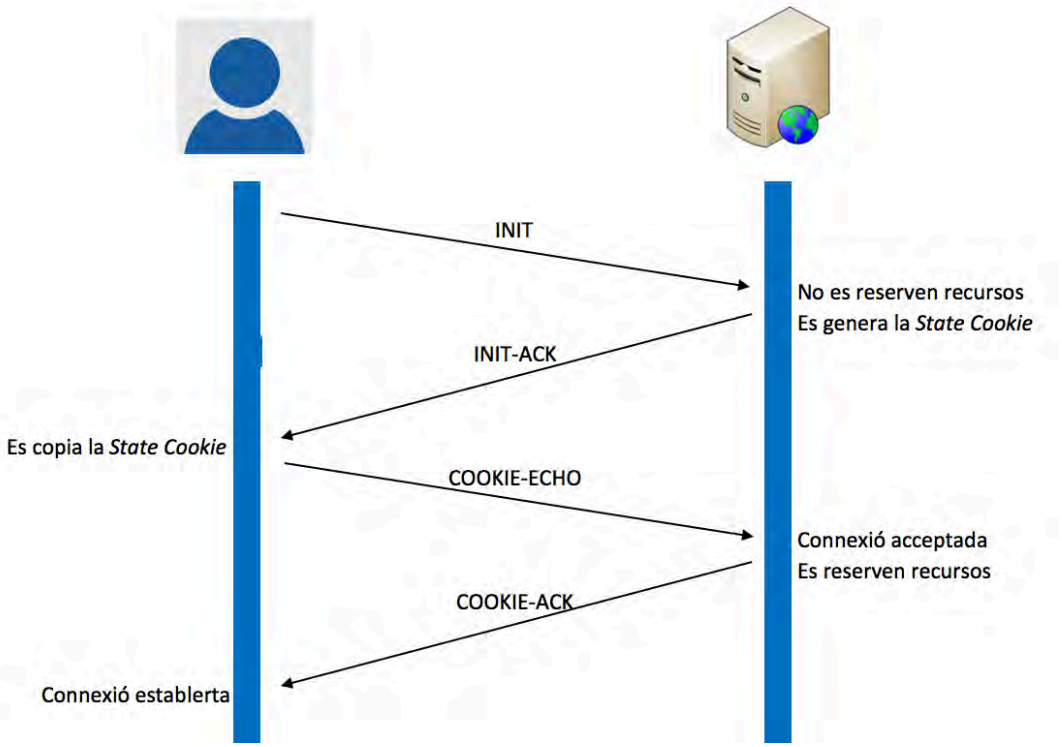

Fig. 27: Diagrama d'intercanvi de missatges en l'establiment de connexió.

3.1.2 Estimació inicial de l'ample de banda

A diferència d'altres mecanismes d'estimació que se centren en trobar l'ample de banda residual (lliure) de l'enllaç, aquest té com a objectiu estimar l'ample de banda total, ja que és un mecanisme pensat per ser utilitzat en un protocol agressiu que busca enviar els seus fitxers el més ràpid possible. A la Fig. 8 es mostra el diagrama d'intercanvi de missatges d'aquest mecanisme.

Un cop completada la connexió entre el client i el servidor, el primer comença a enviar ràfegues de 2 missatges BW (*packet pair*) de mida L_{p1} = 700 bytes cadascun (fase 1). Per cada ràfega rebuda, es mesura la seva dispersió δ (diferència de temps entre l'arribada del primer paquet de la ràfega i l'arribada del segon paquet) i els *One way Delay* (OWD) de cada paquet (diferència entre el temps d'enviament i el temps de recepció del paquet). Amb les mesures dels OWDs es busca el valor mínim, tant per separat com la suma entre els OWDs del primer i segon paquet de cada ràfega. Es considera que una ràfega és la mostra vàlida (és a dir, la que permet calcular la capacitat màxima de l'enllaç) si es compleix que la següent condició durant 40 mostres:

$$\mathrm{OWD}_{min}\{\text{paquet 1} + \text{paquet 2}\} = \mathrm{OWD}_{min}\{\text{paquet 1}\} + \mathrm{OWD}_{min}\{\text{paquet 2}\}.$$

Aleshores, amb la dispersió mesurada de la mostra considerada vàlida es calcula la capacitat de la fase 1:

$$C_{p1}[bps] = \frac{L_{p1}\ [bits]}{\delta\ [segons]}$$

Un cop acabada la fase 1, el servidor ho indica enviant un missatge SACK al client. Si, pel contrari, passat l'enviament de 100 *packet pairs* no s'ha obtingut cap mostra vàlida, les mides L_{p1} i L_{p2} dels paquets s'augmenten o es redueixen un 20% segons si el rati entre la capacitat màxima i la mínima calculades és superior o inferior a 50, respectivament. A continuació, s'haurà de repetir el procés des de l'inici (fase 1).

Quan el client rep el missatge SACK s'inicia la fase 2, que consisteix en el mateix però canviant la mida dels missatges BW a L_{p2} bytes (inicialment són 900 bytes). Així doncs, a partir de l'enviament d'aquests *packet pairs* s'obté la capacitat de la fase 2:

$$C_{p2}[bps] = \frac{L_{p2}\ [bits]}{\delta\ [segons]}$$

Si els valors de C_{p1} i C_{p2} són massa dispars (diferència superior al 5%), caldrà repetir tot el procés des de la fase 1. En cas que siguin prou semblants (diferència inferior al 5%), es pot calcular el valor d'Estimated Link de la següent manera:

$$Estimated\ Link = \frac{C_{p1} + C_{p2}}{2}$$

Finalment, es defineix que en la primera ràfega enviada es comença amb un SR de valor 0,9 vegades l'Estimated Link, de manera que el control de congestió pugui actuar ja des del principi incrementant o reduint el nombre de paquets per ràfega. Així doncs el valor del SR inicial [paquets per segon] es calcula com:

$$SR = \frac{0.9 * Estimated\ Link}{9000 * 8}$$

Un cop obtinguts els dos valors, el servidor envia un missatge SACK al client que conté aquesta informació. El diagrama d'intercanvi de missatge es mostra a la Fig. 28.

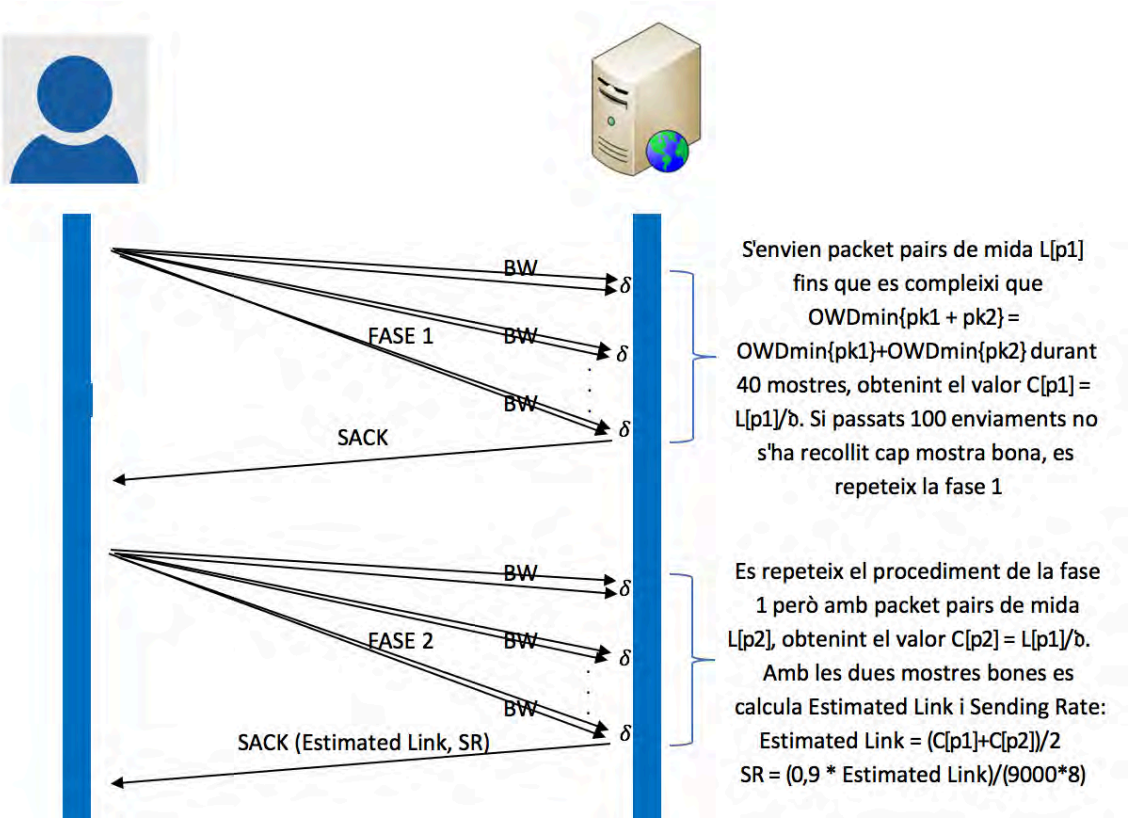

Fig. 28: Diagrama d'intercanvi de missatges del nou mecanisme d'estimació inicial de l'ample de banda.

3.1.3 Intercanvi de dades

Després d'haver calculat una estimació de l'ample de banda màxim de l'enllaç (Estimated Link) i conèixer la velocitat de transferència (Sending Rate) comença l'intercanvi de dades. El seu diagrama d'intercanvi de missatges es pot observar a la Fig. 29. El procés es realitza mitjançant ràfegues de paquets de 9000 bytes separades per un temps T_{burst}, el qual per disseny i després de fer proves en entorns físics es va considerar correcte que fos de 50 ms. A partir del Sending Rate (SR) [paquets per segon] i el T_{burst} [segons] es pot saber el nombre de paquets a enviar en una ràfega ($Packet_{burst}$) [paquets] de la següent manera:

$$Packet_{burst} = SR * T_{burst}$$

El client envia de forma consecutiva $Packet_{burst}$ paquets de 9000 bytes cadascun cap al servidor, qui guarda les dades útils rebudes i va llistant els paquets perduts en forma de *gaps* (grups de paquets consecutius). Un cop rebuda tota la ràfega de paquets, el servidor disminueix o augmenta el valor del SR en funció de si s'han perdut paquets o no, respectivament, segons la següent fórmula (les unitats de BW són bps):

$$SR = \begin{cases} \dfrac{(T_{burst} * SR) + Inc_p}{T_{burst}}, & Pèrdues = FALS \\ \dfrac{SR}{1 + 0.125 * \dfrac{SR * 9000 * 8}{BW}}, & Pèrdues = CERT \end{cases}$$

El valor de l'increment de paquets Inc_p [paquets] ve determinat per:

$$Inc_p = \max\left(\frac{1}{9000}, 10^{\log(BW-(SR*9000*8))-M}\right)$$

On el valor M és un corrector de magnitud [adimensional]:

$$M = \begin{cases} 7, & \frac{SR*9000*8}{BW} < 0.9 \\ \left(\frac{SR*9000*8}{BW}*10\right)-2, & \frac{SR*9000*8}{BW} \geq 0.9 \end{cases}$$

Aquest mètode de càlcul provoca una forma de creixement del SR logarítmica, fent que quan la utilització de l'enllaç sigui baixa, l'increment de la velocitat de transmissió sigui major, i viceversa. D'altra banda, quan es detecten pèrdues, com més alta sigui la utilització de l'enllaç, més pronunciada serà la reducció del SR.

El llistat de *gaps* perduts té la següent estructura:

Gap perdut		**Comptador**
# Primer TSN del gap 1	# Darrer TSN del gap 1	Valor comptador gap 1
...	...	...
# Primer TSN del gap N	# Darrer TSN del gap N	Valor comptador gap N

En el moment que un *gap* s'insereix a la llista, el valor del seu comptador s'inicialitza a 1 i se'n sol·licita la retransmissió. En el cas que en la següent ràfega de paquets no es rebin els paquets sol·licitats, el comptador s'incrementa i fins que aquest no arribi a 4 no es tornarà a demanar la retransmissió dels respectius paquets, moment en que es tornarà a reiniciar a 1 el seu comptador.

Un cop fet el càlcul del SR i actualitzada la llista de gaps perduts, el servidor envia un missatge de confirmació (SACK) on adjunta aquestes tres dades, i que haurà de ser respost per part del client amb el seu missatge de confirmació (ACK). En el moment d'enviar la següent ràfega de paquets un cop passats els T_{burst} segons després de l'última transmissió, el client marcarà els paquets que siguin reenviats amb el tercer bit de menor pes del camp de *flags* a 1. Quan el paquet que s'enviï sigui l'últim de tots (valor de TSN més alt), el 4t bit de menor pes del camp *flags* també es posarà a 1, i es reduirà la mida del missatge de 9000 bytes al necessari.

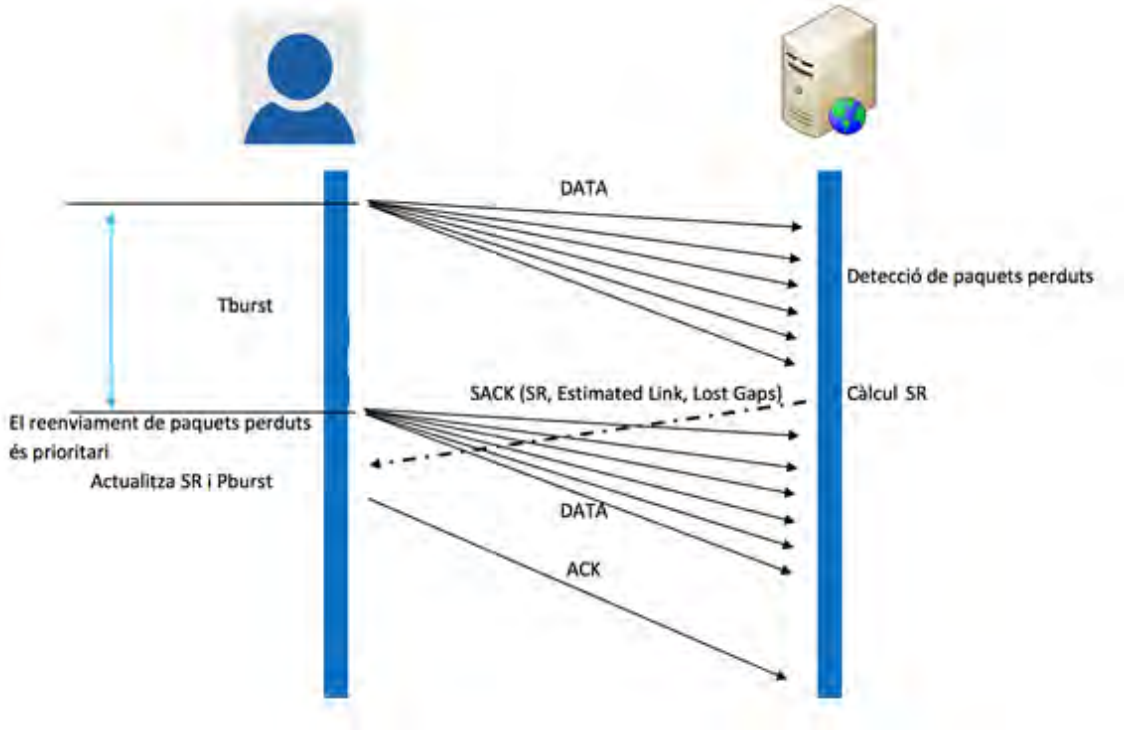

Fig. 29: Diagrama d'intercanvi de missatges en l'intercanvi de dades.

3.1.4 Desconnexió

Un cop acabat l'enviament de dades, el client envia un missatge de petició per finalitzar la comunicació (`SHUTDOWN`). En aquest missatge s'hi insereix l'últim número de seqüència (Transmission Sequence Number o TSN) enviat en la comunicació, ja que en futures implementacions es pretén poder reprendre un intercanvi de dades des del punt on s'hagi deixat si hi ha una desconnexió durant el procés. En rebre el missatge de tancament, el servidor respon amb un de confirmació (`SHUTDOWN-ACK`). Finalment, la sessió acaba quan el client envia el darrer missatge (`SHUTDOWN-COMPLETE`), la recepció del qual implica que el servidor allibera els recursos respectius al client que s'ha desconnectat. A la Fig. 30 es mostra l'intercanvi de missatges d'aquest mecanisme.

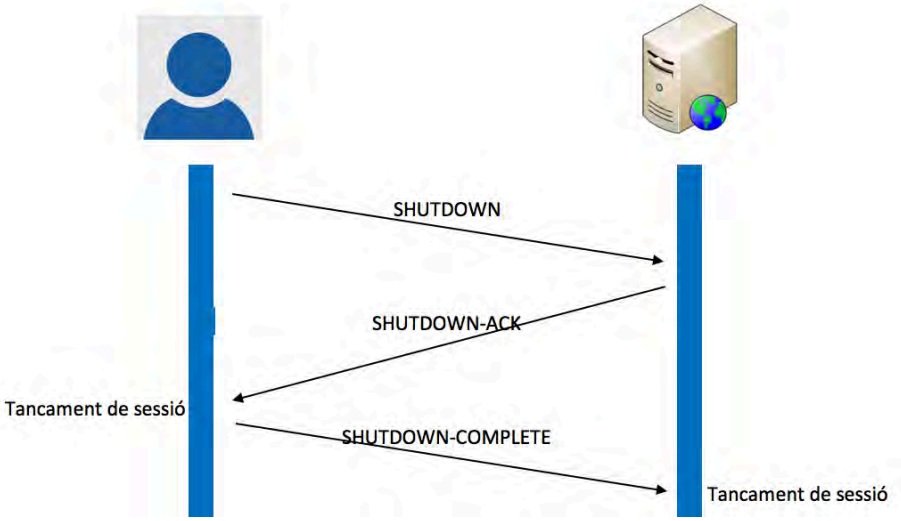

Fig. 30: Diagrama d'intercanvi de missatges en la desconnexió.

3.2 Mecanismes OMBTAP

3.2.1 Modificació de capçaleres i flags OMBTAP

Timestamp - Capçaleres

Els camps associats al *Timestamp* s'incorporen als paquets de `DATA()`, `SACK()` i `ACK()`:

- `DATA()`:
  - Camp `TSVal`: 4 Bytes. Paràmetre de disseny.
- `SACK()`:
  - Camp `TSVal`: 4 Bytes. Paràmetre de dissenyo.
  - Camp `TSEcr`: 4 Bytes. Paràmetre de dissenyo.
  - Camp `Jitter Ratio`: 4 Bytes (Modificació del camp `Jitter`)
- `ACK()`:
  - Camp `TSVal`: 4 Bytes. Paràmetre de disseny.
  - Camp `TSEcr`: 4 Bytes. Paràmetre de disseny.

Explicit Congestion Notification - Flags

S'incorporen noves opcions al `Flags` (8 bits) de la capçalera MBTAP per poder utilitzar ECN. Aquetss dos bits segueixen el mateix codi que ECN.

0 0 X X 0 0 0 0

- Llistat de bits del camp `Flags (8 bits)` a les capçaleres OMBTAP (Taula 2):
  - `Start of Burst (SOB)`
  - `End Of Burst (EOB)`
  - `Non-Connected Abort (NCA)`
  - `Non-Connected Shutdown (NCS)`
  - `ECN-Echo (ECE):`
    - Negociació: Indica que s'habilita l'ús d'ECN.
    - Transmissió: Indica que un router intermedi es troba en un estat de congestió o proper a aquest.
  - `Sending Rate Reduced (SRR):` Indica al receptor que se ha modificado el comportamiento en base a la notificación ECE.

-	-	SRR (1 bit)	ECE (1 bit)	NCS (1 bit)	NCA (1 bit)	EOS (1 bit)	SOS (1 bit)

Taula 2: Camp Flags de la capçalera OMBTAP

Aquestes modificacions tenen afectacions al format de diversos missatges de l'OMBTAP (Taula 3).

Negociació	Missatge (*type*)	Transmissió	Missatge (*type*)
`INIT()`	***0x02***	`DATA()`	***0x01***
`INIT-ACK()`	***0x03***	`SACK()`	***0x0E***
`COOKIE()`	***0x04***	`ACK()`	***0x0F***
`COOKIE-ECHO()`	***0x05***		

Taula 3: Llistat de missatges que incorporen el camp de flags

L'estructura de les capçaleres, incloses les modificacions introduïdes, es detallen a l'Annex II: Format dels missatges OMBTAP.

3.2.2 Funcionament Jitter Ratio - OMBTAP

Un cop introduïts els nous camps (`TSVal, TSecr, Jitter Ratio`) a les capçaleres i especificades les noves opcions del camp `Flags` de la capçalera OMBTAP, es mostra com es realitza el càlcul del *Jitter Ratio* i el RTT, tal i com s'observa a la Fig. 31. Aquesta figura exemplifica l'enviament d'una ràfega del protocol OMBTAP i mostra els valors temporals que es prenen per exemplificar els càlculs. Aquest mètode no requereix de sincronització entre nodes finals, ja que la diferència temporal calculada és a nivell de cada node en particular.

Un cop establerta la comunicació, (1) l'emissor envia una ràfega al receptor de missatges `DATA()`. Els paquets enviats dins la ràfega van marcats amb el moment en que han sigut enviats (`TSVal`). (2) El receptor registra el moment en que rep cada paquet que forma part de la ràfega (inici de la ràfega marcat al camp `Flags = 00XX0001`). (3) Un cop es rep l'últim paquet de la ràfega (final de la ràfega marcat al campo `Flags = 00XX0010`), calcula el Jitter Ratio (Jr) a partit de la *Jitter Window*.

$$Jr = \frac{(t_{rx\acute{U}ltimPkt} - t_{rxPrimerPkt}) - (t_{tx\acute{U}ltimoPkt} - t_{txPrimerPkt})}{(t_{rx\acute{U}ltimPkt} - t_{rxPrimerPkt})} = \frac{(z - x) - (c - a)}{(z - x)}$$

Quan l'emissor rep l'últim paquet de la ràfega i ha realitzat tots els càlculs associats (inclòs el *Jitter* Ratio), (4) envia un missatge `SACK()` a l'emissor. Aquest rep el missatge, poden calcular així el RTT. De forma similar, l'emissor (5) envia un missatge `ACK()` per confirmar la recepció del `SACK()`, podent calcular el receptor el RTT (6).

$$RTT_{Sender} = t_{rxSACK} - t_{tx\acute{U}ltimPkt} = d - c$$

$$RTT_{Receiver} = t_{rxACK} - t_{txSACK} = w - v$$

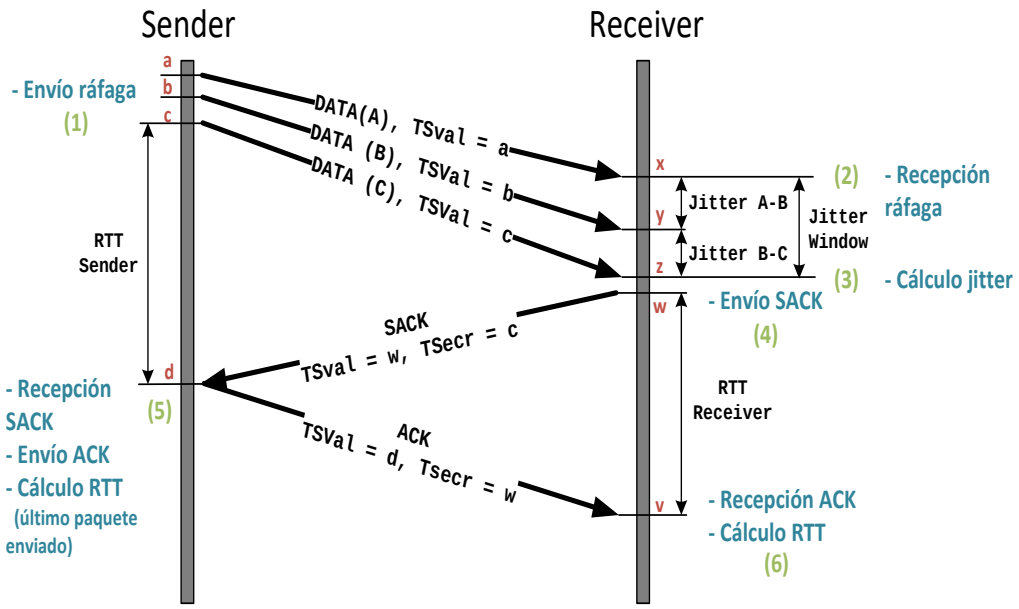

Fig. 31: Càlcul Jitter Ratio de l'OMBTAP

3.2.3 Funcionament Explicit Congestion Notification (ECN) – OMBTAP

Les modificacions per incorporar l'ECN al protocol OMBTAP consta de dues parts diferenciades. La primera part esdevé durant l'establiment de la connexió (Negociació), i la segona durant l'enviament de dades (Transmissió).

Negociació

Durant el període d'establiment de la connexió (Fig. 32), l'emissor indica si utilitzarà ECN o no durant la comunicació. Per a tal finalitat, quan en via el missatge `INIT()` marca el camp `Flags` amb el bit `ECE` actiu (000**1**0000). El receptor, respon amb un `INIT-ACK()`, marcant el mateix bit del camp `Flags` (000**1**0000) per indicar que pot utilitzar ECN.

El procés de marcar el camp `Flags` es repeteix tant en el (3) retorn de la cookie per part de l'emissor amb el missatge `COOKIE-ECHO()`i a la (4) confirmació del receptor amb el missatge `COOKIE-ACK()`.

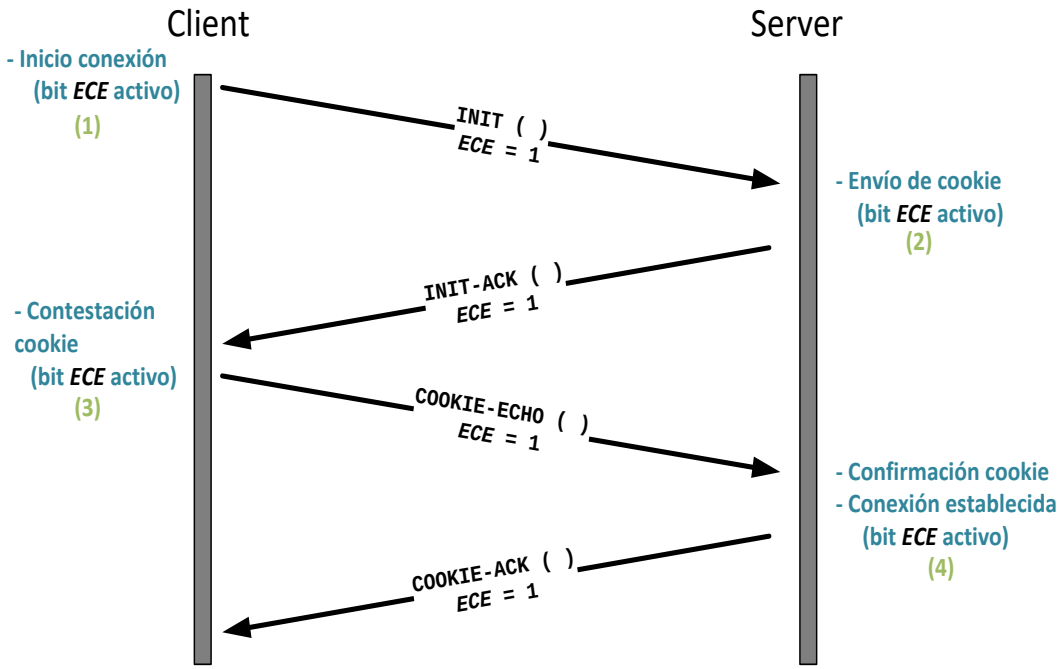

Fig. 32: Establiment de connexió – Negociació ECN

Transmissió

Un cop establerta la connexió i iniciada la transferència entre emissor i receptor, s'obté, la informació sobre la saturació dels nodes intermedis si aquests tenen implementat i configurat ECN.

A la Fig. 33 es mostra l'enviament d'una ràfega a través d'una xarxa congestionada. S'observa com l'emissor (1) envia una rafega al receptor. Durant el procés de transmissió, (2) un dels routers intermedis marca, mitjançant, ECN, el camp CE a 1 (a la capçalera IP), fet que significa l'existència de congestió o que aquesta està a punt de produir-se. Un cop el missatge arriba al receptor, (3) aquest interpreta la informació a nivell IP.

Degut a que s'ha identificat congestió a la connexió (bit CE marcat), (4) el receptor informa a l'emissor mitjançant l'enviament del missatge SACK( ), marcant el bit ECE (camp Flags). (5) Una cop rebut el SACK( ), l'emissor modifica el Sending Rate de l'enviament (depenent de si ha hagut pèrdues) i ho indica mitjançant l'activació del bit *SRR* (camp Flags) al missatge de ACK( ). (6) El receptor rep aquesta informació per saber que s'ha actuat en conseqüència.

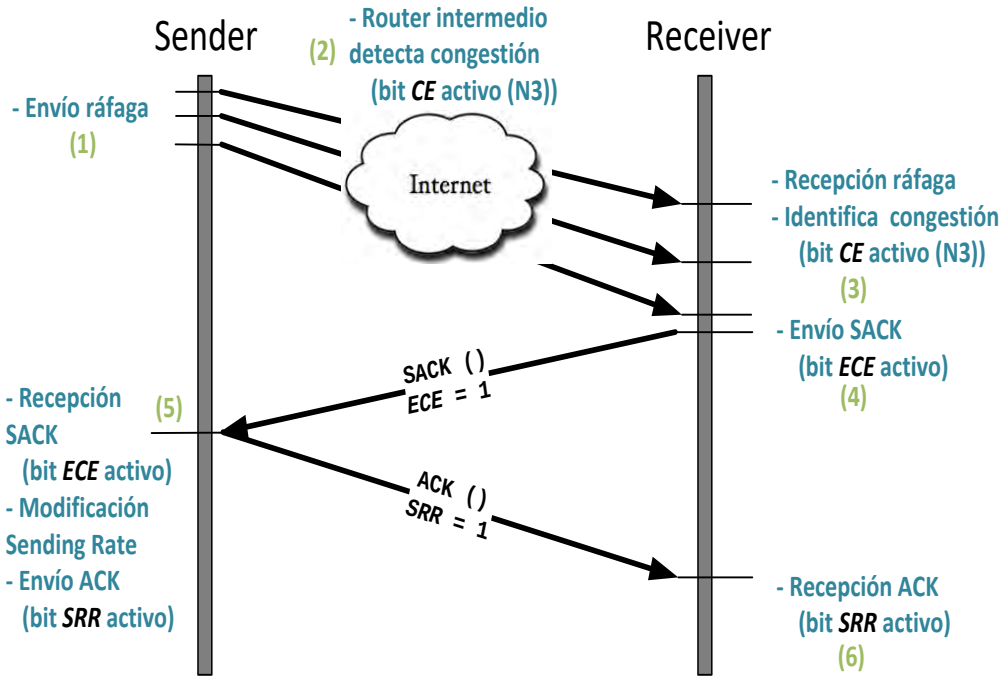

Fig. 33: Intercanvi de dades – Transmissió ECN

3.3 Lògica del control de congestió del protocol OMBTAP

El *Sending Rate* (SR) depèn del valor de las mètriques que marquen el comportament del protocol (ECN, *Jitter Ratio* i les pèrdues en l'enviament de la ràfega anterior). El SR augmenta, disminueix o es manté en funció d'aquests valors. La següent taula (Taula 4) mostra la intel·ligència del protocol OMBTAP.

Estat	Sense pèrdues		Procés	Accions
$E0$	$ECE = 0$	$Jr \leq \gamma$	$SR \uparrow$	$SR = \frac{(T_{burst} * SR) + Inc_p}{T_{burst}}$ $Inc_p = \max(\frac{1}{MTU}, 10^{\log(BW-(SR*MTU*8))-\mathbf{M}})$
$E1$	$ECE = 0$	$Jr > \gamma$	$SR \nearrow$	$SR = \frac{(T_{burst} * SR) + Inc_p}{T_{burst}}$ $Inc_p = \max(\frac{1}{MTU}, 10^{\log(BW-(SR*MTU*8))-\mathbf{9}})$
$E2$	$ECE = 1$	$Jr \leq \gamma$	$SR \equiv$	$SR = \frac{(T_{burst} * SR) + Inc_p}{T_{burst}}$ $Inc_p = 0$
$E3$	$ECE = 1$	$Jr > \gamma$	$SR \equiv$	$SR = \frac{(T_{burst} * SR) + Inc_p}{T_{burst}}$ $Inc_p = 0$

Estat	Amb pèrdues		Procés	Accions
$E4$	$ECE = 0$	$Jr \leq \gamma$	*Canal*	• *Pèrdues per **CANAL*** • *Demanar paquets perduts*
			$SR \equiv$	$SR = \frac{(T_{burst} * SR) + Inc_p}{T_{burst}}$ $Inc_p = 0$
$E5$	$ECE = 0$	$Jr > \gamma$	*Congestió*	• *Pèrdues per **CONGESTIÓN*** • *Demanar paquets perduts (SRR=1)*
			$SR \downarrow$	$SR = \frac{SR}{1 + 0.125 * \frac{SR * MTU * 8}{BW}}$
$E6$	$ECE = 1$	$Jr \leq \gamma$	*Congestió*	• *Pèrdues per **CONGESTIÓN*** • *Demanar paquets perduts (SRR=1)*
			$SR \downarrow$	$SR = \frac{SR}{1 + 0.125 * \frac{SR * MTU * 8}{BW}}$
$E7$	$ECE = 1$	$Jr > \gamma$	*Congestió*	• *Pèrdues per **CONGESTIÓN*** • *Demanar paquets perduts (SRR=1))*
			$SR \downarrow$	$SR = \frac{SR}{1 + 0.125 * \frac{SR * MTU * 8}{BW}}$

Taula 4: Lògica del control de congestió del protocol MBTAP

Sense pèrdues

Quan no es produeixen pèrdues, el comportament del protocol depèn del càlcul del *Jitter Ratio* i la comparació d'aquest valor amb el llindar γ, així com de la notificació per part dels routers intermedis mitjançant ECN.

- `Estat 0 (E0):` Reflecteix el comportament `normal` del protocol (**Sense pèrdues, sense notificació de congestió i Jitter Ratio per sota del llindar**), en el que augmenta el número de paquets a enviar per ràfega, depenent de l'eficiència de l'enviament respecte l'ample de banda estimat.

- `Estat 1 (E1):` Reflecteix el comportament `moderat` del protocol (**Sense pèrdues, sense notificació de congestió i Jitter Ratio per sobre del llindar**), en el que augmenta el número de paquets a enviar per ràfega lleugerament.

$$SR = \frac{(T_{burst} * SR) + Inc_p}{T_{burst}}$$

$$Inc_p = \max\left(\frac{1}{MTU}, 10^{\log(BW - (SR * MTU * 8)) - 9}\right)$$

- S'utilitza `C = 9` a la fórmula d'increment de paquets, $Inc_p()$, degut a que s'aproxima a l'eficiència màxima permesa per l'enllaç sense que aquest generi pèrdues per saturació.

- *Estat 2 (E2):* Reflecteix el comportament `conservador` del protocol (**Sense pèrdues, amb notificació de congestió i Jitter Ratio per sota del llindar**), en què el número de paquets a enviar per ràfega es manté.
  - Això es deu a que els routers intermedis han notificat un estat de congestió i s'ha de prevenir la pèrdua de paquets.

- *Estat 3 (E3):* Reflecteix el comportament `conservador` del protocol (**Sense pèrdues, amb notificació de congestió i Jitter Ratio per sobre del llindar**), en què el número de paquets a enviar per ràfega es manté.
  - Això es deu a que els routers intermedis han notificat un estat de congestió, i amés el *Jitter Ratio* indica que s'estan encuant paquets a les cues dels routers, de manera que és necessari prevenir la pèrdua de paquets.

Amb pèrdues

Quan no es produeixen pèrdues, el comportament del protocol depèn del càlcul del *Jitter Ratio* i la comparació d'aquest valor amb el llindar γ, així com de la notificació de per part dels routers intermedis mitjançant ECN per poder diferenciar si el tipus de pèrdues que s'han produït són degudes al canal o a la congestió.

- *Estat 4 (E4):* Reflecteix el comportament `conservador` del protocol (**Amb pèrdues, sense notificació de congestió i Jitter Ratio per sota del llindar**), en què el número de paquets a enviar per ràfega es manté, ja que s'han produït pèrdues, sol·licitant la retransmissió d'aquests paquets perduts.
  - El J*itter Ratio* indica que no s'aprecia congestió. Tanmateix, el routers intermedis tampoc ho notifiquen amb ECN, associant així les pèrdues al canal.

- *Estat 5 (E5):* Reflecteix el comportament `reductor` del protocol (**Amb pèrdues, sense notificació de congestió i Jitter Ratio per sobre del llindar**), en què el número de paquets a enviar per ràfega es redueix, ja que s'han produït pèrdues, sol·licitant la retransmissió d'aquests paquets perduts.
  - *El Jitter Ratio* indica que existeix congestió, tot i que no hi hagi cap notificació explícita amb ECN (és possible que el router congestionat no tingui habilitat ECN), associant aquestes pèrdues a la CONGESTIÓ.

- *Estat 6 (E6):* Reflecteix el comportament `reductor` del protocol (**Amb pèrdues, amb notificació de congestió i Jitter Ratio per sota del llindar**), en

què el número de paquets a enviar per ràfega es redueix, ja que s'han produït pèrdues, sol·licitant la retransmissió d'aquests paquets perduts.

- Existeix un marcatge ECE, tot i que el *Jitter Ratio* encara no detecti congestió. S'associen les pèrdues a la congestió, ja que els routers intermedis així ho han detectat/notificat.

- *Estat 7 (E7):* Reflecteix el comportament `reductor` del protocol (**Amb pèrdues, amb notificació de congestió i Jitter Ratio per sota del llindar**), en què el número de paquets a enviar per ràfega es redueix, ja que s'han produït pèrdues, sol·licitant la retransmissió d'aquests paquets perduts.
  - Existeix un marcatge ECE, i amés el *Jitter Ratio* indica que hi congestió, de manera que s'associen aquestes pèrdues a la congestió.

Cal remarcar que en el cas que no s'utilitzi ECN, ja sigui perquè no s'habilita als nodes OMBTAP o perquè els routers intermedis no ho suporten, els únics estats possibles són *E0, E1, E4* i *E5.*

3.4 Lògica del control de congestió del protocol OMBTAP

3.4.1 Mida del camp TSVal i TSEcr

La mida dels camps `TSVal` i `TSEcr`, té una influència directa a l'eficiència del protocol OMBTAP degut a que provoca un augment de la capçalera. El valor d'aquests camps és de milisegons. Segons l'estàndard Timestamp [76] la mida indicada és de 4 bytes.

El fet que la mida sigui de 4 bytes, implica que durant la transmissió han de passar 1193 hores abans que el comptador s'hagi de reiniciar a 0.

$$4\ bytes; 32\ bits$$

$$2^{32}\ ms = 4294967296\ ms$$

$$En\ hores \rightarrow 1193\ hores$$

En canvi, si no se segueix l'estàndard i s'utilitzen 3 bytes, es passa a tenir un marge temporal de 4,66 hores.

$$3\ bytes; 24\ bits$$

$$2^{24}\ ms = 16777216\ ms$$

$$En\ hores \rightarrow 4{,}66\ hores$$

Finalment, si es planteja una mida de 2 bytes, el marge temporal passa a ser d'1,09 minuts.

$$2\ bytes; 12\ bits$$

$$2^{16}\ ms = 65536\ ms$$

$$En\ hores \rightarrow 0{,}018\ hores$$

Bytes	Horas	Milisegundos
2	0,018	65536
3	4,66	16777216
4	1193	4294967296

Taula 5: Marge temporal respecte la mida dels camps TSVal i TSEcr.

La mida dels camps `TSVal` i `TSEcr` s'estableix com un paràmetre de disseny, seguint la recomanació que com més petit sigui la seva mida (veure Taula 5), més dades útils poden ser enviades per paquets. En aquest sentit, es proposa utilitzar una mida de **3 bytes**, ja que d'aquesta manera s'aconsegueix un equilibri.

3.4.2 Llindar de decisió (ɣ)

Un dels paràmetre més destacats i rellevants del protocol OMBTAP és el llindar de decisió (ɣ) o *Loss Threshold Decisor* (LTD). El valor de ɣ marca el límit entre el tipus de pèrdues que es generen. Depenent del *Jitter Ratio* calculat i de la seva comparació amb el valor ɣ, es permet discernir entre pèrdues generades per congestió i pèrdues generades pel anal de forma aleatòria.

El valor del *Jitter Ratio* indica el rati entre de paquets encuats, relacionant l'efecte dels paquets en cua d'un router intermedi (*bottleneck*) i el *delay* entre paquets a la recepció. Basant-nos en aquest concepte, si en una iteració no s'han produït pèrdues, implícitament significa que el número de paquets que s'han afegit a la nova ràfega respecte l'anterior no està saturant l'enllaç.

Es per això que el llindar LTD ha de marcar el número de paquets encuats que es considera que marquen un indici de congestió, generada en injectar tots els paquets de la ràfega a l'enllaç. En cas de superar aquest llindar, existirà saturació de l'enllaç.

A JTCP [59], [77], el concepte utilitzat per la diferenciació entre pèrdues és similar al plantejar pel protocol OMBTAP. Com que està basat en TCP, defineix el llindar com l'increment d'un paquet per iteració.

$$LTD_{JTCP} = \frac{k}{Congestion\ Window} = \frac{1}{Congestion\ Window}$$

De forma anàloga, es proposa pel protocol OMBTAP que el número de paquets encuats que es consideren com indici de congestió sigui el número de paquets incrementats respecte el total d'enviats a la ràfega anterior. El fonament principal per marcar aquest llindar és que seran els nous paquets afegits a la ràfega els que saturaran l'enllaç.

$$LTD_{OMBTAP} = \gamma = \frac{\#\ Paquets\ incrementats}{\#\ Paquets\ ràfega}$$

3.4.3 Smooth Jitter Ratio

Una proposta de millora del protocol SCTP [60] planteja l'ús del *Smooth Jitter Ratio* per evitar casos en que es produeix el càlcul d'un valor del *Jitter Ratio* igual a 0.

Demostrada la seva utilitat, es proposa adoptar aquesta mètrica també al protocol OMBTAP.

$$smooth_{jr} = (1 - \alpha) \cdot smooth_{jr} + \alpha \cdot Jr$$

On α adquireix un valor arbitrari. Depenent d'aquest valor, tindrà més pes el valor calculat a la iteració actual, o l'històric de les iteracions anteriors. En el cas del protocol OMBTAP, el valor escollit per α és 0,05, tal i com JSCTP proposa empíricament [60].

Així doncs, el valor que el control de congestió del protocol OMBTAP utilitzarà per comparar-lo amb el LTD i discernir entre pèrdues per congestió i pèrdues per canal no serà el *Jitter Ratio*, sinó el *Smooth Jitter Ratio*.

4 Procés d'implementació del protocol OMBTAP

4.1 Introducció al programa Riverbed Modeler

Riverbed Modeler[5] és un programa que permet dissenyar escenaris de xarxes de dades de qualsevol mida i tipus. A partir d'aquest escenari, es poden fer simulacions sobre el comportament d'aquest escenari sota una gran diversitat de característiques parametritzables. Riverbed Modeler (o Modeler) compta, dins la seva paleta d'objectes, amb tot tipus d'equipament com routers, switchos, servidors, ordinadors, enllaços (links), etc.; els quals contenen dins seu diversos processos que implementen els protocols més importants i rellevants de la torre TCP/IP. Modeler també permet dissenyar i programar dispositius i processos propis amb l'objectiu de poder simular la seva integració dins un escenari real. Cal recordar que a [75] s'explica com instal·lar i muntar l'entorn per poder treballar amb el simulador de forma satisfactòria en Windows 7.

Les simulacions de Modeler es basen en el concepte d'esdeveniments discrets i màquines d'estats finites. Això significa que cada objecte de l'escenari conté diversos processos, cadascun dels quals simula una de les funcionalitats de la torre TCP/IP dins d'aquest. Aquests processos son programats com una màquina d'estats finita amb llenguatge C o C++, on cada estat s'encarrega d'implementar una part específica de la funcionalitat d'aquell procés.

El *kernel* del simulador és l'encarregat de gestionar l'ordre en que s'executen cadascun dels estats continguts en els processos de tots els objectes de l'escenari simulat. La decisió de l'ordre d'execució es pren a partir d'una cua d'interrupcions que van activant aquests estats. Aquestes interrupcions es poden generar a partir de la recepció d'un paquet, de la invocació per part d'un altre procés o de la invocació d'un procés contra ell mateix. D'aquesta manera, tots els processos que s'implementen a Modeler han de tenir un o més estats on aquest es queda en mode repòs (espera) fins a rebre una interrupció, alliberant així el *kernel* per poder executar un altre procés. Modeler compta amb diversos tipus d'editors per tal de poder implementar els processos, els objectes i els escenaris que es volen crear d'una forma més jeràrquica i intuïtiva:

- Editor de Projecte
- Editor de Procés
- Editor de Node
- Editor de Paquets

4.1.1 Editor de Projecte

L'Editor de Projecte (o *Project Editor*) és l'editor principal del programa i que s'utilitza per a qualsevol tipus de simulació. A la Fig. 34 es mostra un projecte d'exemple. Aquest editor permet:

- Dissenyar un model (escenari) de xarxa.
- Configurar la topologia física de l'escenari.

[5] : Riverbed Modeler és un software simulador de xarxa. Tota la informació a: https://www.riverbed.com/es/products/steelcentral/steelcentral-riverbed-modeler.html

- Configurar els paràmetres de tràfic a generar.
- Configurar les característiques de la simulació, executar-la i visualitzar-ne els resultats.

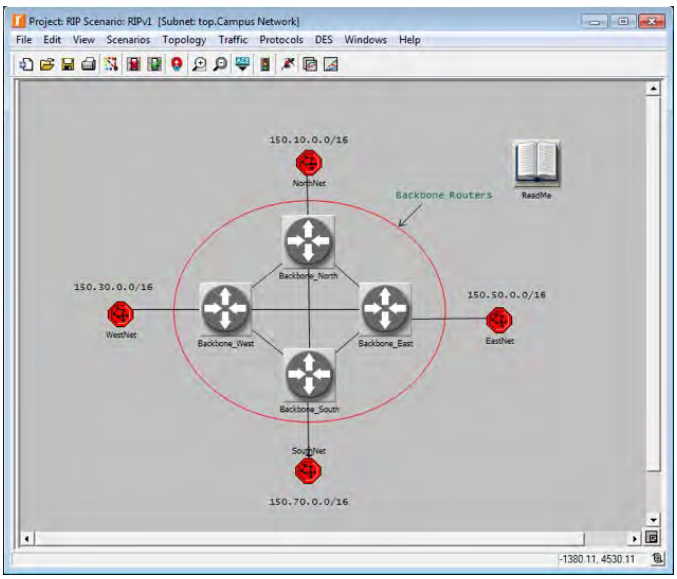

Fig. 34: Project Editor.

4.1.2 Editor de Node

L'Editor de Node (o *Node Editor*) és l'editor on es pot crear i editar l'estructura interna d'un dispositiu o node, és a dir, els processos pels quals està format i la relació entre ells. Aquestes relacions queden representades mitjançant els *streams*, que tenen com a funció transportar els paquets d'un procés cap a un altre. Aquests *streams* són unidireccionals, de manera que el més habitual és que la relació entre dos processos s'implementi mitjançant un parell de *streams*, un per cada sentit de la comunicació. A mode d'exemple, a la Fig. 35. es mostra l'estructura del node "*workstation*" (un ordinador corrent) dins del Node Editor.

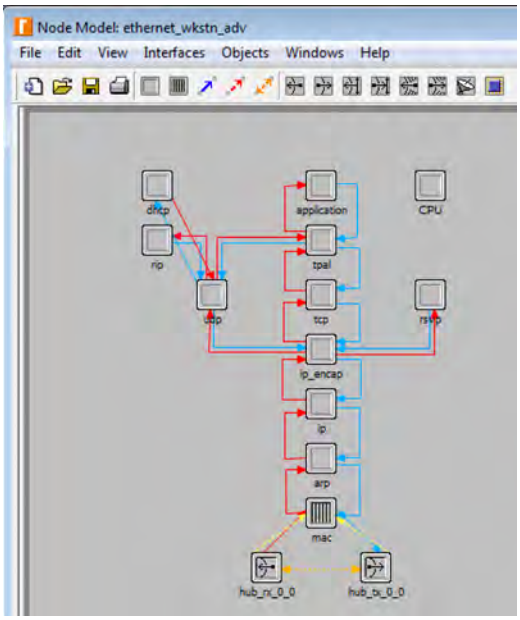

Fig. 35: Node Editor. Estructura del node "*workstation*".

4.1.3 Editor de Procés

L'Editor de Procés (o *Process Editor*) és on precisament es permet escriure el codi per tal d'implementar els algoritmes i protocols propis, així com definir la màquina d'estats que el procés ha de seguir. El llenguatge de programació que s'utilitza s'anomena Proto-C, que no deixa de ser una subcategoria dins el llenguatge genèric C/C++ adaptat a la programació amb màquines d'estats finites.

Els estats d'aquesta màquina tenen dues parts diferenciades per a l'execució del seu codi: la part d'entrada i la part de sortida. Aquesta divisió es deu a que existeixen dos tipus d'estats: els estats forçats (de color verd) i els estats no forçats (de color vermell).

Els estats forçats executen el codi de les parts d'entrada i sortida consecutivament, i a continuació transicionen cap un altre estat determinat. Per conveni, tot el codi d'aquest tipus d'estats s'escriuen únicament a la part d'entrada, ja que no té sentit tenir-lo dividit en dues parts si s'executen consecutivament. En canvi, els estats no forçats només executen el codi de la part d'entrada quan es realitza una transició cap aquest, i a continuació allibera el *kernel* del simulador perquè pugui executar altres processos, quedant així en mode repòs o espera. Quan el *kernel* torna a invocar el procés a causa d'una interrupció, s'executa la part de sortida i es transiciona cap a un altre estat.

Respecte aquestes transicions, des d'un estat en poden sortir una o diverses cap a altres estats. És necessari que quan n'hi hagi més d'una, aquestes estiguin condicionades, ja que només una d'elles pot ser avaluada com a certa en el moment de la transició. Les transicions no condicionades són marcades amb una fletxa amb línia contínua, mentre que les condicionades són marcades amb una fletxa amb línia discontínua.

A més, la transició cap a l'estat inicial del procés (inicialització) es marca amb una fletxa més gruixuda, indicant així que aquesta transició només es duu a terme quan el *kernel* del simulador llença la interrupció inicial (*Begsim Interruption*) cap el procés. A la Fig. 36 es mostra l'Editor de Procés amb la màquina del procés TCP per als nodes "*workstation*".

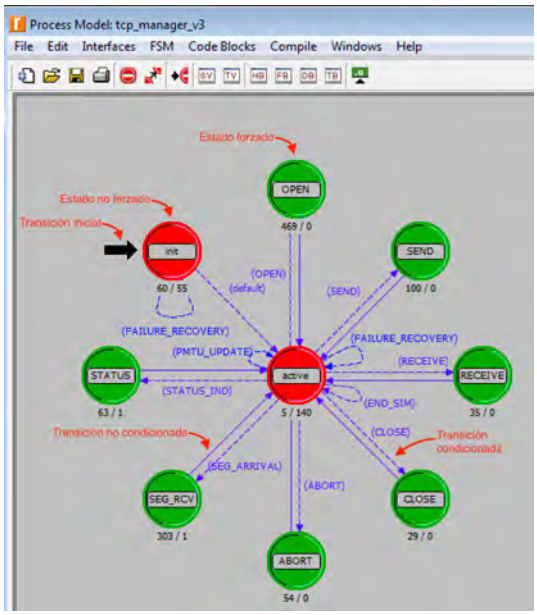

Fig. 36: Process Editor. Màquina d'estats del procés TCP.

A més dels estats, hi ha altres blocs destinats a introduir codi, els quals són:

- ***State Variables Block:*** En aquesta secció es declaren les variables que mantenen el seu valor durant tot el procés, és a dir, que quan hi ha una transició entre estats no es perd el seu valor.
- ***Temporary Variables Block:*** En aquesta secció es declaren les variables que només mantenen el seu valor dins d'un mateix estat, però no entre estats.
- ***Header Block:*** En aquesta secció s'hi introdueixen les llibreries que requereix el procés i es defineixen les macros que s'utilitzaran per les condicions de transició entre estats, les constants i les estructures i tipus propis.
- ***Function Block:*** En aquesta secció es defineixen totes les funcions pròpies que poden ser cridades des del codi dels estats.
- ***Diagnostig Block:*** En aquesta secció s'hi introdueixen sentències per al diagnòstic d'errors i és rarament utilitzat.

4.1.4 Editor de Paquet

L'Editor de Paquet (o *Packet Editor*) es destina a definir el format dels paquets que es volen crear de manera que, un cop definits, des del codi d'un procés es poden crear paquets d'un tipus determinat sense haver de definir el seu format cada vegada. D'aquesta manera, per crear un paquet d'un tipus determinat des del codi del procés només caldrà cridar una de les funcions de la llibreria de Riverbed que apareix a l'Annex I: Glossari de funcions pròpies de Riberbed Modeler utilitzades. Per cada camp d'un paquet es permet definir el seu nom, el tipus de dades que transmet (int, float, char, etc), la codificació (i.e. si el camp és de tipus enter, es pot definir si és amb signe o sense signe i si l'ordre dels bits és *Big Endian* o *Little Endian*), la mida en bits i el seu valor per defecte, entre d'altres. A la Fig. 37 es mostra l'Editor de Paquet.

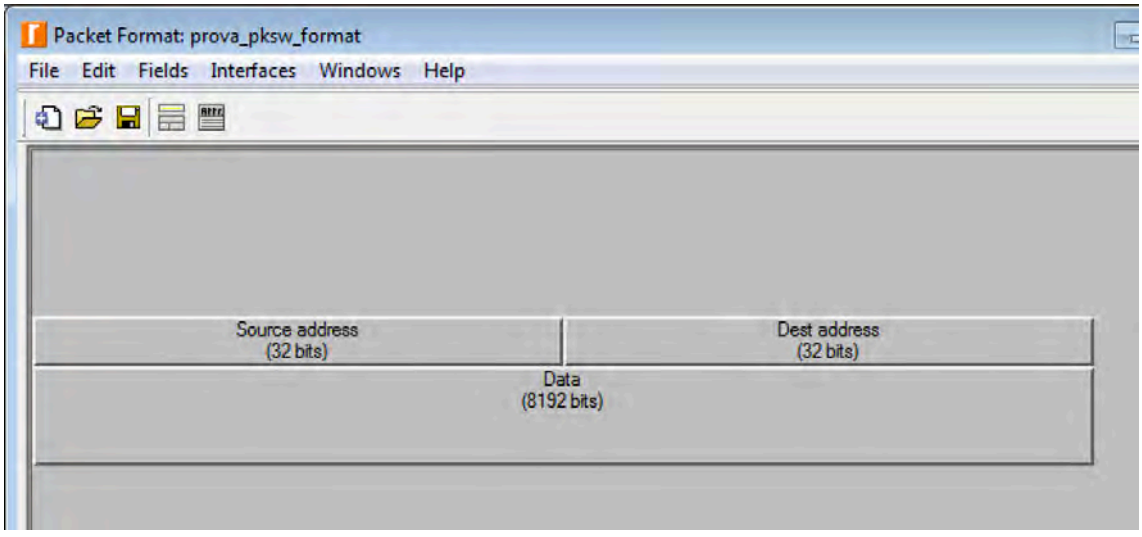

Fig. 37: Packet Editor. Definició d'un paquet de prova.

4.2 Implementació i programació del protocol OMBTAP a Riverbed Modeler

4.2.1 Definició dels paquets

El primer pas del procés d'implementació és definir els paquets (missatges) del protocol OMBTAP utilitzant l'Editor de Node, i així poder crear aquests paquets propis dins del codi del procés. A la Fig. 38 es mostra el paquet DATA_O un cop creat al Packet Editor. Alguns dels paquets que s'havien creat per la implementació del protocol MBTAP no han de ser modificats, i per tants es reaprofiten. Aquests són:

- INIT
- INIT-ACK (amb la *cookie* inclosa)
- COOKIE-ECHO (amb la *cookie* inclosa)
- COOKIE-ACK
- BW
- SHUTDOWN
- SHUTDOWN-ACK
- SHUTDOWN-COMPLETE

En canvi, n'hi ha d'altres que sí s'han creat de nou, ja que el seu format ha variat, ja sigui per l'addició de nous camps a les capçaleres o per la modificació de la mida d'aquests. Els nous paquets definits són:

- DATA_O
- ACK_O
- SACK_O

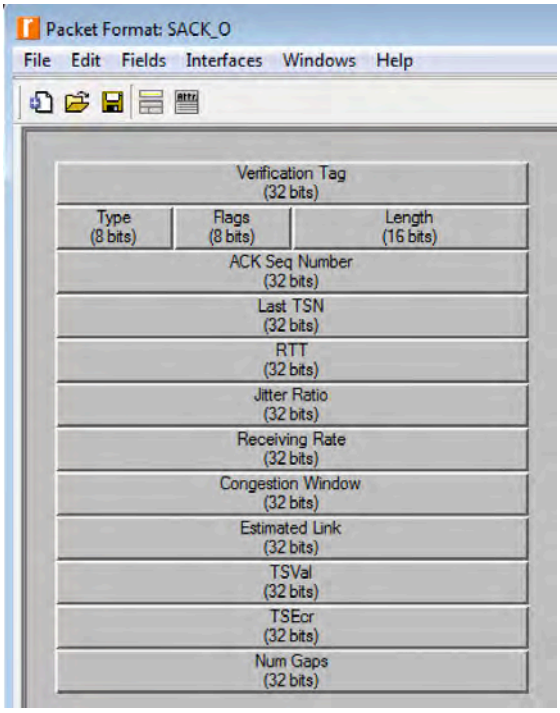

Fig. 38: Missatge DATA_O del protocol OMBTAP creat dins el Packet Editor de Riverbed Modeler.

4.2.2 Creació del node *OMBTAP Workstation*

El següent pas és crear un node que incorpori el protocol OMBTAP i l'integri dins del seu *stack* de protocols. Per aquest motiu, s'ha pres com a base l'estructura del node "wlan_*workstation*" i s'ha col·locat el procés OMBTAP una capa per sobre d'UDP, el qual també interactua amb altres processos superiors, tal i com es mostra a la Fig. 39. La connexió amb el procés UDP es realitza mitjançant un *stream* de pujada i un altre de baixada.

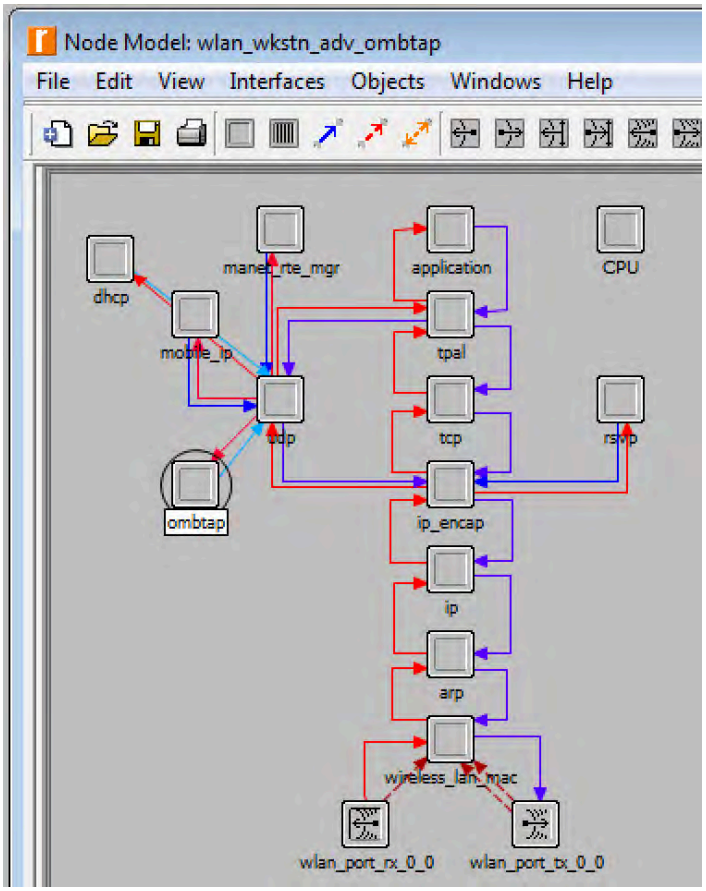

Fig. 39: Estructura del node "OMBTAP WLAN Workstation".

Amb aquesta estructura, les dades que s'envien mitjançant el protocol OMBTAP s'han de generar des del propi procés. Una altra opció hagués estat connectar el procés del protocol OMBTAP amb el procés d'aplicacions i generar des d'allà les dades. A causa que l'objectiu del treball i de les simulacions a realitzar és l'enviament de les dades i no pas el contingut i posterior tractament d'aquestes, s'ha optat per l'opció més eficient i senzilla, que és generar les dades que s'han d'enviar des del propi procés OMBTAP, tal i com ja es va fer anteriorment per la implementació del MBTAP.

4.2.3 Disseny de la màquina d'estats

Per la implementació del protocol a Riverbed s'ha optat pel disseny d'una sola màquina d'estats finita tant pel client com pel servidor. D'aquesta manera, un mateix node pot realitzar qualsevol de les dues funcions segons convingui tal i com s'espera d'un protocol de transport. Per aquest motiu, és necessari que el procés sàpiga si ha d'actuar com a

emissor o receptor a l'hora d'establir la comunicació. Per aconseguir aquest propòsit cal definir un *Model Attribute*, que és un paràmetre del procés que es pot configurar des de l'Editor de Projecte i així modificar les característiques i/o el comportament d'un node. Així doncs, per al procés OMBTAP s'ha definit un atribut a mode de *flag* que indiqui si el node ha d'actuar com a client (emissor) o servidor (receptor). A més d'aquest atribut, també se'n defineixen altres que indiquen la mida de les dades a enviar i el temps que es tarda des de l'inici de la simulació en establir la connexió (veure Fig. 40).

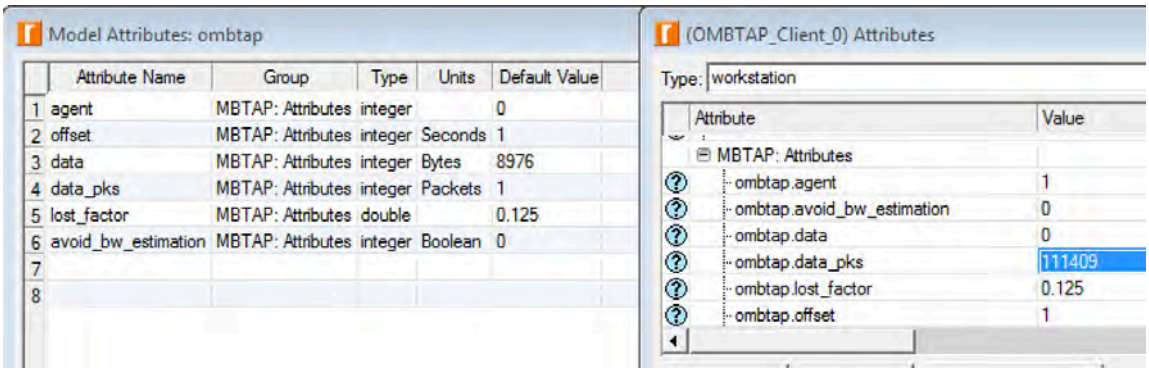

Fig. 40: Definició d'atributs pel protocol OMBTAP. A l'esquerra, la finestra de definició dels atributs del Process Editor. A la dreta, la finestra d'edició dels valors dels atributs en el Project Editor.

Cada estat s'encarrega d'una part diferenciada del protocol, i l'estructura d'aquesta màquina es pot observar a la Fig. 41. Aquesta màquina d'estats manté la mateixa estructura que el protocol MBTAP, tot i que en aquest cas s'han afegit els nous mecanismes comentats dins del procés de l'estat corresponent.

Cal esmentar que en aquesta implementació no s'han programat els estats del control de congestió que contemplen l'ús d'ECN (E2, E3, E6 i E7; veure Taula 4), ja que els nodes intermitjos (routers, switchos i APs) amb els quals s'ha treballat no contemplen aquesta opció. Així doncs, l'únic factor de decisió que determinarà si les pèrdues són degudes a la congestió de l'enllaç o a problemes aleatoris del canal serà el *Smooth Jitter Ratio*.

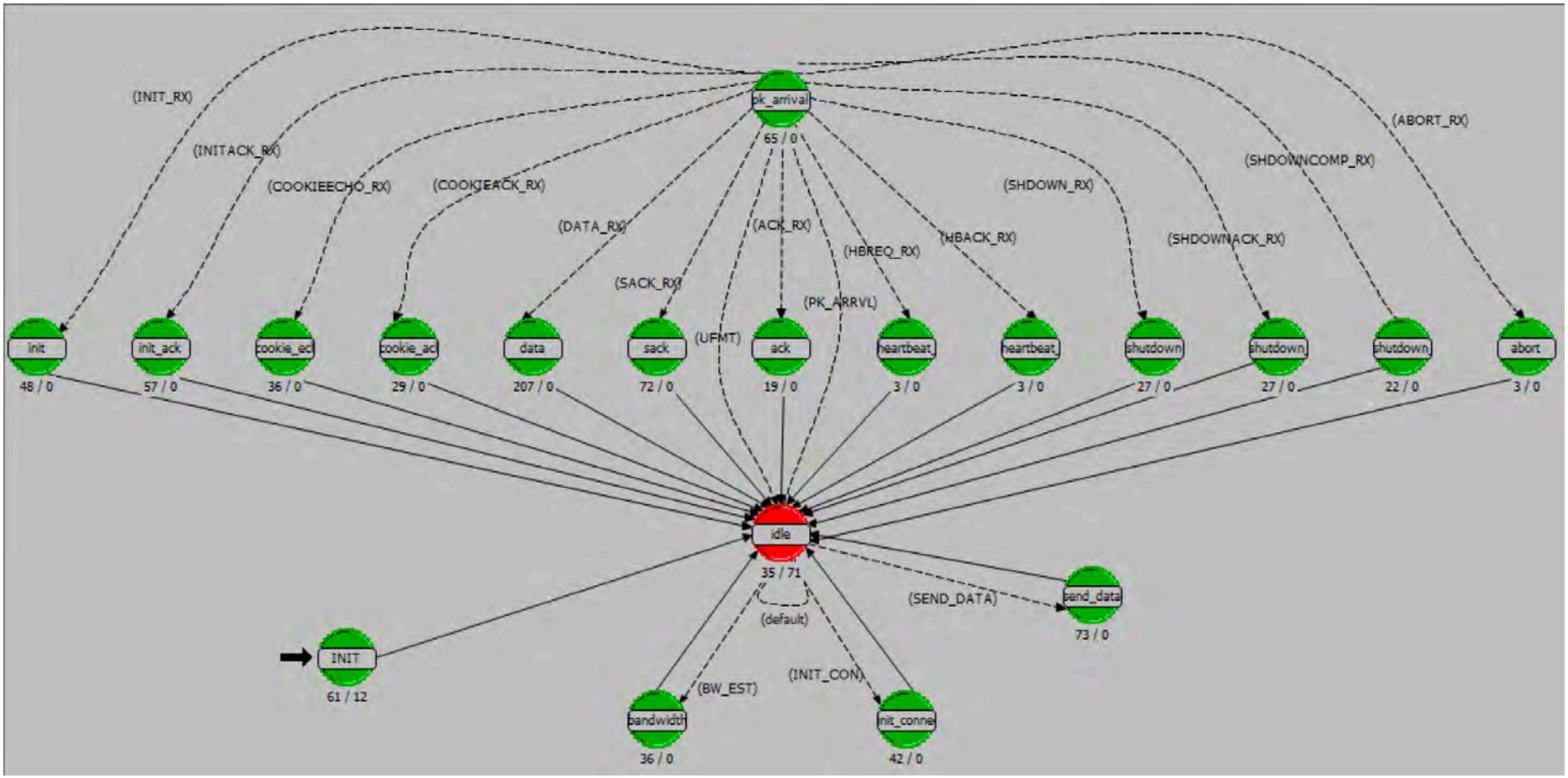

Fig. 41: Màquina d'estats finita del procés OMBTAP.

La funcionalitat de cada estat es descriu a continuació:

- **Estat “INIT” <client/servidor>**
  - S’inicialitza el procés tant del client com del servidor.
  - Es declaren les estadístiques a generar.
  - Es recuperen els ports i les direccions IP per les quals es realitzarà la connexió, a més de descobrir quins són els mòduls UDP i IP del propi node amb els que el procés s’ha de comunicar.
  - En el cas del client, s’activa un *flag* per indicar que s’ha d’iniciar la comunicació.
- **Estat “idle” <client/servidor>**
  - És l’estat on el procés es queda en espera fins rebre una nova interrupció que l’activi, normalment per l’arribada d’un paquet.
  - Abans de bloquejar-se, comprova si els *flags* d’iniciar la comunicació, començar l’estimació inicial de l’enllaç o transmetre dades està activat (només estarà activat un sol *flag* a la vegada).
  - Si hi ha algun *flag* activat, el procés es llença una interrupció sobre ell mateix per poder fer la transició cap a l’estat pertinent.
  - En cas contrari, el procés es bloqueja fins que l’activi una interrupció deguda a l’arribada d’un paquet. Quan es desbloqueja, s’actualitzen les estadístiques de transmissió dels paquets i es transiciona cap a l’estat d’arribada dels paquets (“pk_arrival”).
- **Estat “pk_arrival” <client/servidor>**
  - S’obté quin és el tipus de paquet (missatge) rebut per saber cap a quin estat cal fer la transició.
  - En el cas que el servidor rebi un paquet un cop ja establerta la connexió amb un client, es recuperen les dades d’aquest que es troben a la llista de clients registrats.
- **Estat “init_connection” <client>**
  - S’envia un missatge `INIT` al servidor.
- **Estat “init” <servidor>**
  - Es rep un missatge `INIT`.
  - Es respon amb un missatge `INIT-ACK` que conté la *State Cookie*.
- **Estat “init_ack” <client>**
  - Es rep un missatge `INIT-ACK`.
  - Es respon amb un missatge `COOKIE-ECHO` que conté la còpia de la *State Cookie.*
- **Estat “cookie_echo” <servidor>**
  - Es rep un missatge `COOKIE-ECHO`.
  - Si la *cookie* és correcta, es reserven els recursos per aquell client, se’l registra a la llista de clients i es respon amb un missatge `COOKIE-ACK`. S’activa el *flag* per començar l’estimació inicial de l’ample de banda de l’enllaç.

- En cas d'error, es respon amb un missatge ABORT.

- **Estat "cookie_ack" <client>**
  - Es rep un missatge COOKIE-ACK.
  - S'activa el *flag* d'estimació inicial.
- **Estat "bandwidth" <client>**
  - S'envia una ràfega de missatges DATA per realitzar l'estimació inicial.
  - Si és l'última ràfega, es desactiva el *flag* d'estimació inicial.
- **Estat "send_data" <client>**
  - S'envien els missatges DATA amb les dades al servidor (una ràfega de mida $Packet_{burst}$ – o menys si és la darrera ràfega de l'enviament-).
  - Abans d'enviar noves dades, comprova si hi ha paquets que s'han de reenviar i, si és així, l'enviament d'aquests té prioritat.
  - L'enviament de la següent ràfega serà al cap de T_{burst} segons després d'haver passat per l'estat "idle" i, possiblement, per l'estat "sack".
- **Estat "data" <servidor>**
  - Es rep un missatge DATA.
  - Si el paquet pertany a l'estimació inicial de l'ample de banda:
    - S'acumula la quantitat de dades rebudes amb les que ja s'hagin rebut en aquella mateixa ràfega. Se segueix el mètode d'estimació explicat a l'apartat 3.1.2.
    - Al final de ràfega, es respon amb un missatge ACK i es calcula l'ample de banda estimat per aquella ràfega.
    - Al final de tota l'estimació, es calcula l'ample de banda estimat amb amb el mètode de l'apartat 3.1.2, així com el Sending Rate. Aquests valors s'envien dins el missatge SACK de resposta.
  - Si el paquet pertany a la transmissió de dades útils:
    - Es segueix un procés similar a l'anterior fins al final de ràfega, detectant els salts de TSN com a paquets perduts, que s'insereixen a la llista amb el comptador a 1.
    - Al final de ràfega, es calculen els nous valors d'Estimated Link i Sending Rate, utilitzant el nou mecanisme de control de congestió que contempla l'ús del *Smooth Jitter Ratio*. Si durant la ràfega no s'ha rebut un paquet de la llista de paquets perduts, se n'incrementa el comptador (sense tenir en compte els inserits en la ràfega actual). Finalment, s'envia un missatge SACK on s'inclouen els nous valors d'Estimated Link i Sending Rate calculats i on es sol·licita el reenviament dels *gaps* de paquets perduts amb el comptador a 1 o a 4.
- **Estat "sack" <client>**
  - Es rep un missatge SACK.
  - S'actualitzen els valors d'Estimated Link i Sending Rate pels rebuts en el missatge, i es recalcula el valor de $Packet_{burst.}$

- S'envia un missatge ACK com a resposta.
- Si ja s'han enviat totes les dades i en els tres últims missatges SACK rebuts no es sol·licita el reenviament de cap paquet, es dóna per acabada la transferència de dades, desactivant el *flag* corresponent i enviant el missatge SHUTDOWN.

- **Estat "ack"<client/servidor>**
  - Es rep un missatge ACK.
  - En el cas del servidor:
    - Es té constància que el client ha rebut correctament el missatge SACK.
    - Si el missatge ACK rebut és la resposta a l'últim SACK del servidor (transferència de dades completada), es prepara per a la desconnexió.
  - En el cas del client:
    - Es té constància que la ràfega de missatges DATA de l'estimació inicial s'ha rebut i s'activa el *flag* per enviar la següent (excepte si és l'última, on s'haurà de preparar per rebre el SACK).
- **Estat "heartbeat_req" <servidor> (no implementat)**
  - Es rep un missatge HEARTBEAT-REQUEST.
  - Es respon amb un missatge HEARTBEAT-ACK per mantenir activa la connexió.
- **Estat "heartbeat_ack" <client> (no implementat)**
  - Es rep un missatge HEARTBEAT-ACK.
  - Es té constància que la connexió segueix activa.
- **Estat "abort" <client/servidor>**
  - Es rep un missatge ABORT per alguna circumstància.
  - S'inicia el procés de desconnexió, el client envia el missatge SHUTDOWN i el servidor espera rebre'l.
- **Estat "shutdown" <servidor>**
  - Es rep un missatge SHUTDOWN.
  - Es respon amb un missatge SHUTDOWN-ACK.
- **Estat "shutdown_ack" <client>**
  - Es rep un missatge SHUTDOWN-ACK.
  - Es respon amb un missatge SHUTDOWN-COMPLETE.
- **Estat "shutdown_complete" <servidor>**
  - Es rep un missatge SHUTDOWN-COMPLETE.
  - S'alliberen els recursos que el client consumia i se l'elimina de la llista de clients registrats.

4.2.4 Generació d'estadístiques

Com ja s'ha comentat, Riverbed Modeler permet generar estadístiques com a resultats de la simulació, i així poder veure quin ha estat el comportament del tràfic generat i del protocol que s'estigui provant. Les estadístiques generades més rellevants en aquest projecte són:

- End-to-End Delay (segons)
- Tràfic rebut (en bps i paquets per segon)
- Sending Rate (en bps i paquets per segon)
- Ample de banda estimat (en bps)
- Detecció de pèrdues (booleà)
- Detecció de pèrdues per congestió (booleà)
- Nombre de paquets perduts (paquets)
- Nombre de paquets DATA_O enviats (paquets)
- Nombre de paquets DATA_O rebuts (paquets)
- *Throughput* de l'enllaç (bps). Aquesta no és pròpia del procés OMBTAP.

L'única nova estadística generada per l'OMBTAP és la de detecció de pèrdues per congestió, ja que la resta ja es recol·lectaven amb el MBTAP. Per aquest motiu, s'han reaprofitat i se n'han mantingut els noms. Per poder generar aquestes estadístiques, primer és necessari definir-les a les finestres "Local Statistics" i "Global Statistics" dins de l'Editor de Procés. A la Fig. 42 es mostra la finestra de "Local Statistics" (la de "Global Statistics" conté la mateixa informació).

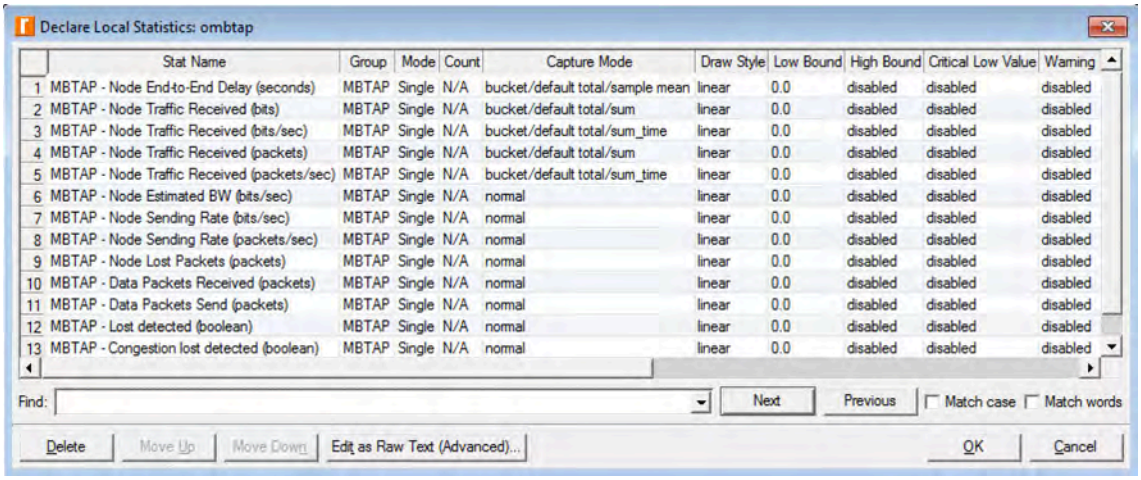

Fig. 42: Finestra de declaració de les "Local Stadistics" dins de l'Editor de Procés.

A continuació, a l'estat d'inicialització del procés s'han de declarar els *handlers* d'aquestes estadístiques, que permetran posteriorment inserir-hi valors, tal i com es mostra a la Fig. 43.

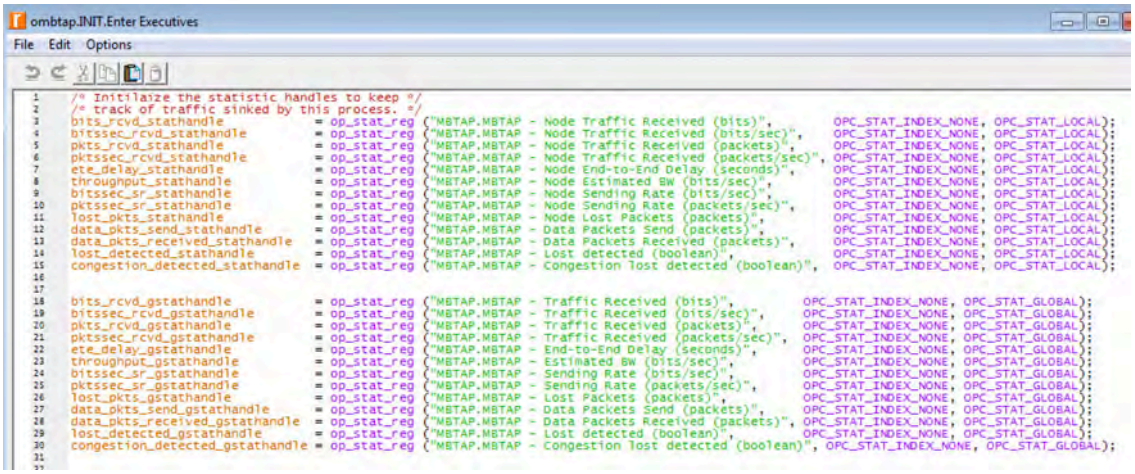

Fig. 43: Declaració dels *handlers* per les estadístiques del protocol OMBTAP.

Finalment, també cal actualitzar les estadístiques quan s'obtenen nous valors. A la Fig. 44 es mostra el moment en que s'actualitzen els valors de les estadístiques de Sending Rate i Estimated Link.

```
op_stat_write(throughput_gstathandle, (estimated_bw*1000000));
op_stat_write(bitssec_sr_gstathandle, (sending_rate*NETSTAT_DATA_SIZE));
op_stat_write(pktssec_sr_gstathandle, sending_rate);

op_stat_write(throughput_stathandle, (estimated_bw*1000000));
op_stat_write(bitssec_sr_stathandle, (sending_rate*NETSTAT_DATA_SIZE));
op_stat_write(pktssec_sr_stathandle, sending_rate);
```

Fig. 44: Actualització dels valors de les estadístiques del procés OMBTAP.

Per tal que les estadístiques locals es puguin visualitzar després d'una simulació, primer cal entrar dins de l'Editor de Node i obrir la finestra "Node Stadistics". En aquesta, cal prémer "Selecet Promoted Statistics..." i seleccionar quines estadístiques es vol que puguin ser visualitzades (veure Fig. 45).

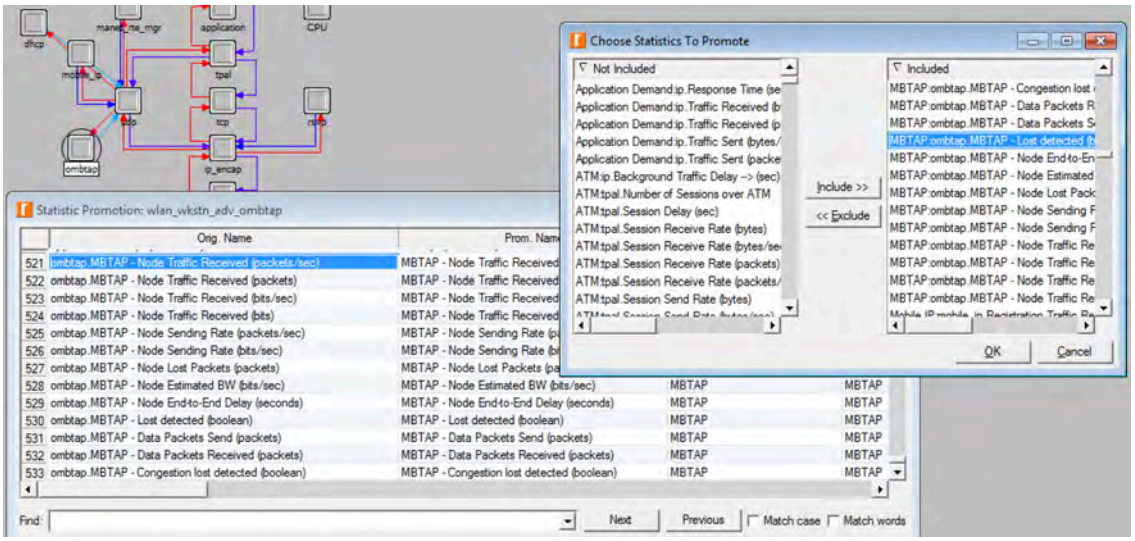

Fig. 45: Promoció de les "Local Statistics".

Finalment, cal triar quines estadístiques es volen visualitzar després de la simulació des de la finestra “Configure/Run Discret Event Simulation” de l’Editor de Projecte. Dins d’aquesta, s’ha anar a “Reports” → “Statistic Groups/SLAs” i prémer “Define Statistics Report”, on es seleccionaran les estadístiques que es volen generar, tal i com es mostra a la Fig. 46.

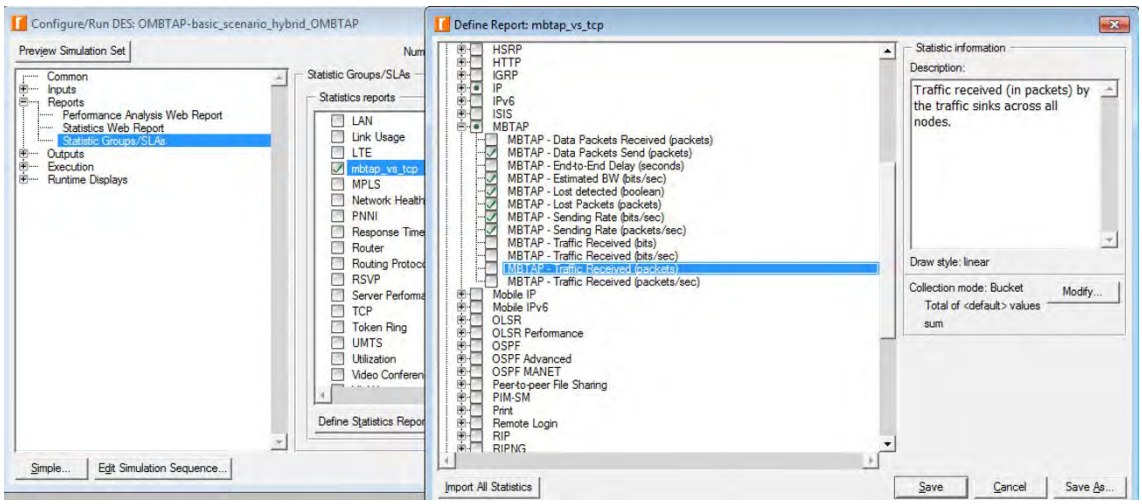

Fig. 46: Selecció de les estadístiques a visualitzar.

5 Simulacions i resultats

5.1 Descripció de l'escenari

En totes les simulacions portades a cap l'escenari és molt similar, seguint el mateix esquema que ja es va utilitzar per realitzar l'estudi del protocol MBTAP. La connexió es realitza entre una LAN situada a Terrassa, on s'hi ubica el servidor OMBTAP i el servidor d'altres aplicacions; i una altra a Barcelona, on s'hi ubiquen el/s client/s OMBTAP i el client d'altres aplicacions. En aquesta darrera LAN, la xarxa d'accés dels nodes finals és una xarxa sense fils. Concretament, s'ha optat per utilitzar l'estàndard IEEE 802.11n (Riverbed Modeler no implementa estàndards superior), amb una velocitat màxima teòrica de 300 Mbps en canal compartit quan s'utilitza MIMO[6] 2x2 [78]. Els routers de cada LAN es connecten entre ells mitjançant un enllaç PPP per formar la WAN (veure Fig. 47). Segons la prova, aquest enllaç WAN o l'enllaç sense fils exerceixen de coll d'ampolla i són el que limiten la capacitat total (ample de banda) màxima de la connexió. Els enllaços dins la LAN són 10Gigabit Ethernet per assegurar que el coll és un dels dos mencionats anteriorment.

Fig. 47: Escenari de la simulació a nivell WAN.

Les simulacions realitzades es poden dividir en 4 bateries de proves diferents:

1. Comunicació entre 1 client OMBTAP i 1 servidor OMBTAP sense generació de pèrdues.
2. Comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb generació de pèrdues aleatòries.
3. Comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb generació de tràfic creuat i pèrdues aleatòries.
4. Comunicació entre 2 clients OMBTAP i 1 servidor OMBTAP.

[6] MIMO: Multiple-input Multiple-output

5. Comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb generació de tràfic creuat i pèrdues aleatòries. Variació del valor del LTD.
6. Comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb generació de pèrdues aleatòries de duració variable.

La composició de les LANs a les diferents proves es mostra a la Fig. 48.

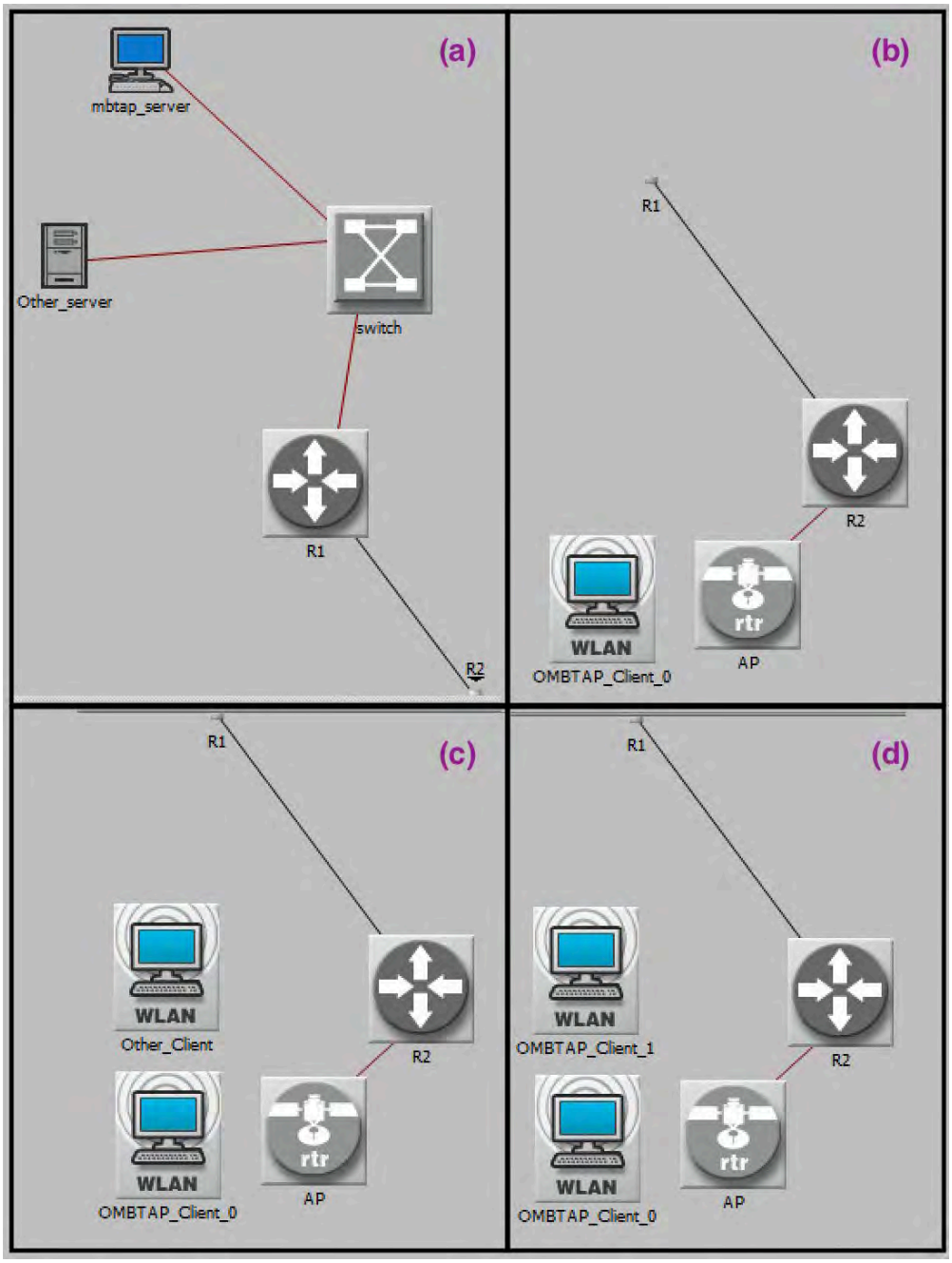

Fig. 48: Xarxes LAN de l'escenari de proves: (a) LAN de Terrassa per a totes les proves. (b) LAN de Barcelona per les bateries de proves 1, 2 i 6. (c) LAN de Barcelona per les bateries de proves 3 i 5. (d) LAN de Barcelona per a la bateria de proves 4.

En tots els escenaris (excepte en la bateria de proves 5) se substituiran els nodes OMBTAP per nodes MBTAP per tal de realitzar una comparativa del comportament i rendiment entre tots dos protocols.

5.1.1 Generació de pèrdues aleatòries

Una de les tasques més complexes de la configuració dels escenaris és la de poder generar pèrdues aleatòries al canal sense fils de forma controlada, ja que és necessari saber a priori quan es produiran aquestes, per així determinar si el protocol OMBTAP està discernint correctament entre pèrdues provocades per congestió i pèrdues provocades aleatòriament per algun error del canal.

Riverbed Modeler no té cap mecanisme que permeti simular aquest procés. A més, el model de canal sense fils que utilitza per implementar aquest tipus de xarxes no té en compte possibles causes d'aquestes pèrdues aleatòries, com podrien ser l'esvaïment (*fading*) o el multicamí. L'únic que permet configurar és la potència de transmissió i la sensibilitat del receptor, de manera que seguint la fórmula de l'atenuació a l'espai lliure[7] es determina si la distància a la que es troben els dos nodes és suficient com per rebre dades. Com es pot observar, aquesta és una implementació molt poc realista del que succeeix a la pràctica.

Així doncs, s'ha optat per un altre mètode. Aquest mètode consisteix en generar interferències a partir de la col·locació d'una segona BSS amb clients (a més de l'AP al qual es connecta el client OMBTAP) que estiguin emetent a la mateixa banda i al mateix canal freqüencial que el node OMBTAP. Riverbed Modeler sí que es capaç de simular correctament aquestes interferències, de forma que quan dos BSS independents emeten a la mateixa freqüència i a la mateixa zona de cobertura, es perden els paquets que s'enviïn en aquell moment. D'aquesta manera, es pot crear una situació en que es perdin paquets de forma "aleatòria" per una causa diferent a la congestió de l'enllaç, en aquest cas un error del canal (interferències).

El següent pas és poder controlar quan es genera aquesta interferència. Si es col·loqués el segon BSS amb els seus clients sempre a la mateixa zona de cobertura que l'AP que utilitza el client OMBTAP, es perdrien tots els paquets durant tota la comunicació, de manera que no se'n podrien extreure conclusions rellevants. Per tant, el que interessa és col·loca aquests nodes generadors d'interferència només durant un interval de temps en la mateixa zona de cobertura que els nodes OMBTAP. Això s'aconsegueix utilitzant un objecte anomenat trajectòria (*Tracejtory*), que permet controlar el moviment d'un o diversos nodes per tot l'escenari al llarg de la simulació, tal i com es mostra a la Fig. 49.

[7] $L_{bf} = 20 \log\left(\frac{4\pi d}{\lambda}\right)$ [dB]

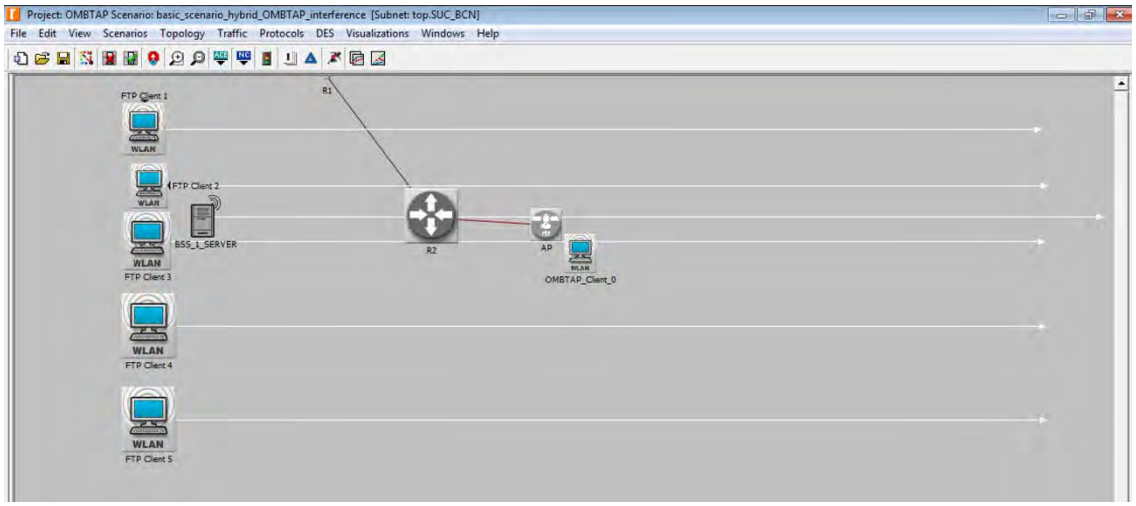

Fig. 49: Generació d'interferències amb el moviment (trajectòria) d'una segona BSS i els seus clients comunicant-se a la mateixa banda freqüencial que l'AP i el client OMBTAP.

Per tal de configurar una trajectòria, només cal definir tants passos com es desitgin i, dins de cada pas, definir la posició (coordenades X,Y relatives a la posició inicial) a la que es mourà el node, quant durarà aquest moviment i quant temps esperarà quiet després de realitzar-lo i abans de procedir al següent pas. En el cas d'aquest treball, per generar interferències era suficient definir 3 passos, tal i com es mostra a la Fig. 50:

1- Restar quiet a la posició inicial

2- Moure's cap a la mateixa zona de cobertura que l'AP que connecta amb el client OMBTAP, i esperar allà el temps durant el qual es vulgui generar interferències.

3- Marxar fora de la mateixa zona de cobertura i esperar fins el final de la simulació.

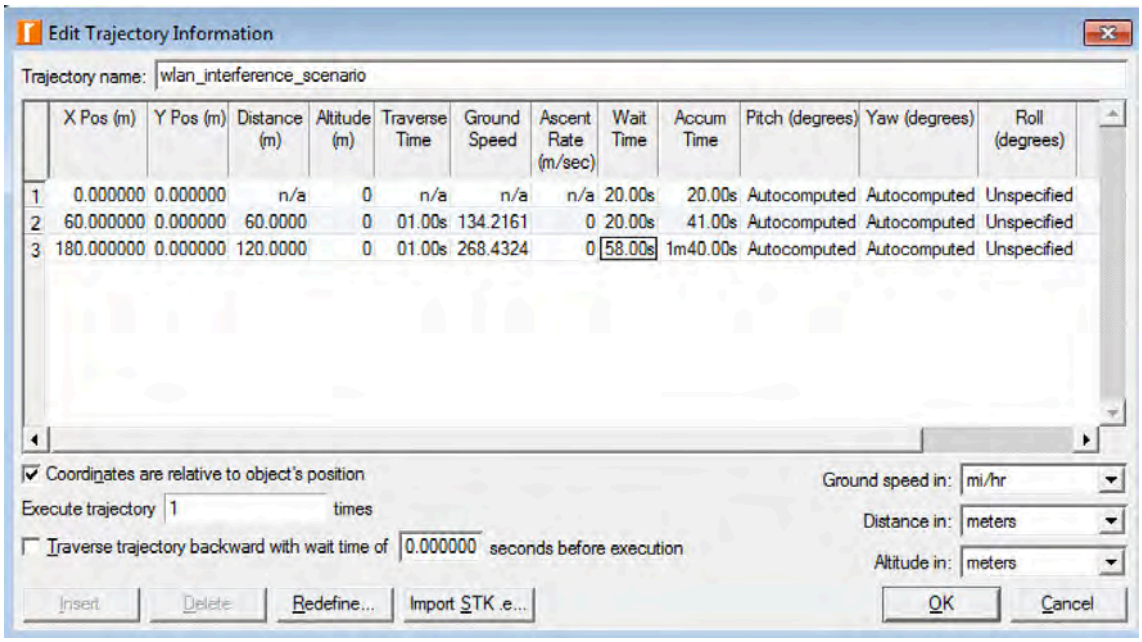

Fig. 50: Definició de la trajectòria (moviments dels nodes) per generar interferències.

S'ha optat per generar en tots els escenaris sempre la mateixa interferència (excepte en la bateria de proves 6), entre els 50 i els 52 segons de la simulació.

5.2 Bateria de proves 1: comunicació entre 1 client OMBTAP i 1 servidor OMBTAP sense generació de pèrdues

Prova	1.A	1.B	1.C
Capacitat enllaç WAN	148,6 Mbps	2377 Mbps	148,6 Mbps
Capacitat enllaç sense fils	300 Mbps	300 Mbps	150 Mbps
Quantitat de dades enviades	1 GB	3 GB	1 GB

Taula 6: Característiques de la bateria de proves 1 (en vermell el coll d'ampolla).

5.2.1 Prova 1.A: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades

5.2.1.1 MBTAP

En aquest escenari el protocol MBTAP estima un ample de banda del canal de 147,3 Mbps, mentre que envia a una velocitat de 147,03 Mbps (veure Fig. 51), aconseguint una utilització del canal del 98,9%. L'enviament és estable i sense pèrdues. Per tant, es pot concloure que l'estimació del canal segueix sent correcta encara que es tracti d'una xarxa heterogènia.

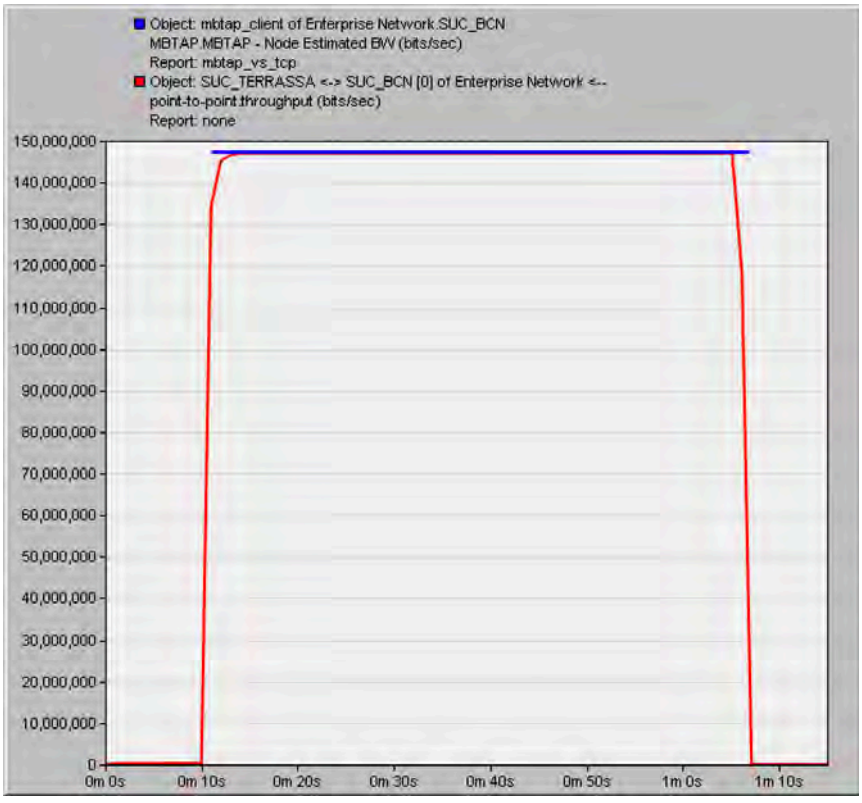

Fig. 51: Prova 1.A.MBTAP - Ample de banda estimat (blau) i *throughput* del coll d'ampolla (vermell).

Resultat: Estimació del canal correcta. Utilització d'un 98,9% de la capacitat total.

5.2.1.2 OMBTAP

En aquest escenari el protocol OMBTAP estima un ample de banda del canal de 147,29 Mbps, mentre que envia a una velocitat de 147,02 Mbps (veure Fig. 52), aconseguint una utilització del canal del 98,9%. L'enviament és estable i sense pèrdues. És lògic que el resultat sigui tant similar a l'anterior, ja que el mecanisme d'estimació d'ample de banda i l'enviament de paquets és el mateix per tots dos protocols.

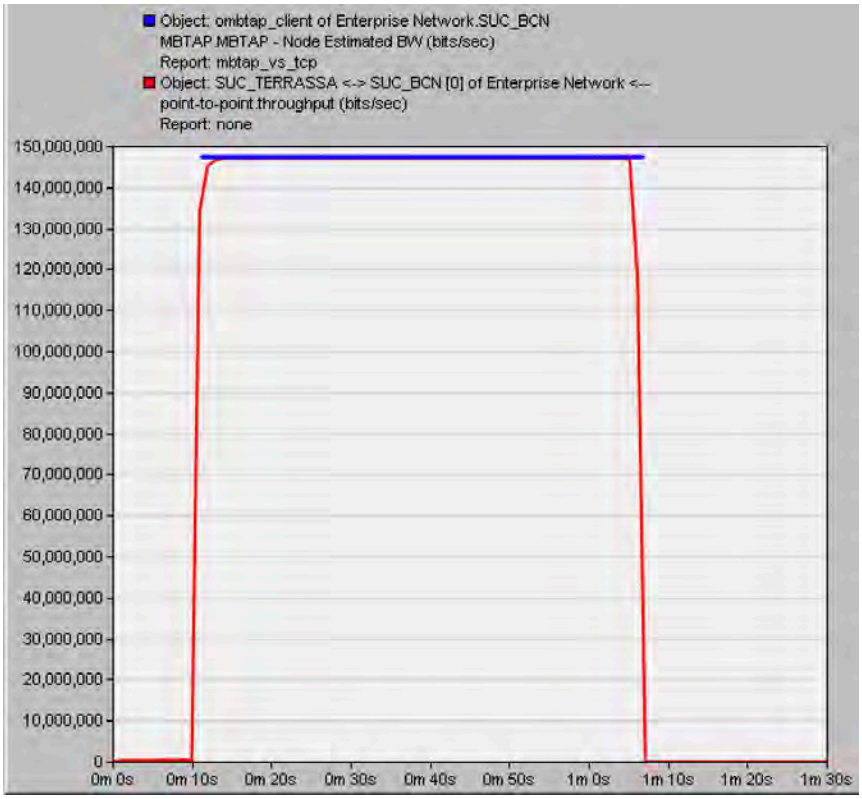

Fig. 52: Prova 1.A.OMBTAP - Ample de banda estimat (blau) i *throughput* del coll d'ampolla (vermell).

Resultat: Estimació del canal correcta. Utilització d'un 98,9% de la capacitat total.

5.2.2 Prova 1.B: Enllaç WAN SONET-48 (2,377 Gbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades

5.2.2.1 MBTAP

En aquest escenari el protocol MBTAP estima un ample de banda del canal de 211,78 Mbps, mentre que envia a una velocitat de 211,72 Mbps (veure Fig. 53), aconseguint una utilització del canal del 70,6%. L'enviament és estable i sense pèrdues. S'observa com en aquest cas, en tractar-se el coll d'ampolla d'un enllaç sense fils, la velocitat real d'aquest deu ser inferior a la teòrica, motiu pel qual s'estima un ample de banda menor. Tot i així, la velocitat real d'enviament s'estabilitza a un 99,9% de la capacitat estimada.

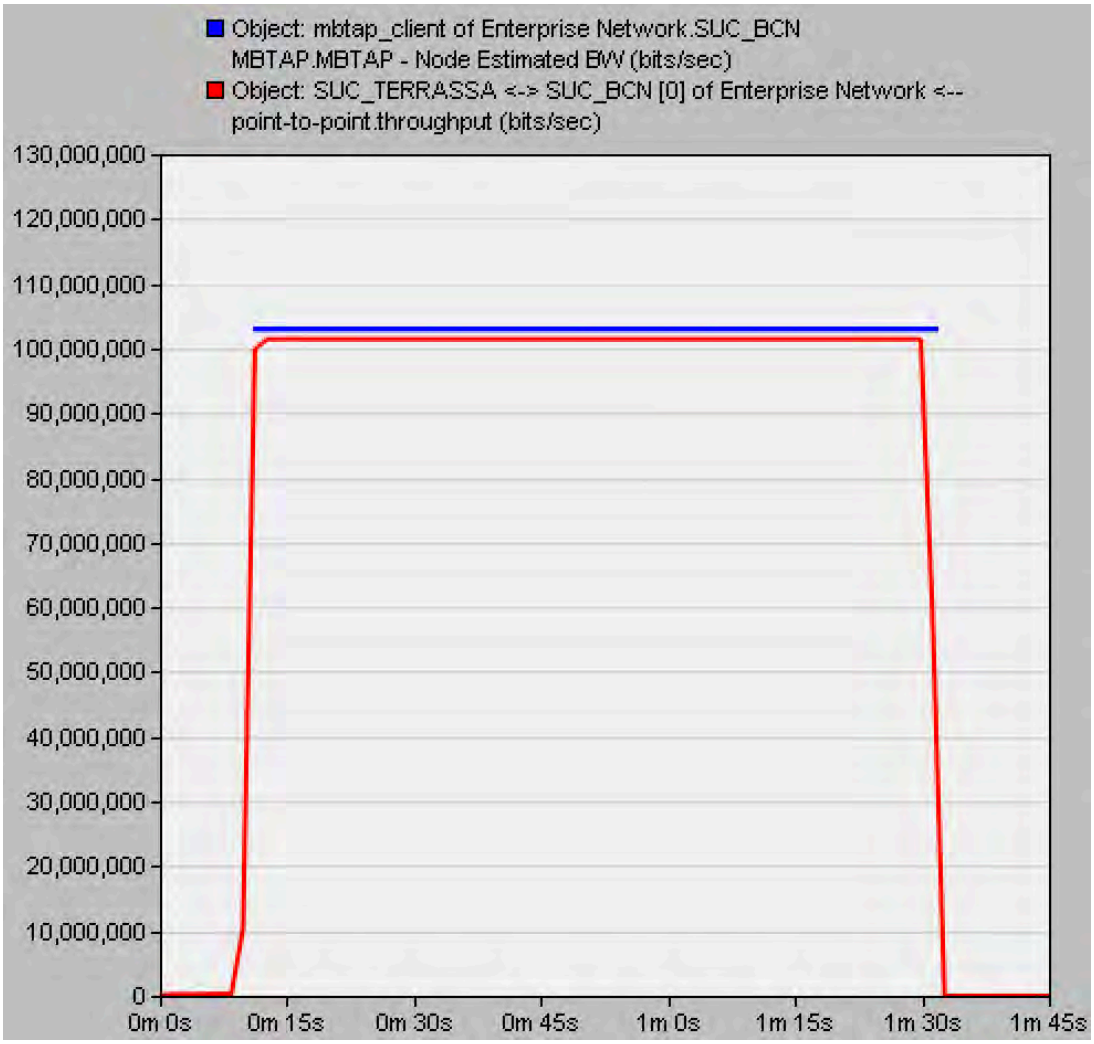

Fig. 53: Prova 1.C.MBTAP - Ample de banda estimat (blau) i *throughput* del coll d'ampolla (vermell).

Resultat: Estimació del canal inferior a la teòrica degut a que el coll d'ampolla és un enllaç sense fils. Utilització d'un 70,6% de la capacitat total. Utilització d'un 99,9% de l'ample de banda estimat.

5.2.2.2 OMBTAP

En aquest escenari el protocol OMBTAP estima un ample de banda del canal de 211,94 Mbps, mentre que envia a una velocitat de 211,73 Mbps (veure Fig. 54), aconseguint una utilització del canal del 70,6%. L'enviament és estable i sense pèrdues. De nou, el resultat és molt similar a l'anterior, ja que el mecanisme d'estimació d'ample de banda i l'enviament de paquets és el mateix per tots dos protocols.

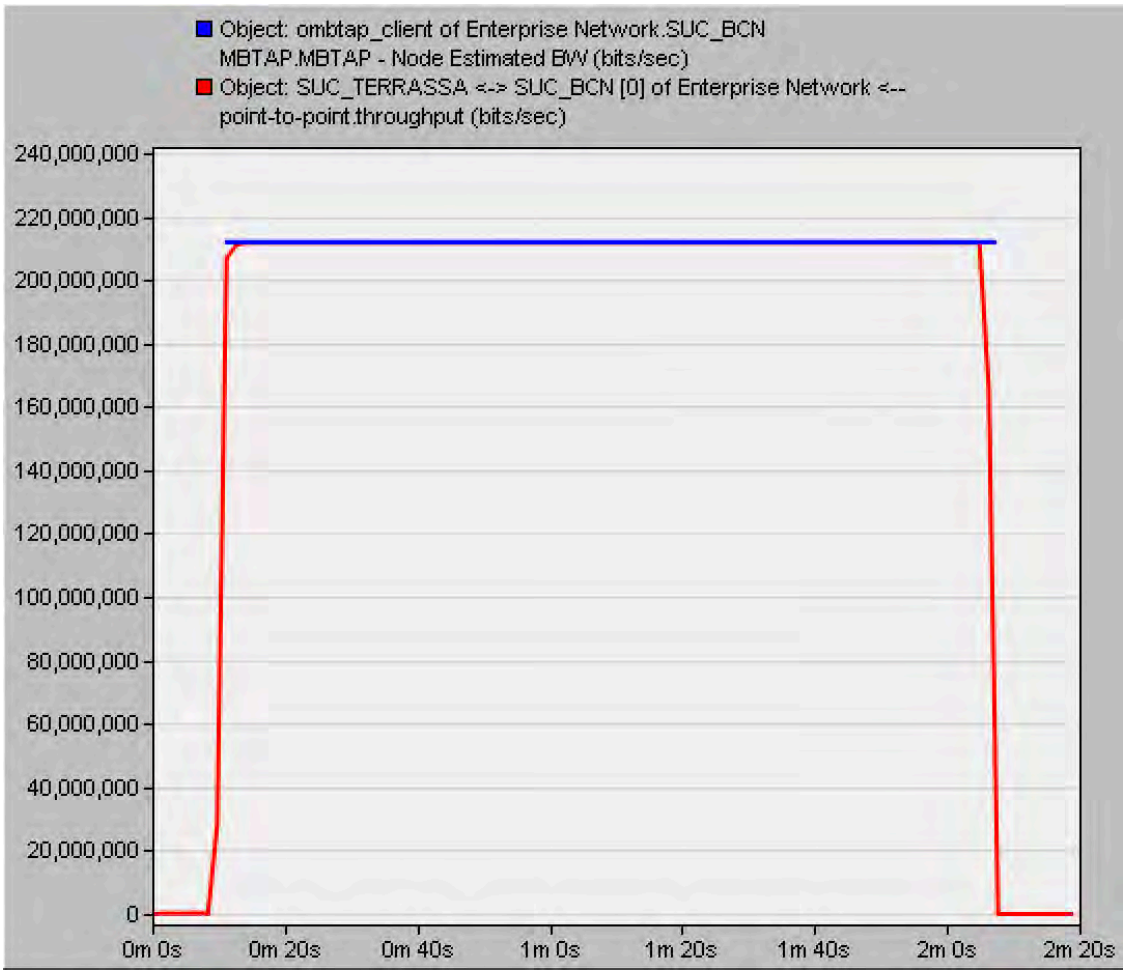

Fig. 54: Prova 1.C.OMBTAP - Ample de banda estimat (blau) i *throughput* del coll d'ampolla (vermell).

Resultat: Estimació del canal inferior a la teòrica degut a que el coll d'ampolla és un enllaç sense fils. Utilització d'un 70,6% de la capacitat total. Utilització d'un 99,9% de l'ample de banda estimat.

5.2.3 Prova 1.C: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 150 Mbps, enviament de 3 GB de dades

5.2.3.1 MBTAP

En aquest escenari el protocol MBTAP estima un ample de banda del canal de 103 Mbps, mentre que envia a una velocitat de 101,45 Mbps (veure Fig. 55), aconseguint una utilització del canal del 68,3%. L'enviament és estable i sense pèrdues. S'observa com en aquest cas, tot i que el coll d'ampolla teòric hauria de ser l'enllaç WAN (el seu ample de banda de 148,6 Mbps és lleugerament inferior que els 150 Mbps de l'enllaç sense fils), l'ample de banda estimat torna a ser aproximadament un 70% de la capacitat màxima teòrica de l'enllaç 802.11n, esdevenint aquest el coll d'ampolla real. Tot i així, la velocitat real d'enviament s'estabilitza a un 98,5% de la capacitat estimada.

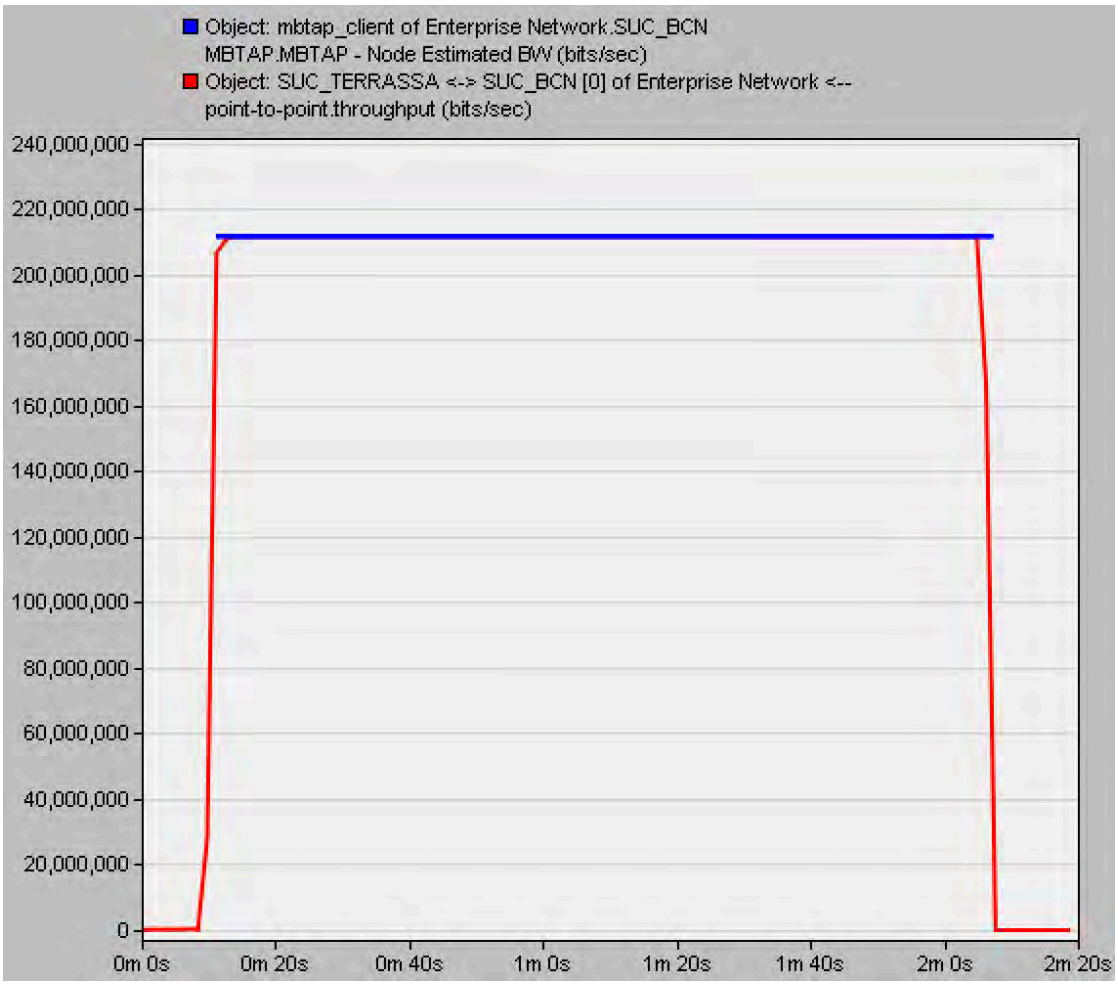

Fig. 55: Prova 1.B.MBTAP - Ample de banda estimat (blau) i *throughput* del coll d'ampolla (vermell).

Resultat: Estimació del canal inferior a la teòrica degut a que el coll d'ampolla és l'enllaç sense fils. Utilització d'un 68,3% de la capacitat total. Utilització d'un 98,5% de l'ample de banda estimat.

5.2.3.2 OMBTAP

En aquest escenari el protocol OMBTAP estima un ample de banda del canal de 103,63 Mbps, mentre que envia a una velocitat de 102,92 Mbps (veure Fig. 56), aconseguint una utilització del canal del 69,3%. L'enviament és estable i sense pèrdues. De nou, el resultat és molt similar a l'anterior, ja que el mecanisme d'estimació d'ample de banda i l'enviament de paquets és el mateix per tots dos protocols.

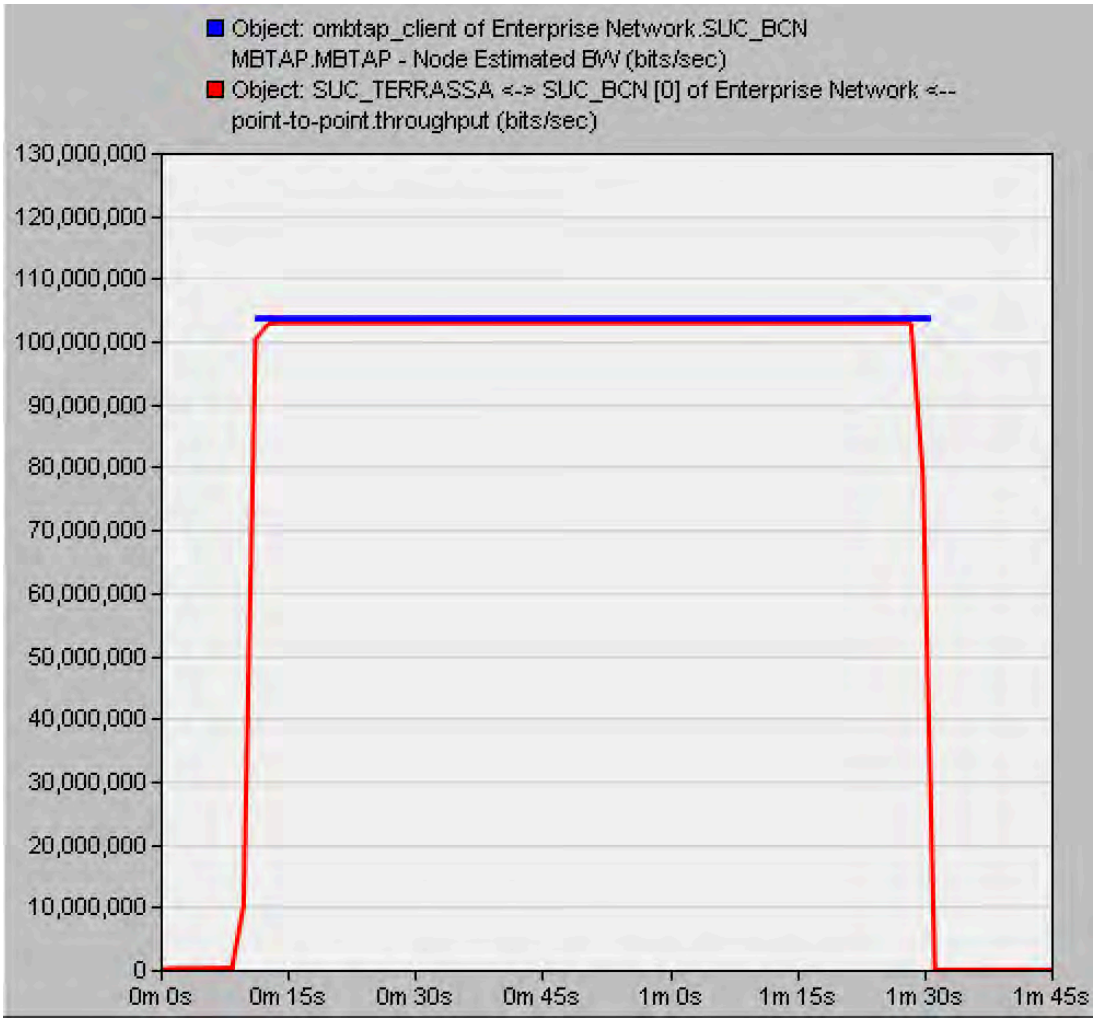

Fig. 56: Prova 1.B.OMBTAP - Ample de banda estimat (blau) i *throughput* del coll d'ampolla (vermell).

Resultat: Estimació del canal inferior a la teòrica degut a que el coll d'ampolla és l'enllaç sense fils. Utilització d'un 69,3% de la capacitat total. Utilització d'un 99,3% de l'ample de banda estimat.

5.2.4 Conclusions de la bateria de proves 1.

L'enviament de dades per un enllaç sense pèrdues ni tràfic creuat és correcte (s'envien tots els paquets a màxima velocitat). Si el coll d'ampolla és un enllaç cablejat, l'estimació de l'ample de banda és pràcticament la mateixa que la velocitat màxima teòrica de l'enllaç, mentre que si el coll d'ampolla es tracta d'un enllaç sense fils, la velocitat real no s'ajusta a la capacitat màxima teòrica i, per tant, l'ample de banda estimat també és inferior (un 70% aproximadament). Això concorda amb els estudis que demostren que l'eficiència de CSMA/CA es pot veure reduïda per diversos factors, com per exemple la distància respecte l'AP [79].

Prova	1.A	1.B	1.C
Capacitat enllaç WAN	148,6 Mbps	2377 Mbps	148,6 Mbps
Capacitat enllaç sense fils	300 Mbps	300 Mbps	150 Mbps
Quantitat de dades enviades	1 GB	3 GB	1 GB
BW[8] estimat (MBTAP)	147,3 Mbps	211,78 Mbps	103 Mbps
BW estimat (OMBTAP)	147,29 Mbps	211,94 Mbps	103,63 Mbps
Utilització de l'enllaç (MBTAP)	98,9%	70,6%	68,3%
Utilització de l'enllaç (OMBTAP)	98,9%	70,6%	69,3%

Taula 7: Resultats de la bateria de proves 1 (en vermell el coll d'ampolla).

[8] BW: Ample de banda

5.3 Bateria de proves 2: comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb generació de pèrdues aleatòries

Per tal de veure el comportament del protocol a l'hora de discernir entre pèrdues aleatòries i pèrdues per congestió, amb la mesura correctiva corresponent per part del control de congestió, es generen pèrdues aleatòries al canal tal i com s'ha explicat a l'apartat 5.1.1. En aquest cas, com que només hi ha interferències i no pas congestió, el Sending Rate no hauria de disminuir mai, i en tot cas hauria de mantenir-se quan es detectessin pèrdues per interferència o augmentar (o mantenir-se si ja es troba al màxim) en cas de no detectar pèrdues.

En aquest cas es comprovarà que aquesta situació es compleix amb dos escenaris diferents. En el primer, el coll d'ampolla és l'enllaç WAN de 148,6 Mbps, mentre que en el segon ho és l'enllaç sense fils de 300 Mbps. Aquesta és la màxima velocitat a la que es pot provar un escenari de xarxes heterogènies dins del simulador Riverbed Modeler. Per tant, aquesta serà la màxima velocitat a la que queda demostrat que funciona el mecanisme definit.

Prova	2.A	2.B
Capacitat coll d'ampolla	148,6 Mbps	300 Mbps
Quantitat de dades enviades	1 GB	3 GB

Taula 8: Característiques de la bateria de proves 2.

5.3.1 Prova 2.A: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades i pèrdues aleatòries

5.3.1.1 MBTAP

La Fig. 57 mostra que en aquesta prova el protocol MBTAP estima un ample de banda del canal de 147,3 Mbps. El Sending Rate es manté estable a 144,09 Mbps (no té en compte les capçaleres) i la velocitat d'enviament és de 147,03 Mbps. (98,9% d'utilització). S'observa que quan apareixen les interferències als 50 segons es perden tots els paquets, la velocitat baixa en picat i el Sending Rate actua en conseqüència (disminuïnt fins a 106 Mbps), ja que el protocol MBTAP no és capaç de discernir que aquestes pèrdues no són degudes a la congestió. Finalment, quan les pèrdues desapareixen, la velocitat d'enviament es va recuperant (es reprèn a 114,6 Mbps) fins arribar de nou als 147,03 Mbps. La transferència del fitxer es completa en 1 minut. A la Fig. 58 s'observa la pèrdua de tots els paquets en un moment puntual a causa de les interferències.

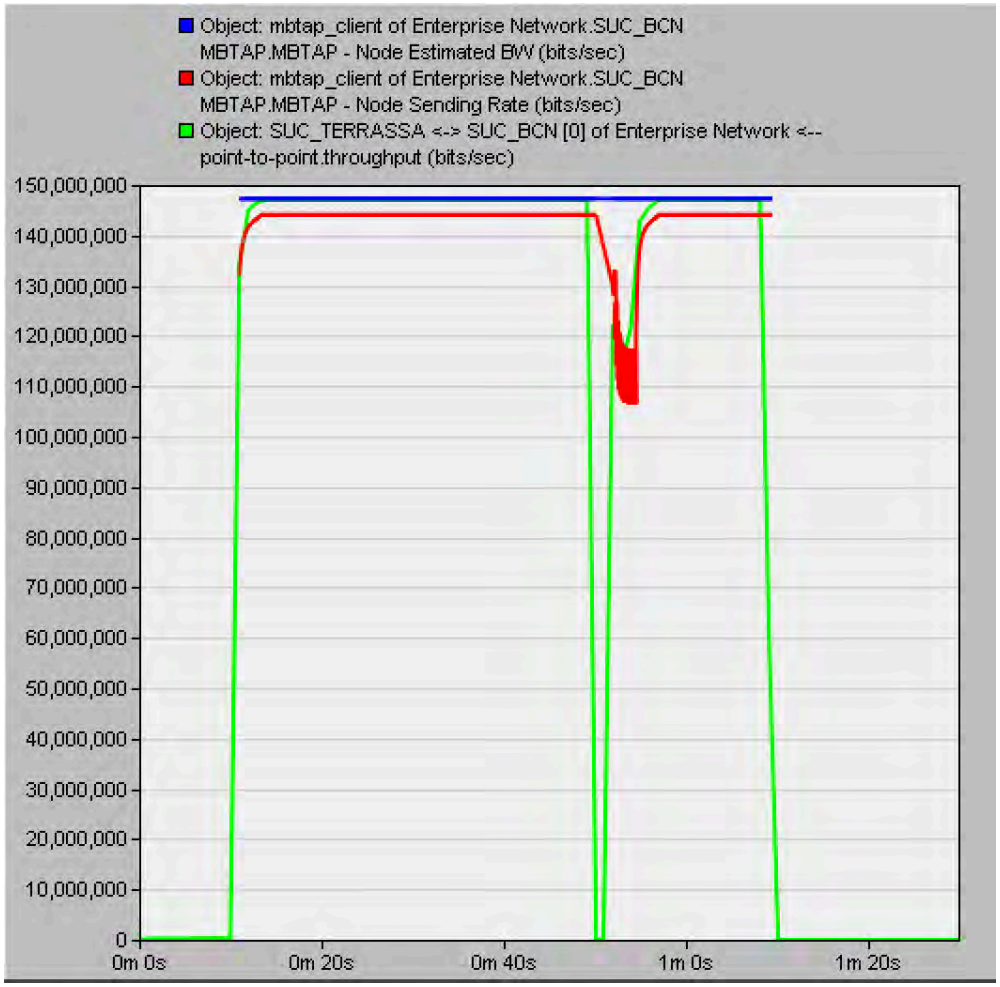

Fig. 57: Prova 2.A.MBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

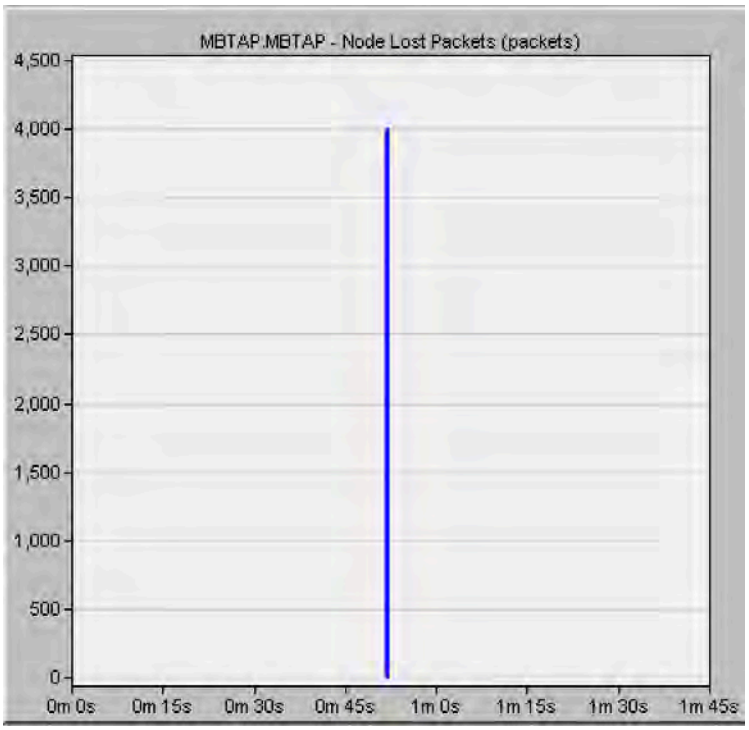

Fig. 58: Prova 2.A.MBTAP - Paquets perduts.

Resultat: Durant el període d'interferències, el control de congestió interpreta les pèrdues com a congestió del canal i el Sending Rate disminueix en conseqüència. Després del període de pèrdues, es torna a recuperar la velocitat d'enviament anterior.

5.3.1.2 OMBTAP

La Fig. 59 mostra que en aquesta cas el protocol OMBTAP estima un ample de banda del canal de 147,29 Mbps. El Sending Rate es manté estable a 144,09 Mbps i la velocitat d'enviament és de 147,02 Mbps. (98,9% d'utilització). La gran diferència respecte la prova anterior es dóna quan apareixen les interferències als 50 segons (es perden tots els paquets), ja que en aquest cas el protocol OMBTAP sí que és capaç d'identificar les pèrdues com a conseqüència d'error del canal (no de congestió), tal i com ho demostra el fet que el Sending Rate es mantingui en lloc de baixar. Aquest fet permet que, un cop desaparegudes les interferències, la velocitat d'enviament torni a recuperar de forma instantània el valor d'abans de les pèrdues (147,02 Mbps). La transferència del fitxer es completa en 59 segons (un 1,67% més ràpida que en el cas anterior). A la Fig. 60 s'observa la pèrdua de tots els paquets en un moment puntual a causa de les interferències.

Fig. 59: Prova 2.A.OMBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

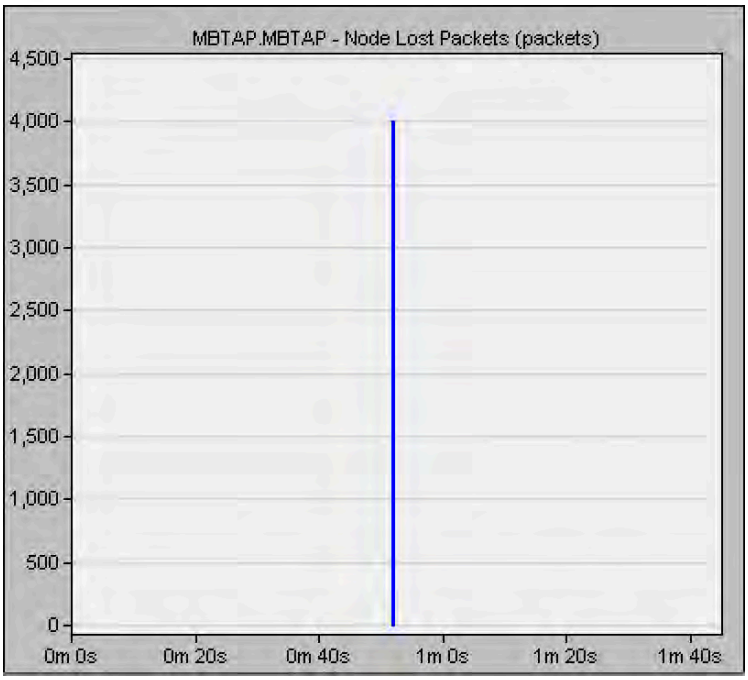

Fig. 60: Prova 2.A.OMBTAP - Paquets perduts.

Resultat: Durant el període d'interferències, el control de congestió interpreta les pèrdues correctament com a error aleatori del canal i el Sending Rate es manté (no disminueix). Després del període de pèrdues, es torna a recuperar la velocitat d'enviament anterior de forma més ràpida.

5.3.2 Prova 2.B: Enllaç WAN SONET-48 (2,377 Gbps) i enllaç sense fils de 300 Mbps, enviament de 3 GB de dades i pèrdues aleatòries

5.3.2.1 MBTAP

La Fig. 61 mostra que en aquesta prova el protocol MBTAP estima un ample de banda del canal de 211,78 Mbps. El Sending Rate es manté estable a 208,2 Mbps (no té en compte les capçaleres) i la velocitat d'enviament és de 211,72 Mbps. (70,6% d'utilització). S'observa que quan apareixen les interferències als 50 segons es perden tots els paquets, la velocitat baixa en picat i el Sending Rate actua en conseqüència (disminuïnt fins a 153,87 Mbps), ja que el protocol MBTAP no és capaç de discernir que aquestes pèrdues no són degudes a la congestió. Finalment, quan les pèrdues desapareixen, la velocitat d'enviament es va recuperant (es reprèn a 149,71 Mbps) fins arribar de nou als 211,72 Mbps. La transferència del fitxer es completa en 2 minuts i 2 segons. A la Fig. 62 s'observa la pèrdua de tots els paquets en un moment puntual a causa de les interferències.

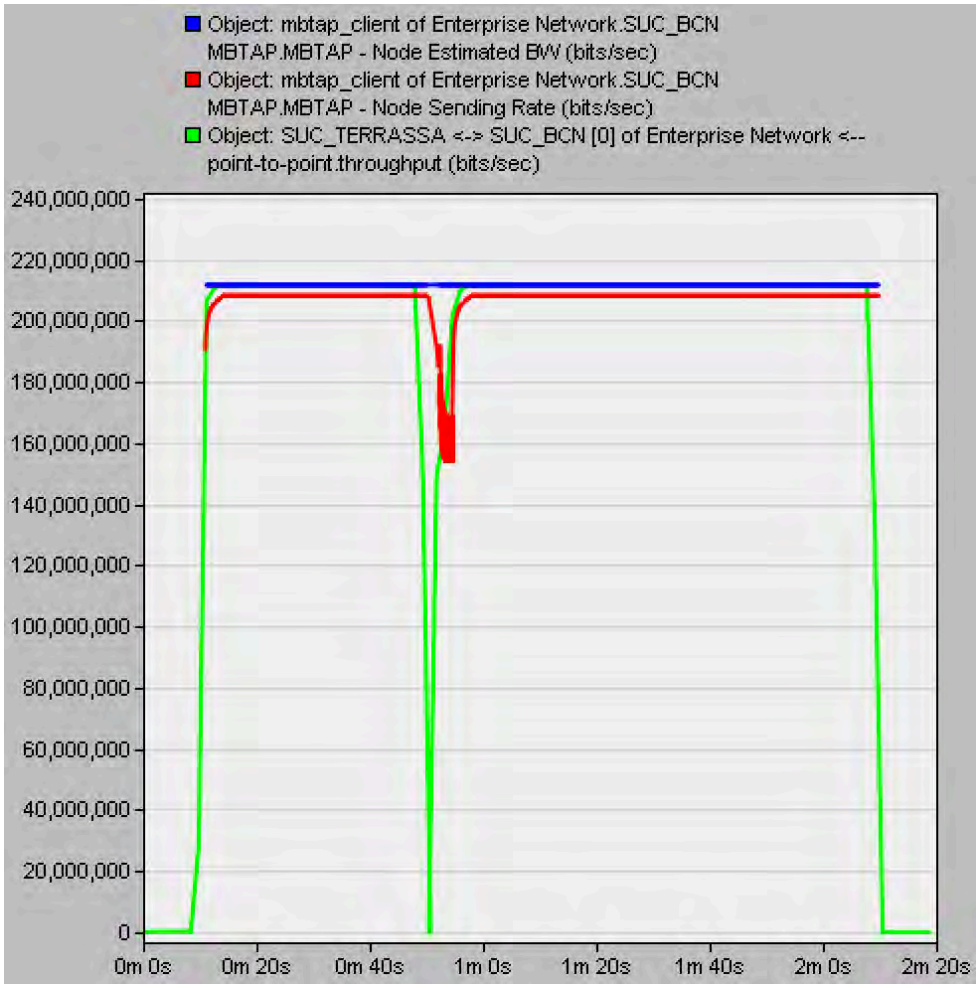

Fig. 61: Prova 2.B.MBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

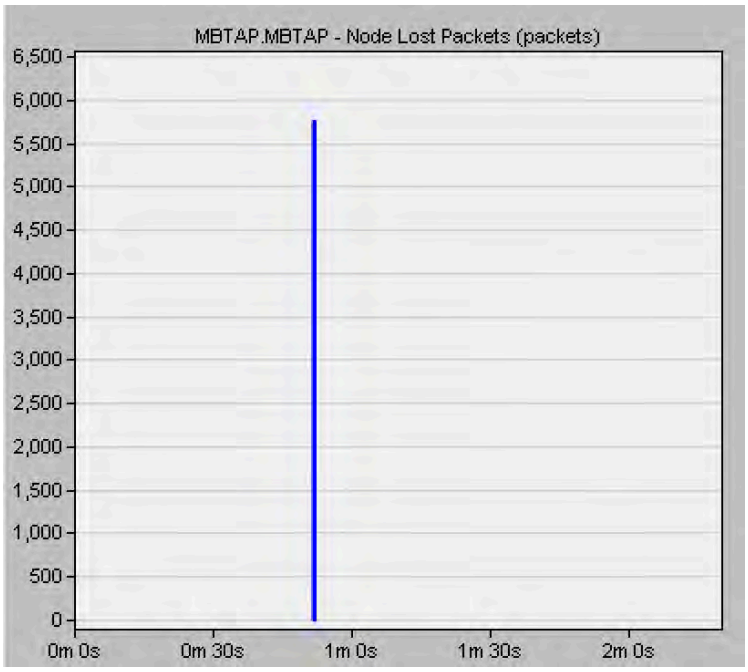

Fig. 62: Prova 2.B.MBTAP - Paquets perduts.

Resultat: Durant el període d'interferències, el control de congestió interpreta les pèrdues com a congestió del canal i el Sending Rate disminueix en conseqüència. Després del període de pèrdues, es torna a recuperar la velocitat d'enviament anterior.

5.3.2.2 OMBTAP

La Fig. 63 mostra que en aquesta cas el protocol OMBTAP estima un ample de banda del canal de 211,94 Mbps. El Sending Rate es manté estable a 208,25 Mbps i la velocitat d'enviament és de 211,73 Mbps. (70,6% d'utilització). La gran diferència es dóna novament quan apareixen les interferències als 50 segons (es perden tots els paquets), ja que en aquest cas el protocol OMBTAP sí que és capaç d'identificar les pèrdues com a conseqüència d'error del canal (no de congestió), tal i com ho demostra el fet que el Sending Rate es mantingui en lloc de baixar. Aquest fet permet que, un cop desaparegudes les interferències, la velocitat d'enviament torni a recuperar de forma instantània el valor d'abans de les pèrdues (211,73 Mbps). La transferència del fitxer es completa en 2 minuts i 0,5 segons (un 1,23% menys que en el cas anterior). A la Fig. 64 s'observa la pèrdua de tots els paquets en un moment puntual a causa de les interferències.

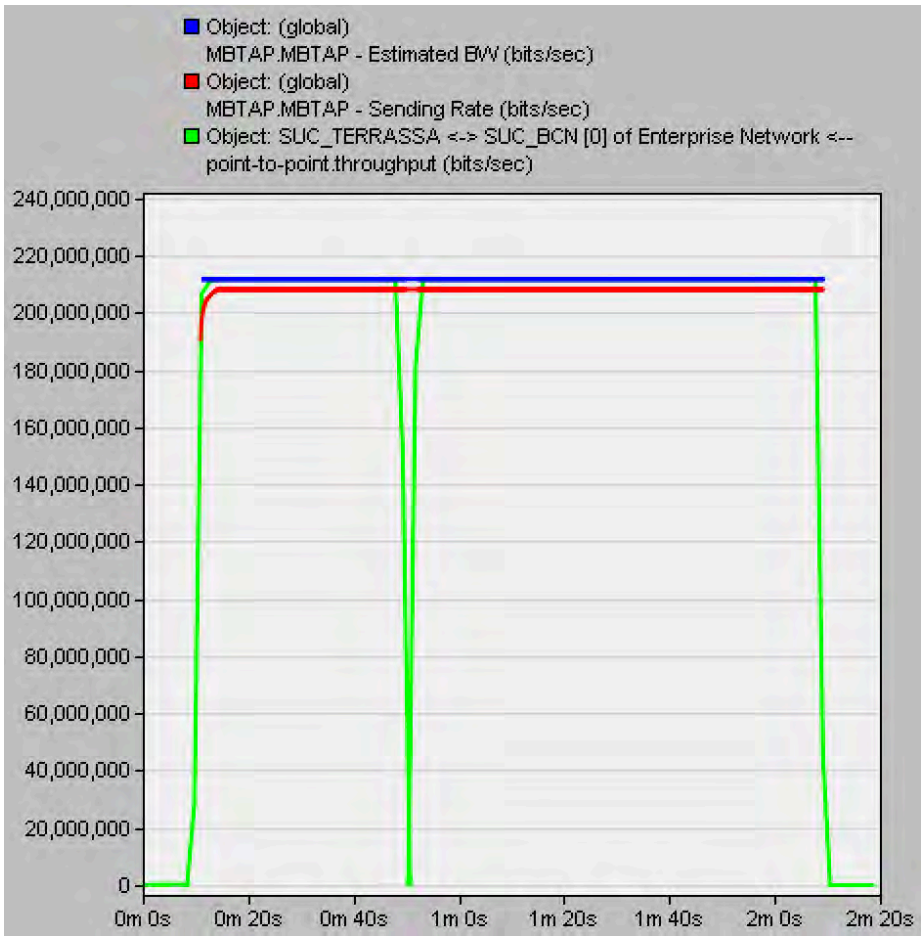

Fig. 63: Prova 2.B.OMBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

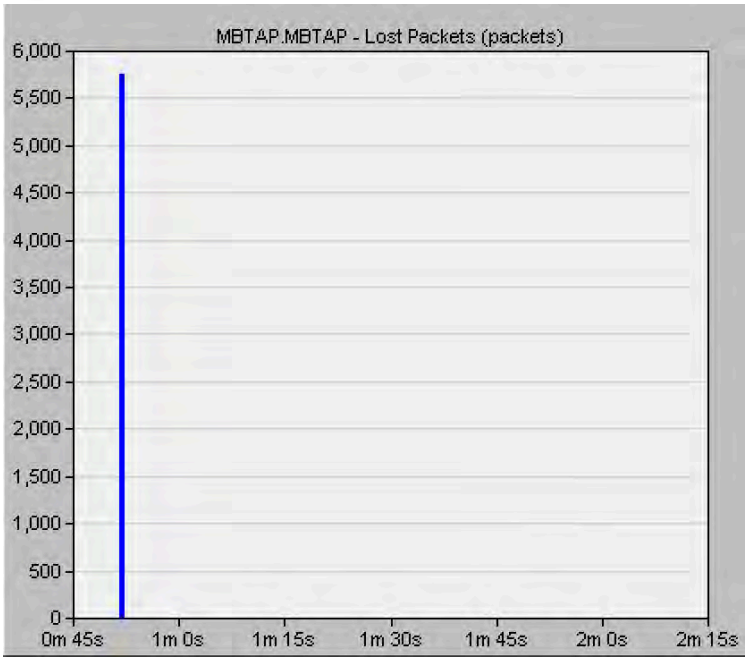

Fig. 64: Prova 2.B.OMBTAP - Paquets perduts.

Resultat: Durant el període d'interferències, el control de congestió interpreta les pèrdues correctament com a error aleatori del canal i el Sending Rate es manté (no disminueix). Després del període de pèrdues, es torna a recuperar la velocitat d'enviament anterior de forma més ràpida.

5.3.3 Conclusions de la bateria de proves 2.

El control de congestió de l'OMBTAP és capaç d'identificar les pèrdues originades per les interferències com a pèrdues aleatòries, en comptes de pèrdues de congestió com fa el protocol MBTAP. Això permet recuperar la màxima velocitat d'enviament de forma més ràpida, acabant amb una utilització del canal del 98,9%, i que la transferència del fitxer sigui més eficient (es redueix el temps de transferència en més d'un 1%).

Prova	2.A	2.B
Capacitat coll d'ampolla	148,6 Mbps	300 Mbps
Quantitat de dades enviades	1 GB	3 GB
Identifica interferències com a pèrdua aleatòria (MBTAP)	No	No
Identifica interferències com a pèrdua aleatòria (OMBTAP)	Sí	Sí
Throughput just després de pèrdues (MBTAP)	114,6 Mbps	149,71 Mbps
Throughput just després de pèrdues (OMBTAP)	147,02 Mbps	211,73 Mbps
Throughput final (MBTAP)	147,03 Mbps	211,72 Mbps
Throughput final (OBTAP)	147,02 Mbps	211,73 Mbps
Duració transferència (MBTAP)	60s	122s
Duració transferència (OMBTAP)	59s	120,5s

Taula 9: Resultats de la bateria de proves 2.

5.4 Bateria de proves 3: comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb tràfic creuat i interferències

L'objectiu d'aquesta bateria de proves és observar com es comporta el protocol OMBTAP davant la presència de tràfic creuat FTP[9] (TCP) i de videoconferència (UDP), els quals generaran congestió, i també davant d'interferències, les quals generaran pèrdues aleatòries. D'aquesta manera, s'hauria de poder veure com el control de congestió actua diferent per cada cas, reduint el Sending Rate en el primer i mantenint-lo en el segon, aconseguint així una transferència més eficient respecte el protocol MBTAP. Per tal propòsit, es col·loca un segon client (en aquest cas, un node "workstation" estàndard) a la LAN de Barcelona que executarà aquestes aplicacions. En totes les proves el coll d'ampolla segueix sent l'enllaç WAN SONET-3 per les raons ja comentades anteriorment.

Prova	3.A	3.B
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps
Quantitat de dades enviades	3 GB	3 GB
Tipus de tràfic creuat	FTP (TCP)	UDP
Quantitat de tràfic creuat	300 MB	30 Mbps

Taula 10: Característiques de la bateria de proves 3.

5.4.1 Prova 3.A: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament de 3 GB de dades OMBTAP i 300 MB de dades FTP.

5.4.1.1 MBTAP

En aquesta prova primer es llença l'enviament de dades FTP. El protocol estima inicialment un ample de banda de 147,3 Mbps tot i la presència de tràfic creuat i la velocitat d'enviament inicial és de 147,03 Mbps (utilització del 98,8%). Això provoca la congestió del coll d'ampolla (utilització del 100%), ja que l'enviament de dades FTP també s'estava fent a alta velocitat, de manera que es perden paquets (veure Fig. 66). A la Fig. 65 s'observa com el control de congestió actua i el SR disminueix, fen-t'ho també la velocitat d'enviament. Com que MBTAP està concebut per ser un protocol agressiu, aquest es queda amb la majoria de l'ample de banda del canal, disminuint molt la velocitat de transferència de FTP. Un cop la congestió acaba (la transferència de FTP finalitza) , el Sending Rate de MBTAP augmenta i es recupera la màxima utilització de l'enllaç. Més endavant, es generen interferències que, tal i com passava en la bateria de proves anterior, el control de congestió del MBTAP no és capaç d'identificar com a pèrdues aleatòries, de forma que interpreta que torna a haver-hi congestió i disminueix el Sending Rate de nou (així com la velocitat d'enviament). Un cop aquestes finalitzen,

[9] File Transfer Protocol (FTP) és un dels protocols de transferència de fitxers més populars. Utilitza TCP com a protocol de transport

el Sending Rate torna a remuntar fins que la velocitat de transferència recupera els 147,03 Mbps. La transferència del fitxer es completa en 3 minuts i 1 segon. En la Fig. 66 s'aprecien clarament els dos moments de pèrdues de paquets (el primer per congestió i el darrer per interferències), mentre que a la Fig. 68 s'observa com el coll d'ampolla sempre presenta una utilització molt alta gràcies a l'agressivitat del protocol (excepte en el moment que el canal sense fils falla per culpa de les interferències).

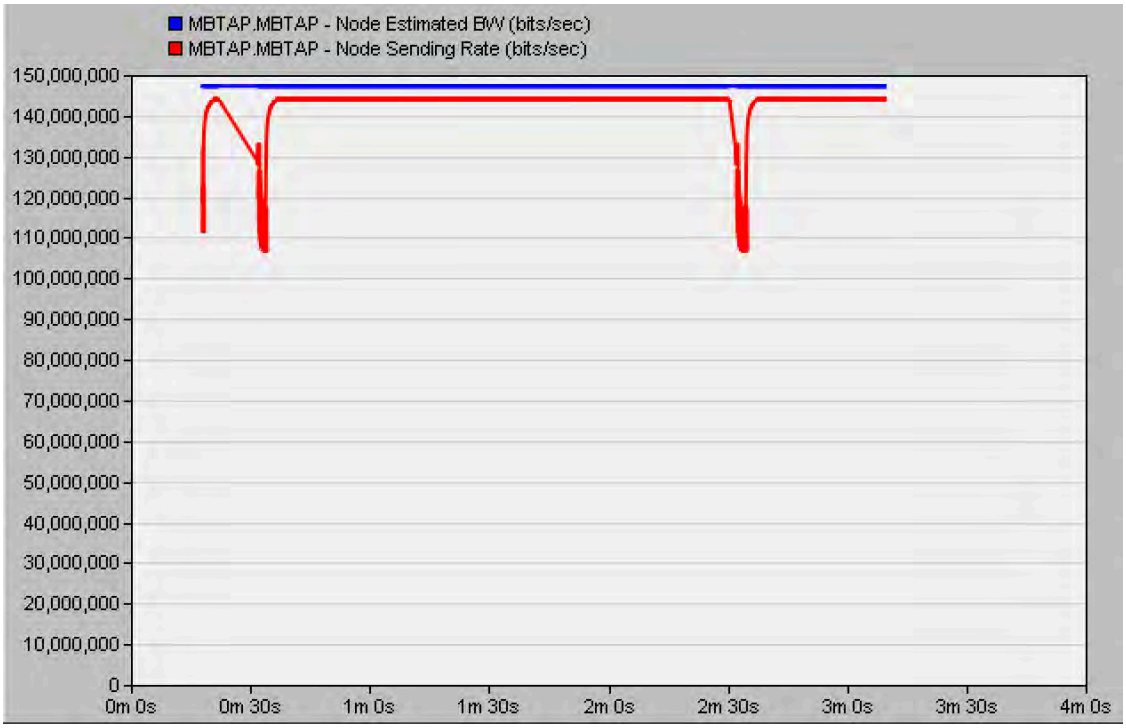

Fig. 65: Prova 3.A.MBTAP - Ample de banda estimat (blau) i Sending Rate calculat (vermell).

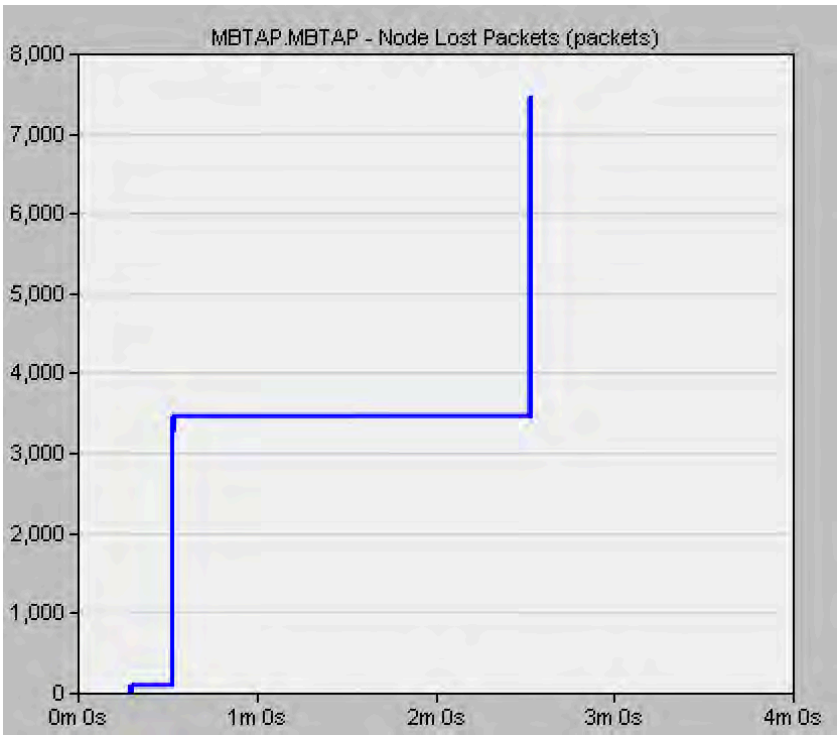

Fig. 66: Prova 3.A.MBTAP - Paquets MBTAP perduts.

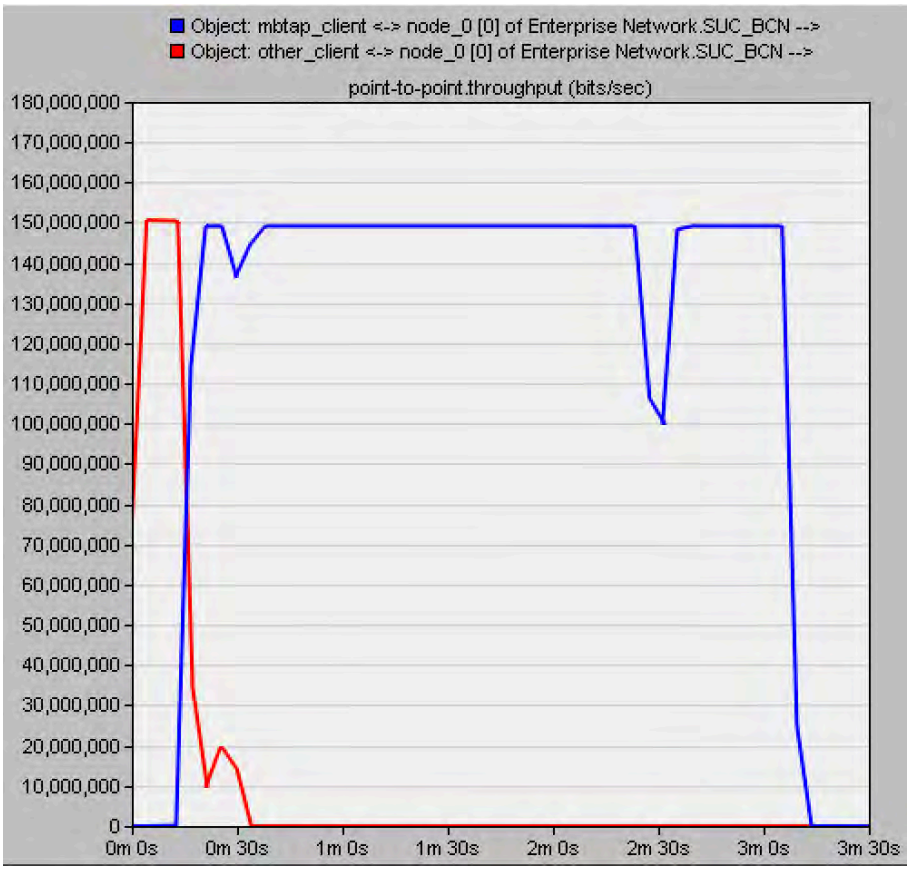

Fig. 67: Prova 3.A.MBTAP - *Througput* del client MBTAP (blau) i del client FTP (vermell).

Fig. 68: Prova 3.A.MBTAP - *Throughput* del coll d'ampolla.

Resultat: Durant el període de congestió el client MBTAP estima correctament la capacitat de l'enllaç i aconsegueix la major part de l'ample de banda amb un *throughput* alt. Durant el període d'interferències, el control de congestió interpreta les pèrdues com a congestió del canal i el Sending Rate disminueix en conseqüència. Després del període de pèrdues, es torna a recuperar la velocitat d'enviament anterior.

5.4.1.2 OMBTAP

De nou, primer es llença l'enviament de dades FTP. El protocol estima inicialment un ample de banda de 147,29 Mbps tot i la presència de tràfic creuat i la velocitat d'enviament inicial és de 147,02 Mbps (utilització del 98,8%). Això provoca la congestió del coll d'ampolla (utilització del 100%), tal i com passava a la prova anterior, i es perden paquets (veure Fig. 70). A la Fig. 69 s'observa com el control de congestió de l'OMBTAP detecta que les pèrdues són degudes a la saturació de l'enllaç, ja que el SR disminueix, així com la velocitat d'enviament. De la mateixa manera que passava amb MBTAP, l'agressivitat de l'OMBTAP fa que aquest es quedi amb la majoria de l'ample de banda del canal, disminuint molt la velocitat de transferència de FTP. Un cop la congestió acaba (la transferència de FTP finalitza) , el Sending Rate de MBTAP augmenta i es recupera la màxima utilització de l'enllaç. Més endavant, es generen interferències que, a diferència de la prova anterior, el control de congestió de l'OMBTAP sí és capaç d'identificar com a pèrdues aleatòries, de forma que interpreta no cal disminuir el SR i el manté. Un cop aquestes finalitzen, al Sending Rate no li cal pujar més i la velocitat d'enviament es recupera amb una sola iteració, arribant als 147,02 Mbps. La transferència del fitxer es completa en 3 minuts (un 0,55% més ràpid que en el cas d'anterior). En la Fig. 70 s'aprecien novament els dos moments de pèrdues de paquets (el primer per congestió i el darrer per interferències), mentre que a la Fig. 72 s'observa com el coll d'ampolla sempre presenta una utilització molt alta gràcies a l'agressivitat del protocol (excepte en el moment que el canal sense fils falla per culpa de les interferències).

Fig. 69: Prova 3.A.OMBTAP - Ample de banda estimat (blau) i Sending Rate calculat (vermell).

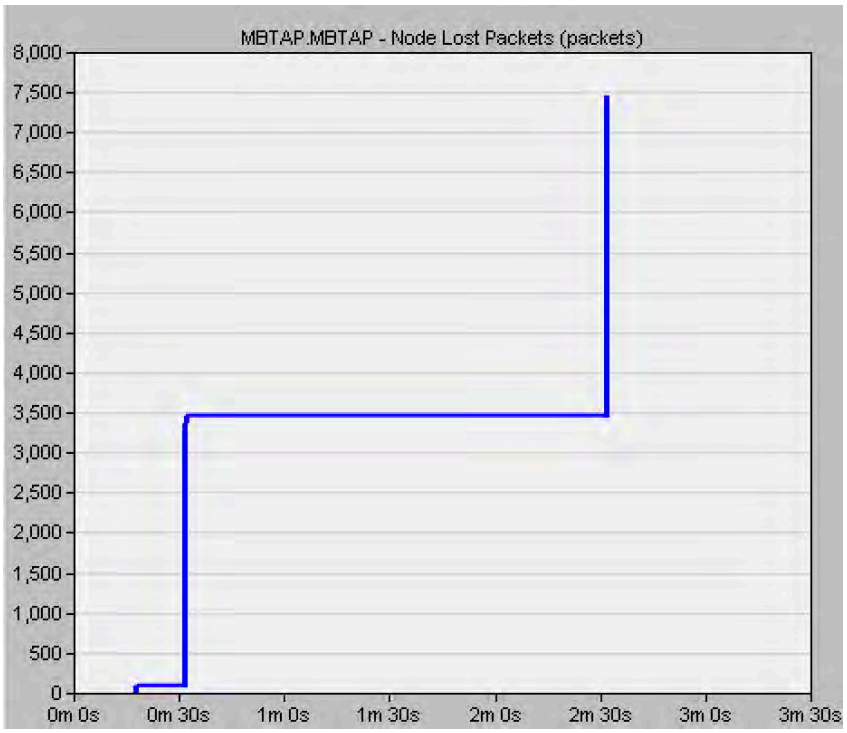

Fig. 70: Prova 3.A.OMBTAP - Paquets OMBTAP perduts.

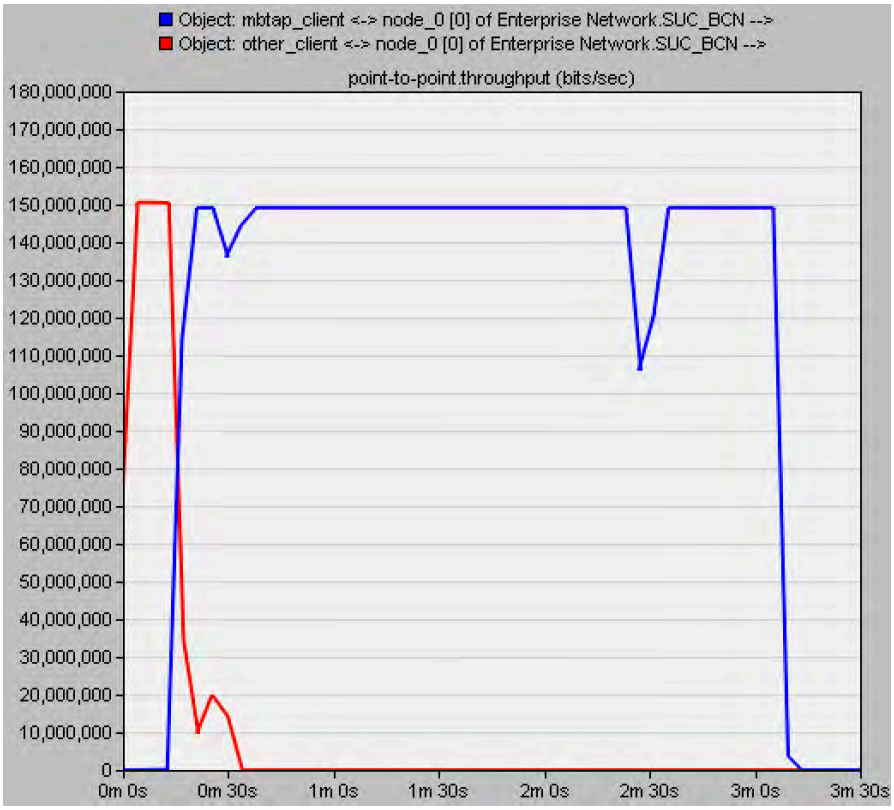

Fig. 71: Prova 3.A.OMBTAP - *Througput* del client OMBTAP (blau) i del client FTP (vermell).

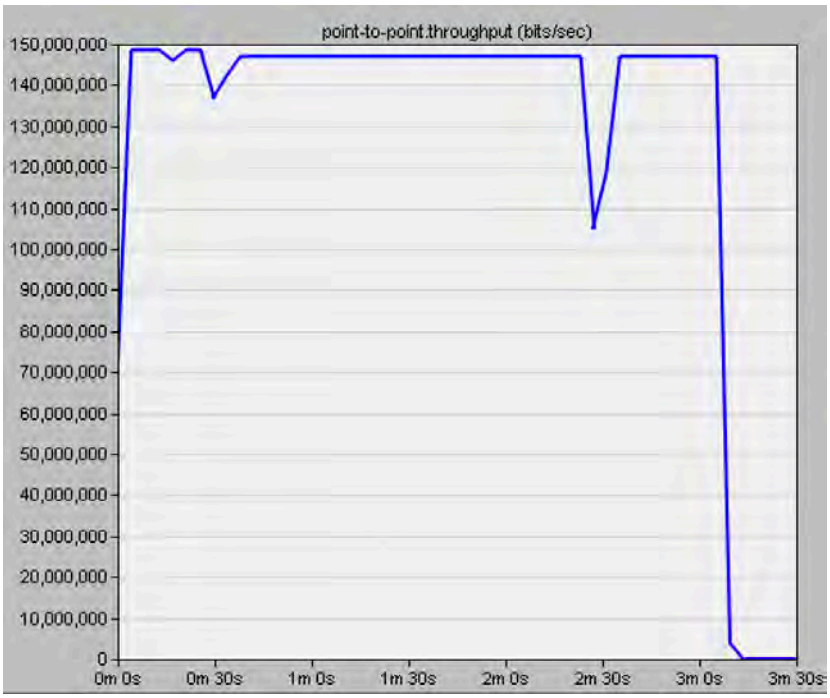

Fig. 72: Prova 3.A.OMBTAP - *Throughput* del coll d'ampolla.

Resultat: Durant el període de congestió el client MBTAP estima correctament la capacitat de l'enllaç i aconsegueix la major part de l'ample de banda amb un *throughput* alt. Durant el període d'interferències, el control de congestió interpreta les pèrdues com aleatòries i el Sending Rate es manté. Després del període de pèrdues, es torna a recuperar la màxima velocitat d'enviament amb una sola iteració.

5.4.2 Prova 3.B: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament de 3 GB de dades OMBTAP i 30 Mbps de videoconferència.

5.4.2.1 MBTAP

En aquesta prova es pot veure a la Fig. 73 que, de nou, tot i haver-hi tràfic creuat (UDP en aquest cas), s'estima correctament la capacitat de l'enllaç (147,3 Mbps). Això provoca que el protocol MBTAP enviï a màxima velocitat (veure Fig. 75), però UDP, per la seva naturalesa (no té control de congestió) segueix enviant a 30 Mbps, provocant la saturació de l'enllaç observada a la Fig. 76. En aquest cas, la velocitat d'enviament durant la congestió és una mica inferior que en el cas de competir amb TCP, ja que la naturalesa d'UDP també és agressiva (envia a la velocitat requerida sense control de congestió). Un cop la congestió acaba (la videoconferència finalitza) , el Sending Rate de MBTAP augmenta i es recupera la màxima utilització de l'enllaç. Més endavant, es generen interferències que, tal i com passava en la bateria de proves anterior, el control de congestió del MBTAP no és capaç d'identificar com a pèrdues aleatòries, de forma que interpreta que torna a haver-hi congestió i disminueix el Sending Rate de nou (així com la velocitat d'enviament). Un cop aquestes finalitzen, el Sending Rate torna a remuntar fins que la velocitat de transferència recupera els 147,03 Mbps. La transferència del fitxer es completa en 2 minuts i 57 segons En la Fig. 74 s'aprecien clarament els dos moments de pèrdues de paquets (el primer per congestió i el darrer per interferències).

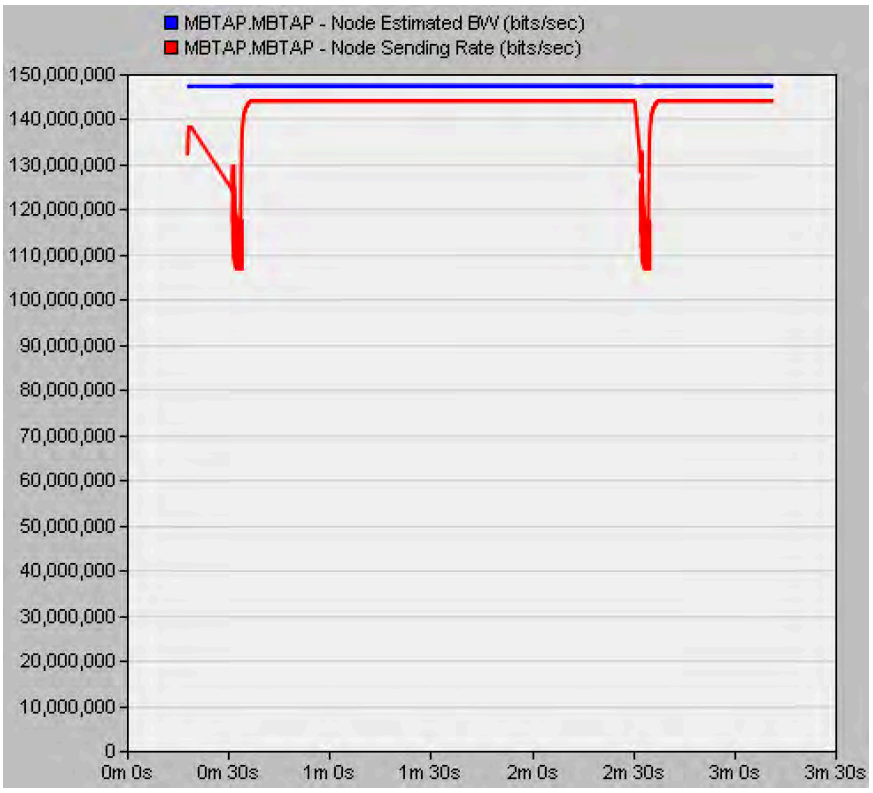

Fig. 73: Prova 3.B.MBTAP - Ample de banda estimat (blau) i Sending Rate calculat (vermell).

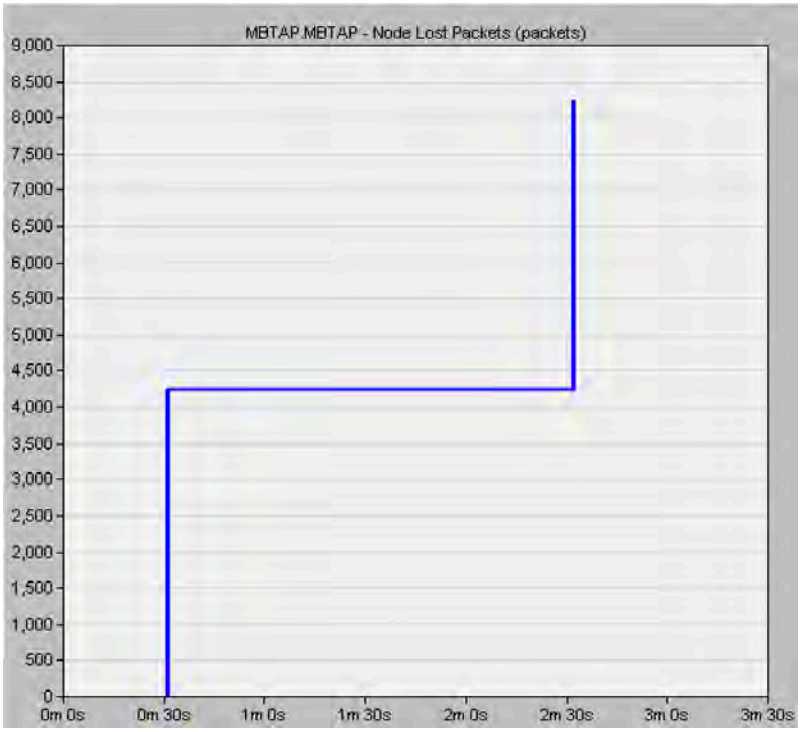

Fig. 74: Prova 3.B.MBTAP - Paquets MBTAP perduts.

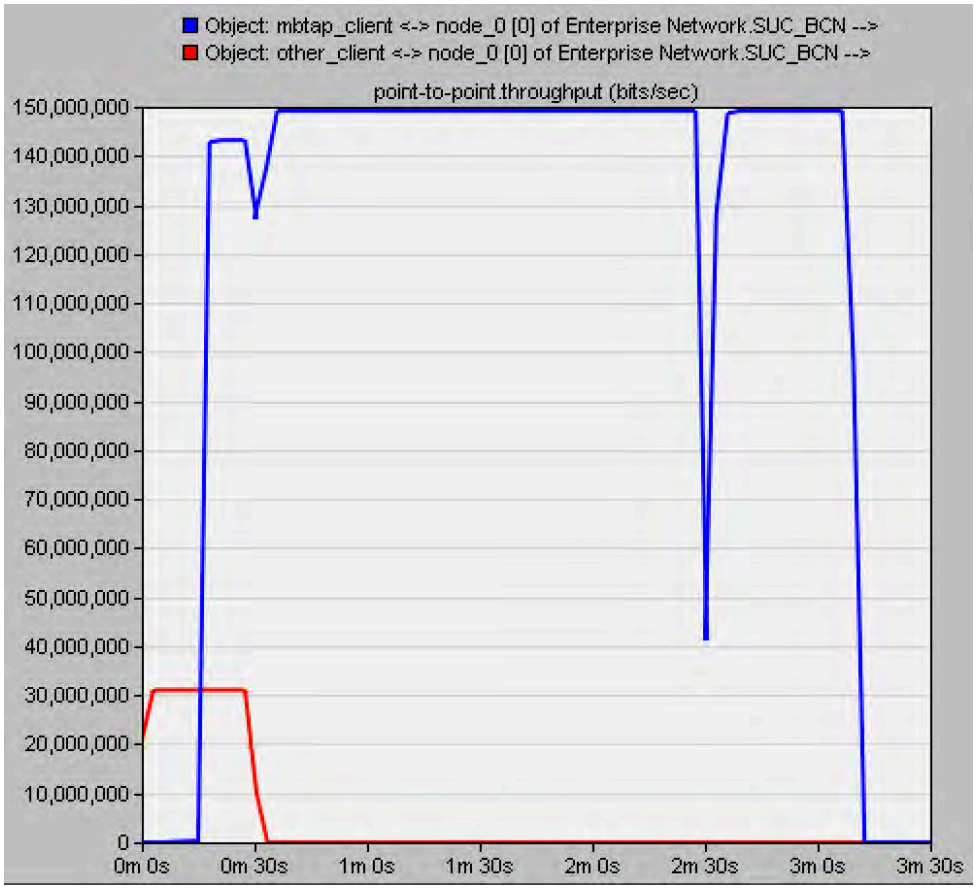

Fig. 75: Prova 3.B.MBTAP - *Througput* del client MBTAP (blau) i del client UDP (vermell).

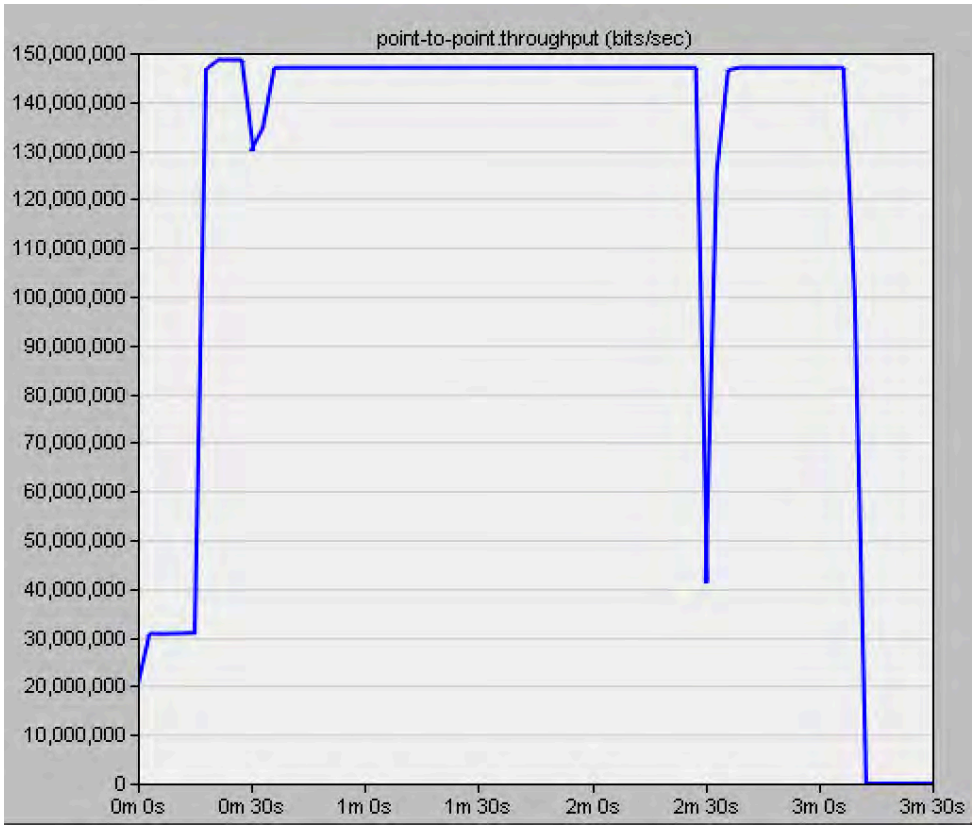

Fig. 76: Prova 3.B.MBTAP - *Throughput* del coll d'ampolla.

Resultat: Durant el període de congestió el client MBTAP estima correctament la capacitat de l'enllaç i envia a màxima velocitat amb un *throughput* de 143 Mbps. Com que el flux UDP està fixat a 31 Mbps, la seva velocitat de transferència no varia i es perden paquets MBTAP, fet que provoca una disminució del Sending Rate. Durant el període d'interferències, el control de congestió interpreta les pèrdues com a congestió del canal i el Sending Rate disminueix en conseqüència. Després del període de pèrdues, es torna a recuperar la velocitat d'enviament màxima (147,03 Mbps).

5.4.2.2 OMBTAP

S'observa de nou a la Fig. 77 que, tot i haver-hi tràfic creuat UDP, l'OMBTAP estima correctament la capacitat de l'enllaç (147,29 Mbps). Això provoca que la velocitat d'enviament sigui màxima (veure Fig. 79), però UDP, de la mateixa manera que en la prova anterior, segueix enviant a 30 Mbps, provocant la congestió de l'enllaç observada a la Fig. 80. Novament, el *throughput* durant la congestió és una mica inferior que en el cas de competir amb TCP a causa del comportament agressiu d'UDP. Un cop acabada la videoconferència , el Sending Rate d'OMBTAP augmenta i es recupera la màxima utilització de l'enllaç. Més endavant, es generen interferències que, tal i com passava en les bateries de proves anterior, el control de congestió de l'OMBTAP sí és capaç d'identificar com a pèrdues aleatòries, de forma que interpreta que no caldisminuir el Sending Rate (el manté). Un cop aquestes finalitzen, la màxima velocitat de transferència (147,02 Mbps) es recupera de forma instantània. La transferència del fitxer es completa en 2 minuts i 55 segons (un 1,3% més ràpid que en el cas d'anterior). En la Fig. 78 s'aprecien clarament els dos moments de pèrdues de paquets (el primer per congestió i el darrer per interferències), mentre que a la Fig. 80 es pot comprovar la pràcticament permanent i màxima utilització del coll d'ampolla (exceptuant el moment de les interferències).

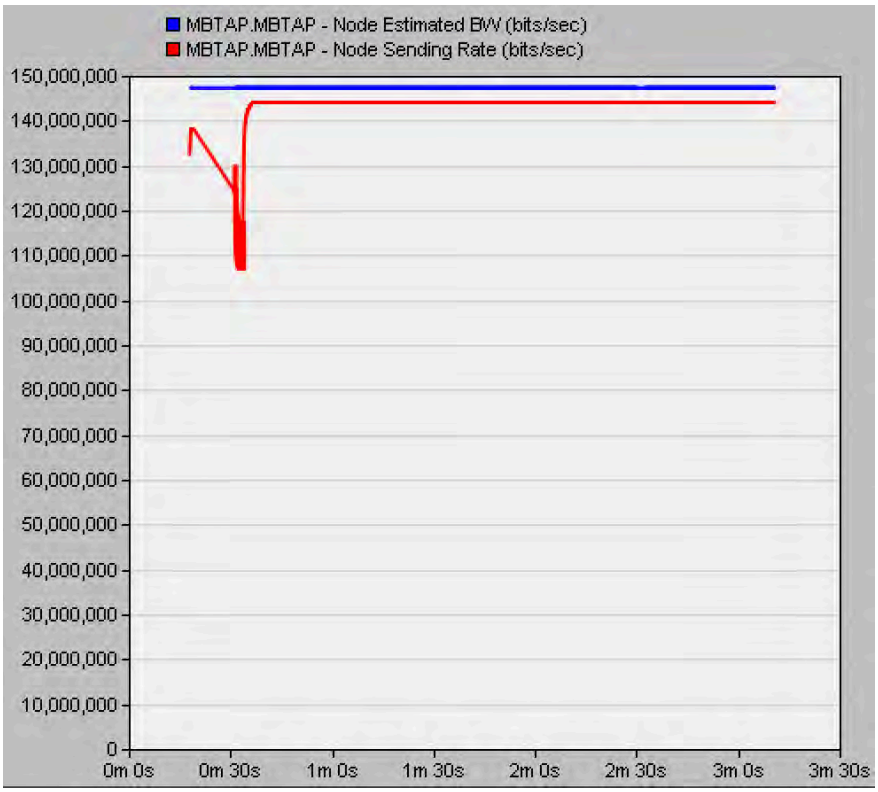

Fig. 77: Prova 3.B.OMBTAP - Ample de banda estimat (blau) i Sending Rate calculat (vermell).

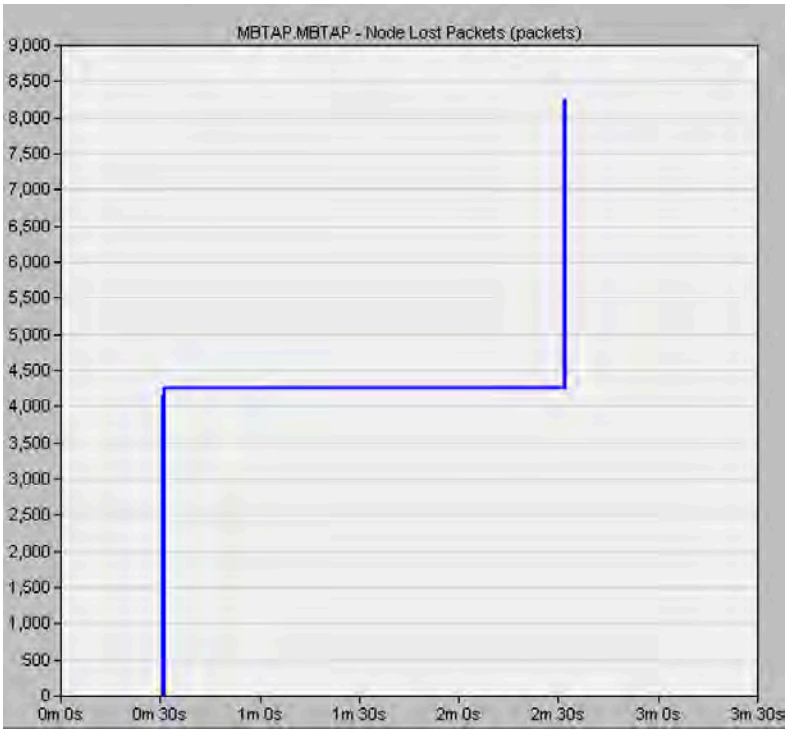

Fig. 78: Prova 3.B.OMBTAP - Paquets OMBTAP perduts.

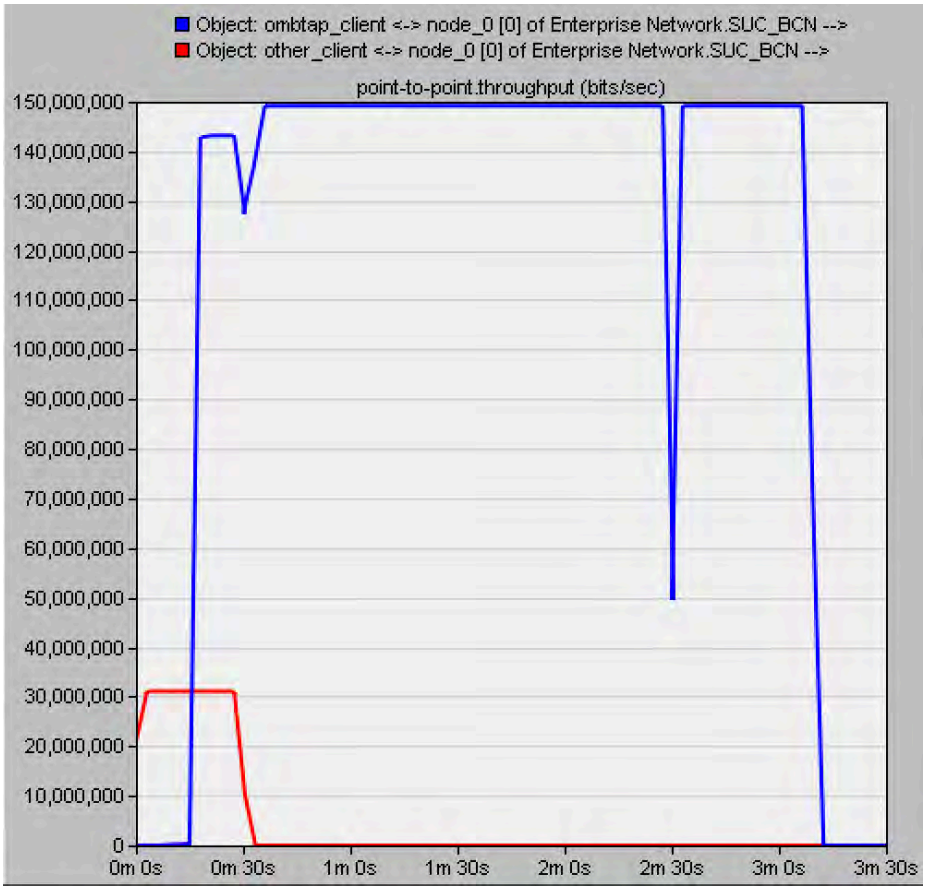

Fig. 79: Prova 3.B.OMBTAP - *Througput* del client OMBTAP (blau) i del client UDP (vermell).

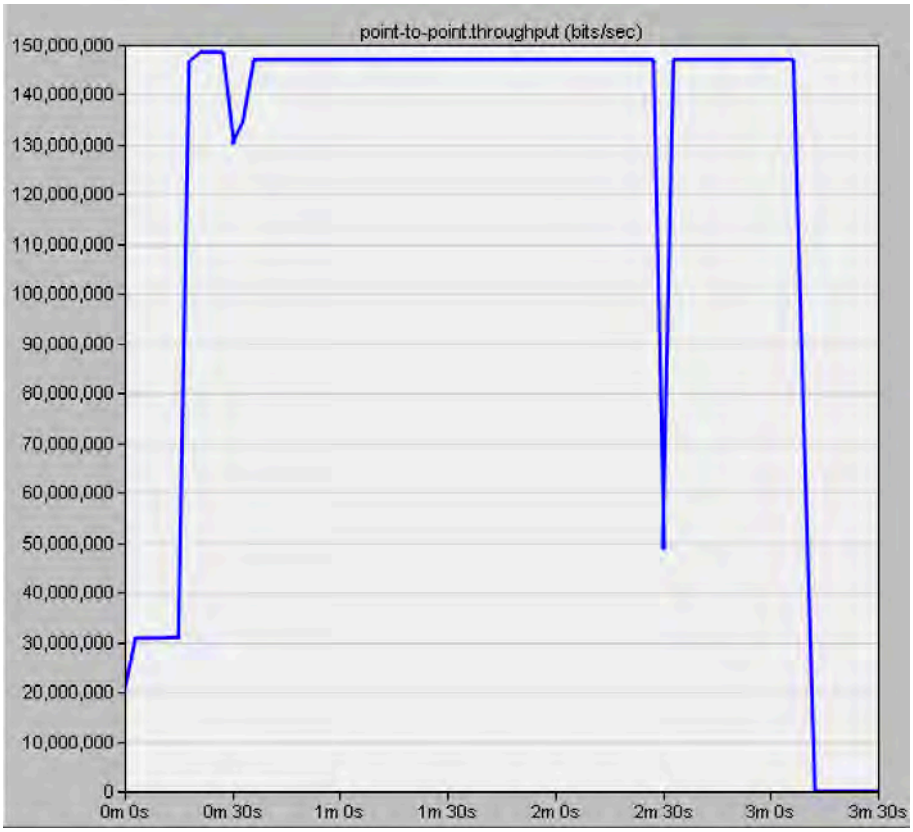

Fig. 80: Prova 3.B.OMBTAP - *Throughput* del coll d'ampolla.

Resultat: Durant el període de congestió el client OMBTAP estima correctament la capacitat de l'enllaç i envia a màxima velocitat amb un *throughput* de 143 Mbps. Com que el flux UDP està fixat a 31 Mbps, la seva velocitat de transferència no varia i es perden paquets MBTAP, fet que provoca una disminució del Sending Rate. Durant el període d'interferències, el control de congestió interpreta que les pèrdues són degudes a un error de canal i el Sending Rate es manté en conseqüència. Després del període de pèrdues, es torna a recuperar la màxima velocitat d'enviament (147,02 Mbps) amb una sola iteració.

5.4.3 Conclusions de la bateria de proves 3.

Davant la presència de tràfic creuat, els protocols MBTAP i OMBTAP són capaços d'estimar correctament la capacitat màxima del canal, fet que els permet enviar a velocitats altes i ocupar la major part de l'ample de banda de l'enllaç. Els resultats obtinguts tant per un protocol com per l'altre en aquest cas són molt semblants. La principal diferència recau a l'hora de trobar-se amb interferències, ja que l'OMBTAP és capaç de detectar aquestes com a pèrdues aleatòries del canal, i no com a pèrdues degudes a congestió, fet que el protocol MBTAP no sap detectar. Això permet que davant d'aquesta situació de pèrdues de paquets aleatòries i la seva posterior desaparició, la velocitat d'enviament màxima es pugui recuperar amb una sola iteració i, per tant, la transferència sigui més eficient (duració de la transferència menor en el cas d'OMBTAP).

Prova	3.A	3.B
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps
Quantitat de dades enviades	3 GB	3 GB
Tipus de tràfic creuat	FTP (TCP)	UDP
Quantitat de tràfic creuat	300 MB	30 Mbps
BW estimat durant congestió (MBTAP)	147,3 Mbps	147,3 Mbps
BW estimat durant congestió (OMBTAP)	147,29 Mbps	147,29 Mbps
***Throughput* MBTAP durant congestió**	147,03 Mbps	143 Mbps
Utilització MBTAP durant congestió	98,9%	96,3%
***Throughput* OMBTAP durant congestió**	147,02 Mbps	143 Mbps
Utilització OMBTAP durant congestió	98,9%	96,3%
Identifica interferències com a pèrdua aleatòria (MBTAP)	No	No
Identifica interferències com a pèrdua aleatòria (OMBTAP)	Sí	Sí
Throughput MBTAP just després d'interferències	146,2 Mbps	128,4 Mbps
Throughput OMBTAP just després d'interferències	147,02 Mbps	147,02 Mbps
Throughput MBTAP final	147,03 Mbps	147,03 Mbps
Throughput OMBTAP final	147,02 Mbps	147,02 Mbps
Duració de la transferència (MBTAP)	181s	177s
Duració de la transferència (OMBTAP)	180s	175s
% millora d'eficiència	0,55%	1,3%

Taula 11: Resultats de la bateria de proves 3.

5.5 Bateria de proves 4: comunicació entre 2 o 3 clients OMBTAP i 1 servidor OMBTAP

L'objectiu d'aquesta bateria de proves és observar com es comporta el protocol OMBTAP en front d'un altre flux OMBTAP, és a dir, com es reparteixen l'ample de banda del canal en el moment de disputa. Per tal propòsit, se substitueix el client d'altres aplicacions per un altre (o dos més) client OMBTAP. Els enllaços WAN i sense fils es mantenen a SONET-3 (148,6 Mbps) i 300 Mbps, respectivament.

Prova	4.A	4.B
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps
Quantitat de dades per cada client OMBTAP	1 GB x 2 clients	1 GB x 3 clients

Taula 12: Característiques de la bateria de proves 4.

5.5.1 Prova 4.A: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades amb cada client OMBTAP (2 clients).

5.5.1.1 MBTAP

A la Fig. 81 es pot observar com el client 2, que inicia la transferència quan el client 1 ja està enviant dades, estima correctament la capacitat màxima del canal tot i haver-hi tràfic creuat, calculant ambdós el mateix valor (147,3 Mbps). Això provoca que els dos enviïn a velocitats altes (veure Fig. 83), tot i que el primer envia a un ritme lleugerament superior perquè el seu SR ja ha tingut temps de pujar respecte el 90% inicial, tal i com s'observa a la Fig. 81. Com que els dos envien gairebé al màxim de la capacitat de l'enllaç, el coll d'ampolla se satura (100% d'utilització, veure Fig. 84) i s'originen molts més paquets perduts que en les proves anteriors (Fig. 82). Concretament, es perden entre 4 i 6 vegades més paquets que en les anteriors proves, de manera que es pot considerar una pèrdua massiva i excessiva dels paquets (el client 1 perd un 30% dels paquets enviats i el client 2 perd un 43% dels paquets enviats).

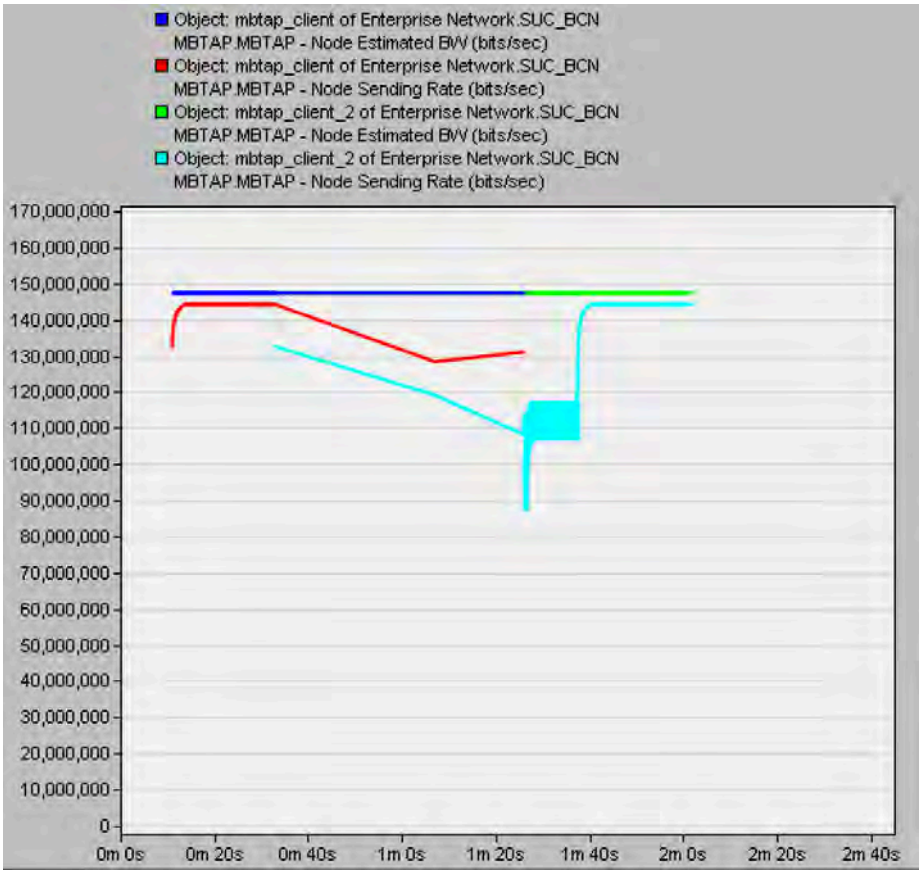

Fig. 81: Prova 4.A.MBTAP - Ample de banda estimat del client 1 (blau) i del client 2 (verd), i Sending Rate del client 1 (vermell) i del client 2 (turquesa).

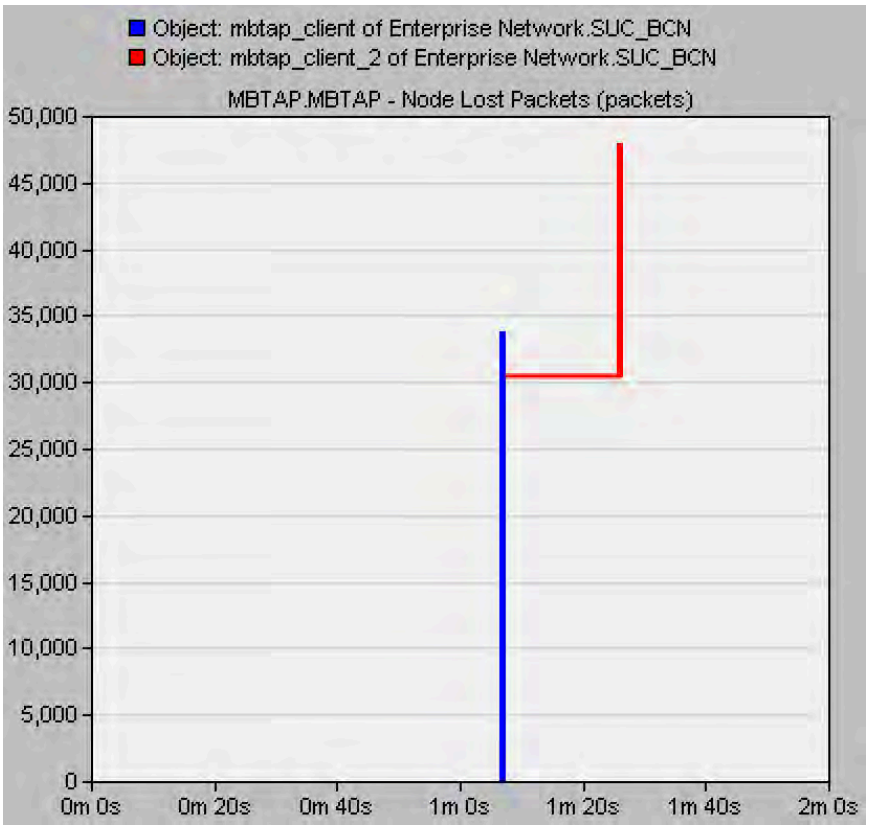

Fig. 82: Prova 4.A.MBTAP – Paquets MBTAP perduts pel client 1 (blau) i pel client 2 (vermell).

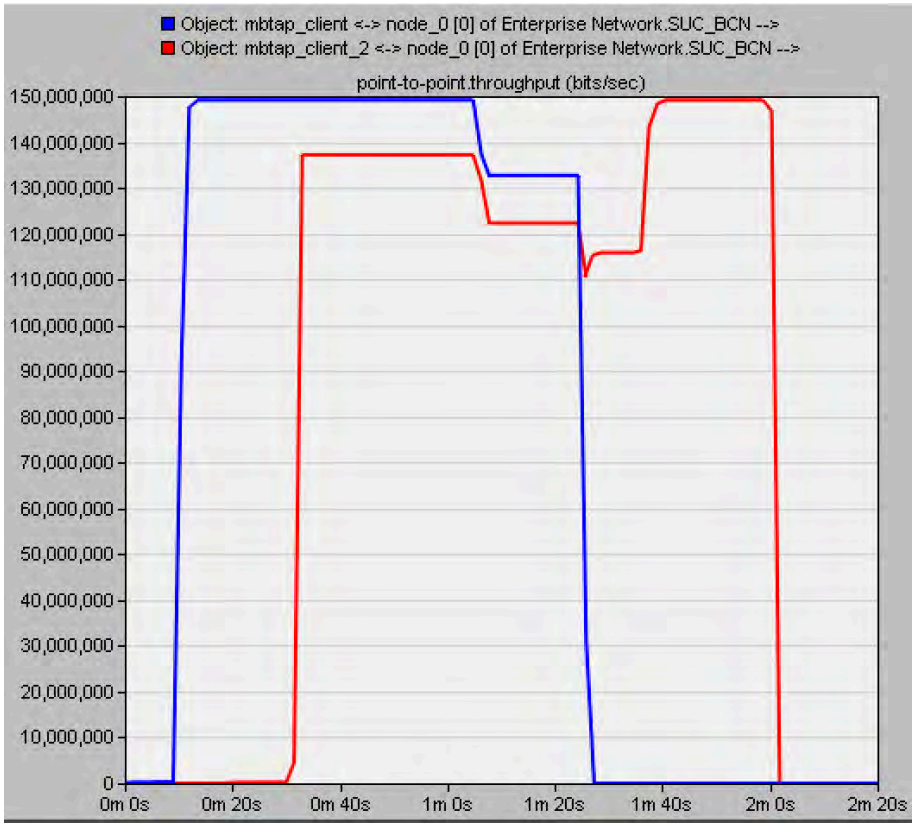

Fig. 83: Prova 4.A.MBTAP - *Througput* del client 1 (blau) i del client 2 (vermell).

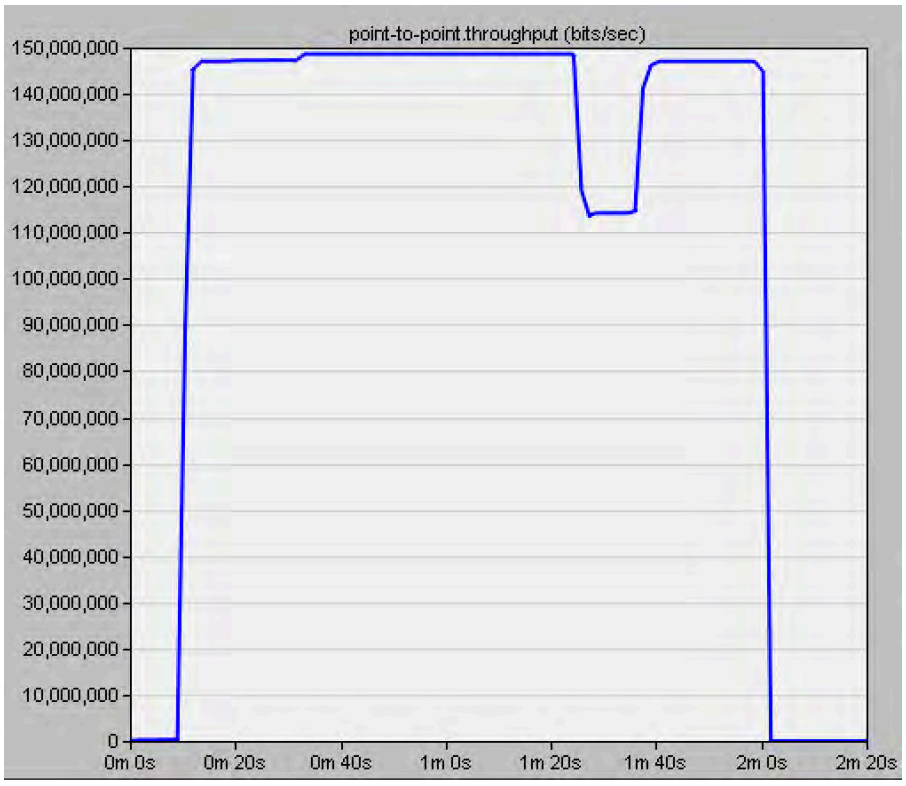

Fig. 84: Prova 4.A.MBTAP - *Througput* del coll d'ampolla.

Resultat: El 2 clients estima correctament la capacitat de l'enllaç (el segon tot i haver-hi tràfic creuat MBTAP). Els 2 clients envien a velocitats altes, obtenint el client 1 un *throughput* més alt perquè el seu SR ha tingut temps d'augmentar abans no arribés el client 2. Es generen moltes pèrdues (36,5%). El repartiment de l'ample de banda és "just" però excessiu tal i com demostra la quantitat de paquets perduts.

5.5.1.2 OMBTAP

En aquest cas, el comportament del protocol OMBTAP respecte el protocol MBTAP (prova anterior) és el mateix, ja que no es generen interferències i totes les pèrdues són degudes a la congestió provocada per la disputa de l'ample agressiva de l'ample de banda entre els dos fluxos.

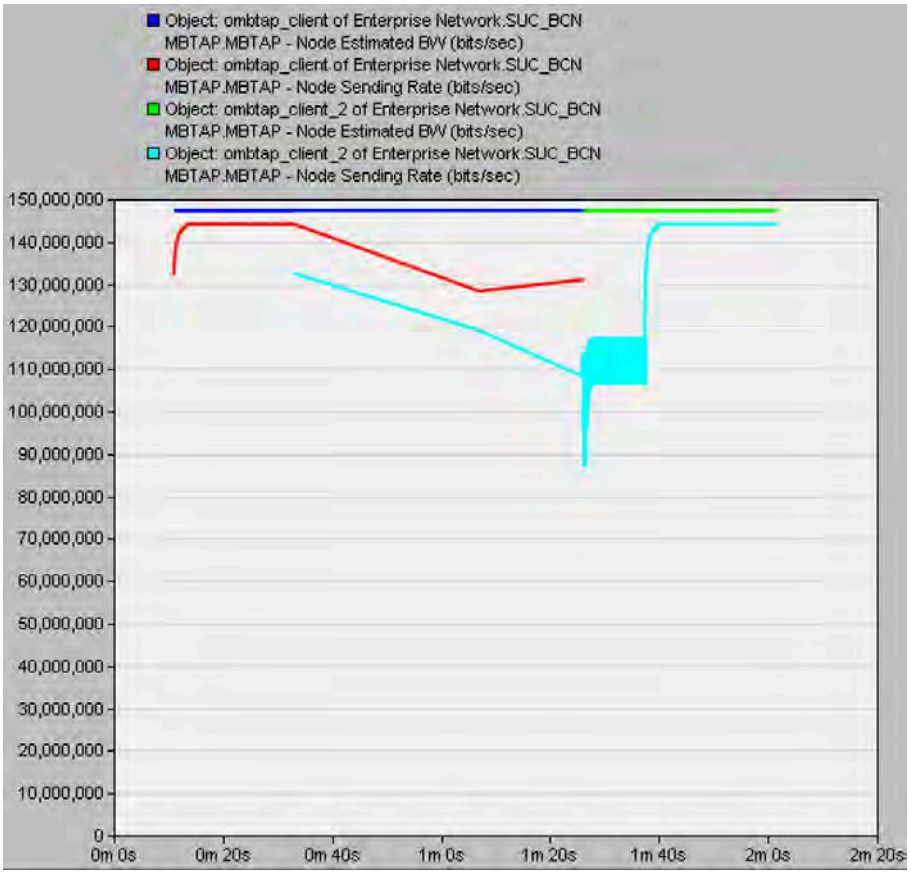

Fig. 85: Prova 4.A.OMBTAP - Ample de banda estimat del client 1 (blau) i del client 2 (verd), i Sending Rate del client 1 (vermell) i del client 2 (turquesa).

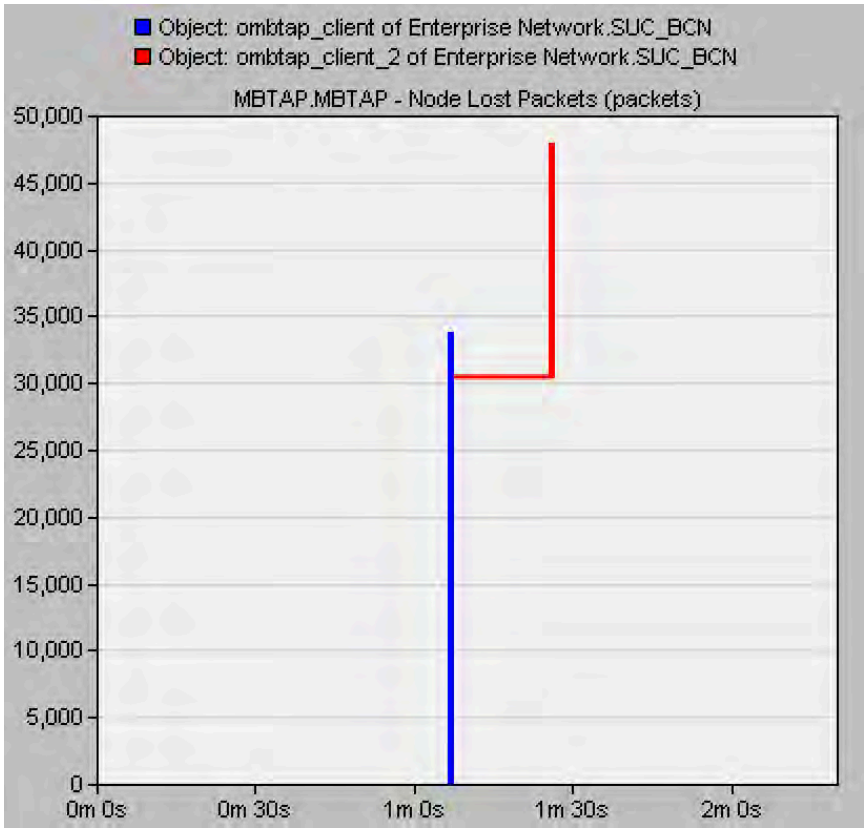

Fig. 86: Prova 4.A.OMBTAP – Paquets OMBTAP perduts pel client 1 (blau) i pel client 2 (vermell).

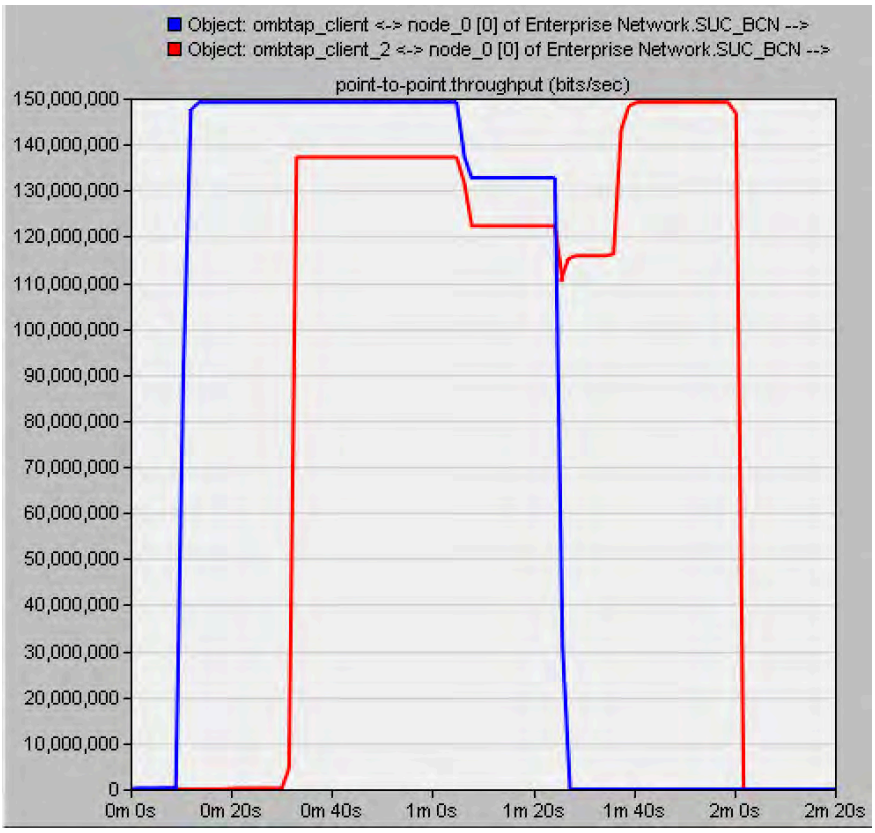

Fig. 87: Prova 4.A.OMBTAP - *Througput* del client 1 (blau) i del client 2 (vermell).

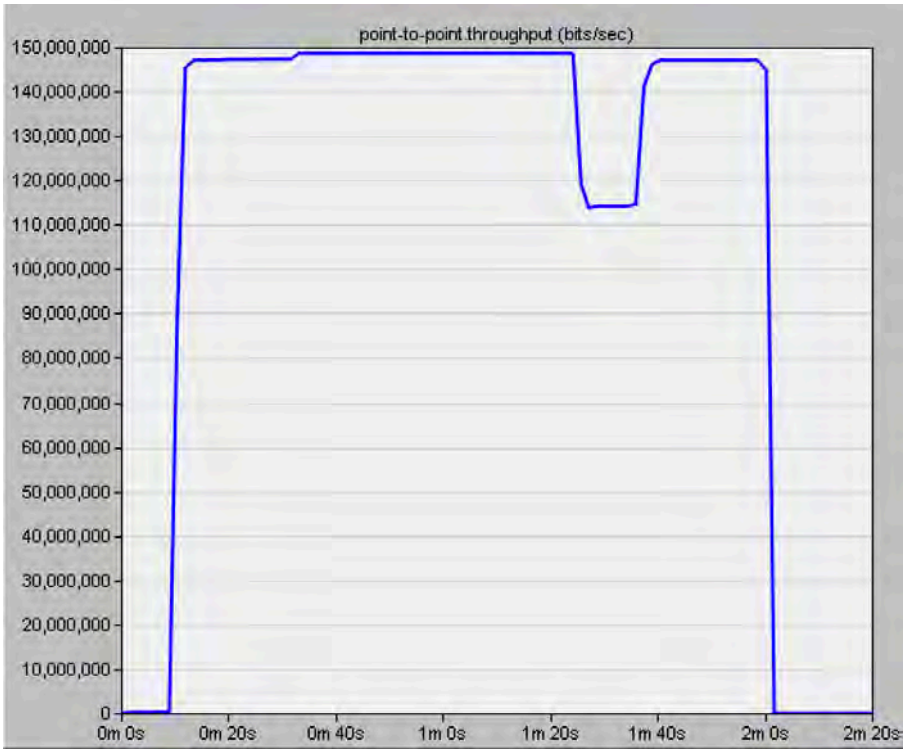

Fig. 88: Prova 4.A.OMBTAP - *Througput* del coll d'ampolla.

Resultat: El 2 clients estimen correctament la capacitat de l'enllaç (el segon tot i haver-hi tràfic creuat MBTAP). Els 2 clients envien a velocitats altes, obtenint el client 1 un *throughput* més alt perquè el seu SR ha tingut temps d'augmentar abans no arribés el client 2. Es generen moltes pèrdues (36,5%). El repartiment de l'ample de banda és "just" però excessiu tal i com demostra la quantitat de paquets perduts..

5.5.2 Prova 4.B: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades amb cada client MBTAP (3 clients).

Aquesta prova no s'ha pogut realitzar amb èxit a causa que el client 2 perd massa paquets quan tots tres clients estan enviant a la vegada i saturen l'enllaç completament (el coll d'ampolla rep el triple de dades de les que pot enviar), de manera que la gestió de tants paquets perduts bloqueja el procés completament i l'enviament de dades deixa de funcionar. Aquesta situació deixa entreveure que el protocol OMBTAP, igual que ja passava amb el MBTAP, necessita un mecanisme de control del *fairness* que sigui capaç de regular l'ample de banda que cada transferència ha d'ocupar sense la necessitat de fer-ho mitjançant la saturació de l'enllaç i la pèrdua massiva de paquets.

Resultat: Simulació fallida.

5.5.3 Conclusions de la bateria de proves 4.

L'estimació de la capacitat del canal amb presència de tràfic creuat OMBTAP és correcta. El repartiment de l'ample de banda és equitatiu, ja que els *throuhputs* dels clients són semblants. Tot i això, com que tots els clients envien a velocitats altes, es perden masses paquets, el que provoca que fins i tot la proba 4.B no es pugui simular a causa de les massives pèrdues. En la següent taula no s'especifica si els resultats són pel protocol OMBTAP o MBTAP perquè els resultats són idèntics.

Prova	4.A	4.B
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps
Quantitat de dades per cada client OMBTAP	1 GB x 2 clients	1 GB x 3 clients
% Pèrdues client 1	30%	-
% Pèrdues client 2	46%	-
BW estimat client 1	147,3 Mbps	-
BW estimat client 2	147,3 Mbps	-
***Throughput* sense congestió client 1**	147,03 Mbps	-
***Throughput* sense congestió client 2**	147,03 Mbps	-
Utilització sense congestió client 1	98,9%	-
Utilització sense congestió client 2	98,9%	-
***Throughput* amb congestió client 1**	147-133 Mbps	-
***Throughput* amb congestió client 2**	136-123 Mbps	-
Utilització amb congestió client 1	98,9-89,5%	-
Utilització amb congestió client 1	91,5-82,8%	-

Taula 13: Resultats de la bateria de proves 4.

5.6 Bateria de proves 5: comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb tràfic creuat i interferències. Variació del valor del LTD

L'objectiu d'aquesta bateria de proves és comprovar com canvia el comportament del control de congestió (diferenciació del tipus de pèrdues) en funció del valor del *Loss Threshold Decissor* (LTD). L'especificació del protocol defineix que aquest valor ha de ser el quocient del número de paquets incrementats per ràfega entre el número de paquets totals per ràfega, basant-se en el fonament teòric que s'explica a l'especificació del protocol JTCP [59]. Així doncs, es provarà quin comportament s'experimenta quan el valor del LTD és inferior o superior a l'especificat. Aquestes proves només es realitzaran amb el protocol OMBTAP, ja que no té sentit realitzar-les també amb el MBTAP (no té implementat el mecanisme de diferenciació de pèrdues).

Prova	5.A	5.B	5.C
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps	148,6 Mbps
Quantitat de dades enviades	1 GB	1 GB	1 GB
Tipus de tràfic creuat	UDP	UDP	UDP
Quantitat de tràfic creuat	30 Mbps	30 Mbps	30 Mbps
Valor LTD	$\frac{0{,}8x\#paquets\ incr.}{\#paquets\ per\ ràfega}$	$\frac{1{,}2x\#paquets\ incr.}{\#paquets\ per\ ràfega}$	$\frac{\#paquets\ incr.}{\#paquets\ per\ ràfega}$

Taula 14: Característiques de la bateria de proves 5.

5.6.1 Prova 5.A: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1GB de dades OMBTAP i 30 Mbps de videoconferència. Valor de LTD inferior a l'especificat.

En aquesta prova es pot observar a la Fig. 89 com, durant el període de congestió, el valor del Sending Rate es manté en lloc de disminuir. Això és degut a que en algunes iteracions el protocol OMBTAP identifica la pèrdua de paquets com a conseqüència d'una interferència, fet que no es així. D'aquesta manera, la velocitat d'enviament no disminueix prou i es perden 4552 paquets (Fig. 90), fet que allarga la transferència de dades (1 minut i 3 segons) . Per altra banda es pot veure que durant el període

d'interferències (segon 50) sí que es detecta correctament la causa de les pèrdues i el Sending Rate es manté en lloc de disminuir.

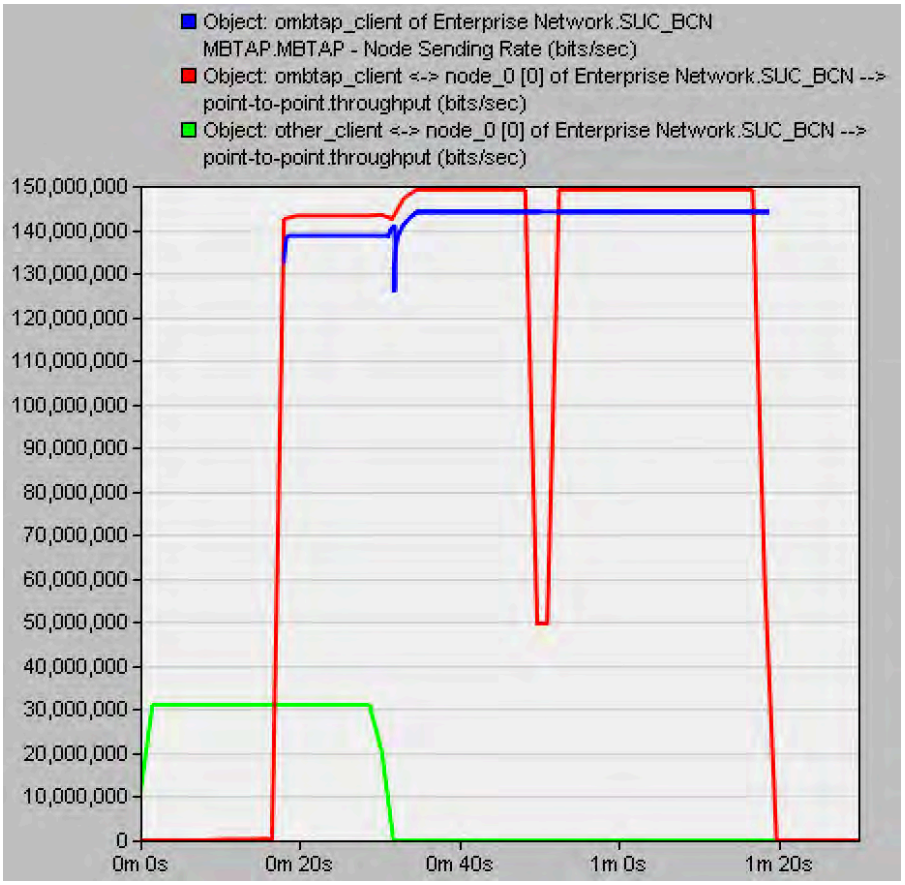

Fig. 89: Prova 5.A.OMBTAP – Sending Rate calculat (blau), *Througput* del client OMBTAP (vermell) i del client UDP (verd).

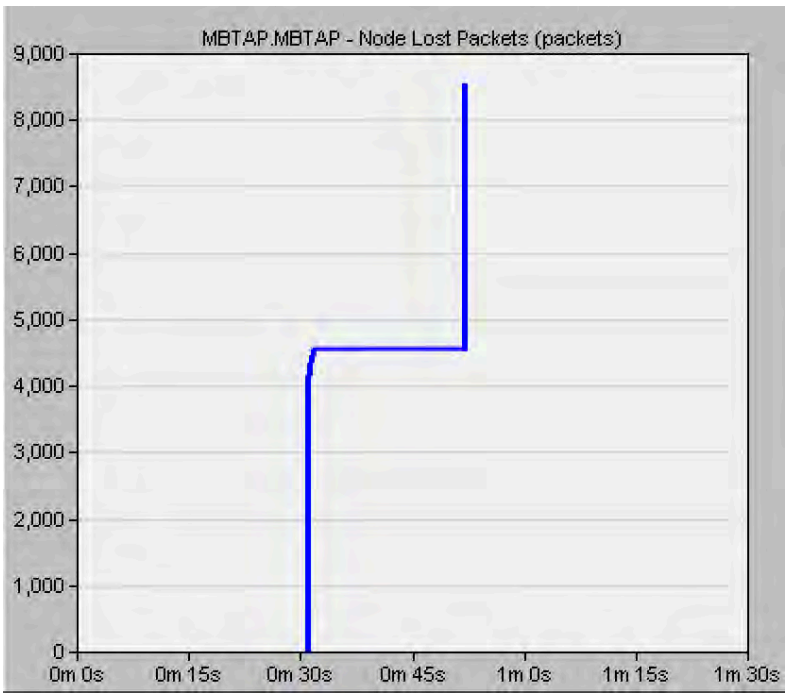

Fig. 90: Prova 5.A.OMBTAP - Paquets OMBTAP perduts.

Resultat: Durant el període de congestió el client OMBTAP no interpreta correctament la causa de les pèrdues, de manera que el Sending Rate es manté en lloc de disminuir, fet que provoca una notable pèrdua de paquets i l'allargament de la transferència. Durant el període d'interferències, sí que s'identifica correctament la causa de les pèrdues en tot moment.

5.6.2 Prova 5.B: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1GB de dades OMBTAP i 30 Mbps de videoconferència. Valor de LTD superior a l'especificat.

En aquesta prova es pot observar a la Fig. 91 com, durant el període de congestió, el valor del Sending Rate dismueix. Per tant, les pèrdues s'identifiquen correctament com a conseqüència d'aquesta congestió, mantenint la transferència estable. La quantitat de paquets perduts en aquesta fase és inferior que en la prova anterior, en aquest cas 4254(veure Fig. 92). D'altra banda, durant el període d'interferències (segon 50), s'observa que en ocasions el Sending Rate disminueix. Això indica que el protocol OMBTAP està determinant incorrectament que la congestió és la causa de les pèrdues. D'aquesta manera, la transferència de fitxers és menys eficient, ja que en baixar la velocitat d'enviament el temps de transferència és superior (1minut i 5 segons).

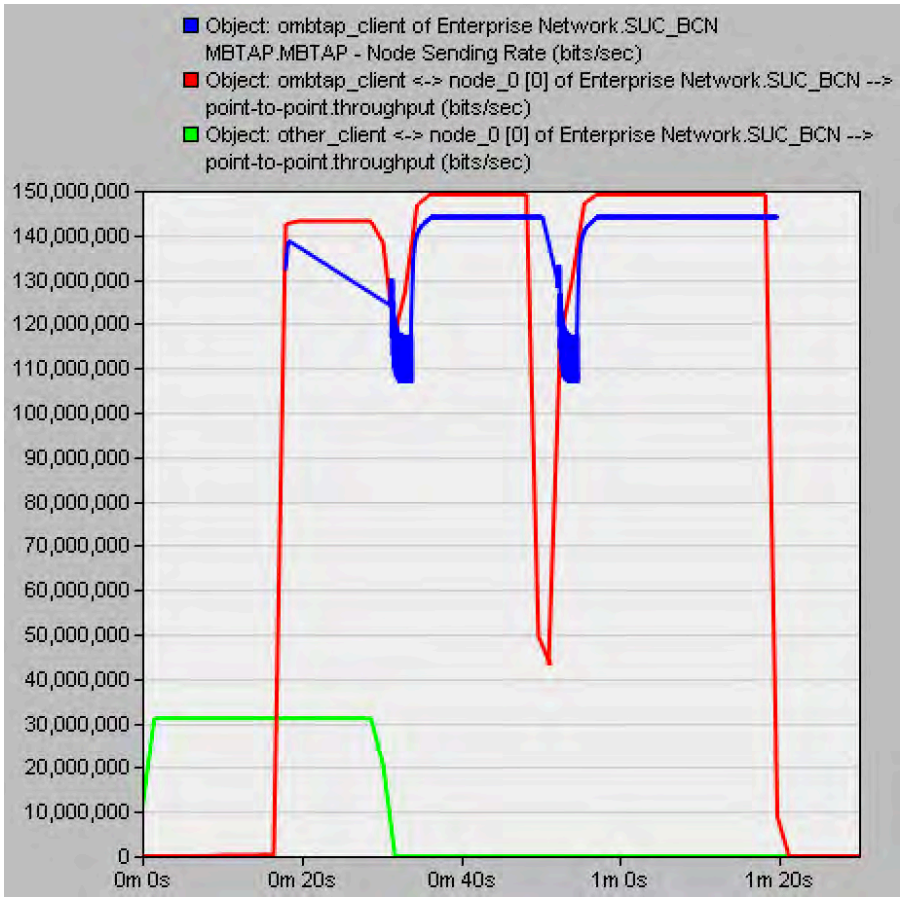

Fig. 91: Prova 5.B.OMBTAP – Sending Rate calculat (blau), *Througput* del client OMBTAP (vermell) i del client UDP (verd).

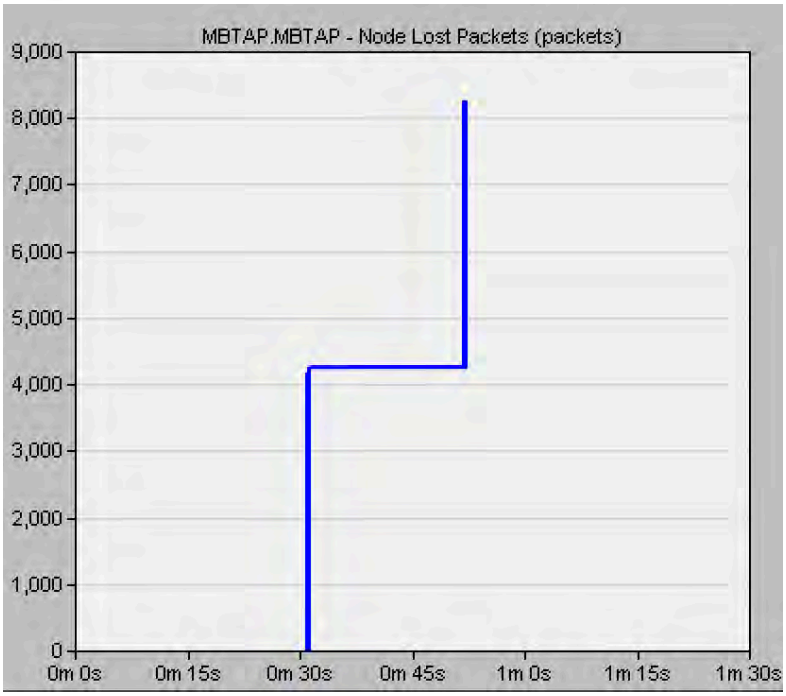

Fig. 92: Prova 5.B.OMBTAP - Paquets OMBTAP perduts.

Resultat: Durant el període d'interferències el client OMBTAP no interpreta correctament la causa de les pèrdues, de manera que el Sending Rate en ocasions disminueix en lloc de mantenir-se, fet que provoca una menor velocitat i un major temps de transferència (menys eficiència). Durant el període de congestió, sí que s'identifica correctament la causa de les pèrdues en tot moment.

5.6.3 Prova 5.C: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1GB de dades OMBTAP i 30 Mbps de videoconferència. Ús del valor de LTD'especificat.

En aquesta prova es pot observar a la Fig. 93 com, tant en el període de congestió com en el d'interferències, es diferencia correctament la causa de les pèrdues. Això provoca que en el primer tram de pèrdues que el Sending Rate disminueixi i, per tant,que es perdin menys paquets (4254, veure Fig. 94). Per altra banda, en el segon tram de pèrdues el Sending Rate es manté, fet que provoca una menor duració i major eficiència de la transferència (1 minut i 2 segons).

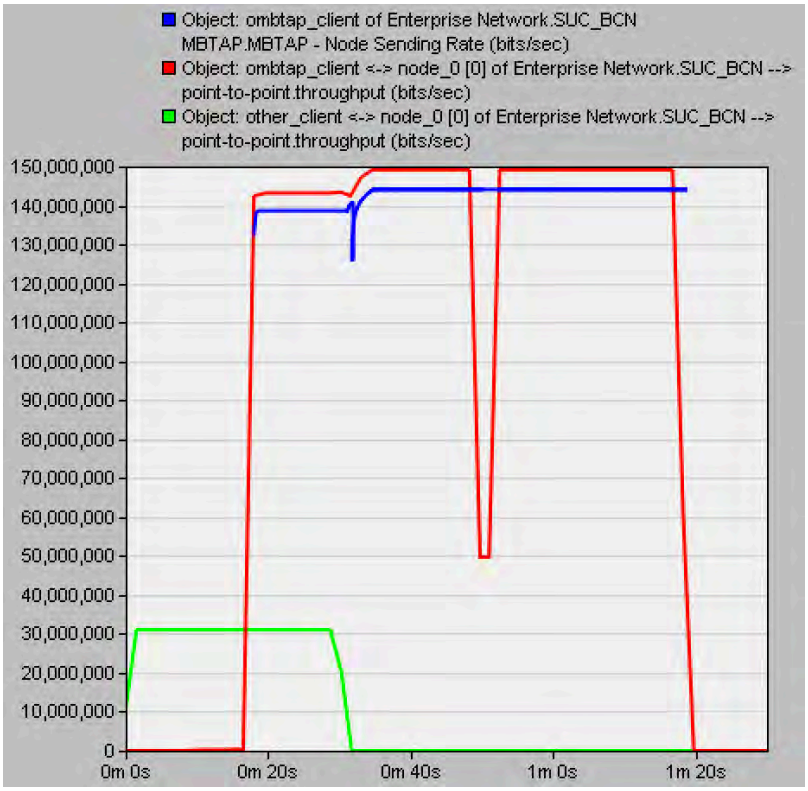

Fig. 93: Prova 5.C.OMBTAP – Sending Rate calculat (blau), *Througput* del client OMBTAP (vermell) i del client UDP (verd).

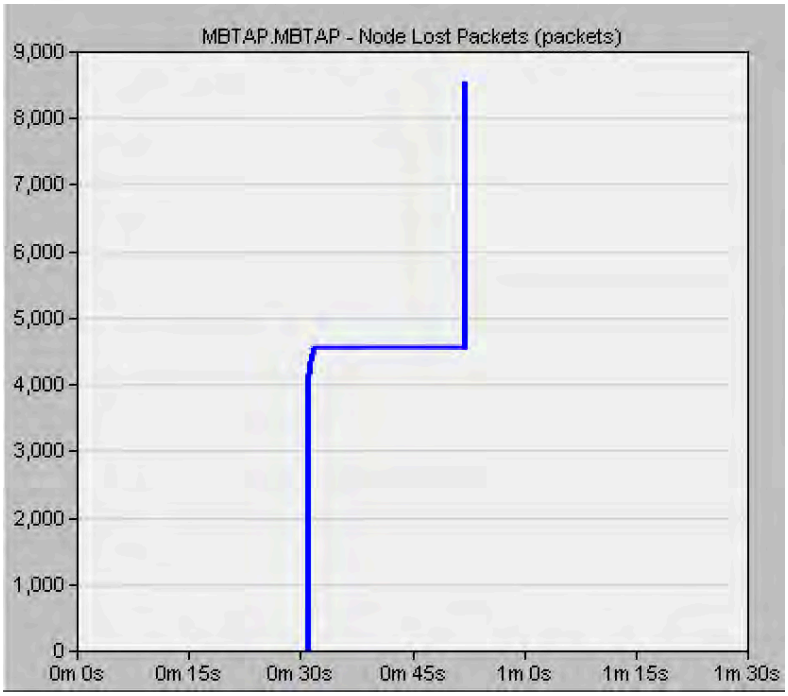

Fig. 94: Prova 5.C.OMBTAP - Paquets OMBTAP perduts.

Resultat: Tant pel període de congestió com pel període d'interferències el protocol OMBTAP discerneix correctament entre pèrdues per congestió i pèrdues aleatòries del canal, fet que ajuda a que es perdin menys paques (més estabilitat) i que disminueixi la duració de la transferència (més eficiència).

5.6.4 Conclusions de la bateria de proves 5.

Quan es defineix per al LTD un valor menor que el definit en l'especificació del protocol OMBTAP, en ocasions s'identifiquen pèrdues originades per la congestió com a pèrdues aleatòries, fet que provoca un augment dels paquets perduts en no disminuir el Sending Rate (desestabilització de la transferència). D'altra banda, quan es defineix per al LTD un valor superior al defnit en l'especificació del protocol OMBTAP, en ocasions s'identifiquen pèrdues originades per interferències com a pèrdues per congestió, fet que provoca una menor *throughput* i un major temps de transferència en disminuir el Sending Rate (pitjor eficiència de transferència). En canvi, quan s'utilitza el valor per al LTD definit a l'especificació, s'observa que s'identifica correctament la causa de les pèrdues de paquets, per tant millora l'estabilització i l'eficiència de la transferència. Això reforça la idea inicial durant el disseny del protocol de fixar el valor del LTD a partir de la fórmula $LTD_{OMBTAP} = \frac{\#\ Paquets\ incrementats}{\#\ Paquets\ ràfega}$, que està basat en els fonaments teòrics i l'especificació del protocol JTCP.

Prova	5.A	5.B	5.C
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps	148,6 Mbps
Quantitat de dades enviades	1 GB	1 GB	1 GB
Tipus de tràfic creuat	UDP	UDP	UDP
Quantitat de tràfic creuat	30 Mbps	30 Mbps	30 Mbps
Valor LTD	$\frac{0{,}8x\#paquets\ incr.}{\#paquets\ per\ ràfega}$	$\frac{1{,}2x\#paquets\ incr.}{\#paquets\ per\ ràfega}$	$\frac{\#paquets\ incr.}{\#paquets\ per\ ràfega}$
Detecció correcta pèrdues per congestió	No	Sí	Sí
Detecció correcta pèrdues per canal	Sí	No	Sí
Velocitat de transferència	63s	65s	62s
% Pèrdues	7,7%	7,4%	7,4%

Taula 15: Resultats de la bateria de proves 5.

5.7 Bateria de proves 6: comunicació entre 1 client OMBTAP i 1 servidor OMBTAP amb interferències de duració variable.

En aquesta sèrie de proves es pretén observar com varia l'eficiència (duració) de la transferència de dades segons la duració del període d'interferències (1s, 10s i 30s), per veure si existeix algun tipus de relació o proporcionalitat entre aquests dos factors. En totes les proves l'escenari és el mateix, amb un enllaç WAN SONET-3 (148,6 Mbps) i un enllaç sense fils de 300 Mbps, variant només la duració de les interferències.

Prova	6.A	6.B	6.C
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps	148,6 Mbps
Quantitat de dades enviades	1 GB	1 GB	1 GB
Duració de les interferències	1s	10s	30s

Taula 16: Característiques de la bateria de proves 6.

5.7.1 Prova 6.A: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades i 1 segon d'interferències.

5.7.1.1 MBTAP

La Fig. 95 mostra de nou que el protocol MBTAP interpreta incorrectament les pèrdues per interferència com a pèrdues de congestió, disminuint el Sending Rate i la velocitat d'enviament durant aquest interval de temps. La duració total de la transferència és de 59 segons.

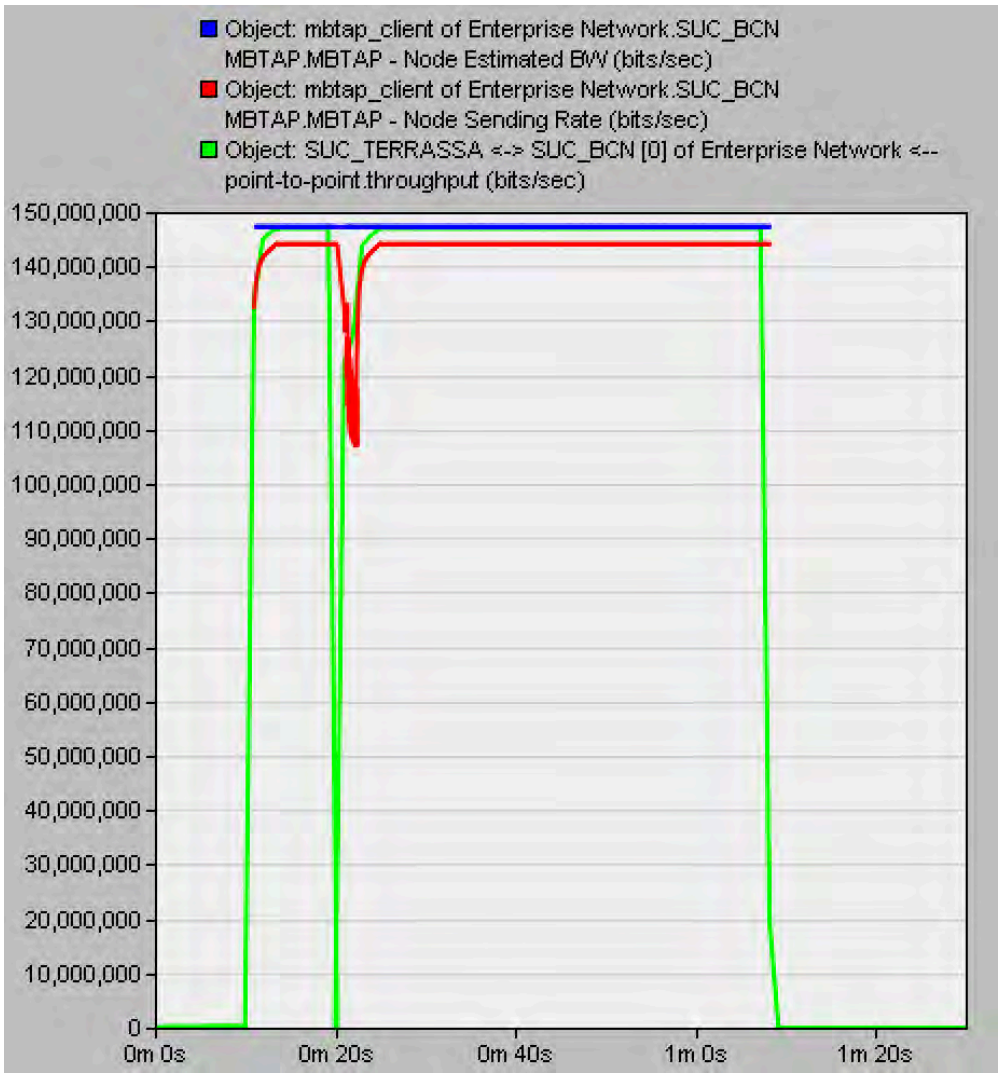

Fig. 95: Prova 6.A.MBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

Resultat: Tal i com ja se sap de les simulacions anteriors, durant el període d'interferències el control de congestió interpreta les pèrdues com a congestió del canal i el Sending Rate disminueix en conseqüència, baixant la velocitat d'enviament.

5.7.1.2 OMBTAP

A la Fig. 96 es veu novament que el protocol OMBTAP sí identifica les pèrdues per interferència com error del canal i manté el Sendig Rate. Això permet que es recuperi la màxima velocitat d'enviament i que, per tant, disminueixi el temps de transferència, que en aquest cas és de 58 segons (un 1,69% més ràpida que en el cas anterior).

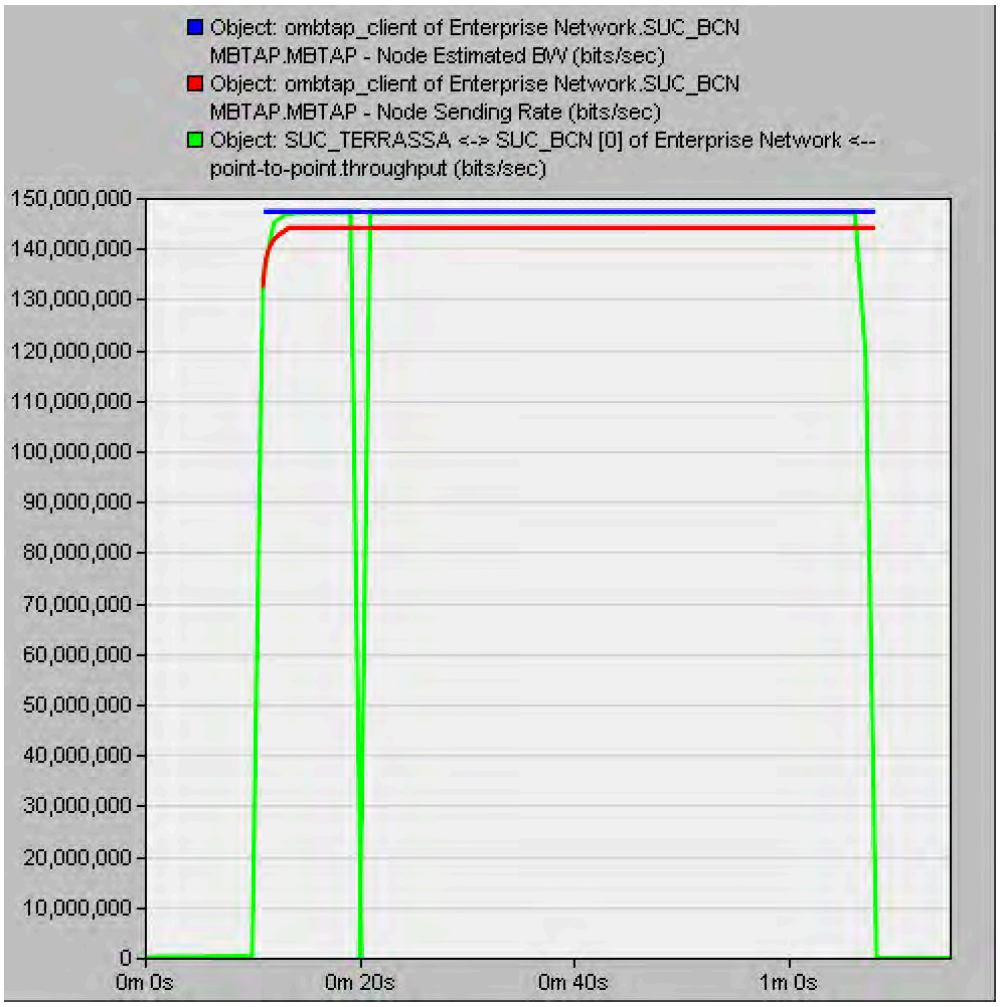

Fig. 96: Prova 6.A.OMBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

Resultat: Gràcies al nou mecanisme de congestió d'OMBTAP s'aconsegueix reduir el temps de transferència quan apareixen interferències de curta duració.

5.7.2 Prova 6.B: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades i 10 segons d'interferències.

5.7.2.1 MBTAP

La Fig. 97 mostra de nou que el protocol MBTAP interpreta incorrectament les pèrdues per interferència com a pèrdues de congestió, disminuint el Sending Rate i la velocitat d'enviament durant aquest interval de temps. La duració total de la transferència és d'1 minut i 11 segons.

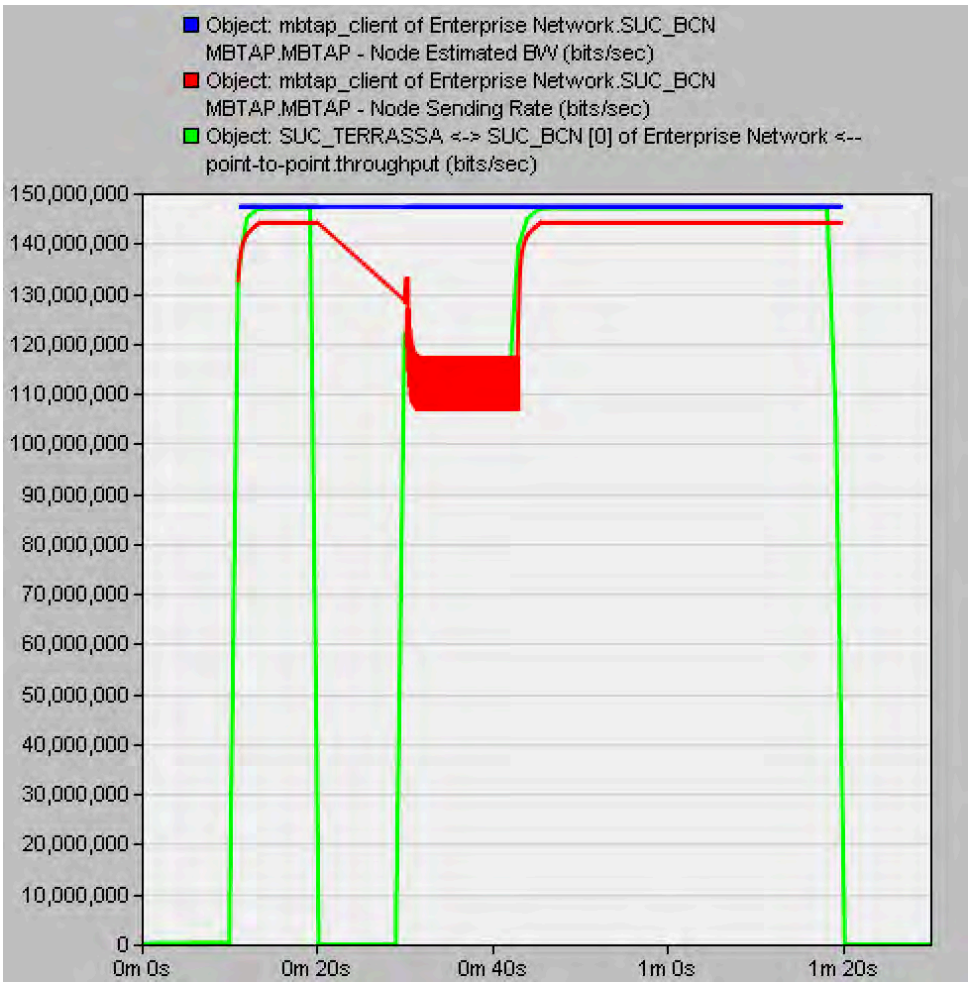

Fig. 97: Prova 6.B.MBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

Resultat: Tal i com ja se sap de les simulacions anteriors, durant el període d'interferències el control de congestió interpreta les pèrdues com a congestió del canal i el Sending Rate disminueix en conseqüència, baixant la velocitat d'enviament.

5.7.2.2 OMBTAP

A la Fig. 98 es veu novament que el protocol OMBTAP sí identifica les pèrdues per interferència com error del canal i manté el Sendig Rate. Això permet que es recuperi la màxima velocitat d'enviament i que, per tant, disminueixi el temps de transferència, que en aquest cas és d'1 minut i 7 segons (un 5,63% més ràpida que en el cas anterior).

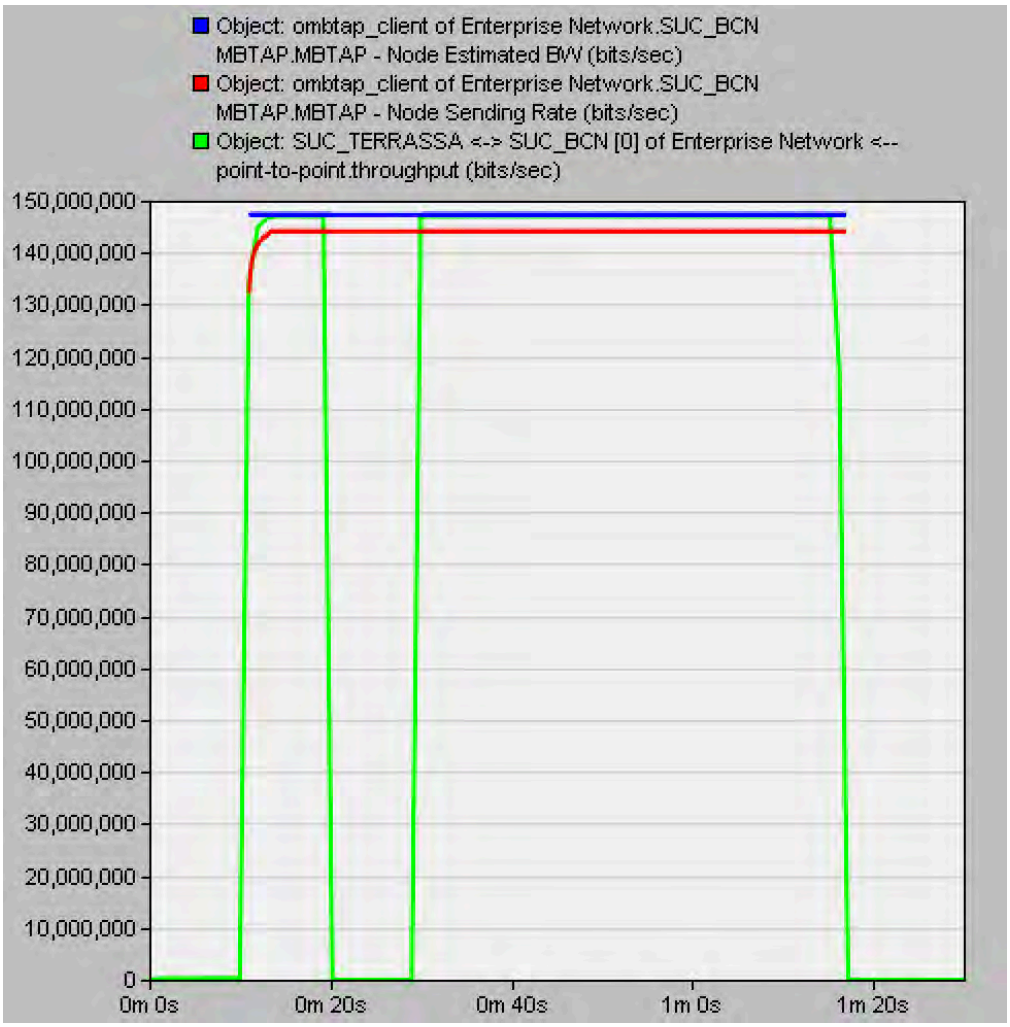

Fig. 98: Prova 6.B.OMBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

Resultat: Gràcies al nou mecanisme de congestió d'OMBTAP s'aconsegueix reduir el temps de transferència quan apareixen interferències de mitjana duració. La millora d'eficiència en valor relatiu és superior respecte les interferències de curta duració.

5.7.3 Prova 6.C: Enllaç WAN SONET-3 (148,6 Mbps) i enllaç sense fils de 300 Mbps, enviament d'1 GB de dades i 30 segons d'interferències.

5.7.3.1 MBTAP

La Fig. 99 mostra de nou que el protocol MBTAP interpreta incorrectament les pèrdues per interferència com a pèrdues de congestió, disminuint el Sending Rate i la velocitat d'enviament durant aquest interval de temps. La duració total de la transferència és d'1 minut i 38 segons.

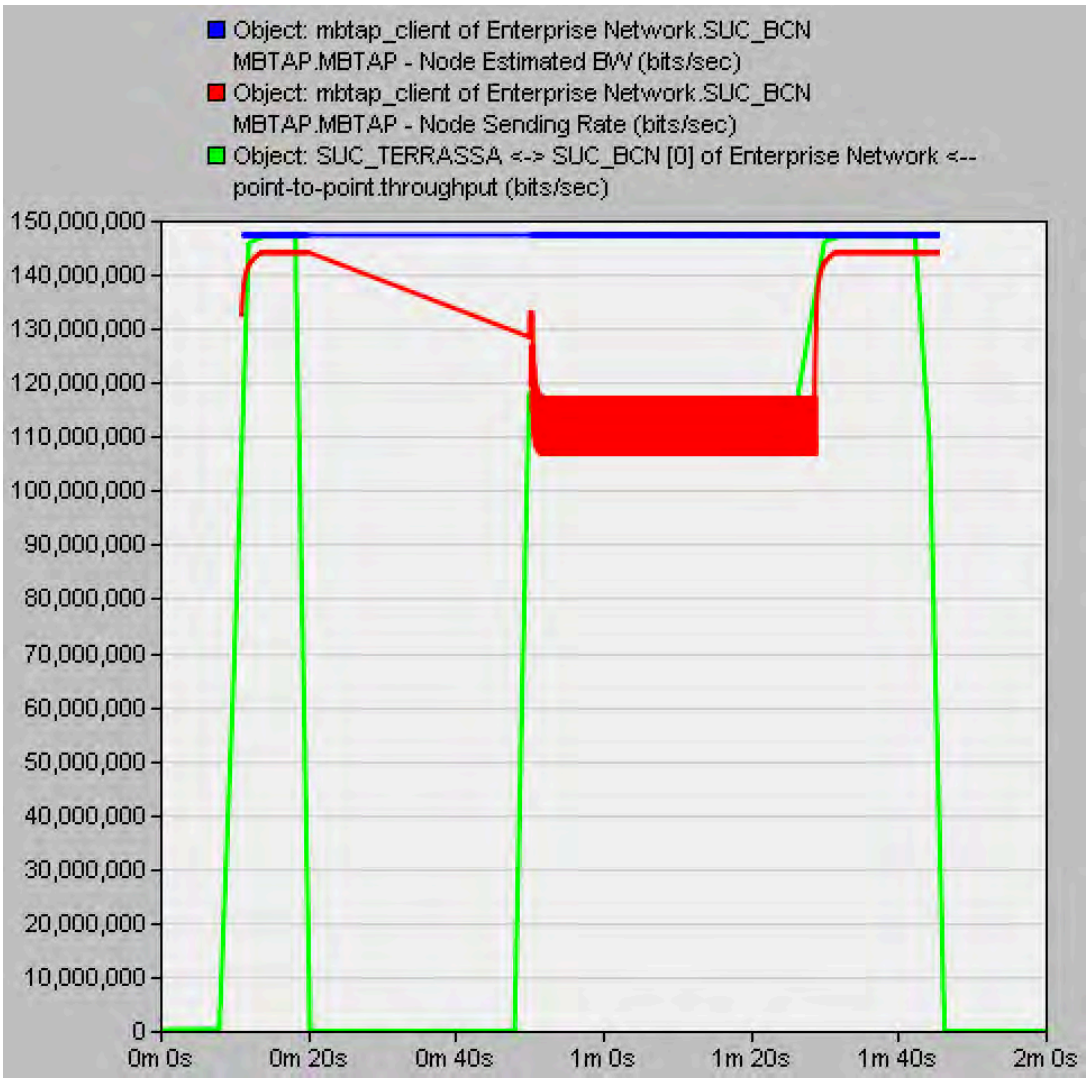

Fig. 99: Prova 6.C.MBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

Resultat: Tal i com ja se sap de les simulacions anteriors, durant el període d'interferències el control de congestió interpreta les pèrdues com a congestió del canal i el Sending Rate disminueix en conseqüència, baixant la velocitat d'enviament.

5.7.3.2 OMBTAP

A la Fig. 100 es veu novament que el protocol OMBTAP sí identifica les pèrdues per interferència com error del canal i manté el Sendig Rate. Això permet que es recuperi la màxima velocitat d'enviament i que, per tant, disminueixi el temps de transferència, que en aquest cas és d'1 minut i 30 segons (un 8,16% més ràpida que en el cas anterior).

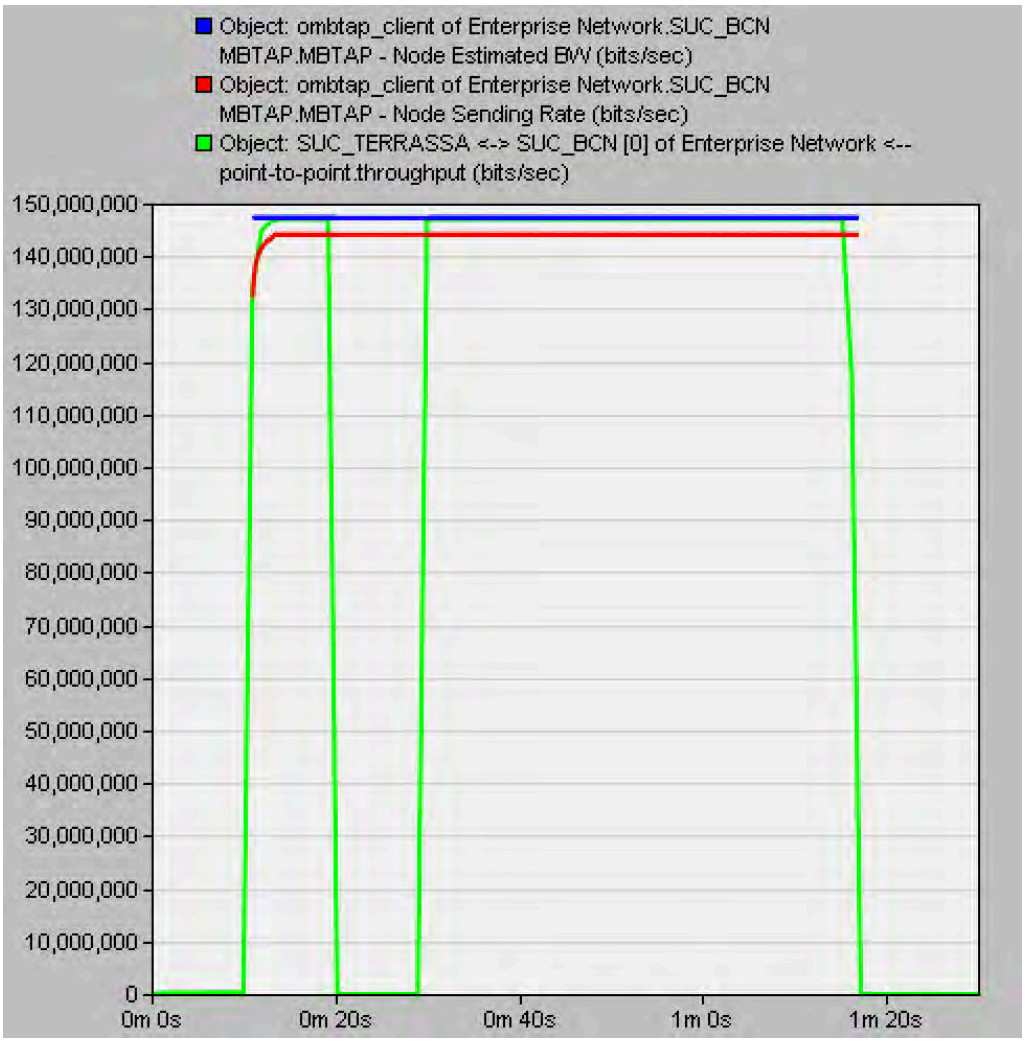

Fig. 100: Prova 6.C.OMBTAP - Ample de banda estimat (blau), Sendin Rate calculat (vermell) i *throughput* del coll d'ampolla (verd).

Resultat: Gràcies al nou mecanisme de congestió d'OMBTAP s'aconsegueix reduir el temps de transferència quan apareixen interferències de llarga duració. La millora d'eficiència en valor relatiu és superior respecte les interferències de mitjana i curta duració.

5.7.4 Conclusions de la bateria de proves 6.

Després d'observar les diferències entre el rendiment dels protocols MBTAP i OMBTAP davant d'interferències, s'observa que aquest segon aconsegueix una millor eficiència (duració de la transferència de dades menor) en tots els casos. A més, el percentatge de millora de l'eficiència és directament proporcional al temps d'interferències.

D'altra banda, resulta obvi que com més temps durin les interferències, més paquets es perdran, ja que l'emissor creu que els paquets que envia s'estan rebent, i no s'adona que no es així fins que no rep el SACK del receptor quan les interferències desapareixen.

Prova	6.A	6.B	6.C
Capacitat coll d'ampolla	148,6 Mbps	148,6 Mbps	148,6 Mbps
Quantitat de dades enviades	1 GB	1 GB	1 GB
Duració de les interferències	1s	10s	30s
Temps de transferència (MBTAP)	59s	71s	98s
Temps de transferència (OMBTAP)	58s	67s	90s
% millora d'eficiència	1,69%	5,63%	8,16%
% pèrdues	1,8%	17,95%	53,86%

Taula 17: Resultats de la bateria de proves 6.

5.8 Conclusions de les simulacions

Després d'analitzar les quatre bateries de proves realitzades, se'n poden treure 4 conclusions principals:

- Tot el mecanisme d'intercanvi de dades funciona correctament: les dades s'envien a la **màxima velocitat** possible (**98,9%** d'utilització del canal, prova 1.A), el control de congestió actua tal i com s'espera a partir de les fórmules especificades i el control de pèrdues actua de forma òptima, detectant els paquets perduts sol·licitant-ne el seu reenviament i retransmetent-los (proves 2.A i 2.B).
- En el cas que el coll d'ampolla sigui un enllaç cablejat, el mecanisme d'**estimació de l'ample de banda** és capaç de mesurar un valor que s'ajusta al real amb molta precisió, fins i tot quan l'enllaç es troba congestionat amb **tràfic creuat** (**147,3 Mbps**, proves 3.A i 3.B). Això permet que l'enviament de dades sigui el màxim de ràpid possible i que el protocol sigui agressiu, tal i com es pretén en el seu disseny inicial (**96,3% d'utilització**, prova 3.B). Per contra, quan el coll d'ampolla és un **enllaç sense fils**, no s'estima el valor teòric màxim del canal, sinó un valor inferior que deu ser pròxim a la velocitat màxima real de l'enllaç (**70,6%** i **68,3%** sobre el **màxim teòric**, proves 1.B i 1.C respectivament), ja que altres fluxos TCP o UDP tampoc són capaços d'augmentar aquest *throughput.* Això és degut a les deficiències que presenta CSMA/CA per diversos factors, com per exemple la distància entre dos nodes.
- El nou control de congestió pel protocol OMBTAP és capaç de **discernir** correctament entre les situacions de **pèrdues** originades per una **congestió** de l'enllaç i les situacions de **pèrdues** originades per un **error aleatori** del canal (deteccions correctes en les proves 3.A i 3.B). Aquest fer permet, davant de situacions de pèrdues aleatòries, **no disminuir el Sending Rate** i recuperar la velocitat de transmissió (*throughput*) màxima de la forma més ràpida possible, **augmentant l'eficiència** de la transferència (**millora** d'eficiència del **0,55%** i de l'**1,3%** en les proves 3.A i 3.B, respectivament) .
- És **necessari** dissenyar i implementar un mecanisme de ***fairness*** per tal que quan diversos fluxos MBTAP/OMBTAP comparteixin el mateix canal, aquest **no se saturi** en excés, originant una **pèrdua massiva** de paquets (pèrdues del **30%** i **46%** de paquets, prova 4.A), i s'aconsegueixi un **repartiment just** de l'ample de banda.
- El **valor del llindar LTD** perquè el control de congestió diferenciï entre pèrdues aleatòries i pèrdues per congestió que es va definir en l'especificació del protocol OMBTAP és **correcte**. Quan s'utilitza el valor de LTD especificat, **s'identifica correctament** la causa de les **pèrdues** de paquets, fent que aquesta pèrdua i la duració de la transferència siguin les òptimes (**7,3% de pèrdues** i **62 segons de duració**, prova 5.C) Quan aquest és inferior al definit (**LTD 20% inferior** a la prova 5.A) les situacions de pèrdues per congestió es poden **mal interpretar**

com a pèrdues aleatòries, causant una **major pèrdua** de paquets (**7,7% de pèrdues**, prova 5.A) i un **major temps** de transferència (**63 segons**, prova 5.A). D'altra banda, quan el valor del LTD és superior al definit (**LTD 20% superior** a la prova 5.B) les situacions de pèrdues per congestió es poden **mal interpretar** com a pèrdues aleatòries, fet que no causa una major pèrdua de paquets (7,4%, prova 5.B) però sí una **duració de la transferència** de dades **superior** (**65 segons**, prova 5.B).

- La **millora percentual de l'eficiència**, és a dir, la reducció del temps de transferència en valor relatiu **entre els protocols MBTAP i OMBTAP** és **directament proporcional** a la **duració de les interferències**, tal i com es demostra en les proves 6.A, 6.B i 6.C. Quan les interferències són de **curta duració** (1 segon, prova 6.A) es **redueix** el temps de transferència un **1,69%**. Quan les interferències són de **mitja duració** (10 segons, prova 6.B) es **redueix** el temps de transferència un **5,63%**. Finalment, quan les interferències són de **llarga duració** (30 segons, prova 6.C) es **redueix** el temps de transferència un **8,16%**.

- La **quantitat de paquets** perduts també és **directament proporcional** a la **duració de les interferències**. Quan les interferències són de **curta** duració (1 segon, prova 6.A) es perden un **1,8%** dels paquets. En interferències de **mitjana** duració (10 segons, prova 6.B) les pèrdues son del **17,95%**. Finalment, amb interferències de **llarga** duració (30 segons, prova 6.C) s'observa un **53,86%** de pèrdues. Això, però, és una característica intrínseca dels canals sense fils, que es veuen exposats davant d'aquests tipus de pèrdues. I per tant, és quelcom que la resta de protocols de transport també experimenten. Per evitar aquesta pèrdua massiva de paquets s'hauria d'implementar un mecanisme de *timeout* que identifiqués un paquet com a perdut si al cap de N segons no s'ha rebut un SACK corresponent al packet, reduint així el Sending Rate (si el SR es mantingués, es perdrien els mateixos paquets, ja que se seguiria enviant a la mateixa velocitat, i la única diferència seria que s'enviarien paquets repetits en lloc de nous paquets, fet que no aporta cap benefici). Tot i que disminuir la velocitat d'enviament ajudaria a reduir el nombre de paquets perduts davant de llargues interferències, això comportaria que quan aquestes desapareguessin es reprendria la transferència a una velocitat menor, resultant en una transferència menys eficient. I això, precisament, és el que pretén evitar el protocol OMBTAP.

6 Conclusions finals i línies de futur

En aquest treball s'ha analitzat l'estat de l'art dels protocols de nivell de transport i els seus controls de congestió, així com la problemàtica de les Long Fast Networks (LFN). En especial, s'ha fet èmfasi en les LFN heterogènies (xarxes amb enllaços sense fils) i la dificultat dels controls de congestió per diferenciar pèrdues originades per congestió i pèrdues originades per errors del canal. Aquesta problemàtica comporta que les pèrdues sempre siguin considerades com a conseqüència d'una congestió, obligant els protocols de transport a reduir el seu *throughput*, fet que suposa una transferència de dades menys eficient.

Per donar resposta a aquesta carència s'ha presentat el protocol OMBTAP, una versió corregida i millorada del seu antecessor MBTAP. Aquest protocol de nivell de transport ha estat dissenyat per tenir un comportament agressiu, i està orientat per escenaris de xarxes heterogènies LFN on s'hi hagin de realitzar transferències d'una gran quantitat de dades amb alta prioritat.

També s'ha descrit el procés d'implementació d'aquest protocol dins del simulador Riverbed Modeler, explicant totes les singularitats que envolten el fet d'adaptar el protocol a un simulador de xarxa així com la millor manera de controlar a voluntat el comportament dels enllaços sense fils.

A continuació, s'han realitzat un seguit de proves (simulacions) i se n'han analitzat els seus resultats. S'ha comprovat que, efectivament, es manté el comportament del protocol MBTAP en aquells casos on el seu funcionament ja era el desitjat. Això implica que davant d'un canal lliure, l'estimació de l'ample de banda i la utilització d'aquest és màxima (98,9%). Davant la presència de tràfic creuat, l'estimació de la capacitat màxima de l'enllaç segueix sent acurada, mentre que el comportament agressiu del protocol permet ocupar la majoria de l'ample de banda de l'enllaç quan s'està competint amb altres fluxos TCP o UDP. A més, davant situacions de congestió que originin pèrdues de paquets, el control de congestió de protocol és capaç de disminuir el Sending Rate, reenviar els paquets perduts i recuperar l'ample de banda perdut quan la situació de pèrdues desapareix.

Gràcies als nous mecanismes de l'OMBTAP, a més de tot el comentat fins ara, el protocol és capaç de diferenciar situacions de pèrdues causades per congestió respecte situacions de pèrdues causades per errors aleatoris del canal (esvaïment, multicamí, obstacles...). S'ha pogut comprovar que el valor definit pel LTD en l'especificació del protocol és el que ajuda a diferenciar millor un tipus de pèrdues de les altres, ja que valors lleugerament inferiors o superiors tendeixen a mal interpretar les situacions d pèrdues. Això permet que, davant les situacions de pèrdues aleatòries (interferències), no faci falta reduir el Sending Rate i, per tant, la transferència de dades encara és més eficient i ràpida que anteriorment. També s'ha demostrat que la millora relativa en la reducció del temps de transferència en aquest tipus de situacions és directament proporcional a la duració del temps d'interferències.

Malgrat això, no es pot considerar que el protocol funcioni 100% a la perfecció ni entorns cablejats ni entorns heterogenis, ja que és necessari implementar un mecanisme que controli el *fairness* de les comunicacions quan hi ha més d'un flux MBTAP/OMBTAP compartint el mateix canal. D'aquesta manera, s'aconseguiria un repartiment just de l'ample de banda evitant una congestió molt accentuada de l'enllaç i una pèrdua massiva de paquets. A més, també cal afegir que, tot i estar dissenyat, no s'ha pogut comprovar el comportament del protocol davant l'ús de l'*Explicit Congestion Notification* (ECN) degut a les limitacions del simulador Riverbed Modeler (aquest no implementa ECN en els nodes intermitjos de xarxa).

Per acabar, els futurs passos a seguir són, per una banda, dissenyar i implementar aquest mecanisme de *fairness* i, per altra, programar i implementar routers dins el simulador Riverbed Modeler que permetin l'ús d'ECN per comprovar-ne el seu funcionament i comportament dins del protocol OMBTAP.

7 Estudi del cost temporal del treball

Tipus de feina	Recerca	Desenvolupament	Simulacions	Documentació	Total
Hores	120	120	140	100	**480**

Taula 18: Estudi del cost temporal del treball.

8 Agraïments

No podria acabar aquest treball sense agrair el suport i l'ajuda rebuda per tot l'equip del grup de recerca GRITS de La Salle, en especial a l'Alan Briones, qui ha estat fent el seguiment del projecte durant tot el seu transcurs.

Referències

[1] V. Jacobson and R. Braden, “TCP Extensions for Long-Delay Paths,” 1988.

[2] H. Bullot, R. Les Cottrell, and R. Hughes-jones, “Evaluation of Advanced TCP Stacks on Fast Long- Distance Production Networks The advanced stacks,” *J. Grid Comput.*, vol. 1, no. 4, pp. 345–359, 2003.

[3] La Salle GRITS, “Diseño de mejoras de la red VSN,” 2013.

[4] La Salle GRITS, “Pre-prototipado MBTAP, Fase 2,” 2016.

[5] A. Capone, L. Fratta, and F. Martignon, “Bandwidth estimation schemes for TCP over wireless networks,” *IEEE Trans. Mob. Comput.*, vol. 3, no. 2, pp. 129–143, 2004.

[6] Universitat de Berkeley, “A Comparative Analysis of TCP Tahoe , Reno , New-Reno , SACK and Vegas.”

[7] V. Jacobson, L. Zhang, and R. T. Braden, “TCP extension for high-speed paths,” 1990.

[8] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, “The NewReno modification to TCP’s fast recovery algorithm,” *No. RFC 6582*, pp. 1–16, 2012.

[9] L. Brakmo and L. L. Peterson, “TCP Vegas: End to end congestion avoidance on a global Internet,” *IEEE J. Sel. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, 1995.

[10] T. Kelly, “Scalable TCP: improving performance in highspeed wide area networks,” *ACM SIGCOMM Comput. Commun. Rev.*, vol. 33, no. 2, pp. 83–91, 2003.

[11] C. Jin, D. X. Wei, and S. H. Low, “FAST TCP: Motivation, Architecture, Algorithms, Performance,” *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, 2006.

[12] S. Floyd, “Highspeed TCP for Large Congestion Windows,” 2002.

[13] A. Kuzmanovic, E. W. Knightly, and R. Les Cottrell, “HSTCP-LP : A Protocol for Low-Priority Bulk Data Transfer in High-Speed High-RTT Networks,” in *The Second Int. Workshop on Protocols for Fast Long-Distance Networks*, 2004.

[14] D. Leith and R. Shorten, “H-TCP protocol for high-speed long-distance networks,” in *Proceedings of PFLDnet*, 2004.

[15] S. Ha, I. Rhee, and L. Xu, “Cubic: a new TCP-friendly high-speed TCP variant,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, 2008.

[16] E. Altman, C. Barakat, S. Mascolo, N. Möller, and J. Sun, “Analysis of TCP Westwood+ in high speed networks,” in *PFLDNet 2006 Workshop Proceedings.*, 2006.

[17] U. Kalim, M. K. Gardner, E. Brown, and W. C. Feng, “Cascaded TCP: Applying pipelining to TCP for efficient communication over wide-area networks,” *GLOBECOM - IEEE Glob. Telecommun. Conf.*, pp. 2256–2262, 2013.

[18] R. Stewart, “Stream Control Transmission Protocol,” 2007.

[19] D. Nagamalai and J. Lee, “Performance of SCTP over high speed wide area networks,” in *2004 IEEE Conference on Cybernetics and Intelligent Systems*, 2004, vol. 2, pp. 890–895.

[20] A. R. Al Caro Jr, K. Shah, J. R. Iyengar, P. D. Amer, and R. R. Stewart, “Congestion Control: SCTP vs TCP,” *Computer and Information Sciences*, 2003.

[21] R. S. Cheng, D. J. Deng, H. C. Chao, and W. E. Chen, “An Adaptive Bandwidth Estimation Mechanism for SCTP over Wireless Networks,” in *2010 5th International Conference on Future Information Technology (FutureTech)*, 2010, pp. 1–6.

[22] T. Liu, C. Xu, J. Guan, and H. Zhang, “Loss detection mechanism in SCTP heterogeneous wireless networks,” in *Proceedings of the 2012 2nd International Conference on Business Computing and Global Informatization, BCGIN 2012*, 2012, pp. 679–682.

[23] Y. Gu and R. L. Grossman, “UDT: UDP-based data transfer for high-speed wide area networks,” *Comput. Networks*, vol. 51, no. 7, pp. 1777–1799, 2007.

[24] K. T. Murata, P. Pavarangkoon, K. Yamamoto, Y. Nagaya, and T. Mizuhara, “A Quality Measurement Tool for High-Speed Data Transfer in Long Fat Networks,” in *2016 24th International Conference on Software, Telecommunications and Computer Networks (SoftCOM)*, 2016, pp. 1–5.

[25] C. Dovrolis, P. Ramanathan, and D. Moore, “Packet-Dispersion Techniques and a Capacity-Estimation Methodology,” *IEEE/ACM Trans. Netw.*, vol. 12, no. 6, pp. 963–977, 2004.

[26] R. Prosad, C. Davrolis, M. Murray, and K. C. Claffy, “Bandwidth estimation: metrics, measurement techniques, and tools,” *IEEE Netw.*, vol. 17, no. 6, pp. 27–35, 2003.

[27] A. Shriram *et al.*, “Comparison of Public End-to-End Bandwidth Estimation Tools on High-Speed Links,” in *PAM: 6th International Workshop on Passive and Active Network Measurement*, 2005, vol. 3431, pp. 306–320.

[28] T. Arsan, “Review of Bandwidth Estimation Tools and Application to Bandwidth Adaptive Video Streaming,” in *2012 9th International Conference on High Capacity Optical Networks and Enabling Technologies (HONET)*, 2012, pp. 152–156.

[29] R. Carter, “Measuring bottleneck link speed in packet-switched networks,” *Perform. Eval.*, vol. 27–28, no. 1, pp. 297–318, 1996.

[30] B. Melander, M. Björkman, and P. Gunningberg, “A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks,” in *IEEE Global Telecommunications Conference (GLOBECOM)*, 2000, vol. 1, pp. 415–420.

[31] Y. Joo and V. Ribeiro, “On the impact of variability on the buffer dynamics in IP networks,” in *ITC Conference on IP Traffic, Modeling and Management*, 2000.

[32] M. Jain and C. Dovrolis, “Pathload: a Measurement Tool for Available Bandwidth Estimation,” in *Proceedings Passive and Active Measurement Workshop (PAM 2002)*, 2002.

[33] J. Navratil and R. Les Cottrell, “ABwE :A Practical Approach to Available Bandwidth Estimation,” in *Proceedings Passive and Active Measurement Workshop (PAM’03)*, 2003.

[34] N. Hu and P. Steenkiste, “Evaluation and characterization of available bandwidth probing techniques,” *IEEE J. Sel. Areas Commun.*, vol. 21, no. 6, pp. 879–894, 2003.

[35] J. Strauss, D. Katabi, and F. Kaashoek, “A measurement study of available bandwidth estimation tools,” in *Proceedings of the conference on Internet measurement conference - IMC ’03*, 2003, pp. 39–44.

[36] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, “pathChirp : Efficient Available Bandwidth Estimation for Network Paths,” in *The 4th*

International workshop on Passive and Active network Measurement PAM 2003, 2003.

[37] K. Lakshminarayanan, V. N. Padmanabhan, and J. Padhye, "Bandwidth Estimation in Broadband Access Networks," in *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 2004, pp. 314–321.

[38] E. Goldoni, G. Rossi, and A. Torelli, "Assolo, a New Method for Available Bandwidth Estimation," in *The Fourth International Conference on InternetMonitoring ICIMP'09*, 2009, pp. 130–136.

[39] K. Lai and M. Baker, "Nettimer: A tool for measuring bottleneck link bandwidth," *Proc. USENIX Symp. Internet Technol. Syst.*, vol. 134, pp. 1–12, 2001.

[40] K. Lai and M. Baker, "Measuring Link Bandwidths Using a Deterministic Model of Packet Delay," in *SIGCOMM '00 Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, 2000, pp. 283–294.

[41] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "SProbe: A Fast Technique for Measuring Bottleneck Bandwidth in Uncooperative Environments," in *IEEE INFOCOM 2002*, 2002.

[42] R. Kapoor, L. Chen, and M. Gerla, "CapProbe : A Simple and Accurate Capacity Estimation Technique," in *SIGCOMM '04 Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, 2004, pp. 67–68.

[43] A. Persson and C. Marcondes, "TCP probe: A TCP with built-in path capacity estimation," in *Proceedings of the 8th IEEE Global Internet Symposium*, 2005.

[44] L. Chen, T. Sun, G. Yang, M. Y. Sanadidi, and M. Gerla, "End-to-End Asymmetric Link Capacity Estimation," in *International Conference on Research in Networking*, 2005, pp. 780–791.

[45] K. M. Salehin and R. Rojas-Cessa, "A Combined Methodology for Measurement of Available Bandwidth and Link Capacity in Wired Packet Networks," *IET Commun.*, vol. 4, no. 2, pp. 240–252, 2010.

[46] R. Ramani and A. Karandikar, "Explicit congestion notification (ECN) in TCP over wireless network," *IEEE Int. Conf. Pers. Wirel. Commun.*, pp. 495–499, 2000.

[47] L. Song, S. Hu, P. Mao, Y. Xiao, and K. Kim, "Performance Analysis of Explicit Control Protocol (XCP)," in *Wireless Mobile and Multimedia Networks (ICWMNN)*, 2010, pp. 40–43.

[48] L. Barreto and S. Sargento, "TCP, XCP and RCP in wireless mesh networks: An evaluation study," *Proc. - IEEE Symp. Comput. Commun.*, pp. 351–357, 2010.

[49] F. Abrantes and M. Ricardo, "A simulation study of XCP-b performance in wireless multi-hop networks," *Q2SWinet'07 Proc. Third ACM Work. Q2S Secur. Wirel. Mob. Networks*, pp. 23–30, 2007.

[50] M. Lestas, A. Pitsillides, P. Ioannou, and G. Hadjipollas, "Adaptive congestion protocol: A congestion control protocol with learning capability," *Comput. Networks*, vol. 51, no. 13, pp. 3773–3798, 2007.

[51] P. Mao and L. Liu, "Enhancements on Adaptive Congestion Protocol for Wireless Networks," no. Meici, pp. 1647–1651, 2015.

[52] K. Tan, F. Jiang, Q. Zhang, and X. Shen, "Congestion control in multihop wireless networks," *IEEE Trans. Veh. Technol.*, vol. 56, no. 2, pp. 863–873, 2007.

[53] C. Parsa and J. J. Garcia-Luna-Aceves, "Improving TCP congestion control over

Internets with heterogeneous transmission media," *Int. Conf. Netw. Protoc.*, pp. 213–221, 1999.

[54] S. Cen, P. C. Cosman, and G. M. Voelker, "End-to-end differentiation of congestion and wireless losses," *IEEE/ACM Trans. Netw.*, vol. 11, no. 5, pp. 703–717, 2003.

[55] Y. Tobe, Y. Tamura, A. Molano, S. Ghosh, and H. Tokuda, "Achieving moderate fairness for UDP flows by path-status classification," *Conf. Local Comput. Networks*, pp. 252–261, 2000.

[56] C. F. Chou, M. W. Hsu, and C. J. Lin, "A trend-loss-density-based differential scheme in wired-cum-wireless networks," *IWCMC 2006 - Proc. 2006 Int. Wirel. Commun. Mob. Comput. Conf.*, vol. 2006, pp. 239–244, 2006.

[57] A. Boukerche, G. Jia, and R. W. N. Pazzi, "Performance evaluation of packet loss differentiation algorithms for wireless networks," *PM2HW2N'07 Proc. Second ACM Work. Perform. Monit. Meas. Heterog. Wirel. Wired Networks*, pp. 50–52, 2007.

[58] M. K. Park, K. H. Sihn, and J. H. Jeong, "A statistical method of packet loss type discrimination in wired-wireless networks," *2006 3rd IEEE Consum. Commun. Netw. Conf. CCNC 2006*, vol. 1, pp. 458–462, 2006.

[59] E. H. K. Wu and M. Z. Chen, "JTCP: Jitter-based TCP for heterogeneous wireless networks," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 4, pp. 757–766, 2004.

[60] E. H. K. Wu, J. M. Chen, C. H. Chu, M. F. Tsai, and J. R. Wang, "Improving SCTP performance by jitter-based congestion control over wired-wireless networks," *Eurasip J. Wirel. Commun. Netw.*, vol. 2011, 2011.

[61] C. P. Fu and S. C. Liew, "TCP Veno : TCP Enhancement for Transmission Over," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 2, pp. 216–228, 2003.

[62] K. Takagaki, H. Ohsaki, and M. Murata, "Analysis of a window-based flow control mechanism based on TCP Vegas in heterogeneous network environment," *IEICE Trans. Commun.*, vol. E85–B, no. 1, pp. 89–97, 2002.

[63] S. Mascolo, C. Casetti, and M. Gerla, "TCP Westwood : Bandwidth Estimation for Enhanced Transport overWireless Links," *Sigmobile*, pp. 287–297, 2001.

[64] C. Casetti, M. Gerla, and S. Mascolo, "TCP Westwood End-to-End Bandwidth Estimation for Enhanced Transport over Wireless Links," *Wirel. Networks*, no. 5091294, pp. 467–479, 2002.

[65] L. a. Grieco and S. Mascolo, "Performance evaluation and comparison of Westwood+, New Reno, and Vegas TCP congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, p. 25, 2004.

[66] K. Xu, Y. Tian, and N. Ansari, "TCP-Jersey for wireless IP communications," *IEEE J. Sel. Areas Commun.*, vol. 22, no. 4, pp. 747–756, 2004.

[67] K. Chandran, S. Raghunathan, S. Venkatesan, and R. Prakash, "A Feedback Based Scheme For Improving TCP Performance In Sudarshan Raghunathan," *IEEE Pers. Commun. Mag. Spec. Issue Ad Hoc Networks*, pp. 34–39, 2001.

[68] J. P. Monks, P. Sinha, and V. Bharghavan, "Limitations of TCP-ELFN for Ad hoc Networks," *MOMUC'00*, 2000.

[69] D. Kim, C. K. Toh, and Y. Choi, "TCP-BuS: Improving TCP performance in wireless Ad Hoc networks," *J. Commun. Networks*, vol. 3, no. 2, pp. 175–185, 2001.

[70] E. He, J. Leigh, and O. Yu, "Reliable Blast UDP : Predictable High Performance

Bulk Data Transfer," *IEEE Int. Conf. Clust. Comput.*, pp. 317–324, 2002.

[71] R. S. Rajaboina, P. C. Reddy, R. A. Kumar, and N. Venkatramana, "Performance comparison of TCP, UDP and TFRC in static wireless environment," *Int. J. Inf. Comput. Secur.*, vol. 8, no. 2, pp. 158–180, 2016.

[72] M. R. Meiss, "Tsunami: A high-speed rate-controlled protocol for file transfer," *Indiana Univ.*, pp. 1–10, 2004.

[73] A. O. F. Atya and J. Kuang, "RUFC: A flexible framework for reliable UDP with flow control," *2013 8th Int. Conf. Internet Technol. Secur. Trans. ICITST 2013*, pp. 276–281, 2013.

[74] P. M. Dickens, "A lightweight, high performance communication protocol for grid computing," *Cluster Comput.*, vol. 13, no. 1, pp. 47–66, 2010.

[75] A. Mallorquí and A. Briones, "Disseny, implemenatció i simulació de mecanismes per a protocols de transport en les Long Fat Networks." 2017.

[76] C. Adams, P. Cain, D. Pinkas, and Z. R., "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)," *RFC 3161*, 2001.

[77] E. H. K. Wu, M. I. Hsieh, M. Z. Chen, and S. Y. Hung, "JTCP: Congestion distinction by the jitter-based scheme over wireless networks," *IEEE Int. Symp. Pers. Indoor Mob. Radio Commun. PIMRC*, vol. 4, no. OCTOBER 2004, pp. 2473–2477, 2004.

[78] E. Perahia and R. Stacey, *Next generation wireless LANs: 802.11n and 802.11ac*. 2005.

[79] J. H. Kim and J. K. Lee, "Performance of Carrier Sense Multiple Access with collision avoidance protocols in wireless LANs," *Wirel. Pers. Commun.*, vol. 11, no. 2, pp. 161–183, 1999.

Annex I: Glossari de funcions pròpies de Riberbed Modeler utilitzades

Número	Funció	Descripció
1	`ip_address_create (const char *addr_str)`	Retorna una direcció IP (`IpT_Address`) a partir de la direcció IP indicada per la *string*.
2	`ip_rte_intf_addr_get (IpT_interface_info* interface_ptr)`	Retorna la direcció IP (`IpT_Address`) de la interfície especificada.
3	`ip_address_create (const char *addr_str)`	Retorna una direcció IP del tipus `IpT_Address` a partir de la direcció IP especificada en format *string*.
4	`ip_rte_intf_tbl_access (IpT_Rte_Module_Data* iprmd_ptr, int index)`	Retorna un punter de la i-èssima interfície (`IpT_Interface_Info*`) del mòdul IP especificat.
5	`ip_rte_num_interfaces_get (IpT_Rte_Module_Data* iprmd_ptr)`	Retorna el número d'interfícies que té el mòdul IP especificadt(ha de ser un node intermig).
6	`ip_rte_parameters_objid_obtain (Objid node_id, Objid module_id, Boolean* gateway_status)`	Obté l'Object ID de l'objecte que conté los paràmetres IP corresponents a l'objecte especificat (una *worksation* o un servidor). Per aquesta finalidat, cal passar la constant `OPC_OBJID_NULL` en el primer paràmetre y `OPC_NIL` en el tercer, indicant l'Object ID de la *workstation* (o servidor) en el segon paràmetre.
7	`ip_support_address_from_node_id_get (Objid node_objid)`	Obté la direcció IP del node especificat. Si aquest té més d'una interfície, es retorna la IP corresponent a la primera interfície de *loopback*.
8	`ip_support_module_data_get (Objid node_objid)`	Retorna el *handle* del mòdul IP (`IpT_Rte_Module_Data*`) del node especificat per posteriorment poder-hi accedir.
9	`ip_support_node_id_from_ip_address_get (IpT_Address ip_addr)`	Retorna l'`Objid` del node que utilitza la direcció IP especificada.
10	`oms_pr_attr_get (OmsT_Pr_Handle pr_handle, char* attr_name, int attr_type, Vartype* attr_value_ptr)`	Obté el valor de l'atribut indicat que forma part d'un procés registrat (especificat pel seu punter).
11	`oms_pr_process_discover (Objid neighbor_objid, List* pr_handle_lptr, char* attr0_name, int attr0_type, Vartype attr0_value,...., OPC_NIL)`	Descobreix el(s) procés/sos dins dels processos registrats que fa(n) *match* amb els atributs especificats, i omple la llista especificada amb aquest(s) procés/sos.
12	`oms_pr_process_register (Objid node_objid, Objid module_objid, Prohandle pro_handle, char* process_name)`	Registra un procés en el registre del model i retorna un identificador d'aquest registro (`OmsT_Pr_Handle`) per poder accedir-hi posteriorment.

13	`oms_tan_neighbor_streams_find (Objid module_objid, Objid neighbor_objid, int* in_strm_ptr, int* out_strm_ptr)`	Determina si hi ha una connexió de *packet stream* entre dos mòduls especificats. En cas afirmatiu (es retorna el codi `OPC_COMPCODE_SUCCESS`), indica els índexs del *stream* pe a cada mòdul en els punters especificats. En cas contrari, retorna `OPC_COMPCODE_FAILURE`.
14	`op_dist_load (const char* dist_name, double dist_arg0, double dist_arg1)`	Carrega una distribució determinada utilitzada para generar valors estocàstics. Retorna un `Distribution*` amb la distribució generada. En cas d'error, retorna `OPC_NIL`.
15	`op_dist_outcome (Distribution* dist_ptr)`	Retorna un `double` amb un valor generat a partir de la distribució especificada (normalment generada amb la funció `op_dist_load()`).
16	`op_dist_uniform (double limit)`	Retorna un `double` amb un valor generat a partir d'una distribució uniforme entre 0 (inclòs) i el límit especificat (no inclòs).
17	`op_ici_attr_exists (Ici* iciptr, const char* attr_name)`	Retorna `OPC_TRUE` si l'atributo indicat existeix a l'ICI especificada, i `OPC_FALSE` en cas contrari.
18	`op_ici_attr_get (Ici* iciptr, const char* attr_name, Vartype* value)`	Retorna el valor de l'atribut indicada de l'ICI especificada.
19	`op_ici_attr_set (Ici* iciptr, const char* attr_name, Vartype value)`	Assigna el valor especificat a l'atribut definit de l'ICI especificada.
20	`op_ici_create (const char* fmt_name)`	Crea una nova ICI amb l'estructura del formato especificat. Retorna un `Ici*` que apunta a la nova ICI creada.
21	`op_ici_destroy (Ici* iciptr)`	Destrueix l'ICI especificada.
22	`op_ici_format (Ici* iciptr, char* format_name)`	Obté el nom del formato de l'ICI especificada.
23	`op_ici_id (Ici* iciptr)`	Retorna l'ID (`OpT_Ici_Id`) de l'ICI especificada.
24	`op_ici_install (Ici* iciptr)`	Estableix l'ICI especificada com l'ICI instal·lada. Això provoca que l'ICI s'associï automàticament amb las interrupcions de sortida que s'invoquen en el procés.
25	`op_ici_print (Ici* iciptr)`	Escriu per pantalla la informació de l'ICI especificada.
26	`op_id_self ()`	Obté l'Object ID del *surrounding processor*.

27	`op_ima_obj_attr_get_int32(Objid objid, const char* attr_name, int* value_ptr)`	Obté el valor d'un atributo enter especificat de l'objeto especificat. Retorna `OPC_COMPCODE_SUCCESS` si el valor és obtingut amb èxit o `OPC_COMPCODE_FAILURE` en cas d'error.
28	`op_intrpt_force_remote (int code, Objid mod_objid)`	Força una interrupció al mòdul especificat. Immediatament s'invoca el procés corresponent (indicat pel codi enter) i se suspèn temporalment l'execució del procés actual.
29	`op_intrpt_schedule_self (double time, int code)`	Programa una interrupció contra el propi procés, que s'executarà en el temps de simulació indicat amb el codi d'interrupció especificat.
30	`op_intrpt_strm ()`	Retorna un `int` amb l'índex del *stream* associat amb l'última interrupció que ha invocat el procés.
31	`op_intrpt_type ()`	Retorna un `int` que simbolitza el tipus d'interrupció que ha invocat el procés.
32	`op_pk_create_fmt (const char* format_name)`	Retorna un `Packet*` apuntant a un nou paquet generat amb el format especificat. En cas d'error retorna `OPC_NIL`.
33	`op_pk_create (Opt_Packet_Size bulk_size)`	Retorna un `Packet*` apuntant a un nou paquet generat sense format de la mida especificada (en bits). En cas d'error retorna `OPC_NIL`.
34	`op_pk_creation_mod_get (Packet* pkptr)`	Obté l'Object ID del mòdul on el paquet especificat ha estat creat. Aquest Object ID es pot modificar amb la funció `op_pk_mod_set()`.
35	`op_pk_creation_time_get (Packet* pkptr)`	Retorna un `double` amb el temps de simulació (en segons) de creació del paquet especificat. Aquest valor es pot canviar amb la funció `op_pk_creation_time_set()`.
36	`op_pk_deliver_delayed (Packet* pkptr, Objid mod_objid, int instream_index, double delay)`	Realitza la mateixa tasca que la funció `op_pk_deliver()`, però es pot indicar quant temps s'ha d'esperar abans d'enviar el paquet.
37	`op_pk_deliver (Packet* pkptr, Objid mod_objid, int instream_index)`	Envia el paquet especificat cap a un mòdul remot a través del *stream* indicat. Programa l'arribada del paquet pel temps actual de simulació i en perd la propietat (li dóna al procés d'invocació).
38	`op_pk_destroy (Packet* pktptr)`	Destrueix el paquet especificat.
39	`op_pk_encap_flag_is_set (Packet* pkptr, int index)`	Comprova si un *flag* (indicat per l'índex) del paquet especificat està activat. Retorna `OPC_TRUE` en cas afirmatiu i `OPC_FALSE` en cas contrari.

40	`op_pk_encap_flag_set (Packet* pkptr, int index)`	Activa un *flag* (indicat per l'índex) del paquet especificat.
41	`op_pk_format (Packet* pkptr, char* fmt_name)`	Obté el nom del format del paquet especificat.
42	`op_pk_get (int instream_index)`	Retorna un `Packet*` que apunta al paquet que ha arribat per l'*input stream* especificat i elimina aquest paquet del *buffer* del *stream*. Si el *stream* està buit, la funció retorna `OPC_NIL`. Per evitar aquest error, prèviament es pot fer la comprovació amb la funció `op_strm_empty()`.
43	`op_pk_ici_get (Packet* pkptr)`	Retorna un `Ici*` que apunta a l'ICI que s'hagi associat amb el paquet especificat. Si el paquet no està associat amb cap ICI, es retorna `OPC_NIL`.
44	`op_pk_ici_set (Packet* pkptr, Ici* iciptr)`	Crea una associació entre el paquet i l'ICI especificats, guardant un `Ici*` dins del paquet. Aquest procés sobreescriu qualsevol associació prèvia entre el paquet especificat i una altra ICI.
45	`op_pk_id (Packet* pkptr)`	Retorna l'ID (`OpT_Packet_Id`) del paquet especificat.
46	`op_pk_id (Packet* pkptr)`	Retorna l'`OpT_Packet_Id` del paquet especificat. Per veure l'ID en format *string* s'ha d'utilitzar la funció `op_pk_id_str_get(OpT_Packet_Id pkid, char* buffer)`.
47	`op_pk_name_to_index (const char* format_name, const char* field_name)`	Retorna un `int` amb l'equivalència (índex numèric) del camp especificat del format de paquet especificat.
48	`op_pk_nfd_get_int32 (Packet* pkptr, const char* field_name, int* value_ptr)`	Obté el valor enter de 32 bits del camp especificat del paquet especificat. Retorna `OPC_COMPCODE_SUCCESS` en cas d'èxit i `OPC_COMPCODE_FAILURE` en cas d'error.
49	`op_pk_nfd_set_int32 (Packet* pkptr, const char* field_name, int value)`	Assigna un valor enter de 32 bits al camp especificat del paquet especificat. Retorna `OPC_COMPCODE_SUCCESS` en cas d'èxit i `OPC_COMPCODE_FAILURE` en cas d'error.
50	`op_pk_num_fds (Packet* pkptr)`	Retorna el número de camps que té el paquet especificat.
51	`op_pk_num_fields_get (Packet* pkptr, int* num_fmd_fields, int* num_unfmd_fields)`	Retorna el número de camps amb format i el número de camps sense format que té el paquet especificat.
52	`op_pk_print (Packet* pkptr)`	Escriu la informació del paquet per pantalla.

53	`op_pk_send_delayed (Packet* pkptr, int outstream_index, double delay)`	Realitza la mateixa tasca que la funció `op_pk_send()`, però es pot indicar quant temps cal esperar abans d'enviar el paquet
54	`op_pk_send (Packet* pkptr, int outstream_index)`	Envia el paquet especificat cap al *stream* de sortida especificat.
55	`op_pk_total_size_get (Packet* pkptr)`	Retorna un `OpT_Packet_Size` amb el valor de la mida (en bits) del paquet especificat. Aquest valor es pot canviar amb la funció `op_pk_total_size_set()`.
56	`op_pk_transfer (Packet* from_pkptr, Packet* to_pkptr)`	Reemplaça el contingut d'un paquet especificat pel d'un altre paquet especificat.
57	`op_pk_tree_id (Packet* pkptr)`	Retorna l'ID (`OpT_Packet_Id`) de "l'arbre" de paquets al qual pertany el paquet especificat.
58	`op_prg_list_access (List* list_ptr, int pos_index)`	Fa una lectura no destructiva de l'element de la llista en la posició indicada, retornant un punter cap a aquest element.
59	`op_prg_list_init (List* list_ptr)`	Inicialitza la llista especificada per poder començar a operar amb ella.
60	`op_prg_list_insert (List* list_ptr, void* element_ptr, int pos_index)`	Insereix l'element indicat a la posició desitjada de la llista especificada. Per indicar una posició, es pot utilitzar la macro `OPC_LISTPOS_HEAD` per referir-se al primer element de la llista o `OPC_LISTPOS_TAIL` per l'últim.
61	`op_prg_list_remove (List* list_ptr, int pos_index)`	Fa una lectura destructiva de l'element de la llista en la posició indicada, retornant un punter cap a aquest element.
62	`op_prg_list_size (List* list_ptr)`	Obté la longitud de la llista especificada.
63	`op_prg_odb_bkpt (const char* label)`	Defineix un *breakpoint* etiquetat que succeirà en cas que la simulació s'executi amb l'ODB (Opnet Debugger) habilitat i el *breakpoint* estigui habilitat. La *string* que es passa per arguments es compara amb les etiquetes de *breakpoint* habilitades.
64	`op_prg_odb_ltrace_active (const char* label)`	Determina si l'etiqueta especificada ha estat habilitada a través d'una comanda de l'ODB.
65	`op_prg_odb_print_major (const char* string0, const char* string 1, … , const char* stringn, OPC_NIL)`	Escriu per pantalla una seqüència de *strings* a l'output del ODB començant pel nivell d'indentació major.
66	`op_pro_self ()`	Obté un *handle* del propi procés que s'està executant. Aquest *handle* es pot utilitzar per fer peticions al *Kernel* sobre el propi procés.

67	`op_sim_end (const char* line0, const char* line1, const char* line2, const char* line3)`	Força l'acabament de la simulació, mostrant per pantalla els missatges especificats.
68	`op_sim_message (char* line0, char* line1)`	Escriu per pantalla un missatge.
69	`op_sim_time ()`	Retorna un `double` amb el temps de simulació actual (en segons).
70	`op_stat_reg (const char* stat_name, int stat_index, int type)`	Retorna un *handle* utilitzat per fer referència a l' estadística desitjada dins del Process Model.
71	`op_stat_write (Stathandle stat_handle, double value)`	Escriu el parell (temps, valor) a l'estadística especificada. El valor s'especifica per arguments i el temps és el temps de simulació actual.
72	`op_topo_child (Objid parent_objid, int child_type, int child_index)`	Retorna l'objecte fill del pare especificat, segons l'índex i el tipus d'objecte que es demanen.
73	`op_topo_child_count (Objid parent_objid, int child_type)`	Donat l'Object ID de l'objecte pare, retorna el número d'objectes fill que aquest té. Si no es vol especificar quin tipus d'objecte fill es busca, s'ha de passar la constant `OPC_OBJTYPE_GENERIC` en el segon paràmetre.
74	`op_topo_parent (Objid child_objid)`	Retorna l'Object ID del pare de l'objecte fill especificat. Retorna `OPC_OBJID_NULL` en cas que l'objecte indicat no tingui pare.

Taula 19: Glossari de funcions pròpies de Riverbed Modeler utilitzades.

Annex II: Format dels missatges OMBTAP

Missatge INIT (0x02)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (client)
	Destination Port	2	Port destí associat (servidor)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació Datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag S enviat pel peer remot. En el missatge INIT el seu valor es 0
	Type	1	Tipus de missatge. INIT = 0x02
	Flags	1	No definit
	Length	2	Tamany UDP Data
	Initiation Tag C	4	Valor aleatori, marca el Verification Tag que haurà de posar el peer remot per identificar l'associació
	#Out Streams C	2	No utilitzat en la implementació en Riverbed Modeler Cada connexió correspon a un sol arxiu o flux de dades.
	Max #In Streams C	2	No utilitzat en la implementació en Riverbed Modeler Cada connexió correspon a un sol arxiu o flux de dades.
	Initial TSN C	4	Marca el número TSN pel qual començarà el client
	Optional Variable Length Parameters	Variable	Defineix paràmetres opcionals en format TLV (no utilitzat)

Fig. 101: Missatge INIT.

Missatge INIT-ACK (0x03)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (servidor)
	Destination Port	2	Port destí associat (client)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag C enviat pel peer remot.
	Type	1	Tipus de missatge. INIT-ACK = 0x03
	Flags	1	No definit
	Length	2	Tamany UDP Data
	Initiation Tag S	4	Valor aleatori, marca el Verification Tag que haurà de posar el peer remot per identificar l'associació
	#Out Streams S	2	No utilitzat en la implementació en Riverbed Modeler Cada connexió correspon a un sol arxiu o flux de dades.
	Max #In Streams S	2	No utilitzat en la implementació en Riverbed Modeler Cada connexió correspon a un sol arxiu o flux de dades.
	Initial TSN S	4	Marca el número TSN pel qual començarà el servidor
	Optional Variable Length Parameters	Variable	Defineix paràmetres opcionals en format TLV. En aquest missate s'hi col·loca la State Cookie

Fig. 102: Missatge INIT-ACK.

Missatge COOKIE-ECHO (0x04)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (client)
	Destination Port	2	Port destí associat (servidor)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag S enviat pel peer remot.
	Type	1	Tipus de missatge. COOKIE-ECHO = 0x04
	Flags	1	No definit
	Length	2	Tamany UDP Data
	State Cookie	156	Copia directa de les dades enviades en el paràmetre State Cookie del missatge INIT-ACK

Fig. 103: Missatge COOKIE-ECHO.

Paràmetre State Cookie (0x0007)

	Camp	Tamany (bytes)	Descripció
Parameter Header	Parameter type	2	Tipus de paràmetre (0x0007)
	Parameter length	2	Tamany del paràmetre (160 bytes)
Parameter Data	Initiation Tag C	4	Initiation Tag enviat pel client en el missatge INIT
	Initiation Tag S	4	Initiation Tag enviat pel client en el missatge INIT-ACK
	Initial TSN C	4	Valor aleatori que indica el nombre de TSN pel que començarà el client (no s'utilitza)
	Initial TSN S	4	Valor aleatori que indica el nombre de TSN pel que començarà el servidor
	# Out Streams	2	No utilitzat en la implementació en Riverbed Modeler Cada connexió correspon a un sol arxiu o flux de dades.
	Max # In Streams	2	No utilitzat en la implementació en Riverbed Modeler Cada connexió correspon a un sol arxiu o flux de dades.
	Timestamp	6	Moment de creació de la Cookie [HH:MM:SS DD/MM/AA]
	Lifespan	2	Temps de vida de la Cookie [MM:SS]
	Hash MD5	128	Hash creat a partir de la resta de valors del "Parameter Data"

Fig. 104: Paràmetre *State Cookie.*

Missatge COOKIE-ACK (0x05)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (servidor)
	Destination Port	2	Port destí associat (client)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag C enviat pel peer remot.
	Type	1	Tipus de missatge. COOKIE-ACK = 0x05
	Flags	1	No definit
	Length	2	Tamany UDP Data

Fig. 105: Missatge COOKIE-ACK.

Missatge BW (0x20)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (client)
	Destination Port	2	Port destí associat (servidor)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació Datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag S enviat pel peer remot
	Type	1	Tipus de missatge. BW = 0x20
	Flags	1	1r bit de menor pes = 1 si és el primer paquet del *packet pair* 2n bit de menor pes = 1 si és segon paquet del *packet pair*
	Length	2	Tamany UDP Data
	TSN Value	4	Valor global dels paquets que rep el receptor. Cada paquet rebrà una numeració genèrica independentment de l'arxiu o fluxe al que pertanyi
	Stream Identifier	2	Nombre d'arxiu o fluxe al que es pertany (no s'utilitza en la implementació en Riverbed Modeler)
	Stream Seq Num	2	Indica la posició del paquet dins un arxiu o fluxe
	Data	Variable	Padding que s'afegeix per aconseguir la mida de paquet desitjada. Les mides inicials són: P1 = 700 bytes P2 = 900 bytes

Fig. 106: Missatge BW.

Missatge DATA (0x01)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (client)
	Destination Port	2	Port destí associat (servidor)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag S enviat pel peer remot
	Type	1	Tipus de missatge. DATA = 0x01
	Flags	1	Us principal per: - ECN (0 0 X X 0 0 0 0) - Transmissió - Inici i final de ràfega (0 0 0 0 0 0 Y Y) (Veure *Flags*)
	Length	2	Tamany UDP DATA
	TSN Value	4	Valor global dels paquets que rep el receptor. Cada paquet rebrà una numeració genèrica independentment de l'arxiu o fluxe al que pertanyi
	Stream Identifier	2	Nombre d'arxiu o fluxe al que es pertany (no s'utilitza en la implementació en Riverbed Modeler)
	Stream Seq Num	2	Indica la posició del paquet dins un arxiu o fluxe
	TS Value (TSVal)	3	Timestamp que indica el clock (UnixEpochTime) en el moment que l'emisor envia el paquet. El valor s'obté de la fórmula $TSVal = a \cdot s + b$
	Data	Variable	Informació de l'arxiu que s'està enviant. Durant la transmissió, la mida del paquet és la d'un Jumboframe (9000 Bytes), però sí aquest és l'últim paquet de dades del fluxe, el seu tamany es veu reduit al necessari.

Fig. 107: Missatge DATA.

Missatge SACK (0x0E)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (servidor)
	Destination	2	Port destí associat (client)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag S enviat pel peer remot
	Type	1	Tipus de missatge. SACK = 0x0E
	Flags	1	Us principal per: - ECN (0 0 X X 0 0 0 0) - Transmissió - Inici i final de ràfega (0 0 0 0 0 0 Y Y) (Veure *Flags*)
	Length	2	Tamany UDP DATA
	ACK Seq Number	4	Asigna un valor a l'ACK per controlar posteriorment la resposta.
	Last TSN	4	Últim valor de TSN correcte rebut
	RTT	4	Valor del RTT de l'última ràfega rebuda, en microsegons
	Jitter Ratio	4	Valor del Jitter Ratio calculat
	Sending Rate (SR)	4	Velocitat de transmissió de paquets, en paquets per segon
	Congestion Window	4	Min (Buffer disponible en recepció, RR*(SYN+RTT))
	Estimated Link (BW)	4	Valor de l'ample de banda estimat de l'enllaç. En Mbps
	TS Value (TSVal)	3	Timestamp que indica el clock (UnixEpochTime) en el moment que l'emisor envia el paquet. El valor s'obté de la fórmula $TSVal = a \cdot s + b$
	TS Echo Reply (TSEcr)	3	Timestamp echo de l'últim TSVal rebut
	# Gaps	4	Nombre de blocs de paquets perduts indicats en el propi SACK
	Inici gap 1	4	TSN del primer paquet del primer gap de paquets perduts
	Final gap 1	4	TSN de l'últim paquet del primer gap de paquets perduts
	...		Array de gaps perduts ordenats
	Inici gap N	4	TSN del primer paquet de l'últim gap de paquets perduts
	Final gap N	4	TSN de l'últim paquet de l'últim gap de paquets perduts

Fig. 108: Missatge SACK.

Missatge ACK (0x0F)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (client)
	Destination	2	Port destí associat (servidor)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag S enviat pel peer remot
	Type	1	Tipus de missatge. ACK = 0x0F
	Flags	1	Us principal per: - ECN (0 0 X X 0 0 0 0) - Transmissió - Inici i final de ràfega (0 0 0 0 0 0 Y Y) (Veure *Flags*)
	Length	2	Tamany UDP DATA
	ACK Seq Number	4	En enviament de dades: valor que contenia el SACK rebut. En estimació del canal: Número de ràfega per la qual s'envia l'ACK
	CW	4	Tamany de la finestra de congestió
	TS Value (TSVal)	4	Timestamp que indica el clock (UnixEpochTime) en el moment que l'emisor envia el paquet. El valor s'obté de la fórmula TSVal = a · s + b
	TS Echo Reply (TSEcr)	4	Timestamp echo de l'últim TSVal rebut

Fig. 109: Missatge ACK.

Missatge ABORT (0x14)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat
	Destination Port	2	Port destí associat
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació Datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag enviat pel peer remot
	Type	1	Tipus de missatge. ABORT = 0x14
	Flags	1	No definit
	Length	2	Tamany UDP Data
	Error Cause Identifier	2	Indica quina ha estat la causa de l'error
	Length	Variable	Tamany de la informació addicional de l'error en bytes
	Error Cause Identifier	Variable	Informació addicional de l'error

Fig. 110: Missatge ABORT.

Missatge SHUTDOWN (0x10)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (client)
	Destination Port	2	Port destí associat (servidor)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació Datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag S enviat pel peer remot
	Type	1	Tipus de missatge. SHUTDOWN = 0x10
	Flags	1	No definit
	Length	2	Tamany UDP Data
	Cumulative TSN	4	Conté el TSN del darrer missatge de dades rebut

Fig. 111: Missatge SHUTDOWN.

Missatge SHUTDOWN-ACK (0x11)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (servidor)
	Destination Port	2	Port destí associat (client)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació, marcat per l'Initiation Tag C enviat pel peer remot.
	Type	1	Tipus de missatge. SHUTDOWN-ACK = 0x11
	Flags	1	No definit
	Length	2	Tamany UDP Data

Fig. 112: Missatge SHUTDOWN-ACK.

Missatge SHUTDOWN-COMPLETE (0x12)

	Camp	Tamany (bytes)	Descripció
UDP Header	Source Port	2	Port origen associat (client)
	Destination Port	2	Port destí associat (servidor)
	Length	2	Tamany UDP Header + UDP Data
	Checksum	2	Comprovació Datagrama
UDP Data	Verification Tag	4	Camp identificador de l'associació. En aquest cas correspon al mateix valor rebut en el Verification Tag del missatge SHUTDOWN-ACK
	Type	1	Tipus de missatge. SHUTDOWN-COMPLETE = 0x12
	Flags	1	Bit de menor pes = 1 Indica que el camp Verification Tag del missatge es una còpia del Verification Tag del missatge SHUTDOWN-ACK
	Length	2	Tamany UDP Data

Fig. 113: Missatge SHUTDOWN-COMPLETE.

Flags
(0 1 2 3 4 5 6 7)

	Camp	# bit	Descripció
Flags	-	0	Lliure
	-	1	Lliure
	SRR	2	Sending Rate Reduced (SRR): Indica al receptor que ha modificat el seu comportament a causa de la notificació ECE.
	ECE	3	ECN-Echo (ECE): - Negociació: Indica que s'utilitzarà ECN. -Transmisió: Indica que un router intermedi es troba en estat de congestió o pròxim a aquest.
	NCS	4	Non-Connected Shutdown (NCS): Indica que en el camp de Verif Tag del SHUTDOWN-COMPLETE s'envia el Verification Tag rebut en SHUTDOWN-ACK anterior
	NCA	5	Non-Connected Abort (NCA): Si existeix associació i Verification Tag, T=0. Si no existeix asociació o Verification Tag vàlid, el verification tag de l'ABORT serà igual al del missatge que es respongui
	EOB	6	End of Burst (EOB): Indica final de ràfega
	SOB	7	Start of Burst (SOB): Indica inici de ràfega

Fig. 114: Camp FLAGS.

Índex de figures

Índex de taules