Universitatea “Politehnica” din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Algoritmi de tip RED pentru combaterea congestiei

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de
Inginer în domeniul *Calculatoare şi Tehnologia Informaţiei*
programul de studii de licenţă *Ingineria Informaţiei*

Conducător ştiinţific
Conf. dr. ing. Ştefan Stăncescu

Absolvent
Balint George-Adrian

Anul 2012

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul Electronică Aplicată şi Ingineria Informaţiei

Aprobat Director de Departament:
Prof. Dr. Ing. Sever Paşca

TEMA PROIECTULUI DE DIPLOMĂ
a studentului
Balint I. George-Adrian, grupa 441A

1. Titlul temei: Algoritmi de tip RED pentru combaterea congestiei

2. Contribuţia practică, originală a studentului va consta în: Studiul unor formule noi de calcul a probabilitatii de pierdere de pachete in cozi de asteptare, realizare de simulari comparative cu NS 2.3 (Network Simulator), a algoritmilor de tip RED; optimizari de algoritmi de tip RED pentru combaterea congestiei traficului in retele de calculatoare.

3. Realizarea practică/ proiectul rămân în proprietatea: UPB, Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

4. Proprietatea intelectuală asupra proiectului aparţine: UPB, Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

5. Locul de desfăşurare a activităţii: UPB, Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

6. Data eliberării temei: 15.05.2012

CONDUCĂTOR LUCRARE: *Conf. dr. ing. Ştefan Stăncescu*

STUDENT: *George-Adrian Balint*

Anexa 5

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Algoritmi de tip RED pentru combaterea congestiei*", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul *Calculatoare si Tehnologia Informatiei*, programul de studii *Ingineria Informatiei* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor constituie fraudă şi se sancţionează conform regulamentelor în vigoare.

Bucureşti, *15.09.2012*

Absolvent *George-Adrian BALINT*

Cuprins

CAPITOLUL 1

1. Introducere. Congestia traficului.

1.1 Definiția congestiei

Deoarece congestia apare odată cu supraîncărcarea rețelelor, definițiile congestiei se concentrează asupra comportamentului rețelelor la încărcări mari sau foarte mari.

Supraîncărcarea resurselor apare atunci când pe o legătura se doreşte a fi transmis un volum de date mai mare decât capacitatea sa. Ca o definiție obiectivă, putem spune că sarcina de a transmite un volum de date mai mare decât capacitatea legăturii pe care se va transmite poate duce la o **congestie.** În prezent congestia este semnalată pe rețele prin detecția pierderilor de pachete ca rezultat al aşteptării în coada ce depăşeşte o anumită limită (specificată în stackul protocolului TCP). Algoritmul de control al congestiei specificat în TCP a rămas în proporții mari neschimbat de la varianta sa inițială. Bucla de control închisă a TCP-ului nu permite pierderea de pachete fără semnalizare şi elimină astfel posibilitatea unui colaps din cauza congestiei pe rețea (eveniment ce a apărut în trecut în momentul când congestia a fost lăsată la voia întâmplării nefolosindu-se nici un protocol de control al transmisiilor).

Prin urmare, termenul de rețea congestionată poate să însemne atât supraîncărcarea bufferelor echipamentelor de rețea şi implicit a cozilor (fapt ce duce la pierderi de pachete), dar totodată congestia poate fi remarcată şi în cazul în care bufferele nu sunt supra-încărcate ci doar utilizatorul nu are o disponibilitatea a serviciilor rețelei ce îi poate satisface nevoile aplicației folosite.

Pentru exemplificare, să considerăm o rețea în care, dintr-un anumit motiv, rata de sosire a pachetelor la un ruter depăşeşte rata de deservire a acestuia. În acest moment, pachetele sunt introduse într-o coadă de aşteptare, ceea ce conduce la întârzieri. Întârzierea adițională poate determina sursele să retransmită, crescând astfel şi mai mult încărcarea rețelei. Acest tip de feedback conduce la o deteriorare rapidă a situației, unde traficul este dominat de retransmisii şi productivitatea efectivă – banda efectivă (throughput) scade brusc. Mai departe, dacă este implementat un mecanism de control al traficului „router-to-router", noi pachete pot să nu mai fie admise în coadă, aşa că acestea pot inunda un ruter precedent de asemenea. Se poate ajunge astfel la o situație de tipul „**deadlock**", în care întreg traficul este înghețat.

Avem de-a face cu trei evenimente ce apar simultan. În primul rând, întârzierea datorată cozilor de aşteptare creşte. În al doilea rând, pot apărea pierderi de pachete. În final, în starea congestionată, traficul este dominat de retransmisii, astfel încât eficiența scade. Definițiile standard ale congestiei sunt de forma următoare: „O rețea este congestionată dacă, datorită supraîncărcării, condiția X se îndeplineşte", unde X reprezintă creşterea excesivă a întârzierii, pierderi de pachete sau scăderi ale benzii efective. [1]

Aceste definiții nu sunt satisfăcătoare din mai multe motive. În primul rând, întârzierile şi pierderile de pachete sunt indicatori de performanță ce sunt impropriu folosiți drept indicatori pentru

congestie, dat fiind că modificarea acestor indicatori se poate datora unor fenomene diferite de congestie. În al doilea rând, definiția nu specifică în mod exact punctul de la care o rețea poate fi considerată congestionată. De exemplu, în timp ce o rețea care are timpi medii de întârziere în fiecare ruter de 1 până la 10 durate de deservire nu este cu siguranță congestionată, nu este clar dacă o rețea ce are timpi medii de întârziere de 1000 durate de deservire este sau nu congestionată. Nu pare posibil să se stabilească o valoare de prag care să determine congestia!

Nu în ultimul rând, o rețea congestionată din punctul de vedere al unui utilizator nu este neapărat congestionată din punctul de vedere al altuia. De exemplu, dacă utilizatorul A poate tolera o rată de pierdere a pachetelor de 1 la 1000, şi utilizatorul B poate tolera o rată de pierdere de 1 la 100, şi rata de pierdere efectivă este de 1 la 500, atunci A vă considera că rețeaua este congestionată, în timp ce pentru B rețeaua va funționa în parametri normali. O rețea trebuie considerată necongestionată numai dacă toți utilizatorii săi sunt de acord cu presupusa stare. [1]

Din cele prezentate mai sus, este clar faptul ca starea de congestionare a unei rețele depinde de perspectiva utilizatorului. Un utilizator care cere puțin de la o rețea poate tolera o pierdere de performanță mult mai bine decât un utilizator cu cerințe mari. De exemplu, un utilizator care foloseşte rețeaua numai pentru e-mail va fi mulțumit cu o întârziere de o oră, pe când acestă „performanță" nu este acceptabilă pentru un utilizator care foloseşte rețeaua pentru transimiuni de voce în timp real. Punctul cheie este noțiunea de ***folos***, folos pe care un utilizator îl obține de la rețea, şi modul în care acest folos se degradează odată cu creşterea încărcării rețelei.

Definiție

O rețea este congestionată din perspectiva utilizatorului *i* dacă utilitatea lui *I* scade ca urmare a unei creşteri a încărcării rețelei.

Observații:

1. O rețea poate fi congestionată din perspectiva unui utilizator şi necongestionată din perspectiva altuia.
2. O rețea poate fi considerată strict necongestionată numai dacă nici un utilizator nu o percepe că fiind congestionată.

O rețea care controlează congestia trebuie să fie sensibilă la funcțiile de utilitate ale utilizatorilor şi să fie capabilă să îşi direcționeze resursele astfel încât să nu existe pierderi de utilitate odată cu creşterea încărcării. Aşadar, rețeaua trebuie să fie capabilă să facă diferența între conversații şi să prioritizeze conversațiile în funcție de stringența utilității participanților la o anumită conversație.

Congestia poate fi produsă de mai mulți factori. Dacă dintr-o dată încep să sosească şiruri de pachete pe trei sau patru linii de intrare şi toate necesită aceiaşi linie de ieşire, atunci se va forma o coadă. Dacă nu există suficientă memorie pentru a le păstra pe toate, unele se vor pierde. Adăugarea de memorie poate fi folositoare până la un punct, dar Nagle a descoperit în 1987 că dacă ruterele ar avea o cantitate infinită de memorie, congestia s-ar înrăutăți în loc să se amelioreze, deoarece până să ajungă la începutul cozii pachetele au fost deja considerate pierdute (**timeout** repetat) şi s-au trimis duplicate. Toate aceste pachete vor fi trimise cu conştiinciozitate către următorul ruter, crescând încărcarea de-a lungul căii către destinație.

În figura 1.1 se prezintă simptomele unei rețele congestionate.

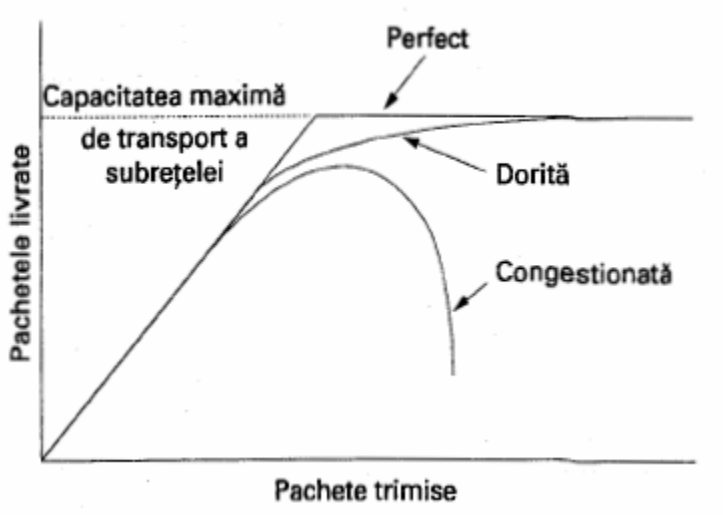


Figura 1.1: Dacă traficul este prea intens, apare congestia şi performaţele se degradează puternic. [1]

Şi procesoarele lente pot cauza congestia. Dacă unitatea centrală a ruter-ului (CPU) este lentă în execuția funcțiilor sale (introducerea în cozi, actualizarea tabelelor etc.), se pot forma cozi, chiar dacă linia de comunicație nu este folosită la capacitate maximă. Similar şi liniile cu lățime de bandă scăzută pot provoca congestia. Schimbarea liniilor cu unele mai performante şi păstrarea aceluiaşi procesor sau vice-versa de obicei ajută puțin, însă de cele mai multe ori nu fac decât să deplaseze punctul critic. Adevărata problemă este de multe ori o incompatibilitate între părți ale sistemului. Ea va persista până ce toate componentele sunt în echilibru. [1]

Este important de subliniat diferența dintre controlul congestiei şi controlul fluxului, deoarece relația este subtilă. Controlul congestiei trebuie să asigure că subrețeaua este capabilă să transporte întreg traficul implicat. Este o problemă globală, implicând comportamentul tuturor calculatoarelor gazdă, al tuturor ruterelor, prelucrarea de tip store-and-forward din rutere şi toți ceilalți factori care tind să diminueze capacitatea de transport a subrețelei.

Controlul fluxului, prin contrast, se referă la traficul capăt la capăt între un expeditor şi un destinatar. Rolul său este de a împiedica un expeditor rapid să trimită date continuu, la o viteză mai mare decât cea cu care destinatarul poate consuma datele. Controlul fluxului implică frecvent existența unui feed-back de la receptor către emițător, pentru a spune emițătorului cum se desfăşoară lucrurile la celălalt capăt.

Pentru a vedea diferența dintre aceste două concepte, să considerăm o rețea cu fibre optice cu o capacitate de 1000 Gbps pe care un calculator încearcă să transfere un fişier către un calculator personal la 1 Gbps. Deşi nu există nici o congestie (rețeaua nu are nici un fel de probleme), controlul fluxului este necesar pentru a forța supercalculatorul să se oprească des, pentru a permite calculatorului personal să primească datele.

La cealaltă extremă, să considerăm o rețea de tip store-and-forward cu linii de 1 Mbps şi 1000 de calculatoare mari., din care jumătate încearcă să transfere la 100 kbps către cealaltă jumătate. Aici problema nu apare datorită unui emițător rapid care surclasează un receptor lent, ci pentru că traficul cerut depăşeşte posibilitățile rețelei.

Motivul pentru care controlul congestiei şi controlul fluxului sunt adesea confundate este acela că unii algoritmi pentru controlul congestiei funcţionează trimiţând mesaje înapoi către diferitele surse, spunându-le să încetinească atunci când reţeaua are probleme. Astfel, un calculator gazdă poate primi un mesaj de încetinire fie din cauză că receptorul nu suportă încărcarea, fie pentru că reţeaua este depăşită.

1.2 Principii generale ale controlului congestiei [1]

Controlul congestiei este un proces foarte complex care nu şi-a găsit o rezolvare cât de cât mulţumitoare nici în ziua de azi. Chiar dimpotrivă, acest subiect este din ce în ce mai des abordat în cercetările din informatică din cauza aglomerării din ce în ce mai mări a reţelelor de calculatoare. Putem extinde problema congestiei chiar la nivelul general al unei reţele de telecomunicaţii. Aspectul care complică şi mai mult mecanismele de control al congestiei este acela că nu se poate localiza pe un nivel particular al stivei de protocoale. Algoritmii de control al congestiei pot fi implementaţi atât la nivelul routerelor de reţea cât şi la nivelul protocoalelor de transport implementate în softul de reţea al dispozitivelor comunicante. O soluţie exclusivă, bazată doar pe controlul unui singur nivel nu duce la o rezolvare mulţumitoare. De aceea se caută soluţii de compromis între cele două niveluri.

Multe din problemele care apar în sistemele complexe, cum ar fi reţelele de calculatoare, pot fi privite din punctul de vedere al unei teorii a controlului. Această abordare conduce la împărţirea tuturor soluţiilor în două grupe: în buclă deschisă şi în buclă închisă. Soluţiile în buclă deschisă încearcă să rezolve problema printr-o proiectare atentă, în esenţă să se asigure că problema nu apare. După ce sistemul este pornit şi funcţionează, nu se mai fac nici un fel de corecţii.

Instrumentele pentru realizarea controlului în buclă deschisă decid când să se accepte trafic nou, când să se distrugă pachete şi care să fie acestea, realizează planificarea deciziilor în diferite puncte din reţea. Toate acestea au ca numitor comun faptul că iau decizii fără a ţine cont de starea curentă a reţelei.

Prin contrast, soluţiile în buclă închisă se bazează pe conceptul de reacţie inversă (feedback loop). Această abordare are trei părţi, atunci când se foloseşte pentru controlul congestiei:

1. Monitorizează sistemul pentru a detecta când şi unde se produce congestia.
2. Trimite aceste informaţii către locurile unde se pot executa acţiuni.
3. Ajustează funcţionarea sistemului pentru a corecta problema.

În vederea monitorizării subreţelei pentru congestie se pot folosi diverse de metrici. Cele mai utilizate sunt procentul din totalul pachetelor care au fost distruse din cauza lipsei spaţiului temporar de memorare, lungimea medie a cozilor de aşteptare, numărul de pachete care sunt retransmise pe motiv de timeout, întârzierea medie a unui pachet, deviaţia standard a întârzierii unui pachet. În toate cazurile, valorile crescătoare indică creşterea congestiei.

Al doilea pas în bucla de reacţie este transferul informaţiei legate de congestie de la punctul în care a fost depistată la punctul în care se poate face ceva. Varianta imediată presupune trimiterea unor pachete de la ruterul care a detectat congestia către sursa sau sursele de trafic, pentru a raporta

problema. Evident, aceste pachete suplimentare cresc încărcarea rețelei exact la momentul în care acest lucru era cel mai puțin dorit, subrețeaua fiind congestionată.

Există însă şi alte posibilități. De exemplu, poate fi rezervat un bit sau un câmp în fiecare pachet, pentru a fi completat de rutere dacă congestia depăşeşte o anumită valoare de prag. Când un ruter detectează congestie, el completează câmpurile tuturor pachetelor expediate, pentru a-şi preveni vecinii.

O altă abordare este ca ruterele sau calculatoarele gazdă să trimită periodic pachete de probă pentru a întreba explicit despre congestie. Aceste informații pot fi apoi folosite pentru a ocoli zonele cu probleme. Unele stații de radio au elicoptere care zboară deasupra oraşelor pentru a raporta congestiile de pe drumuri şi a permite ascultătorilor mobili să îşi dirijeze pachetele (maşinile) astfel încât să ocolească zonele fierbinți.

În toate schemele cu feedback se speră că informarea asupra producerii congestiei va determina calculatoarele gazdă să ia măsurile necesare pentru a reduce congestia. Pentru ca o schemă să funcționeze corect, duratele trebuie reglate foarte atent. Dacă un ruter strigă STOP de fiecare dată când sosesc două pachete succesive şi PLEACĂ (eng.: GO) de fiecare dată când este liber mai mult de 20 μsec, atunci sistemul va oscila puternic şi nu va converge niciodată. Pe de altă parte, dacă va aşteaptă 30 de minute pentru a fi sigur înainte de a spune ceva, mecanismul pentru controlul congestiei reacționează prea lent pentru a fi de vreun folos real. Pentru a funcționa corect sunt necesare unele medieri, dar aflarea celor mai potrivite constante de timp nu este o treabă tocmai uşoară.

Se cunosc numeroşi algoritmi pentru controlul congestiei. Pentru a oferi o modalitate de organizare a lor, Yang şi Reddy (1995) au dezvoltat o taxonomie pentru algoritmii de control al congestiei. Ei încep prin a împărți algoritmii în cei în buclă deschisă şi cei în buclă închisă, aşa cum s-a precizat anterior. În continuare împart algoritmii cu buclă deschisă în unii care acționează asupra sursei şi alții care acționează asupra destinației. Algoritmii în buclă închisă sunt de asemenea împărțiți în două subcategorii, cu feedback implicit şi cu feedback explicit. În algoritmii cu feedback explicit, pachetele sunt trimise înapoi de la punctul unde s-a produs congestia către sursă, pentru a o avertiza. În algoritmii impliciți, sursa deduce existența congestiei din observații locale, cum ar fi timpul necesar pentru întoarcerea confirmărilor.

Prezența congestiei înseamnă că încărcarea (momentană) a sistemului este mai mare decât cea pe care o pot suporta resursele (unei părți a sistemului). Pentru rezolvare vin imediat în minte două soluții: sporirea resurselor sau reducerea încărcării. De exemplu subrețeaua poate începe să folosească linii telefonice pentru a creşte temporar lățimea de bandă între anumite puncte. În sistemele bazate pe sateliți, creşterea puterii de transmisie asigură de regulă creşterea lățimii de bandă. Spargerea traficului pe mai multe căi în locul folosirii doar a celei mai bune poate duce efectiv la creşterea lățimii de bandă. În fine, ruterele suplimentare, folosite de obicei doar ca rezerve pentru copii de siguranță (backups) (pentru a face sistemul tolerant la defecte), pot fi folosite pentru a asigura o capacitate sporită atunci când apar congestii serioase.

Totuşi, uneori nu este posibilă creşterea capacității sau aceasta a fost deja crescută la limită. Atunci singura cale de a rezolva congestia este reducerea încărcării. Sunt posibile mai multe metode pentru reducerea încărcării, cum ar fi refuzul servirii anumitor utilizatori, degradarea serviciilor pentru o parte sau pentru toți utilizatorii şi planificarea cererilor utilizatorilor într-o manieră mai previzibilă.

Unele dintre aceste metode, pe care le vom studia pe scurt, pot fi aplicate cel mai bine circuitelor virtuale. Pentru subrețelele care folosesc intern circuite virtuale aceste metode pot fi utilizate la nivelul rețea. Pentru subrețele bazate pe datagrame ele pot fi totuşi folosite uneori pentru conexiuni la nivelul transport. În acest capitol, ne vom concentra pe folosirea lor în cadrul nivelului rețea. În următorul, vom vedea ce se poate face la nivelul transport pentru a controla congestia.

O schemă de control al congestiei trebuie să îndeplinească anumite cerințe fundamentale. Acestea sunt:

1) Eficiență
2) Heterogenitate
3) Capacitatea de a funcționa cu surse defecte
4) Stabilitate
5) Scalabilitate
6) Simplitate
7) Echitate

Eficiența

Sunt două aspecte de menționat în ceea ce priveşte eficiența unei scheme de control a congestiei. În primul rând trebuie să se țină cont de cantitatea de trafic suplimentar pe care schema de control a congestiei o aduce în rețea. Am dori desigur că o asemenea schemă să presupună o povară minimă pentru rețea.

În al doilea rând trebuie să ne gândim dacă schema de control nu conduce la o subutilizare a resurselor critice ale rețelei. Scheme de control ineficiente pot "gâtui" sursele chiar dacă nu există pericol de congestie, conducând la o subutilizare a resurselor. Nu se doreşte să se lucreze suboptimal într-o rețea.

Heterogenitatea

Odată cu creşterea dimensiunilor rețelelor cât şi a acoperirii lor, acestea tind să înglobeze o varietate din ce în ce mai mare de arhitecturi hardware şi software. O schemă de control care presupune o singură dimensiune de pachet, un singur tip de protocol la nivelul transport sau un singur tip de serviciu nu poate funcționa în regulă într-un aşa mediu. Aşadar, ne dorim o schemă de control implementabilă peste o largă varietate de arhitecturi de rețea.

Capacitatea de a funcționa cu surse defecte

În rețelele actuale, administratorul de rețea nu deține mijloace administrative de control asupra surselor de mesaje. Deci, sursele (de exemplu, utilizatorii de PC-uri) sunt liberi să manipuleze protocoalele de rețea pentru a maximiza utilitatea obținută de la rețea. Atunci când un ruter informează o sursă să-şi reducă rata de trimitere, o sursă funcțională va face acest lucru. Totuşi, este posibil că o sursă „defectă" să ignore aceste semnale, în speranța că acest lucru îi va permite să trimită mai multe date. O schemă de control a congestiei nu trebuie să eşueze în prezența unor asemenea surse (o modalitate este "pedepsirea" surselor defecte).

Stabilitatea

Controlul congestiei poate fi privit că o problemă clasică de feed-back negativ. Complexitatea

suplimentară se datorează faptului că semnalele de control sunt întârziate. Cu alte cuvinte, există o întârziere finită între momentul detectării congestiei şi momentul recepționării acestui semnal de către sursă. Mai mult, sistemul prezintă zgomot iar observările parametrilor sistemului pot fi corupte. Această complexitate poate induce instabilitate în rețea. Astfel, vrem că schema de control să fie robustă şi dacă este posibil, dovedit stabilă.

Scalabilitatea

Este însuşi natura sistemelor distribuite să crească odată cu trecerea timpului. Cheia succesului într-un asemenea mediu este scalabilitatea. O schemă de control performantă a congestiei ar trebui să fie scalabilă în ceea ce priveşte doi parametri: lățimea de bandă şi dimensiunea rețelei.

Transmisiunile pe fibră optică au făcut posibilă dezvolarea de rețele cu lățimi de bandă cu trei ordine de mărime mai mari decât predecesorii lor nu foarte îndepărtați (45000 kbps față de 56 kbps). În altă ordine de idei, sute de mii de rețele locale au fost interconectate într-o enormă inter-rețea ce acoperă întregul glob. Această expansiune face că scalabilitatea unei scheme de control a congestiei să fie imperativă.

Simplitatea

Simplitatea este întotdeauna un bun de preț. Protocoale mai simple sunt mai uşor de implementat şi pot suporta mai uşor creşterile de lățime de bandă. De asemenea, protocoalele simple sunt mai probabil acceptate că standarde. Aşadar, o schemă de control al congestiei ideală trebuie să fie uşor de implementat.

Echitatea

Poate fi necesar că anumite surse să-şi reducă transmisiile datorită congestiei. Modalitatea de a alege care surse vor fi restricționate (fie cerându-le să-şi reducă transmisiile, fie renunțând la unele dintre pachetele trmise de acestea) determină cât de echitabil îşi alocă rețeaua resursele. De exemplu, dacă o cerere excesivă din partea sursei A „sufocă" sursa B, este clar că rețeaua nu îl tratează echitabil pe B. Nu este de dorit o schemă de control care să genereze o alocare neechitabilă a resurselor.

Când se vorbeşte despre echitate, sunt două aspecte de care trebuie ținut cont: definirea şi implementarea. Numeroase criterii de definire a echității au fost propuse de-a lungul timpului dar nici unul nu este absolut. Implementarea unui crieteriu de echitate generează probleme care uneori depăşesc domeniul controlului congestiei. De exemplu, s-ar putea decide că este echitabil să se avantajeze anumite surse care sunt dispuse să plătească pentru acest lucru sau care trimit un anumit tip de pachete (mai importante - email).

1.3 Politici pentru prevenirea congestiei

Voi începe studiul nostru asupra metodelor de control al congestiei prin analiza sistemelor cu buclă deschisă. Aceste sisteme sunt proiectate astfel încât să minimizeze congestia în loc să o lase să se producă şi apoi să reacționeze. Ele încearcă să-şi atingă scopul folosind politici corespunzătoare, la diferite niveluri.

Prevenirea congestiei se bazează pe un set de mecanisme ce ajută cozile echipamentelor să evite

congestia. Cozile se supraîncărca atunci când traficul sosit pe interfețele de intrare depăşeşte capacitatea interfeței/interfețelor de ieşire . Atunci când mai mult trafic soseşte pe interfețele de ieşire decât acestea pot suporta, cozile (memoriile buffer ale) routerelor se încarcă . Mecanismele de management al cozilor ne pot ajuta să evităm supraîncărcarea cozilor şi respectiv congestia. Practic aceste mecanisme ne oferă posibilitatea de a alege între pierdere de pachete şi întârzierea sau jitterul – variațiile în timp ale întârzierilor (parametri critici pe rețea).

Să începem cu nivelul legătură de date şi să ne continuăm apoi drumul spre niveluri superioare. Politica de retransmisie stabileşte cât de repede se produce timeout la un emițător şi ce transmite acesta la producerea timeout-ului. Un emițător vioi, care produce repede timeout şi retransmite toate pachetele în aşteptare folosind retrimiterea ultimelor *n,* va produce o încărcare mai mare decât un emițător calm, care foloseşte retrimiterea selectivă. Strâns legată de acestea este politica de memorare în zonă tampon. Dacă receptorii distrug toate pachetele în afara secvenței, acestea vor trebui retransmise ulterior, introducând o încărcare suplimentară. În ceea ce priveşte controlul congestiei, repetarea selectivă este, în mod sigur, mai bună decât retrimiterea ultimelor *n*.

În figura 1.2 sunt prezentate diferite politici pentru nivelul legătură de date, rețea şi transport, care pot influența congestia (Jain, 1990).

Nivel	**Politică**
Transport	Politica de retransmisie Politica de memorare temporară a pachetelor în afară de secvență (out-of-order caching) Politica de confirmare Politica de control al fluxului Determinarea timeout-ului
Rețea	Circuite virtuale contra datagrame în interiorul subrețelei Plasarea pachetelor în cozi de aşteptare şi politici de servire Politica de distrugere a pachetelor Algoritmi de dirijare Gestiunea timpului de viață al pachetelor
Legătură de date	Politica de retransmitere Politica de memorare temporară a pachetelor în afară de secvență (out-of-order caching) Politica de confirmare Politica de control al fluxului

Figura 1.2 - Politici care influențează congestia. [1]

Şi politica de confirmare afectează congestia. Dacă fiecare pachet este confirmat imediat, pachetele de confirmare vor genera un trafic suplimentar. Totuşi, în cazul în care confirmările sunt preluate de traficul de răspuns, se pot produce timeout-uri şi retransmisii suplimentare. O schemă prea strânsă pentru controlul fluxului (de exemplu fereastră mică) reduce volumul de date şi ajută în lupta cu congestia.

La nivelul rețea, alegerea între folosirea circuitelor virtuale şi datagrame influențează congestia, deoarece mulți algoritmi pentru controlul congestiei funcționează doar pe subrețele cu circuite virtuale. Plasarea în cozi de aşteptare a pachetelor şi politicile de servire specifică dacă ruterele au o coadă pentru fiecare linie de intrare, o coadă pentru fiecare linie de ieşire sau ambele. Mai precizează ordinea în care se prelucrează pachetele (de exemplu round robin sau bazată pe priorități). Politica de distrugere a pachetelor este regula care stabileşte ce pachet este distrus dacă nu mai există spațiu. O politică bună va ajuta la eliminarea congestiei, pe când una greşită o va accentua.

Un algoritm de dirijare bun poate ajuta la evitarea congestiei prin răspândirea traficului de-a lungul tuturor liniilor, în timp ce un algoritm neperformant ar putea trimite toate pachetele pe aceeaşi linie, care deja este congestionată. În fine, gestiunea timpului de viață asociat pachetelor stabileşte cât de mult poate trăi un pachet înainte de a fi distrus. Dacă acest timp este prea mare, pachetele pierdute vor încurca pentru mult timp activitatea, iar dacă este prea mic, este posibil să se producă timeout înainte de a ajunge la destinație, provocând astfel retransmisii.

La nivelul transport apar aceleaşi probleme ca la nivelul legăturii de date dar, în plus, determinarea intervalului de timeout este mai dificil de realizat, deoarece timpul de tranzit prin rețea este mai greu de prezis decât timpul de tranzit pe un fir între două rutere. Dacă intervalul de timeout este prea mic, vor fi trimise inutil pachete suplimentare. Dacă este prea mare, congestia se va reduce, însă timpul de răspuns va fi afectat de pierderea unui pachet.

CAPITOLUL 2

2. Tehnici cunoscute de prevenire a pierderii pachetelor în rețele

2.1. Introducere

Principalul scop al asigurării QoS este acela al controlării pierderilor de pachete.

Aceasta se realizează printr-un management adecvat al cozilor. Pachetele sunt pierdute din două motive: datorită erorilor de transmisie (fenomen relativ rar), sau datorită congestiilor din rețea. Pentru a controla și a evita congestiile din rețea trebuie implementate unele mecanisme în punctele de acces și în ruterele intermediare.

La capetele rețelei se află TCP-ul sau aplicațiile client care trebuie să implementeze un mecanism de adaptare la condițiile rețelei. În interiorul rețelei ruterele sunt cele care asigură, pe baza cozilor, un management al traficului. Cozile din arhitectura ruterelor sunt gândite să absoarbă vârfurile de trafic. Limitând dimensiunile cozilor scad și întârzierile pachetelor prin rețea. Din nefericire, atunci când cozile sunt pline, pachetele sunt eliminate.

În rețelele de mare viteză, gateway-urile sunt proiectate cu cozi de lungime maximă pentru a controla congestiile. Protocolul de transport TCP detectează o congestie numai după ce un pachet a fost aruncat de către gateway. Cu toate acestea, este evident că deși cozile au dimensiuni mari nu este de dorit ca acestea să fie pline în majoritatea timpul – dacă s-ar întampla acest lucru, întârzierea medie din rețea ar crește semnificativ. Prin urmare, odată cu creșterea vitezei rețelei, este din ce în ce mai importantă prezența unor mecanisme care să mențină throughput-ul ridicat, însă dimensiunea cozilor mică.

În absența unui feedback explicit din partea gateway-ului, protocolul de nivel 4, transport, ar putea deduce apariția congestiei prin apariția gâtuirilor (bottlenecks), prin modificările ce apar în throughput, prin modificările întarzierilor punct la punct, dar și prin aruncarea pachetelor. De asemenea, perspectiva pe care o are o conexiune individuală este limitată de modificările ce apar în traficul acestei conexiuni, de modelul traficului, de lipsa cunoașterii numărului de gateway-uri ce sunt afectate de congestii, de schimbările de rutare posibile, dar și de propagarea întârzierilor.

Cea mai eficientă detecție a apariției congestiei o realizează gateway-ul. Numai gateway-ul are o vedere unică a modului în care evoluează cozile în timp. În plus, gateway-ul este împărțit între mai multe conexiuni cu timpi de răspuns variați, cu toleranțe față de întârzieri diferite și cerințe legate de throughput de asemenea diferite. Deciziile legate de durata și dimensiunea congestiei permise la nivelul gateway-ului sunt luate cel mai bine chiar de către gateway.

2.2. Protocolul TCP

2.2.1 Informații generale

TCP este un protocol punct la punct, orientat pe conexiune, introdus pentru a proteja rețeaua împotriva supraîncărcării puternice și pentru a realiza o împărțire echitabilă a lățimii de bandă pentru diferite surse TCP. Protocolul TCP detectează erori precum pachete eronate, pierdute sau duplicate. Emițătorul atribuie o secvență numerică fiecărui pachet (segment) și solicită o confirmare pozitivă (ACK) de la receptor. Receptorul detectează pierderea de pachete verificând secvența numerică și confirmând, la fiecare pachet primit, ultimul pachet recepționat în ordinea corectă. Pentru fiecare pachet care nu este în ordine, o confirmare duplicat este trimisă. Acest ACK duplicat are rolul de a notifica perechea de faptul că a fost detectată o pierdere, și pentru a îi transmite emițătorului următoarea secvență numerică așteptată.

O sursă TCP este considerată saturată precum în cazul unei sesiuni FTP, unde toate segmentele au dimensiunea maximă posibilă, rezultând pachete de lungimi constante. Termenul de pachet este echivalent cu termenul segment. TCP-ul deține o fereastră de congestie și o limită pentru „slow-start" ce sunt descrise de variabilele CWND și SSTHRESH. CWND, determină numărul maxim de pachete ce pot fi neconfirmate în rețea, iar SSTHRESH controlează creșterea dimensiunii CWND. Dacă CWND este mai mică decât SSTHRESH, în faza de „slow-start", fereastra de congestie este incrementată cu câte un pachet pentru fiecare confirmare neduplicat primită. Aceasta duce la o creștere exponențială a CWND. Când CWND este mai mare sau egală cu SSTHRESH, în faza de evitare a congestiei, CWND crește cu câte un pachet la fiecare RTT (round trip time), ceea ce corespunde unei creșteri liniare a CWND (*Figura 2.1*). [2]

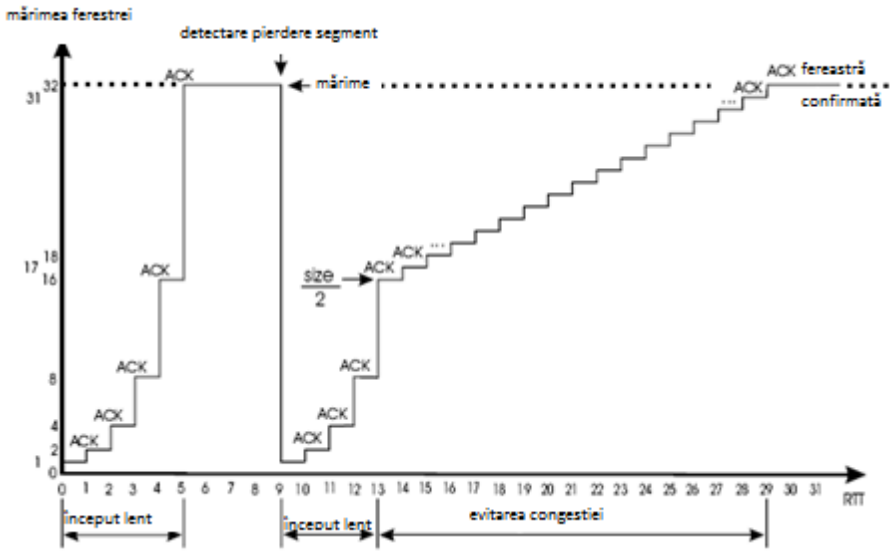


Figura 2.1. Ilustrarea fazelor „slow-start" și de evitare a congestiei pentru TCP
Analytic Performance Evaluation of the RED Algorithm for QoS in TCP/IP Networks

TCP reacționează la pierderea pachetelor în două feluri diferite. Setează câte un cronometru pentru fiecare pachet trimis. În cazul în care nu este primită o confirmare pentru acel pachet, înainte de expirarea timpului, CWND își reduce dimensiunea la dimensiunea maximă a segmentului iar SSTHRESH este setat la *max* (flight size / 2, 2 * dimensiunea segmentului). Flight size reprezintă cantitatea de informație neconfirmată în rețea. Presupunând că emițătorul este saturat, valoarea CWND este egală cu valoarea flight size. Când emițătorul recepționează trei ACK-uri duplicat, înainte de expirarea timpului, este realizată o recuperare rapidă (fast recovery): într-o manieră simplificată, TCP trimite pachetul pierdut din nou, reduce SSTHRESH la *max* (flight size/2, 2 * dimensiunea segmentului) și setează CWND la noua valoare calculată SSTHRESH (*Figura 2.2*) [2].

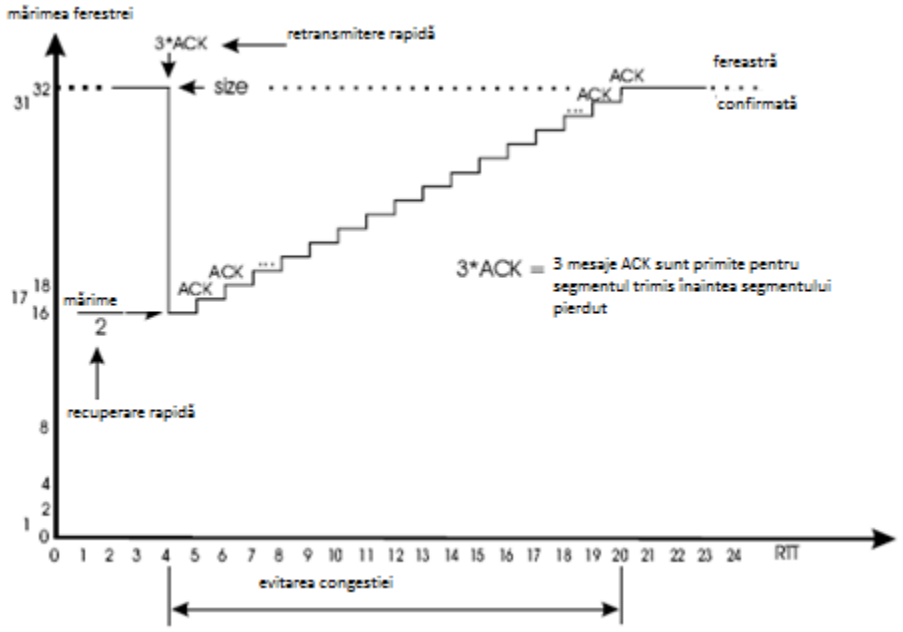


Figura 2.2. Algoritmul de retransmitere rapidă a TCP
Analytic Performance Evaluation of the RED Algorithm for QoS in TCP/IP Networks

2.2.2 Algoritmi TCP de prevenire a congestiei

Controlul congestiei se referă la controlarea traficului ce pătrunde într-o rețea de telecomunicații, ceea ce presupune evitarea supraîncărcării legăturilor dintre nodurile intermediare ale rețelei, prin reducerea pachetelor trimise. Așa cum s-a prezentat anterior, controlul congestiei nu trebuie confundat cu controlul fluxului care protejează receptorul de a fi „copleșit” de către emi’ător.

Există mai mulți algoritmi de prevenire a congestiei prevăzuți de TCP. Primii algoritmi apăruți sunt TCP Tahoe, și TCP Reno, ambii având la bază regulile de „slow-start” și „fast retransmit” prezentate anterior. Cei doi algoritmi, Tahoe și Reno, diferă prin modul în care reacționează la pierderea de pachete:

Tahoe: pierderea este detectată atunci când timpul de timeout expiră înainte ca o confirmare să fie recepționată. În acel moment, Tahoe reduce fereastra de congestie la valoarea maximă a dimensiunii unui segment (MSS – maximum segment size) și se resetează la faza de „slow-start".

Reno: în cazul în care sunt recepționate 3 ACK duplicat, Reno înjumătățește fereastra de congestie, efectuează o retransmitere rapidă (fast retransmit), și intră în recuparare rapidă (fast recovery).

Până la mijlocul anilor 1990, toate timeout-urile și întarzierile dus-intors erau bazate numai pe ultimul pachet din buffer. Cercetătorii Larry Peterson și Lawrence Brakmo de la Universitatea din Arizona, au introdus ***TCP Vegas*** în care timeout-ul este setat iar întarzierea dus-întors este măsurată pentru fiecare pachet din buffer. De asemenea, acest algoritm folosește o creștere aditivă a ferestrei de congestie. Această varianta nu a cunoscut o răspândire foarte mare, în afara laboratorului cercetătorilor.

TCP New Reno îmbunătățește retransmisia în timpul fazei de recuperare rapidă (fast recovery) a TCP Reno. Pe parcursul recuperării rapide, pentru fiecare ACK duplicat, este transmis un nou pachet netrimis de la sfârșitul ferestrei de congestie, pentru a menține fereastra plină. Pentru fiecare ACK parțial din spațiul de secvențe numerice, emițătorul trimite următorul pachet cu secvența numerică următoare ACK-ului.

Întrucât cronometrul de timeout este resetat ori de cate ori există un progres în buffer-ul de transmisie, New Reno poate umple găuri mari sau mai multe găuri în spațiul de secvențe. Deoarece New Reno poate trimite pachete noi la sfârșitul ferestrei de congestie în timpul recuperării rapide, este menținut un throughput mare pe durata procesului de umplere a găurilor, chiar și atunci când există mai multe găuri, de mai multe pachete fiecare. Când TCP intră în recuperare rapidă, acesta înregistrează cea mai mare secvență numerică neconfirmată. Când această secvență numerică este confirmată, TCP revine la stare de evitare a congestiei.

Atunci când nu au loc pierderi de pachete, dar in schimb pachetele sunt reordonate cu mai mult de 3 secvențe numerice, apare o problemă în New Reno. Când acest fenomen are loc, New Reno intră greșit în recuperare rapidă, însă când pachetul reordonat este livrat, are loc progresul secvenței numerice ACK și din acel moment până la sfârșitul recupeăării rapide, fiecare bit al secvenței numerice produce un duplicat și retransmisia nenecesară.

TCP Hybla are rolul de a elimina penalizarea conexiunilor TCP ce încorporează legături terestre cu latențe foarte mari sau legături radio, datorită timpilor dus-întors mai mari. Provine dintr-o evaluare analitică a dinamicii ferestrei de congestie, ce sugerează înlăturarea dependenței performanței de timpul dus-intors (RTT – round trip time).

TCP BIC (binary increase congestion control) este o implementare a TCP cu un algoritm de control al congestiei optimizat pentru rețele de viteze mari, cu latențe mari (numite LFN – long fat networks, in RFC 1072). BIC este folosit implicit în kernel-urile Linux, de la 2.6.8 la 2.6.18.

TCP Cubic este o versiune mai puțin agresivă a BIC, în care fereastra este o funcție cubică de timp, de la momentul ultimei congestii, cu punctul de inflexiune setat la fereastra anterioară evenimentului.

2.3. Algoritmul de prevenire a congestiei DECbit

În cazul schemei DECbit, gateway-ul folosește un bit în antetul pachetului pentru a furniza un feedback rețelei cu privire la apariția congestiei. Când un pachet ajunge la gateway, aceasta calculează lungimea medie a cozii pentru ultima perioadă (*busy+idle*) plus perioada *busy* curentă (gateway-ul este *busy* atunci când sunt transmise pachete, altfel este *idle*). Când lungimea medie a cozii depășește unu, gateway-ul setează bitul de indicare a congestiei în antetul pachetelor ce sosesc.

Sursa folosește ferestre pentru controlul fluxului, pe care le updatează o dată la fiecare două perioade dus-întors. Dacă cel puțin jumătate din pachetele din ultima fereastră au avut bitul de indicare al congestiei setat, atunci fereastra este scăzută exponențial. Altfel, aceasta este crescută liniar.

Există câteva diferențe semnificative între gateway-urile ce au implementat algoritmul DECbit și gateway-urile ce au implementat algoritmul RED, prezentat mai târziu în această lucrare.

Prima diferență se referă la metoda de calculare a dimensiunii medii a cozii. Deoarece schema DECbit folosește ultimul ciclu (busy+idle) plus perioada busy curentă, pentru calcularea dimensiunii medii a cozii, dimensiunea cozii poate fi mediată după o perioadă de timp destul de scurtă. În rețele de mare viteză, cu buffer-e foarte mari la nivelul gateway-ului, este de dorit ca controlul constantei de timp folosite pentru calculul dimensiunii medii a cozii, să fie făcut explicit. Acest lucru este realizat în cazul gateway-urilor RED. În rețele DECbit, sursele se uită la fracțiunea de pachete care au fost marcate în ultimul interval dus-întors. În cazul rețelelor cu gateway RED, sursa își reduce fereastra chiar și în cazul în care există un singur pachet marcat.

A doua diferență dintre cele două scheme, este metoda prin care gateway-ul alege conexiunile pe care le anunță despre apariția congestiei. În schema DECbit nu este nici o diferență conceptuală între algoritmul pentru detecția congestiei și algoritmul pentru setarea bitului de congestie. Când un pachet sosește la gateway iar dimensiunea medie a cozii este prea mare, bitul pentru indicarea congestiei este setat în antetul pachetului. Din cauza acestei metode de marcare a pachetelor, rețele DECbit pot întampina probleme în fața traficului în rafale. Acest lucru este evitat în cazul gateway-urilor RED marcarea pachetelor făcându-se aleator. Pentru gateway-uri ce evită congestia, destinate pentru a lucra cu TCP, o motivație în plus pentru marcarea aleatoare este evitarea sincronizării globale ce poate rezulta în urma reducerii simultane a mai multor conexiuni TCP, a ferestrei de comunicație. Acest aspect nu este la fel de important în cazul DECbit, deoarece fiecare sursă își micșorează moderat fereastra ca răspuns la apariția congestiei.

CAPITOLUL 3

3. Random Early Drop (RED)

3.1 Introducere

În 1993, Sally Floyd și Van Jacobson au venit cu o lucrare interesantă despre "cum să detectezi congestia în rețele folosind ruterele, cu ajutorul unui mecanism de detecție aleatoare timpurie".

Algoritmul RED – Random Early Detection, cunoscut și ca Random Early Discard sau Random Early Drop, este un mecanism de evitare a congestiei, care are în plus câteva metode pentru detectarea congestiei și pentru alegerea conexiunii care să notifice această congestie. Ruterele RED sunt folosite de obicei cu rețele TCP/IP, fiind proiectate pentru rețele unde un singur pachet marcat sau aruncat este suficient pentru semnalarea congestiei la nivelul transport.

Random Early Detection (RED) reduce congestia în cozi aruncând pachete astfel încât anumite conexiuni TCP pe o perioadă limitată de timp transmit mai puține pachete pe rețea. În loc să se aștepte până când o coadă se supraîncarcă, cauzând un număr mare de pachete aruncate, RED în mod intenționat aruncă o parte din pachete înainte ca bufferul să fie plin. Această acțiune este menită să facă transmițătorii ce generează traficul să reducă volumul de date transmis pe rețea.

Numele "detecție aleatoare timpurie" descrie funcționarea algoritmului. RED alege aleator pachete de descărcat după ce a luat decizia de descărcare de pachete. Logica de funcționare a RED conține 2 aspecte principale: RED trebuie să detecteze congestia înainte ca aceasta să se intample, cu alte cuvinte, RED trebuie să decidă care sunt condițiile care vor duce la decizia de aruncare de pachete, iar când se ia această decizie, trebuie să fie definit numărul de pachete de aruncat. În primul rând RED verifică adâncimea cozii echipamentului. Apoi RED calculează adâncimea medie pentru acea interfață, apoi decide dacă va apărea fenomenul de congestie. RED folosește încărcarea medie pentru că încărcarea reală își schimbă valoarea mult mai des. Pentru că RED evită efectele sincronizării, este necesar ca aruncare de pachete să fie egal proporționată.

Avantajul acestui mecanism de control al congestiei la nivelul ruterului este că funcționează cu actualele protocoale de transport, și nu necesită ca toate ruterele din Internet să folosească același mecanism de control al congestiei. Ruterele RED reprezintă un mecanism simplu de evitare a congestiei, și poate fi implementat gradual în actualele rețele TCP/IP fără a schimba protocolul de transport.

Scopul ruterelor RED este de a preveni apariția congestiei prin controlul dimensiunii medii a cozii de pachete.

Dimensiunea medie a cozii este calculată de RED folosind un filtru trece-jos cu o medie ponderată mobilă, exponențială. Dimensiunea medie a cozii este comparată cu 2 praguri, un prag *minim*

și un prag *maxim*. Atunci când dimensiunea medie a cozii este mai mică decat pragul minim, nu este marcat nici un pachet. Atunci când dimensiunea medie a cozii este mai mare decât pragul maxim, fiecare pachet ce sosește este marcat. Dacă pachetele marcate sunt de fapt aruncate, sau dacă toate nodurile sursă cooperează, acest procedeu poate asigura ca dimensiunea medie a cozii să nu depășească semnificativ pragul maxim.

Atunci când dimensiunea medie a cozii se situează între minim și maxim, fiecare pachet ce sosește este marcat cu probabilitatea p_a, unde p_a este o funcție de dimensiunea medie a cozii ***avg***. De fiecare dată când un pachet este marcat, probabilitatea ca acel pachet să aparțină unei anumite conexiuni, este proporțională cu cota pe care acea conexiune o deține din lățimea de bandă totală ce traversează gateway-ul.

```
Pentru fiecare pachet sosit
     Calculează dimensiunea medie a cozii avg
     Dacă min_th <= avg < max_th
          Calculează probabilitatea p_a
          Marchează pachetul sosit cu probabilitatea p_a
     Altfel dacă max_th <= avg
          Marchează pachetul sosit
```

Figura 3.1 Algoritmul RED [4]

Prin urmare este alcătuit din 2 algoritmi separați. Algoritmul care calculează dimensiunea medie a cozii determină gradul maxim de *rafale* ce va fi permis în coada gateway-ului. Algoritmul care calculează probabilitatea de marcare a unui pachet, determină cât de des marchează gateway-ul pachete, la un anumit nivel de congestie. Obiectivul este ca gateway-ul să marcheze pachete la intervale de timp egal distanțate, pentru a evita sincronizarea globală, și de a marca pachete suficient de des pentru a putea controla dimensiunea medie a cozii.

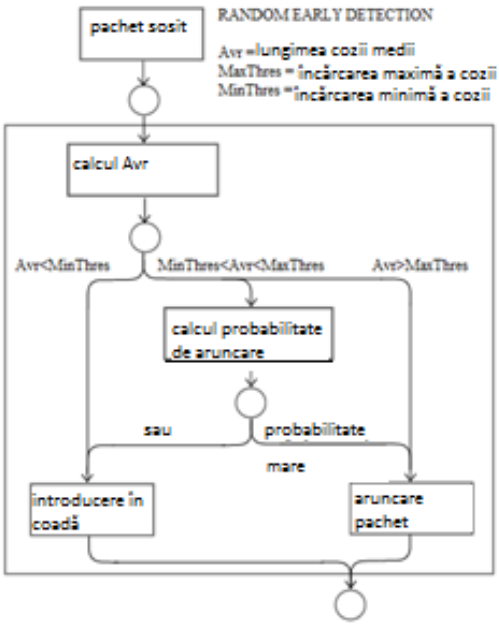


Figura 3.2 Schema algoritmului RED [4]

Calculul dimensiunii medii a cozii, are în vedere perioada cât timp coada este goală (perioadă *idle*), estimând numărul ***m*** de pachete care *ar fi putut* să fie transmise de către gateway pe durata acesteia. După perioada *idle*, gateway-ul calculează dimensiunea medie a cozii ca și când ***m*** pachete ar fi ajuns pe durata acesteia, într-o coadă goală.

Cum ***avg*** variază între $\boldsymbol{min_{th}}$ și $\boldsymbol{max_{th}}$, probabilitatea de marcare a pachetelor $\boldsymbol{p_b}$ variază liniar între ***0*** și $\boldsymbol{max_p}$:

$$p_b = max_p \frac{avg - min_{th}}{max_{th} - min_{th}}$$

Probabilitatea finală de marcare a pachetelor $\boldsymbol{p_a}$ crește încet pe măsură ce numărătoarea crește față de ultimul pachet marcat.

$$p_a = \frac{p_b}{1 - count * p_b}$$

În acest mod, se asigură că gateway-ul nu așteaptă prea mult pentru a marca un nou pachet.

Gateway-ul marchează fiecare pachet ce ajunge aici, atunci când dimensiunea medie a cozii ***avg*** depășește $\boldsymbol{max_{th}}$.

O opțiune pentru algoritmul RED este măsurarea cozii în *bytes* și nu în *pachete*. În acest mod, dimensiunea medie a cozii reflectă precis întârzierea medie la gateway. Când este folosită această opțiune, algoritmul este modificat pentru a asigura faptul că probabilitatea cu care este marcat un pachet, este proporțională cu dimensiunea pachetului în bytes.

$$p_b = max_p \frac{avg - min_{th}}{max_{th} - min_{th}}$$

$$p_b = p_b \frac{PacketSize}{MaximumPacketSize}$$

$$p_a = \frac{p_b}{1 - count * p_b}$$

În acest fel, un pachet FTP are șanse mai mari să fie marcat în comparație cu un pachet TELNET.

Inițializare
  $avg \leftarrow 0$
  $count \leftarrow -1$
Pentru fiecare pachet sosit
  Calculează noua dimensiune medie a cozii ***avg***
    Dacă coada **nu** este goală
      $avg \leftarrow (1 - w_q)\ avg + w_q q$
    Altfel
      $m \leftarrow f(time - q_{time})$
      $avg \leftarrow (1 - w_q)^m avg$
  Dacă $min_{th} \leq avg < max_{th}$
    Incrementează ***count***
    Calculează probabilitatea $\boldsymbol{p_a}$:

$$p_b = max_p \frac{avg - min_{th}}{max_{th} - min_{th}}$$

$$p_a = \frac{p_b}{1 - count * p_b}$$

    Marchează pachetul sosit cu probabilitatea $\boldsymbol{p_a}$
    $count \leftarrow 0$
  Altfel dacă $max_{th} \leq avg$
    Marchează pachetul sosit
    $count \leftarrow 0$
  Altfel $count \leftarrow -1$
Când coada devine goală
  $q\_time \leftarrow time$

Figura 3.3 Algoritmul Random Early Drop (RED) detaliat [4]

Variabile salvate:

avg: dimensiunea medie a cozii

q_time: momentul de început al perioadei *idle*
count: pachete de la ultimul pachet marcat

Parametrii fixați

w_q**:** ponderea cozii
min_{th}**:** pragul minim al dimensiunii cozii
max_{th}: pragul maxim al dimensiunii cozii
max_p: valoarea maximă pentru p_b

Alți parametri

p_a: probabilitatea de marcare a pachetelor, curentă
q: dimensiunea curentă a cozii
time: timpul curent
f(t): o funcție liniară de timp ***t***

Ponderea cozii w_q este determinată de către mărimea și durata *rafalelor* în coadă, ce sunt permise de către gateway. Pragurile min_{th} și max_{th} sunt determinate de dimensiunea medie a cozii dorite. Dimensiunea medie a cozii ce stabilește compromisurile dorite (cum ar fi compromisul dintre maximizarea throughput-ului și minimizarea întârzierii), depinde de caracteristicile rețelei.

3.2 Calculul lungimii medii a cozii

Pentru calcularea lungimii medii a cozii, este folosit un filtru trece-jos. Astfel, creșterile de scurtă durată ale lungimii cozii datorate traficului în rafale nu vor duce la o creștere semnificativă a lungimii medii a cozii.

Filtrul trece jos este o medie ponderată mobilă,exponențială:

$$avg \leftarrow \ 1 - w_q \ avg + w_q q$$

Ponderea w_q determină constanta de timp a filtrului trece jos. Mai jos sunt discutate limitele de sus și de jos ale lui w_q.

3.2.1. w_q - valori posibile

Dacă w_q este prea mare, atunci procedura de mediere nu va filtra congestia tranzitorie de la nivelul gateway-ului.

Presupunem că la început coada este goală, cu o lungime medie a cozii de valoare zero, după care, coada crește de la 0 la ***L*** pachete, după ***L*** pachete sosite. După sosirea celui de-al ***L*** pachet, dimensiunea medie a cozii ***avg*** este [4]:

$$avg_L = \sum_{i=1}^{L} i w_q (1 - w_q)^{L-i}$$

$$= w_q (1 - w_q)^L \sum_{i=1}^{L} i \left(\frac{1}{1 - w_q}\right)^i$$

$$= L + 1 + \frac{(1-w_q)^{L+1} - 1}{w_q} \text{ (1) [4]}$$

În figura de mai jos [4] este prezentată dimensiunea medie a cozii $\boldsymbol{avg_L}$ pentru mai multe valori ale lui $\boldsymbol{w_q}$ și ***L***. Pe axa ***x*** este reprezentat $\boldsymbol{w_q}$, cu valori între 0.001 și 0.005, iar pe axa ***y*** este reprezentat ***L*** cu valori cuprinse între 10 și 100. De exemplu, pentru $\boldsymbol{w_q}$ = 0.001, după o creștere a cozii de la 0 la 100 de pachete, dimensiunea medie a cozii $\boldsymbol{avg_{100}}$ este 4.88 pachete.

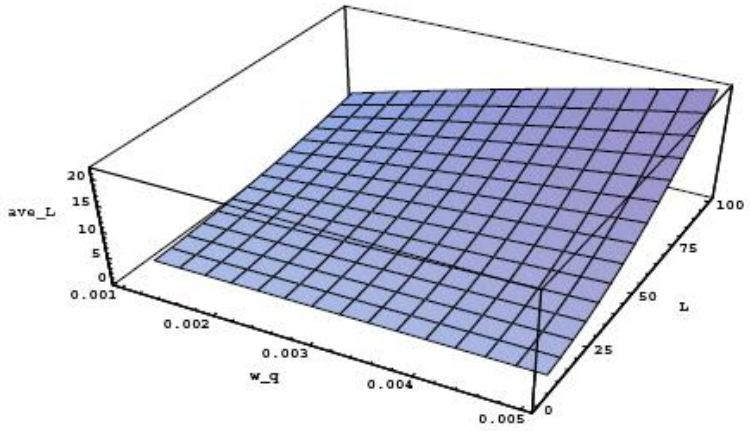


Figura 3.4. avg_L ca funcție de w_q și L

Având setat un prag minim $\boldsymbol{min_{th}}$ și știind că dorim să permitem rafale de câte ***L*** pachete, atunci $\boldsymbol{w_q}$ ar trebui ales asftel încât să fie satisfăcută ecuația de mai jos, pentru $\boldsymbol{avg_L < min_{th}}$:

$$L + 1 + \frac{(1-w_q)^{L+1} - 1}{w_q} < min_{th} \text{ (2) [4]}$$

De exemplu, pentru $\boldsymbol{min_{th}}$ = 5 și ***L***=50, $\boldsymbol{w_q}$ trebuie ales <=0.0042.

Gateway-urile ce au implementat algoritmul RED trebuie să mențină valoarea calculată a lungimii medii a cozii, ***avg*** sub un anumit prag. Cu toate acestea, aceasta nu are nici un sens dacă media calculată ***avg*** nu este o reflecție rezonabilă a lungimii medii curente. Dacă w_q este setat prea jos, atunci ***avg*** răspunde prea încet la schimbările actuale ale lungimii cozii. În acest caz, gateway-ul este incapabil să sesizeze apariția unei congestii.

Presupunem trecerea cozii de la 0 la 1 pachet, și că pe masură ce pachetele ajung și pleacă cu aceeași rată, coada rămâne la un pachet. De asemenea presupunem că inițial, dimensiunea medie a cozii era zero. În acest caz, vor fi necesare $\frac{-1}{ln\ 1-w_q}$ sosiri de pachete (cu dimensiunea cozii rămânând la 1) până când dimensiunea medie a cozii ***avg*** devine 0.63=1-1/e. Pentru w_q = 0.001, sunt necesare 1000 de sosiri de pachete; pentru w_q = 0.002 sunt necesare 500 de sosiri de pachete; pentru w_q = 0.003 sunt necesare 333 sosiri de pachete.

3.2.2 Alegerea pragurilor min_{th} și max_{th}

Valorile optime pentru min_{th} și max_{th} depind de dimensiunea medie a cozii. Dacă traficul este în general în rafale, atunci min_{th} trebuie să fie corespunzător de mare pentru a permite menținerea utilizării legăturii la un nivel suficient de înalt. Pentru trafic tipic cu întârzieri destul de mari, un prag minim de un pachet ar duce la o legătură inaccetabil de proastă.

Valoarea optimă pentru max_{th} depinde în parte de întârzierea medie maximă ce este permisă de către gateway.

Algoritmul funcționează eficient atunci când $max_{th} - min_{th}$ este mai mare decât creșterea dimensiunii medii a cozii într-o perioadă de timp. De obicei, max_{th} este cel puțin de 2 ori min_{th}.

3.3 Calculul probabilității de marcare a pachetelor

Probabilitatea inițială de marcare a pachetelor p_b este calculată ca o funcție liniară de lungimea medie a cozii. Mai jos sunt comparate două metode pentru calculul probabilității finale de marcare a pachetelor și este ilustrat avantajul folosirii celei de-a doua metode. În prima metodă, când lungimea medie a cozii este constantă, numărul de pachete sosite între 2 pachete marcate, este o *variabilă aleatoare geometrică*. În cea de-a doua metodă, numărul de pachete sosite între 2 pachete marcate este o *variabilă aleatoare uniformă*.

Probabilitatea inițială de marcare a pachetelor este calculată cu ajutorul relației

$$p_b = max_p \frac{avg - min_{th}}{max_{th} - min_{th}}$$

Parametrul $\boldsymbol{max_p}$ reprezintă valoarea maximă pentru probabilitatea de marcare a pachetelor $\boldsymbol{p_b}$ atinsă când lungimea medie a cozii ajunge la pragul maxim.

3.3.1 Metoda 1: Variabilă aleatoare geometrică

Să presupunem că fiecare pachet este marcat cu probabilitatea $\boldsymbol{p_b}$. Considerăm timpul dintre marcări $\boldsymbol{X}$, ca fiind dat de numărul de pachete care sosesc, după ce a fost marcat un pachet, până când este marcat un altul. Deoarece fiecare pachet este marcat cu probabilitatea $\boldsymbol{p_b}$,

$$Prob\ X = n\ = \ 1 - p_b\ ^{n-1} p_b$$

Prin urmare, $\boldsymbol{X}$ este o variabilă aleatoare *geometrică*, cu parametru $\boldsymbol{p_b}$ și $\boldsymbol{E[X]=1/p_b}$.

Cu o dimensiune medie a cozii constantă, scopul este de a marca pachetele la intervale destul de regulate. Nu este de dorit să avem prea multe pachete marcate foarte apropiate, și de asemenea nu este de dorit ca intervalele de timp dintre marcări să fie foarte mari. Ambele variante pot duce la sincronizări globale, cu mai multe conexiuni reducându-și ferestrele în același timp, și ambele variante se pot intampla când $\boldsymbol{X}$ este o variabilă aleatoare geometrică.

3.3.2 Metoda 2: Variabilă aleatoare uniformă

O variantă mai bună este ca $\boldsymbol{X}$ să fie o variabilă aleatoare *uniformă* din *{1, 2, ..., 1/p_b}* (presupunând pentru simplitate că 1/p_b este un întreg). Acest lucru se întâmplă când probabilitatea de marcare pentru fiecare pachet sosit este $\frac{p_b}{1-count*p_b}$, unde ***count*** este numărul de pachete nemarcate ce au sosit de când a fost marcat ultimul pachet. În acest caz,

$$Prob\ X = n\ = \frac{p_b}{1 - \ n-1\ p_b} \prod_{i=0}^{n-2} 1 - \frac{p_b}{1 - ip_b}$$

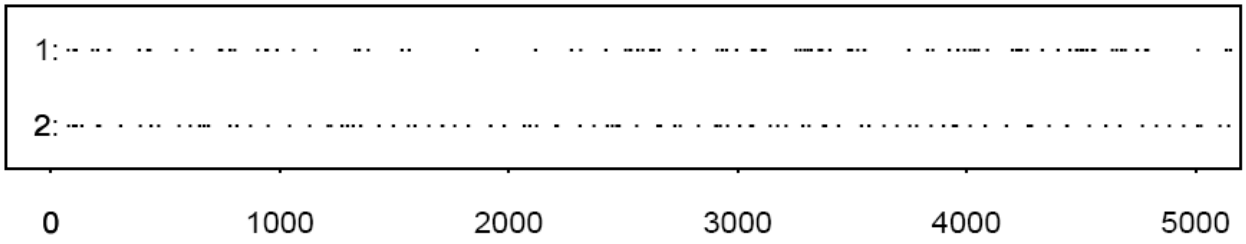
$$= p_b pentru\ 1 \leq n \leq \frac{1}{p_b}$$

Și

$$Prob\ X = n\ = 0\ pentru\ n > \frac{1}{p_b}$$

În acest caz $E\ X\ = \frac{1}{2p_b} + \frac{1}{2}$.

În figura de mai jos, este prezentat un experiment ce compară cele două metode de marcare a pachetelor. Linia de sus reprezintă metoda 1, unde fiecare pachet este marcat cu probabilitatea ***p***, pentru ***p*** = 0.02. Linia de jos, reprezintă metoda 2, unde fiecare pachet este marcat cu probabilitatea $\frac{p}{1+ip}$, pentru ***p*** = 0.01 și cu ***i*** numărul de pachete nemarcate de la ultimul pachet marcat. Ambele metode au marcat aproape 100 de pachete din 5000 sosite. Axa ***x*** reprezintă numărul de pachete. Pentru fiecare metodă, este reprezentat câte un punct pentru fiecare pachet marcat. Așa cum era de așteptat, pachetele marcate sunt mai grupate în cazul primei metode, decât în cea de-a doua.



Numărul de pachete
(rândul de sus pentru metoda 1, rândul de jos pentru metoda 2)

Figura 3.5 Pachete marcate aleator, comparație între 2 metode de marcare

3.4 Alte versiuni de implementare ale algoritmului RED

Există mai multe versiuni ale acestui algoritm, cele mai cunoscute fiind WRED, FRED și ARED.

3.4.1 WRED (Weighted RED):

WRED este implementarea Cisco pentru Random Early Detection (RED) și este folosită ca o alternativă pentru modul implicit de aruncare a pachetelor de la sfârșitul unei cozi atunci când coada este plină (tail-drop behaviour). WRED este o metodă de evitare a congestiei. Este adresată problema sincronizării globale, când mai multe transmisii TCP traversează un ruter și ies pe aceeași interfață. În momentul începerii comunicației TCP viteza este relativ mică și dacă banda încă nu este ocupată, fiecare comunicație va crește în viteză. Când suma tututror transmisiilor depășește lățimea de bandă a interfeței de ieșire, pachetele vor începe să fie aruncate. Acest lucru va determina toate comunicațiile să-și micșoreze viteza de transmisie în același timp, ceea ce va duce la o scădere semnificativă a vitezei globale. Din cauza că sesiunile TCP au avut viteza scăzută în același timp, se vor întoarce la situația inițială în care viteza fiecărei sesiuni este mică iar lățimea de bandă este încă suficientă. După cum se poate observa procesul se va relua.

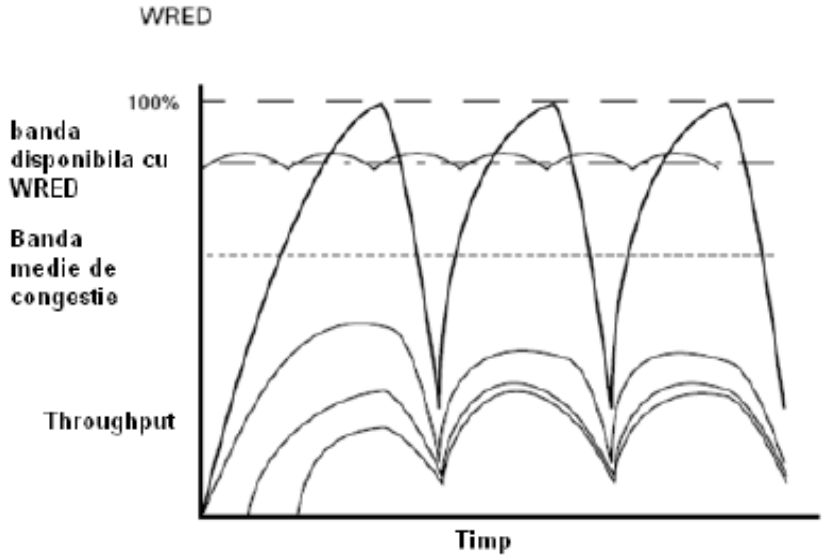


Figura 3.6 Weighted Random Early Detection

Astfel, în medie, procentul din banda ocupată va fi mult sub 100%. Cu WRED acest procent va crește semnificativ. WRED nu permite scăderea vitezei tuturor sesiunilor TCP în același timp. WRED stabilește niște praguri pentru fiecare coadă creată de sistem. Înainte de a pune un pachet în coadă se uită la aceste praguri. Dacă numărul de pachete din coadă este mai mic decât pragul inferior, pachetul este pus în coada; dacă este peste pragul superior, pachetul este aruncat; daca este între cele 2 praguri este folosită o funcție pentru a se stabili dacă pachetul va fi pus în coadă sau aruncat.

WRED se comportă aproape într-un mod identic ca RED. Calculează dimensiunea medie a cozii și decide dacă se vor arunca pachete, și dacă da, ce procent din pachete se vor arunca, decizie luată pe baza acelorași variabile descrise anterior în cazul RED.

Diferența între RED și WRED este aceea că WRED construiește un profil WRED pentru fiecare valoare de clasificare DSCP. Un profil WRED este în fapt un set de praguri minime și maxime și un procent asociat fiecărei clase de trafic. Valorile pragurilor sunt definite ca număr de pachete în coadă. Folosind aceste profile,WRED poate trata diferit anumite tipuri de pachete atunci când are loc o congestie. Un alt concept de importanță majoră ce trebuie tratat, înainte de a discuta configurația WRED, este referitor la locul unde poate fi folosit WRED și cum interacționează cu uneltele de management al cozilor.

WRED se bazează în decizia asupra când și câte pachete să fie descărcate pe următoarele criterii:

- Dimenisunea medie a cozii
- Pragul minim
- Pragul maxim

În primul rând, precum RED, WRED calculează dimensiunea medie a cozii. Apoi compară această dimensiune cu pragul minim și maxim de marcare. Dacă valoarea medie este între valorile

limită ale celor 2 praguri, WRED descarcă un procent din pachete, dacă valoarea medie depășește pragul maxim, WRED descarcă toate pachetele noi sosite.

RED face ca diferențierea QoS să fie imposibilă. WRED însă, oferă o detecție cu considerente față de QoS.

3.4.2 FRED (Fair RED)

Fair RED folosește algoritmul RED la care introduce parmetrii $\mathbf{min_q}$ și $\mathbf{max_q}$ – numărul minim și maxim de pachete ce pot fi buffer-ate de fiecare link. De asemenea FRED introduce variabila globală $\mathbf{avg_{cq}}$, un estimator al mediei pachetelor din buffer per canal (link). Link-urile ce trimit mai puține pachete decât $\mathbf{avg_{cq}}$ au prioritate. Numărul de pachete curente din buffer al fiecărei conexiuni cu ruterul este reținut în variabila **qlen**. De asemenea FRED poate reține numărul de încercări eșuate a fiecarui link de a răspunde la notificarea congestiei - acesta e reținut în variabila **strike.** Astfel FRED poate penaliza link-urile cu valori ale variabilei strike mari [5].

3.4.3 ARED (Adaptive RED)

Adaptive RED își configurează parametrii bazându-se pe încărcarea din trafic. Astfel, dacă coada medie se situează între cele 2 praguri atunci $\mathbf{max_p}$ este crescut aditiv sau scăzut multiplicativ în funcție de încarcarea pe trafic.

Ideea de bază este de a prezice dacă RED trebuie să devină mai mult sau mai puțin agresiv prin examinarea lungimii medii a cozii de pachete. Dacă această lungime depășește în mod continuu pragul minim, atunci algoritmul este prea agresiv. Dacă însă lungimea medie depaşeşte constant pragul maxim atunci algoritmul nu este suficient de agresiv. Pe baza valorilor medii ale lungimii cozii algoritmul va ajusta corespunzător valoarea lui $\mathbf{max_p}$. Pentru cazul prezentat în continuare $\boldsymbol{max_p}$ este pur și simplu scalat cu unul din factorii $\boldsymbol{\alpha}$ sau $\boldsymbol{\beta}$ în funcție de pragul depăşit.

Every $Q(ave)$ Update:

If ($\min_{th} < Q(ave) < \max_{th}$)

status=Between;

If ($Q(ave) < \min_{th}$ && status!=Below)

status=Below;

$\max_p = \max_p / \alpha$;

If ($Q(ave) > \max_{th}$ && status!=Above

status=Above;

$\max_p = \max_p \times \beta$

Principalul avantaj al acestui algoritm este faptul că îşi modifică singur parametrii în funcție de încărcarea pe rețea. Dezavantajul lui este că încă nu s-au stabilit clar parametrii optimi cu care să lucreze astfel încât poate avea rezultate contradictorii. [2]

3.5 Caracteristici ale algoritmului RED

Evitarea congestiei

Gateway-ul RED garantează faptul că dimensiunea medie calculată a cozii nu depășește pragul maxim prin aruncarea pacheteleor ce ajung la gateway atunci când dimensiunea medie a cozii atinge acest prag. Dacă ponderea w_q pentru procedura mediei mobile ponderate exponențial a fost stabilită corespunzător, atunci ruterul RED va fi cel care va controla de fapt dimensiunea medie a cozii. Atunci când ruterul RED setează un bit în antetul pachetelor, în momentul în care dimensiunea medie a cozii depășește pragul maxim, înseamnă că se bazează pe colaborarea dintre surse pentru a controla dimensiunea medie a cozii.

Scală de timp adecvată

După ce este notificată conexiunea asupra apariției congestiei, prin marcarea unui pachet, intervalul de timp necesar gateway-ului pentru a sesiza o descreștere a ratei de primire a pachetelor, este de cel puțin o perioadă dus-întors. În cazul gateway-urilor RED, scala de timp pentru detecția congestiei se potrivește aproximativ cu scala de timp necesară conexiunilor să răspundă la congestie. Gateway-urile RED nu notifică conexiunile să-și reducă ferastra ca rezultat al congestiei tranzitorii de la gateway.

Lipsa sincronizării globale

Rata cu care gateway-ul RED marchează pachetele, depinde de nivelul congestiei. Pe durata unei congestii scăzute, gateway-ul are o probabilitate mică de a marca fiecare pachet ce sosește. Pe măsură ce congestia crește, probabilitatea de a marca fiecare pachet crește și ea. Gateway-urile RED, evită sincronizarea globală marcând pachetele cu o rată cât mai mică posibil.

Simplitate

Algoritmul RED poate fi implementat fără costuri ridicate în rețelele curente.

Putere globală cât mai mare

Gateway-ul RED controlează în mod explicit dimensiunea medie a cozii. În figura de mai jos se poate observa că pentru legături cu utilizare foare mare, puterea globală este mai mare în cazul folosirii algoritmului RED, decât în situația folosirii algoritmului Drop Tail. Determinarea

dimensiunii medii optime a cozii reprezintă însă un obiectiv ce trebuie studiat în continuare. Aceasta depinde de rețele și condițiile de trafic.

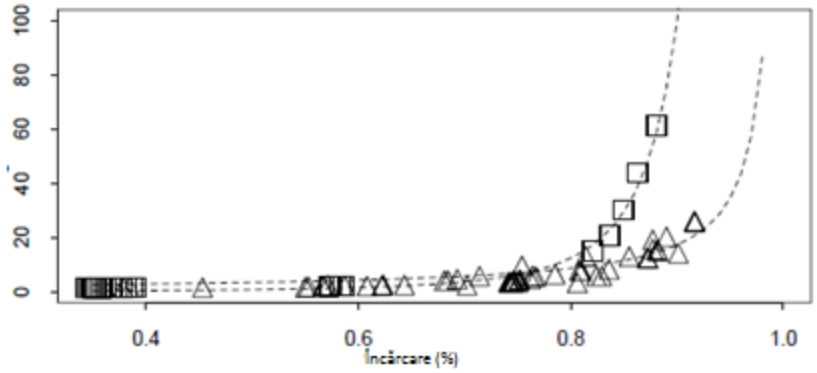


Figura 3.7 Compararea throughput-ului în cazul gateway-urilor RED și Drop Tail (triunghi pentru RED, pătrat pt Drop Tail)

Corectitudine

Acest obiectiv este unul foarte important pentru mecanismele de evitare a congestiei. Obiectivul corectitudinii nu este bine definit, motiv pentru care, acest aspect va fi tratat cu privire la performanțele algoritmului RED. Gateway-ul RED nu discriminează anumite conexiuni sau clase de conexiuni particulare. Pentru acesta, fracțiunea de pachete marcate pentru fiecare conexiune este aproximativ proporțională cu cota pe care acea conexiune o are la lățimea de bandă totală. Cu toate acestea, gateway-urile RED nu încearcă să se asigure că fiecare conexiune primește aceeași fracțiune din throughput-ul total și nu controlează explicit utilizatorii ce acționează necorespunzător. Gateway-urile RED furnizează un mecanism de identificare a nivelului congestiei. De asemenea, aceste gateway-uri ar putea fi folosite pentru identificarea conexiunilor cu o cotă mare la lățimea de bandă totală. Controlul throughput-ului acestor conexiuni ar putea fi realizat prin adăugarea unor mecanisme noi.

Posibilitatea implementării în numeroase situații

Mecanismul aleator de marcare de pachete este adecvat pentru rețele ale căror conexiuni sunt descrise de o gamă largă de timpi de rountrip şi de troughtput; este adecvat de asemenea şi pentru o gamă largă a numărului de conexiuni active simultan. Schimbările de încărcătură a rețelei sunt detectate prin modificările dimensiunii medii a cozii, iar rata de marcare a pachetelor este ajustată corespunzător.

Chiar şi în cadrul unei rețele în care ruterul RED semnalează congestia prin aruncarea de pachete marcate, sunt multe situații în care aruncarea unui pachet nu produce o descreştere a încărcăturii la router. Dacă ruterul aruncă un pachet aparținând unei conexiuni TCP, aruncarea pachetului respectiv va fi detectată de sursă, cel mai probabil după expirarea timpului de retransmitere.

Aruncarea unui pachet ar putea trece neobservată de sursă în cazul în care ruterul aruncă un pachet ACK corespunzând unei conexiuni TCP sau non-TCP. În ciuda tuturor acestor lucruri, şi chiar în cazul unei rețele congestionate caracterizată de un trafic alcătuit dintr-o combinație de conexiuni scurte TCP sau non-TCP, routerul are controlul asupra dimensiunii medii a cozii prin faptul că aruncă toate pachetele primite în momentul în care este depăşit pragul de maximum.

Senzitivitatea parametrilor

Spre deosebire de ruterele Drop Tail în cazul cărora singurul parametru variabil este dimensiunea bufferului, ruterele RED au parametri suplimentari care determină pragul superior pentru dimensiunea medie a cozii, intervalul de timp pe care se calculează această dimensiune, şi frecvența maximă cu care se marchează pachetele. Mecanismul de evitare al congestiei ar trebui să aibă o sensibilitate scăzută a parametrilor, iar aceşti parametrii ar trebui să poată fi aplicați rețelelor cu o gamă foarte largă de variație a lărgimii de bandă.

În cazul ruterelor RED parametrii w_q, min_{th} şi max_{th} sunt necesari pentru că persoana care proiectează rețeaua să poată lua decizii conştiente în legătură cu dimensiunea medie dorită a cozii şi în legătură cu dimensiunea şi durata rafalelor ce vor fi acceptate de buffer.

Parametrul max_p poate fi ales dintr-o gamă destul de largă de valori, deoarece reprezintă doar limita superioară pentru probabilitatea p_b de marcare a pachetului. Dacă nivelul congestiei este atât de mare încât ruterul să nu poată controla dimensiunea medie a cozii prin marcarea a cel mult unei fracțiuni de max_p dintre pachete atunci dimensiunea medie a cozii va depăşi pragul de maximum, iar ruterul va marca fiecare pachet până când congestia va putea fi controlată.

În continuare sunt date câteva reguli de urmat pentru ca algoritmul RED să ofere o performanță adecvată în cazul unei game variate de condiții de trafic şi de parametrii ai ruterelor.

- **Asigurarea unui calcul adecvat pentru dimensiunea medie a cozii**: stabilim $w_q \geq 0{,}001$. Dimensiunea medie a cozii ruterului este limitată de max_{th} atât timp cât dimensiunea medie calculată a cozii ***avg*** reflectă dimensiunea reală. Valoarea ponderii w_q nu ar trebui stabilită prea mică pentru ca lungimea medie a cozii să nu întârzie prea mult în evidențierea dimensiunilor reale ale cozii.

$$L + 1 + \frac{1 - w_q{}^{L+1} - 1}{w_q} < min_{th}$$

- **Valoarea atribuită lui min_{th} trebuie să fie suficient de mare astfel încât să maximizeze puterea rețelei.** Valorile pragurilor min_{th} şi max_{th} ar trebui să fie suficient de mari pentru a maximiza puterea rețelei. Din cauza faptului că traficul în rețea este de cele mai multe ori în rafale, dimensiunea reală a cozii poate fi de asemenea destul de variabilă; dacă lungimea medie a cozii este stabilită la un nivel prea mic atunci legătura de ieşireva fi utilizată ineficient.

- **Diferența max_{th}-min_{th} trebuie să fie suficient de mare astfel încât să evite sincronizarea globală.** Pentru a împiedica sincronizarea globală diferența max_{th}-min_{th} trebuie să fie mai mare decât variația tipică a dimensiunii medii a cozii pe parcursul duratei dus-întors. Dacă această diferență este prea mică atunci dimensiunea medie calculată a cozii poate să oscileze cu regularitate până la ***maxth***, având practic aceeaşi comportare cu Drop Tail.

3.6 Identificarea utilizatorilor cu comportament necorespunzător

Ruterele RED oferă un mecanism eficient pentru identificarea conexiunilor care utilizează un procent ridicat din lățimea de bandă atunci când se produce congestia. Deoarece ruterele RED aleg în mod aleator pachetele care să fie marcate în momentul producerii congestiei, ruterele pot identifica cu uşurință care dintre conexiuni a primit o parte semnificativă dintre pachetele marcate recent. Atunci când numărul de pachete marcate este suficient de mare, conexiunea care a primit un procent ridicat din pachetele marcate este foarte probabil să fie şi conexiunea care a utilizată cea mai mare parte a lărgimii de bandă. Această informație ar putea fi utilizată de către nivelele superioare în scopul de a restricționa lățimea de bandă a acelor conexiuni în timpul congestiei.

Ruterele RED notifică conexiunile apariția congestiei prin marcarea pachetelor. Probabilitatea de a marca un pachet dintr-o anumită conexiune este direct proporțională cu procentul din lățimea de bandă pe care-l utilizează conexiunea respectivă în acel moment, spre deosebire de Tail Drop.

Ruterele RED ar putea păstra cu uşurință o listă cu cele mai recente *n* pachete marcate. Dacă o mare parte din pachetele marcate aparțin unei anumite conexiuni, este probabil că aceea conexiune să fi avut şi un procentaj mare din lățimea de bandă medie. Dacă unele din conexiunile TCP primesc procente considerabile din lățimea de bandă, atunci este foarte posibil că acea conexiune să fie o sursă care nu ține cont de protocoalele TCP curente, sau este o conexiune cu o perioadă dus-întors mai scurtă sau cu o fereastră mai largă decât cea a altor conexiuni. În oricare dintre aceste cazuri, ruterul RED poate fi setat astfel încât să acorde o prioritate mai scăzută acelor conexiuni ce primesc un procent mai mare din lățimea de bandă în timpul producerii congestiei.

CAPITOLUL 4

4. Modificarea algoritmului RED

4.1 Introducere

Aşa cum s-a putut observa în secțiunea anterioară, probabilitatea cu care sunt marcate pachetele, este dependentă de valoarea dimensiunii medii a cozii.

$$p_b = max_p \frac{avg - min_{th}}{max_{th} - min_{th}}$$

$$p_a = \frac{p_b}{1 - count * p_b}$$

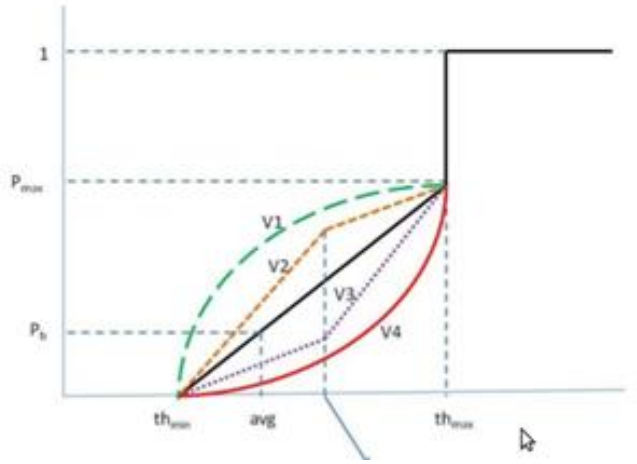
RED calculeaza probabilitatea de aruncare a pachetelor prin 2 paşi atunci când lungimea cozii medii scade intre th_min şi th_max. Primul pas este să calculăm o probabilitate numită Pb, care este bazată pe formula următoare:

Pb = Pmax(avg – th_min)/(th_max-th_min)*

Se observă că probabilitatea Pb se calculează pe baza unei funcții liniare.
Al doilea pas presupune numărarea numărului de pachete care trec prin gateway (count) până când ultimul pachet este aruncat, aplicând următoarea formulă:

*Pa = Pb / (1 – count*Pb)*

Urmărind articolul [3] am modificat primul pas aplicand 4 funcții diferite pentru calcularea Pb şi compararea rezultatelor. Graficele celor 4 funcții sunt reprezentate mai jos:



Rulând simulari pentru aceste functii am obținut urmatoarele grafice, unde culoarea roşie repreyinta valoarea curentă a cozii iar culoarea verde reprezintă dimensiunea cozii medii:

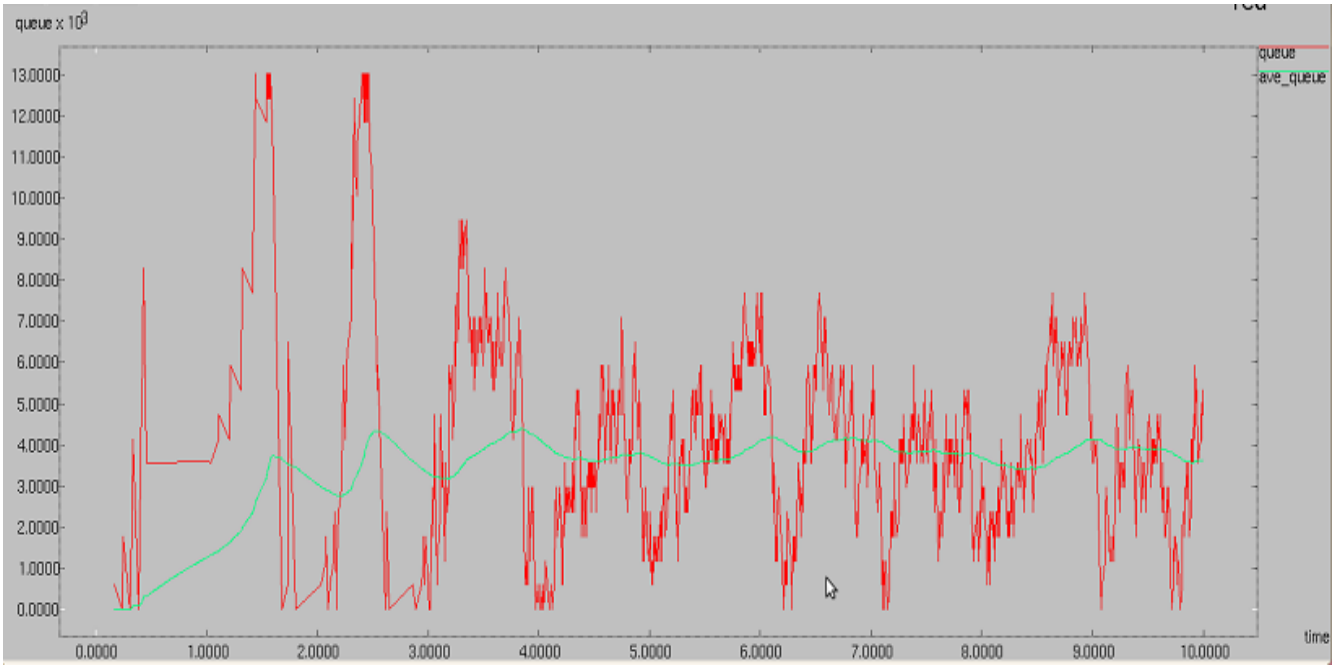


Figura 4.1 Grafic RED nemodificat

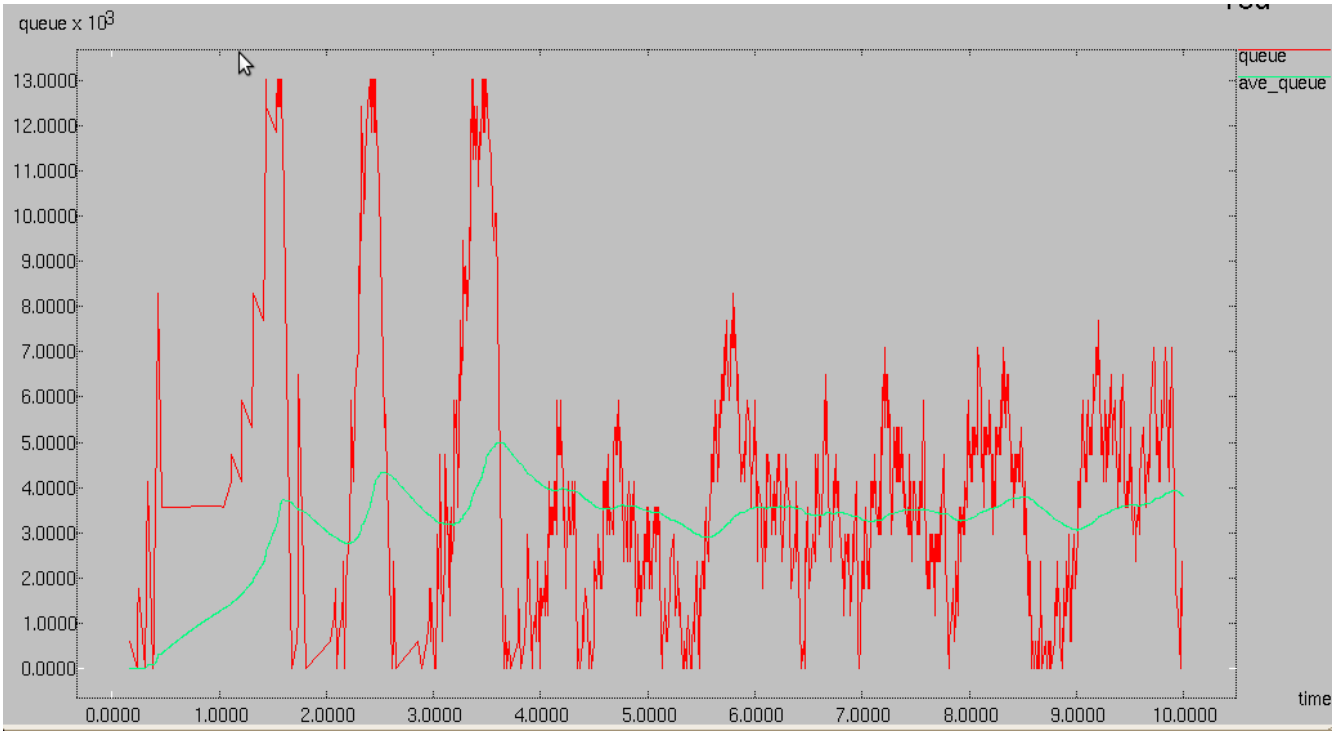


Figura 4.2 Grafic folosind funcția v1

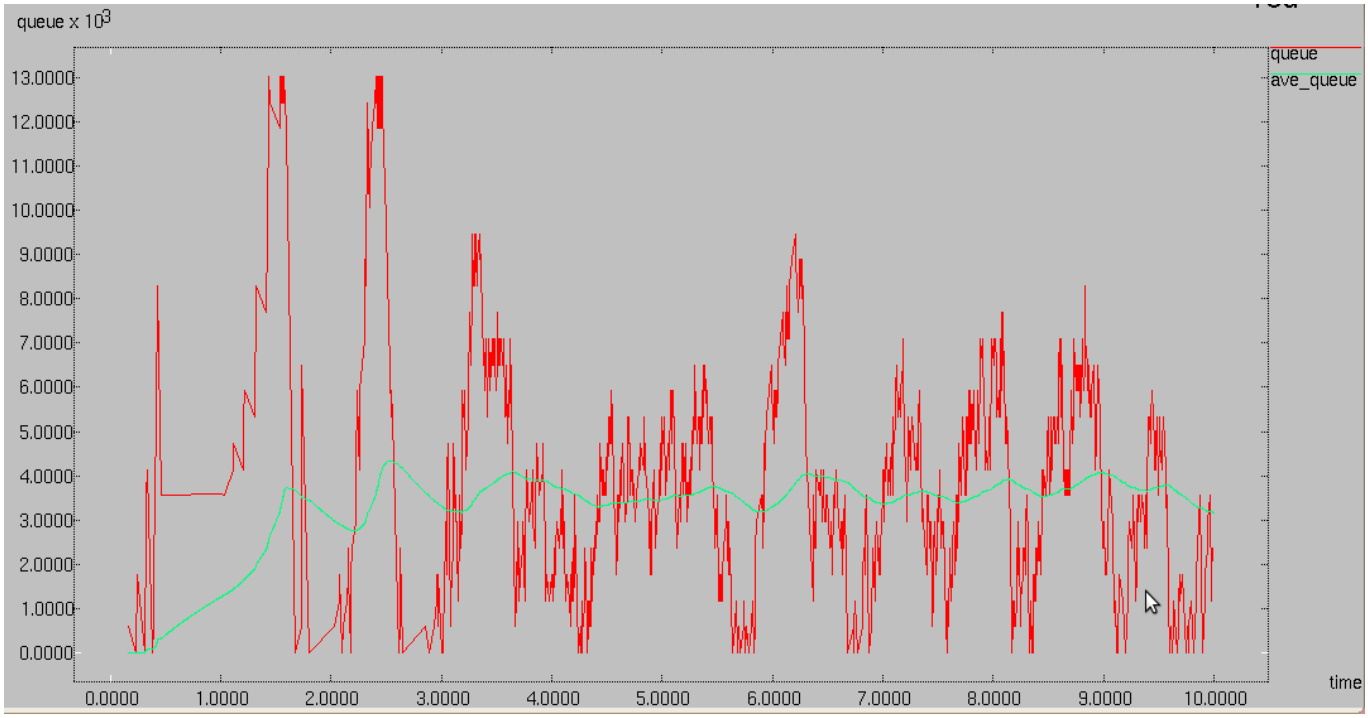


Figura 4.3 Grafic folosind funcția v2

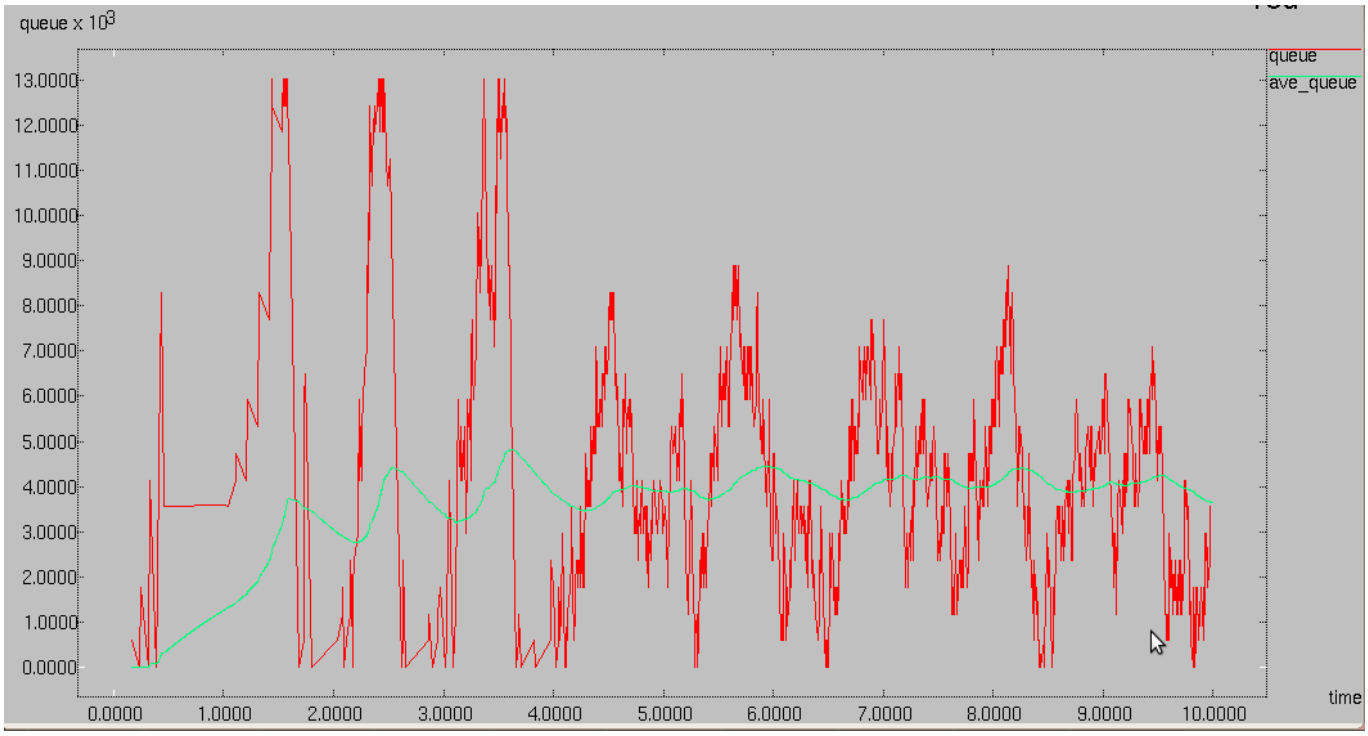


Figura 4.4 Grafic folosind functia v3

Se observă că folosind funcțiile v1 si v2 valoarea cozii medii este ținută la un nivel mai scăzut decât in cazul folosirii algoritmului RED nemodificat, în timp ce folosind funția v3 se ajunge la un nivel mai ridicat. În concluzie v1 şi v2 sunt mult mai agresive în aruncarea pachetelor atunci când valoarea cozii medii este in intervalul [th_min; th_max].

4.2 Soluția propusă pentru modificarea algoritmului RED

Având in vedere soluțiile studiate anterior am propus o modificare a funcției cu care se calculează probabilitatea de aruncare a pachetelor Pb în cazul în care valoarea cozii medii se află în intervalul stabilit de noi, [1.3*th_min; 0.7*th_max].

*Pb = Pmax * func(avg);*

Noua funcție intersectează un set de puncte ale căror coordonate se pot modifica pentru a se obține cele mai bune performanțe. Următorul grafic arată punctele prin care poate trece dreapta care descrie funcția folosită:

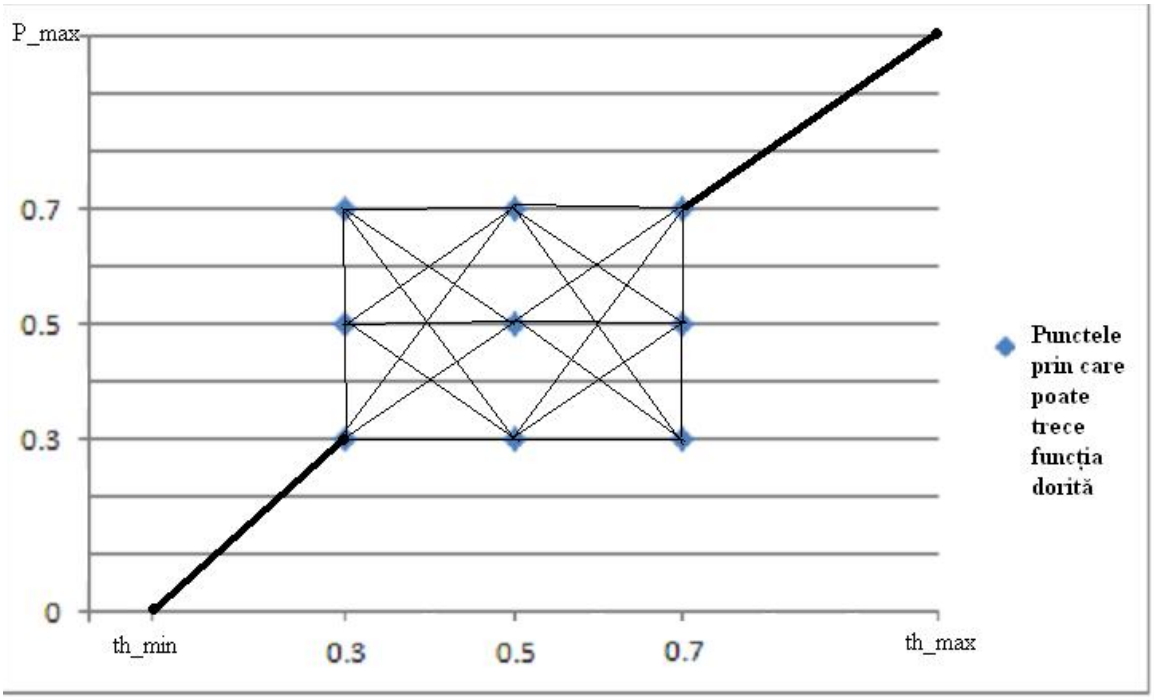


Figura 4.5

Traseul optim, pentru care se obțin performanțele cele mai bune, se caută făcând toate cele 27 de încercări posibile. În *Figura 4.5* sunt ilustrate toate traseele pe care le poate urma funcția folosită la calcularea probabilității de aruncare a pachetelor. Traseul funcției se stabilește în funcție de coordonatele date punctelor pe care le intersectează.

CAPITOLUL 5

5. Simulări

5.1 Network Simulator

Scurt istoric

NS2 a apărut în 1989, ca o variantă a simulatorului de rețele REAL (simulator conceput inițial în scopul analizei comportamentului dinamic al fluxurilor și al schemelor de control ale congestiei). Network Simulator 2 este disponibil pentru diferite platforme cum ar fi: FreeBSD, Linunx, SunOS, Solaris. El poate rula și pe windows prin intermediul unui mediu Linux-like, Cygwin.

NS este folosit foarte mult în cercetare, oferind un suport substanțial în simularea comunicării TCP, routing-ului și a protocoalelor multicast, atât în rețele cablate cât și wireless (locale sau prin sateliți).

OTcl

Network Simulator este un simulator orientat pe obiecte, scris în C++, cu o interfață făcută de un interpretor OTcl. Simulatorul suportă o ierarhie a claselor în C++, și o ierarhie a claselor asemănătoare pentru interpretorul OTcl. Cele două ierarhii sunt strâns legate una de cealaltă. Din perspectiva utilizatorului, există o corespondență unu la unu, între o clasă din cadrul ierarhiei interpretorului și una din cadrul ierarhiei compilate. Rădăcina acestei ierarhii este clasa TclObject. Utilizatorii creează simulări noi prin intermediul interpretorului. Aceste obiecte sunt instanțiate în cadrul interpretorului și sunt oglindite de un obiect corespondent în ierarhia compilată. Ierarhia claselor interpretorului este realizată automat prin metode definite în clasa TclClass. Obiectele instanțiate de utilizator sunt oglindite prin metode definite in clasa TclObject. Există și alte ierarhii în codul C++ și scripturile OTcl. Aceste alte ierarhii nu sunt oglindite în maniera TclObject.

De ce două limbaje?

Network Simulator folosește două limbaje deoarece simulatorul are două tipuri de obiective pe care trebuie să le realizeze. Pe de o parte, simulări detaliate ale protocoalelor necesită un limbaj de programare al sistemului, care poate manipula eficient bytes, pachete, anteturi și care poate implementa algoritmi ce rulează cu seturi foarte mari de date. Pentru aceste obiective, viteza la rulare este importantă, în timp ce rularea simulării, găsirea bug-urilor și repararea acestora este mai puțin importantă. Pe de altă parte, o mare parte a cercetărilor în rețele implică variații mici ale parametrilor sau configurațiilor sau numărului de scenarii. În aceste cazuri, timpul necesar iterației (schimbarea

modelului și rularea acestui din nou) este mai importantă. Din moment ce configurația rulează o singură dată (la începutul simulării), timpul de rulare pentru această parte este mai puțin important.

NS împacă cele două cerințe cu cele două limbaje, C++ și OTcl. C++ este rapid la rulare, însă mai greoi la schimbare, făcându-l potrivit pentru implementările detaliate ale protcoalelor. OTcl rulează mult mai greu, însă poate fi schimbat foarte rapid (și interactiv), făcându-l ideal pentru configurarea simulărilor.

Network Animator

Nam este o unealtă pentru animații, bazată pe Tcl/TK , folosită pentru vizualizarea datelor simulărilor de rețele. Teoria de implementare din spatele Nam a fost crearea unui animator care să poată citi seturi foarte mari de date pentru animații și care să poată fi îndeajuns de extensibil pentru a putea vizualiza o viariatate foarte mare de situații de rețele. Având această constrângere, **nam** a fost creat să poată citi comenzi simple de animație din interiorul unor fișiere foarte mari de urmărire. Pentru a putea lucra cu seturi de date foarte mari pentru animații, cantitatea de informație reținută în memorie este foarte mică. Comenzile sunt reținute în fișier și sunt recitite ori de câte ori este necesar.

Primul pas pentru a putea folosi **nam** este crearea unui fișier de urmărire. Acest fișier conține informații referitoare la topologii, noduri, legături dar și pachete. De obicei, acest fișier este creat de către Network Simulator.

După ce a fost generat fișierul pentru urmărire, acesta este pregătit pentru a putea fi animat de către nam. La pornire, nam citește fișierul, crează topologia, creează o fereastră pop-up, realizează layout-ul necesar și se oprește la momentul de timp 0. Network animator are o interfață pentru utilizator însă permite controlul asupra multor aspecte ale animației.

Instalarea simulatorului

Simulările au fost realizate cu ajutorul simulatorului Network Simulator v2.35 rulând pe un sistem de operare Ubuntu 10 instalat pe o maşină virtuală creată pe un Notebook HP 4520s, având următoarele specificații: processor Intel Core i3, memorie RAM 4GB, memorie externa 400GB.

Pentru instalarea simulatorului am urmat următorii paşi:

-descărcarea şi instalarea fişierului ns-2 all-in-one cu următoarele comenzi:

```
$ wget http://nchc.dl.sourceforge.net/sourceforge/nsnam/ns-allinone-
2.34.tar.gz
$ tar -xzvf ns-allinone-2.34.tar.gz
$ cd ns-allinone-2.34
```

```
$ sudo apt-get install build-essential autoconf automake libxmu-dev
$ ./install
```

-setarea variabilelor [6]

```
$ gedit ~/.bashrc
$ source ~/.bashrc
```

-validare

```
$ cd ns-2.34
$ ./validate
```

5.2 Modificarea simulatorului

Pentru modificarile prezentate in capitolul anterior, pentru calculul funcției necesare pentru determinarea probabilității de aruncare a pachetelor, Pb, am modificat fişierele sursă red.cc şi red.h ale simulatorului NS2.

În fişierul sursă a algoritmului RED, red.cc, am făcut următoarea modificare pentru calculul probabilității de aruncare a pachetelor Pb:

```
REDQueue::calculate_p_new(double v_ave, double th_max, int gentle, double v_a,
      double v_b, double v_c, double v_d, double max_p)
{
          double p;

    double l1, l2, l3, l4,l5,l6, d;

    if (gentle && v_ave >= th_max) {

                // p ranges from max_p to 1 as the average queue

                // size ranges from th_max to twice th_max

       p = v_c * v_ave + v_d;

       } else if (!gentle && v_ave >= th_max) {

                // OLD: p continues to range linearly above max_p as

               // the average queue size ranges above th_max.

               // NEW: p is set to 1.0

               p = 1.0;

       } else {
```

```
            d= edp_.th_max - edp_.th_min;

            l1=edp_.th_min+edp_.p1*d;
            l2=edp_.th_min+edp_.p2*d;
            l3=edp_.th_min+edp_.p3*d;
            l4=edp_.p4*max_p;
            l5=edp_.p5*max_p;
            l6=edp_.p6*max_p;


            if ((v_ave <= l1)&&(v_ave>= edp_.th_min)){

                p = v_a * v_ave + v_b;
}


            else if ((v_ave <= l2)&&(v_ave > l1)) {

            double slope, intercept, f;

            double dx, dy;

            dx = l2 - l1;

            dy = l5 - l4;

            slope = dy / dx;

            intercept = l4 - slope * l1;

            p=slope*v_ave + intercept;
}
            else if ((v_ave <= l3)&&(v_ave > l2)) {

            double slope, intercept, f;

            double dx, dy;

            dx = l3 - l2;

            dy = l6 - l5;

            slope = dy / dx;

            intercept = l5 - slope * l2;

            p=slope*v_ave + intercept;
}
            else if ((v_ave > l3)&&(v_ave<th_max)) {
```

```
            p = v_a * v_ave + v_b;
}

}

                // p = (v_ave - th_min) / (th_max - th_min)

                p *= max_p;


    if (p > 1.0)

        p = 1.0;

    return p;
}
```

5.3 Scenariul simulat

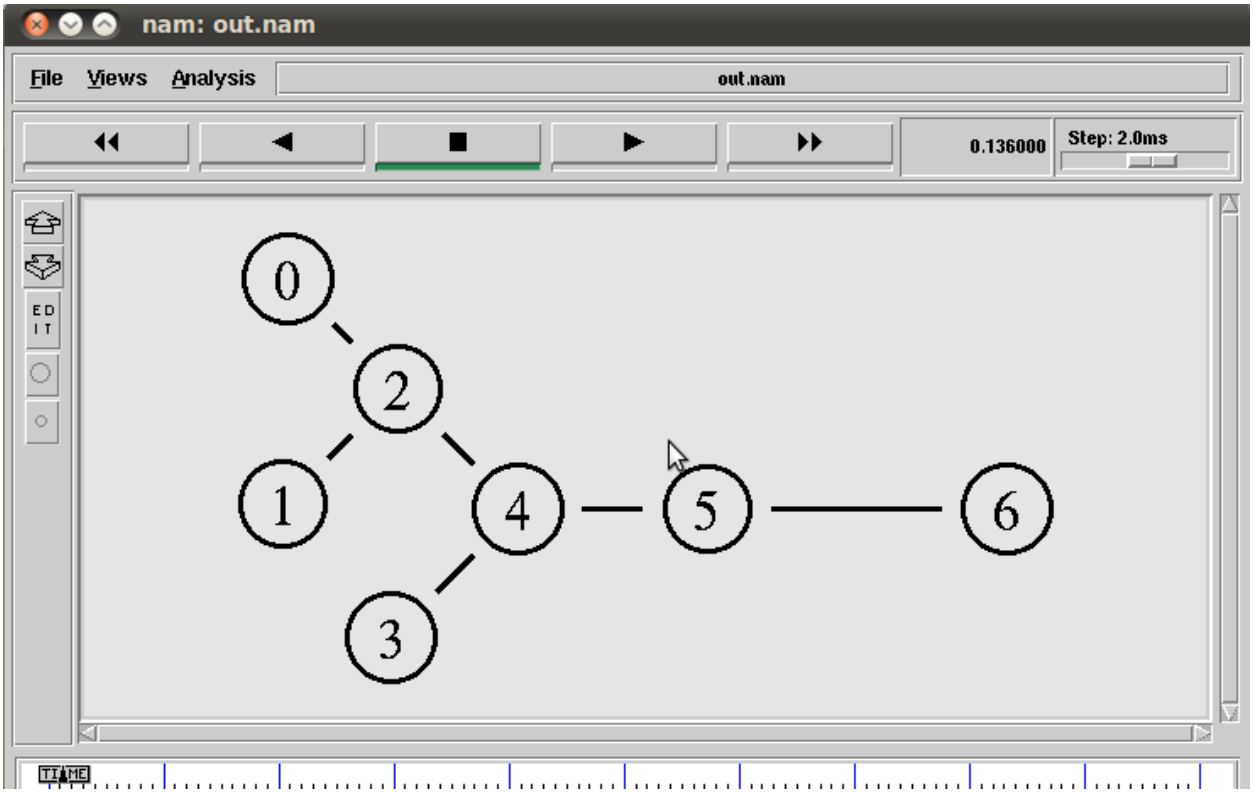


Figura 5.1 Topologia utilizată

Topologia utilizată este alcătuită din:

- 5 link-uri cu o lățime de bandă de 10 Mbps

- 1 link bottleneck cu o lățime de bandă de 1,5 Mbps şi un delay de 20 ms

Considerăm că avem sase conexiuni FTP ce utilizează TCP, şi setăm dimensiunea maximă a ferestrei la 30. Fiecare dintre cele patru noduri trimit trafic FTP pe parcursul întregii perioade de simulare (10 secunde). Dimensiunea cozii de bottleneck este 25.

Se alege dimensiunea de 552 de bytes pentru pachetul TCP (pachetul efectiv creat în timpul simulării are 592 bytes deoarece se adaugă 40 bytes ce corespund headerului).

Valorile utilizate pentru parametrii RED sunt cele standard : min_{th}=5, max_{th}=15, w_q=0,002, unde min_{th} reprezintă coada minimă, max_{th} reprezintă coada maximă, iar w_q reprezintă încărcarea cozii.

Pentru realizarea scenariului am creat un script folosind limbajul TCL. Scriptul este ataşat in Anexa1. Pentru rularea scriptului se apeleaza comanda `$ ns RED.tcl x1 x2 x3 y1 y2 y3`, unde x1, x2, x3, y1, y2, y3 reprezintă parametrii punctelor care descriu funcția folosită.

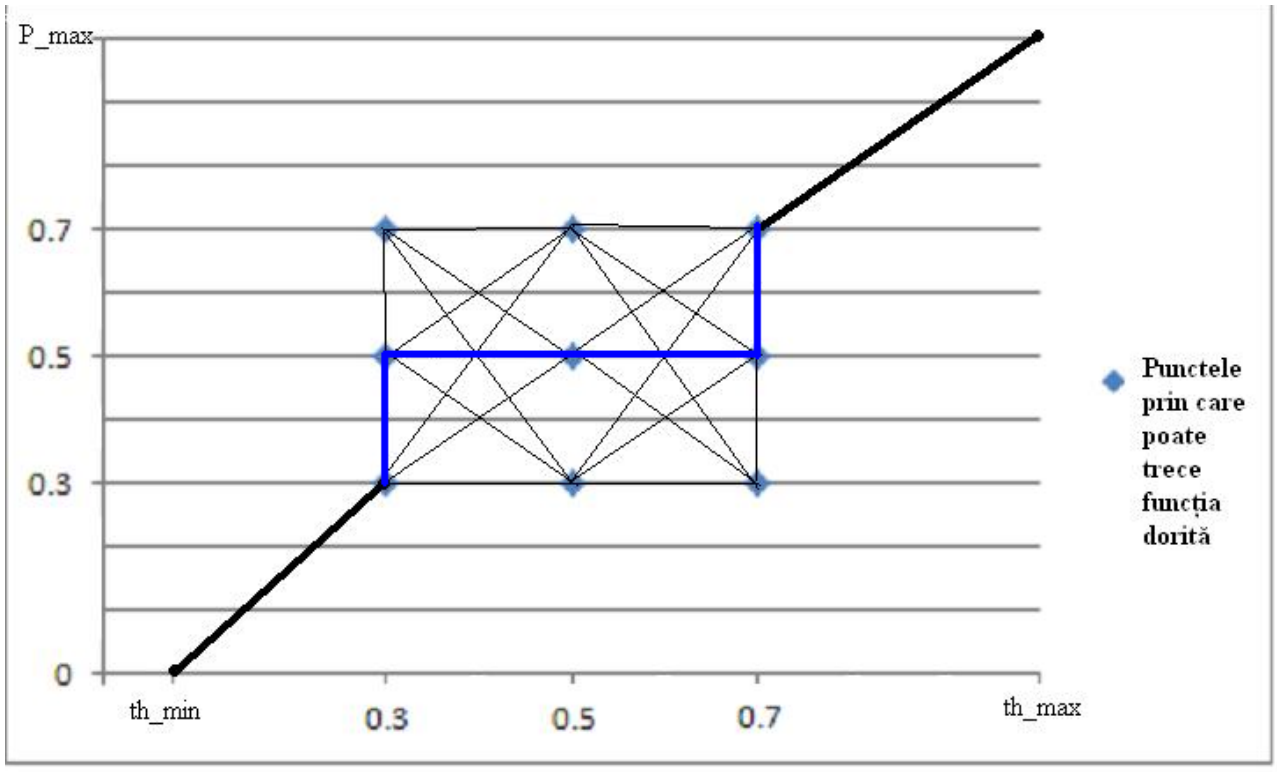


Figura 5.2 Exemplu: `$ ns RED.tcl 3 5 7 5 5 5`

CAPITOLUL 6

6 Prezentarea rezultatelor

Am rulat simulari pentru toate cele 27 de cazuri de alegere ale punctelor pentru construirea funcției, iar în continuare voi prezenta graficele care descriu variația cozii medii, numărul de pachete aruncate si frecvența de aruncare a pachetelor, pentru cele mai semnificative dintre aceste cazuri.

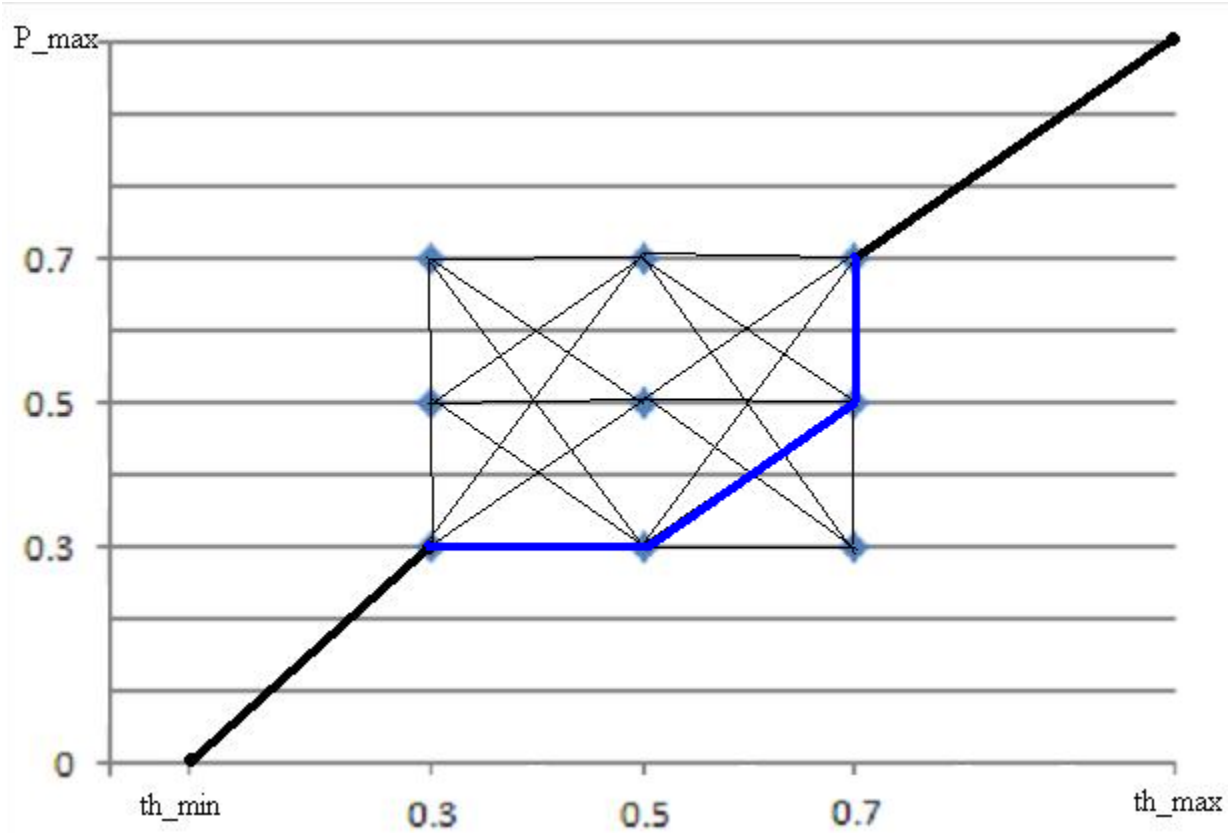


Figura 6.1 Grafic funcție în cazul configurației 357335

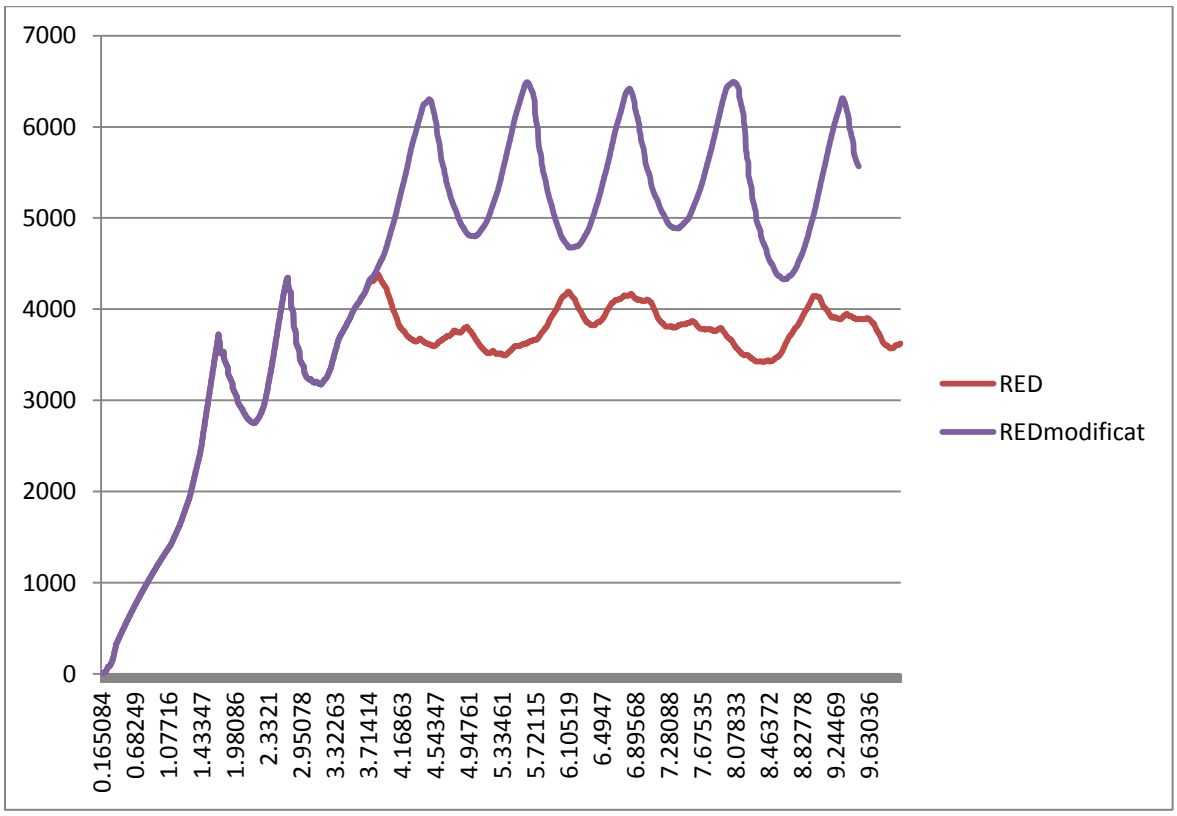


Figura 6.2 Variația cozii medii şi a cozii curente în timp pentru configurația 357335

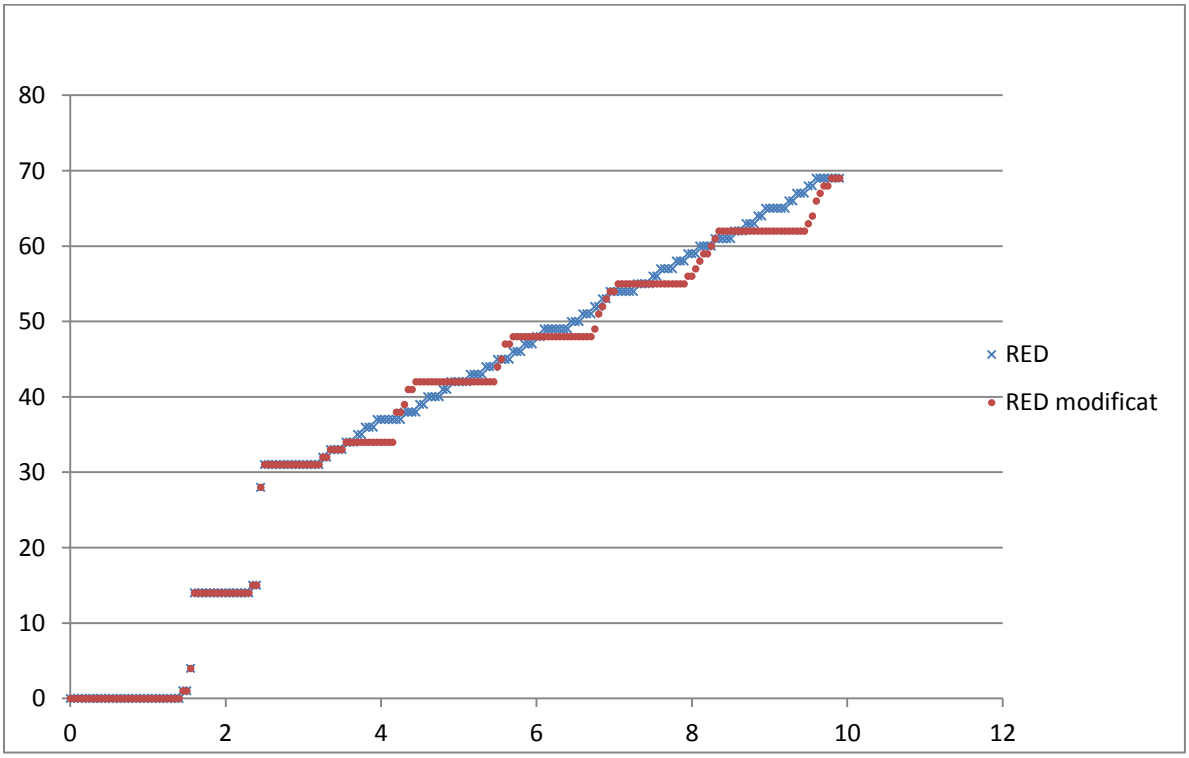


Figura 6.3 Variația numărului de pachete aruncate în timp pentru configurația 357335

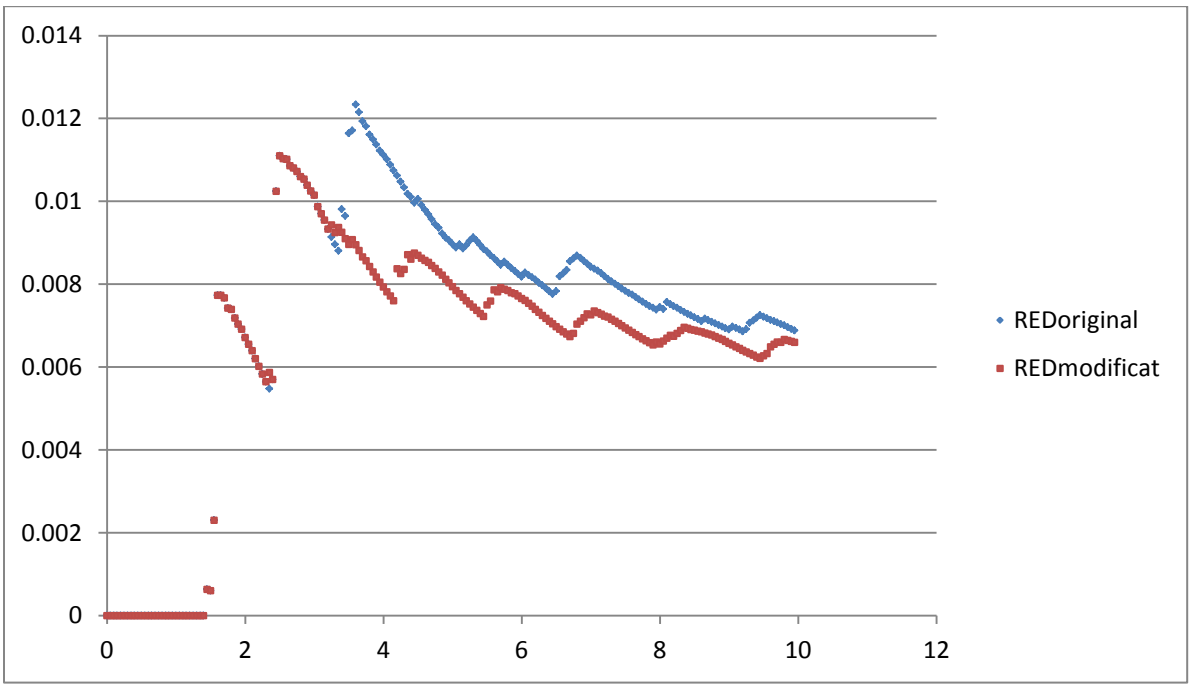


Figura 6.4 Frecvența de aruncare a pachetelor în timp pentru configurația 357335

Se observă o frecvență de aruncare a pachetelor mai mica decât in cazul folosirii algoritmului RED nemodificat.

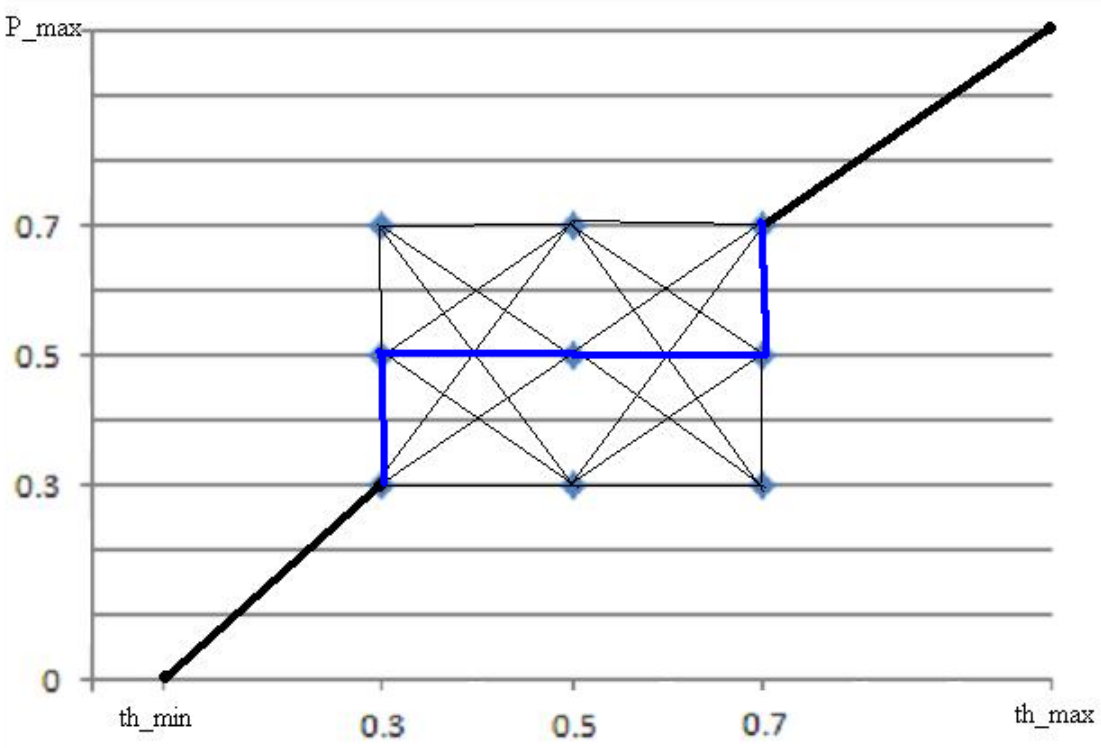


Figura 6.5 Grafic funcție în cazul configurației 357555

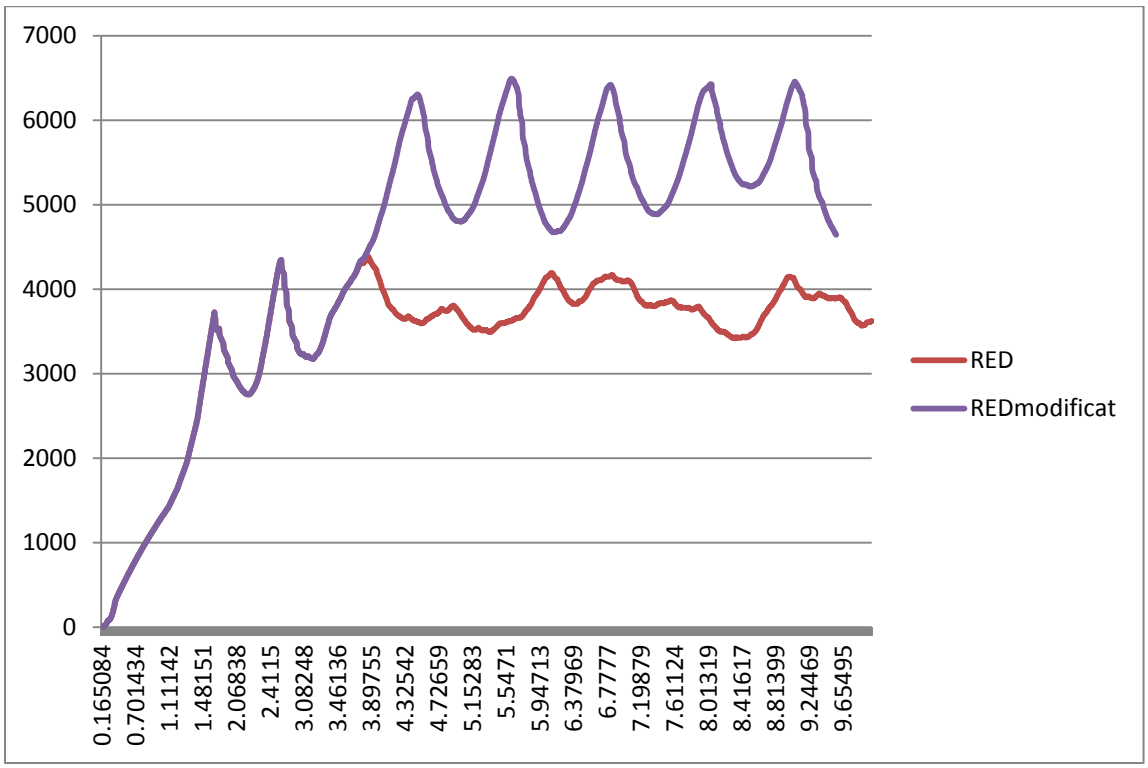


Figura 6.6 Variația cozii medii şi a cozii curente în timp pentru configurația 357555

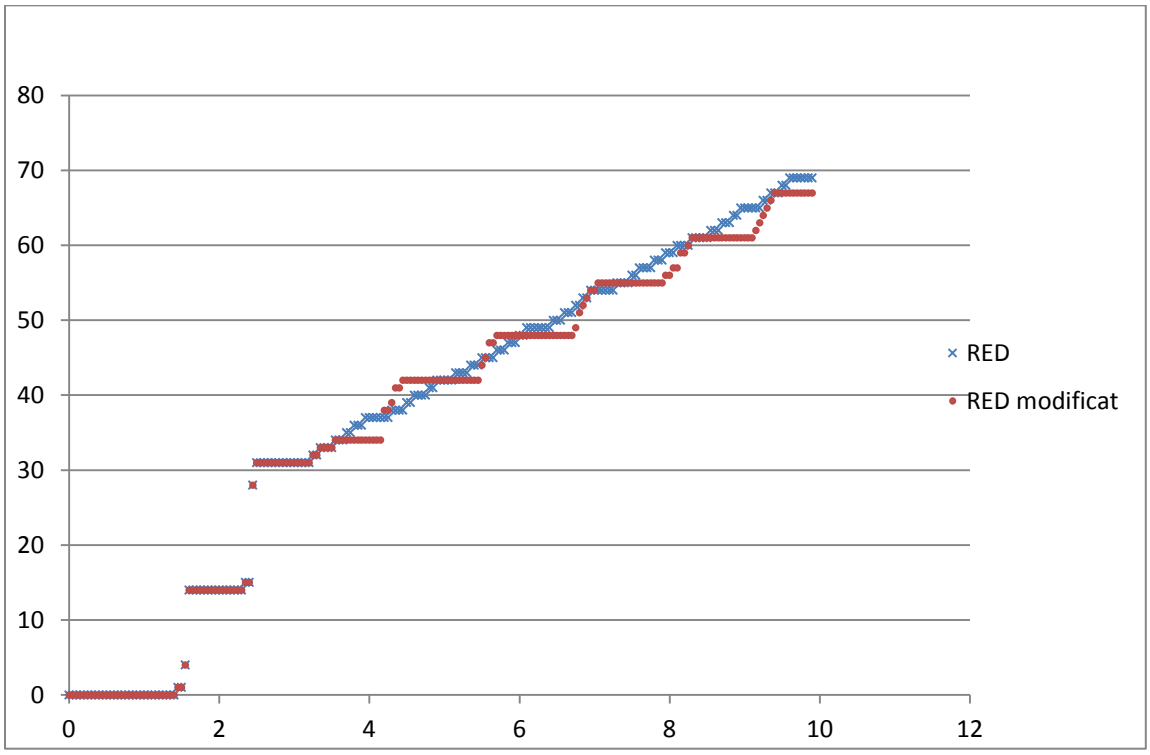


Figura 6.7 Variația numărului de pachete aruncate în timp pentru configurația 357555

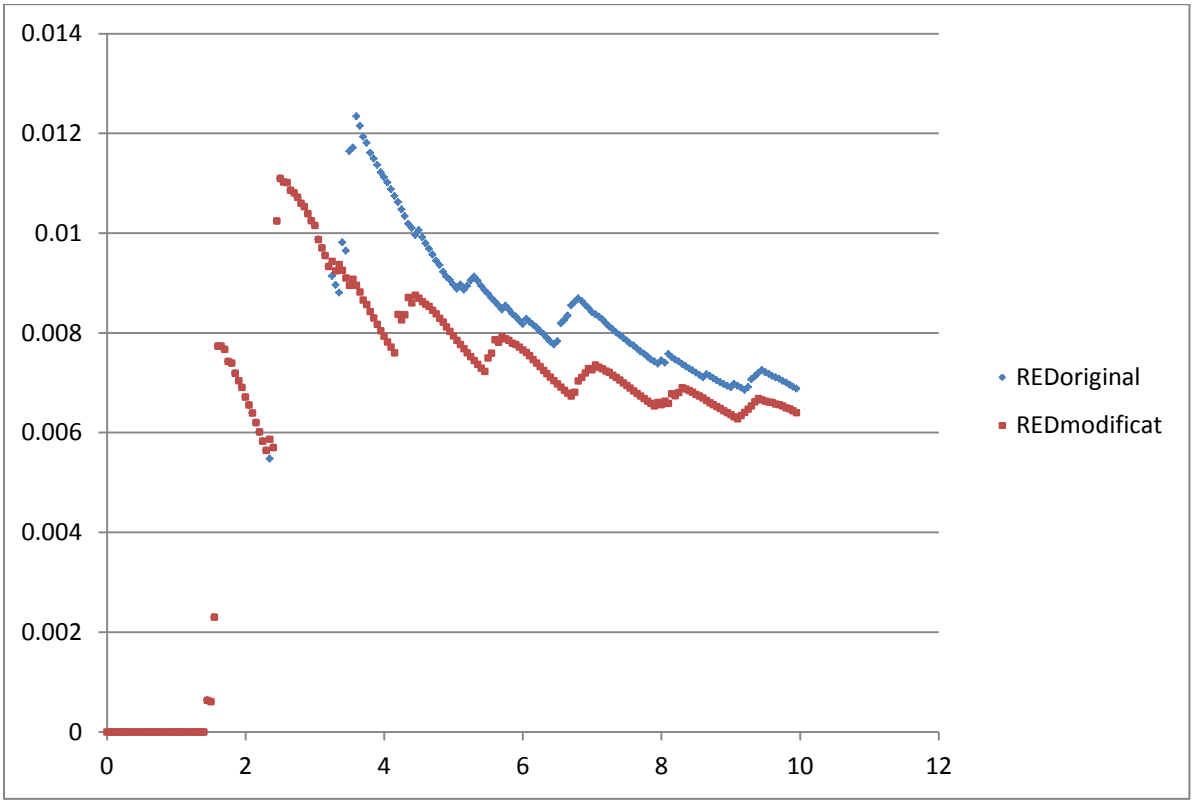


Figura 6.8 Frecvența de aruncare a pachetelor în timp pentru configurația 357555

Se observă că în această configurație am obținut cele mai bune rezultate pentru frecvența de aruncare a pachetelor.

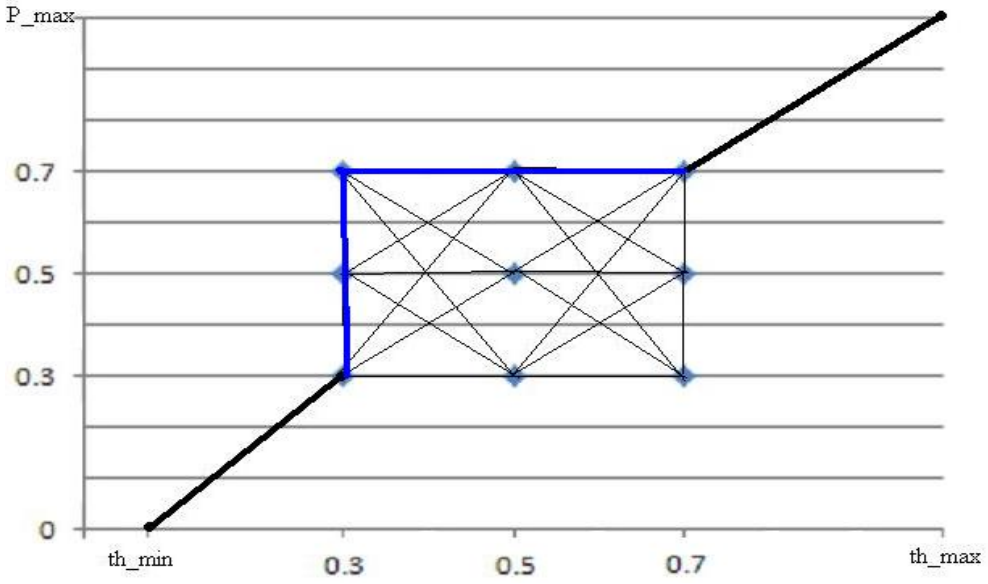


Figura 6.9 Grafic funcție în cazul configurației 357777

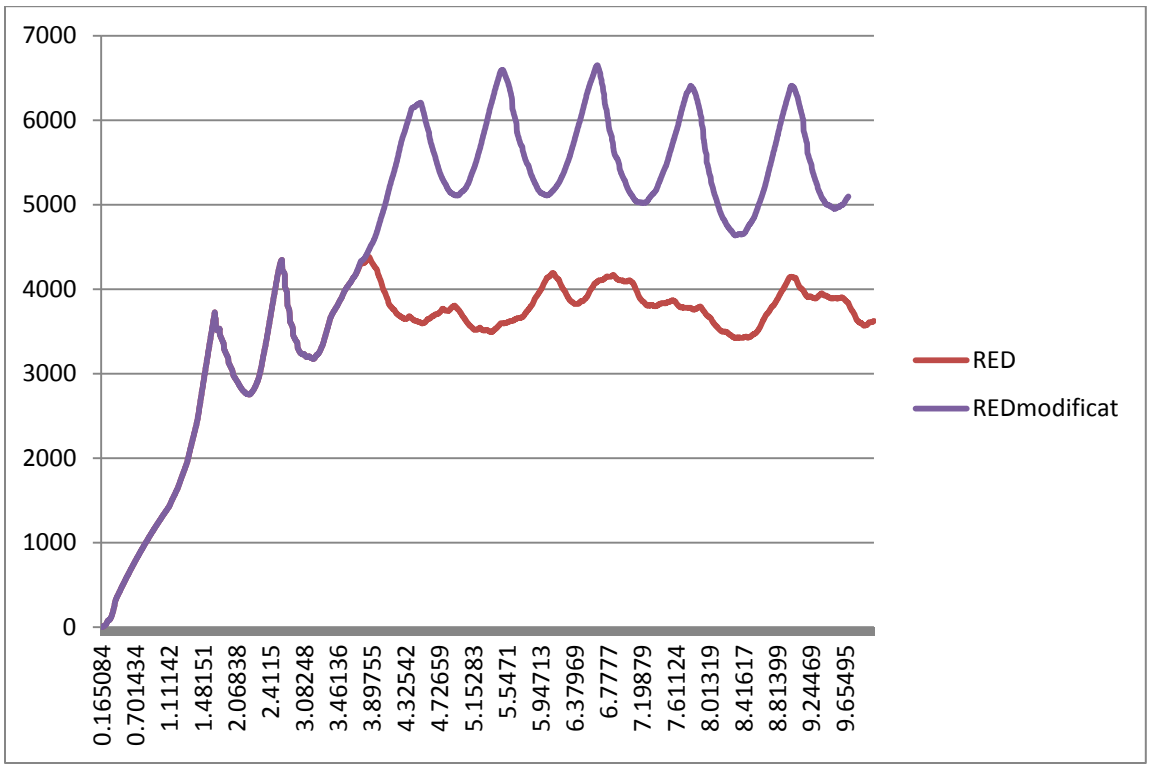


Figura 6.10 Variația cozii medii şi a cozii curente în timp pentru configurația 357777

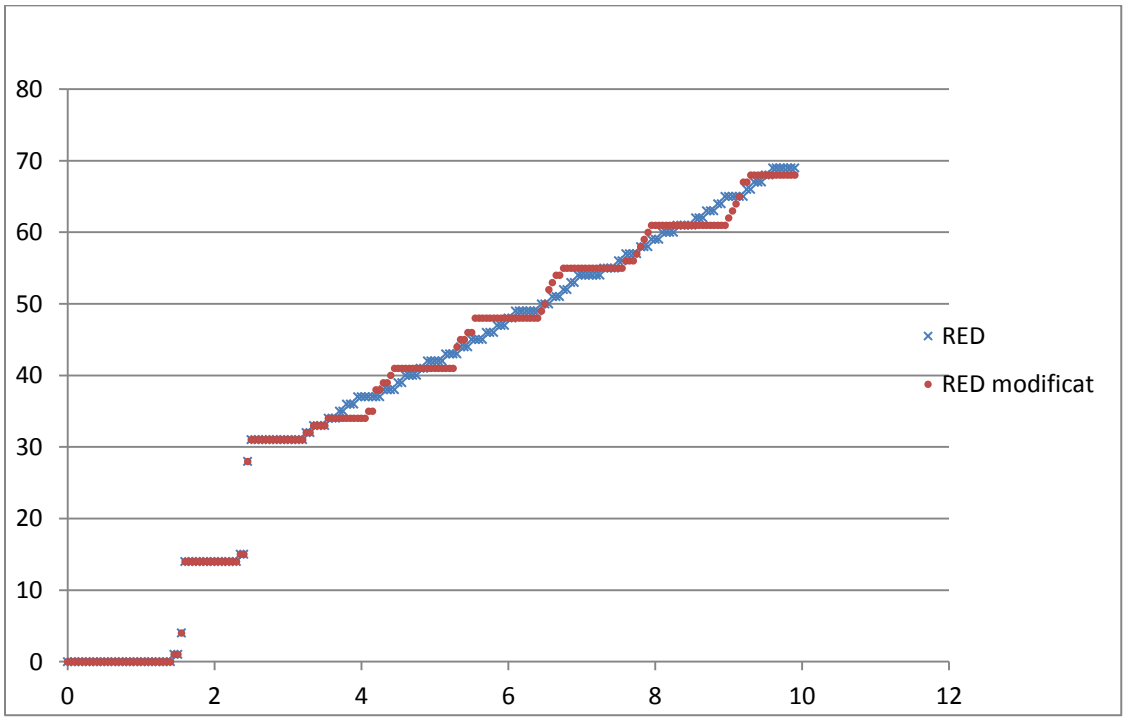


Figura 6.11 Variația numărului de pachete aruncate în timp pentru configurația 357777

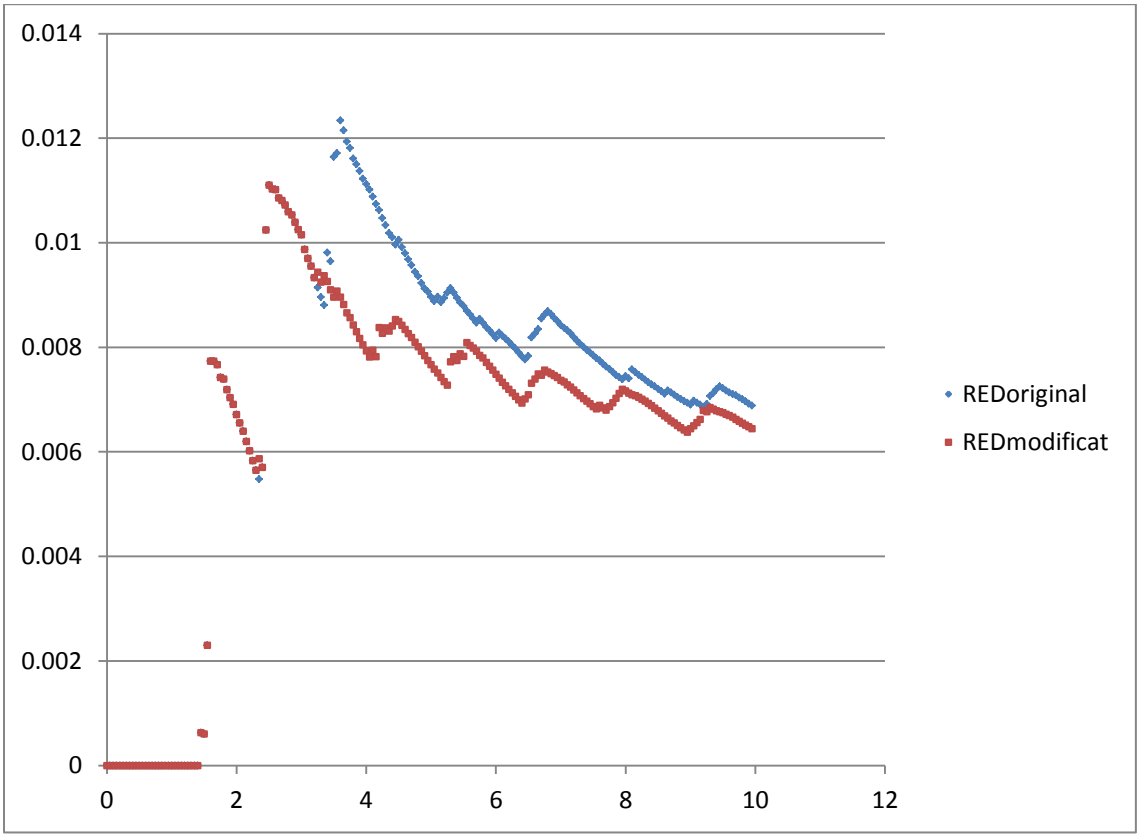


Figura 6.12 Frecvența de aruncare a pachetelor în timp pentru configurația 357777

În continuare voi prezenta un tabel cu date centralizate în urma simularii cu mai multe tipuri de configurații pentru construirea funcției de care ne folosim pentru calculul probabilității de aruncare.

Variantă algoritm	Variația cozii medii față de RED	Frecvența de aruncare a pachetelor	Variația frecvenței de aruncare față de RED	I(alpha=0.5)
RED original	0	0.00688642	0	1
357357	0.30514258	0.006449777	-0.06340637	0.939804168
357333	0.296452406	0.006895891	0.001375344	0.958704134
357335	0.292464612	0.006600976	-0.041450304	0.931150361
357373	0.320120375	0.006828236	-0.008449064	0.990420391
357555	0.310936354	0.006402293	-0.070301643	0.945326038
357753	0.296452406	0.006895891	0.001375344	0.958704134
357777	0.323141197	0.00644672	-0.063850338	0.967431951

Pentru o evaluare mai buna a performanței noilor algoritmi s-a introdus in tabel un parametru integrator I = α*A + (1-α)*B, calculat în funcție de parametrul A, care reprezintă o valoare scalată a variației cozii medii față de dimensiunea cozii medii folosind algoritmul RED original, si parametrul B, care reprezintă o valoare scalată a variației frecvenței de aruncare față de variația frecvenței de aruncare

a algoritmului RED original. Cu cât acest parametru integrator are o valoare mai mică cu atât algoritmul este mai eficient decât algoritmul RED original.

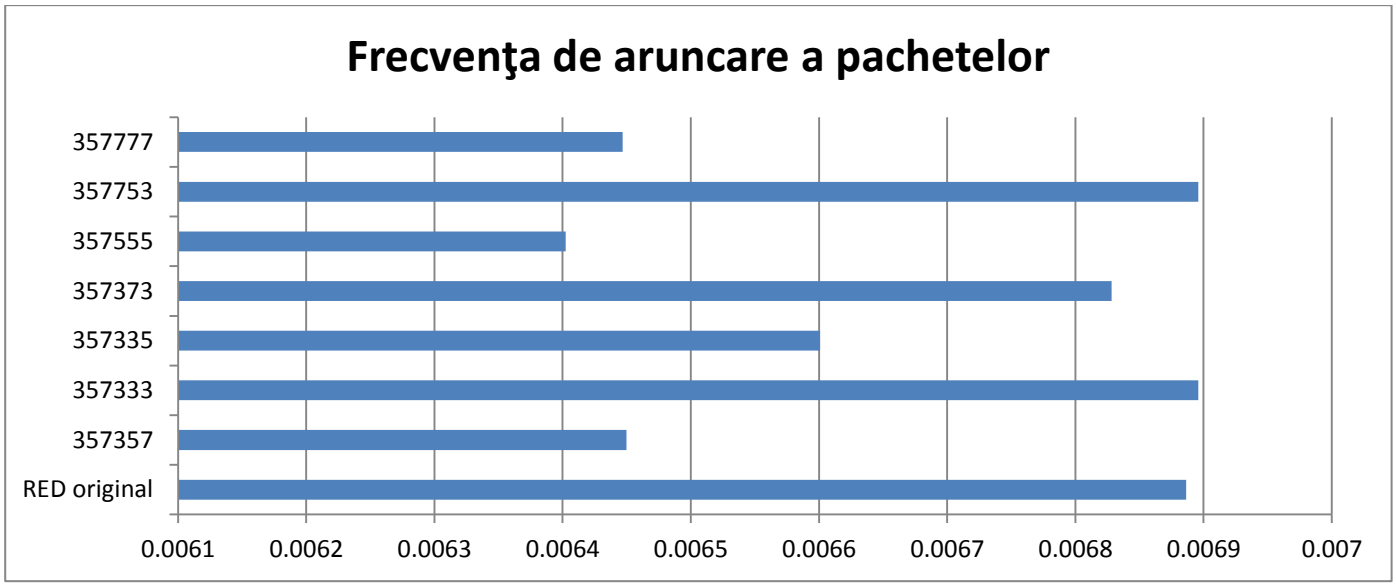


Figura 6.13

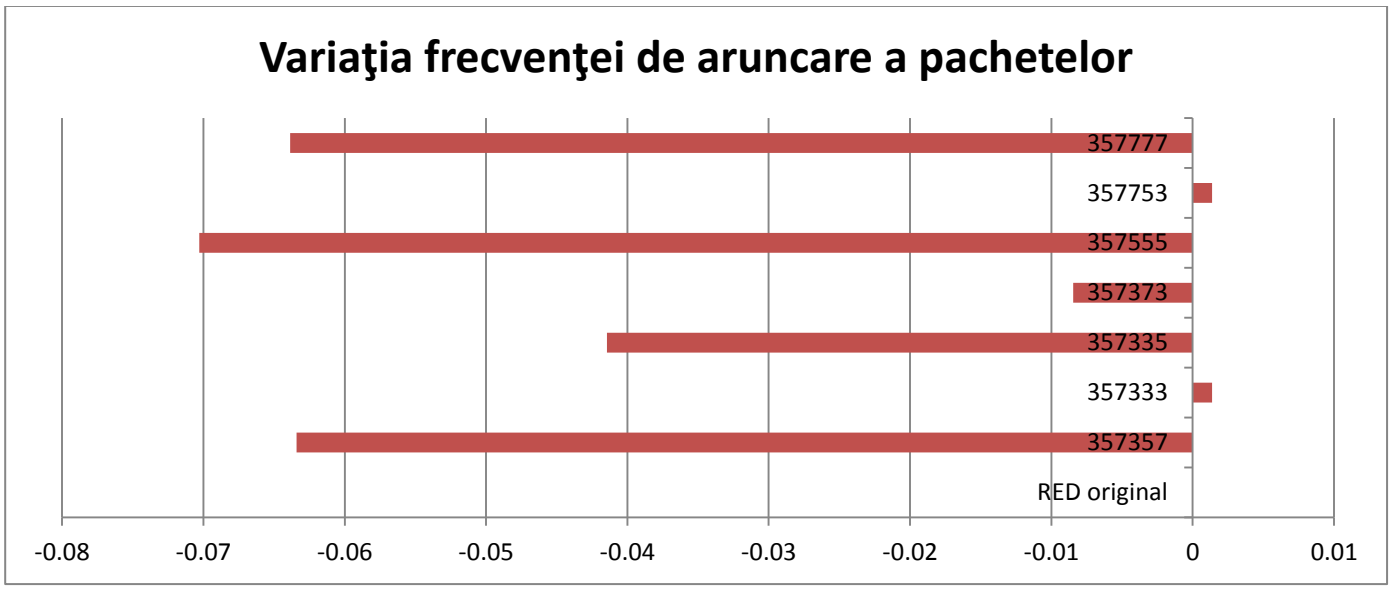


Figura 6.14

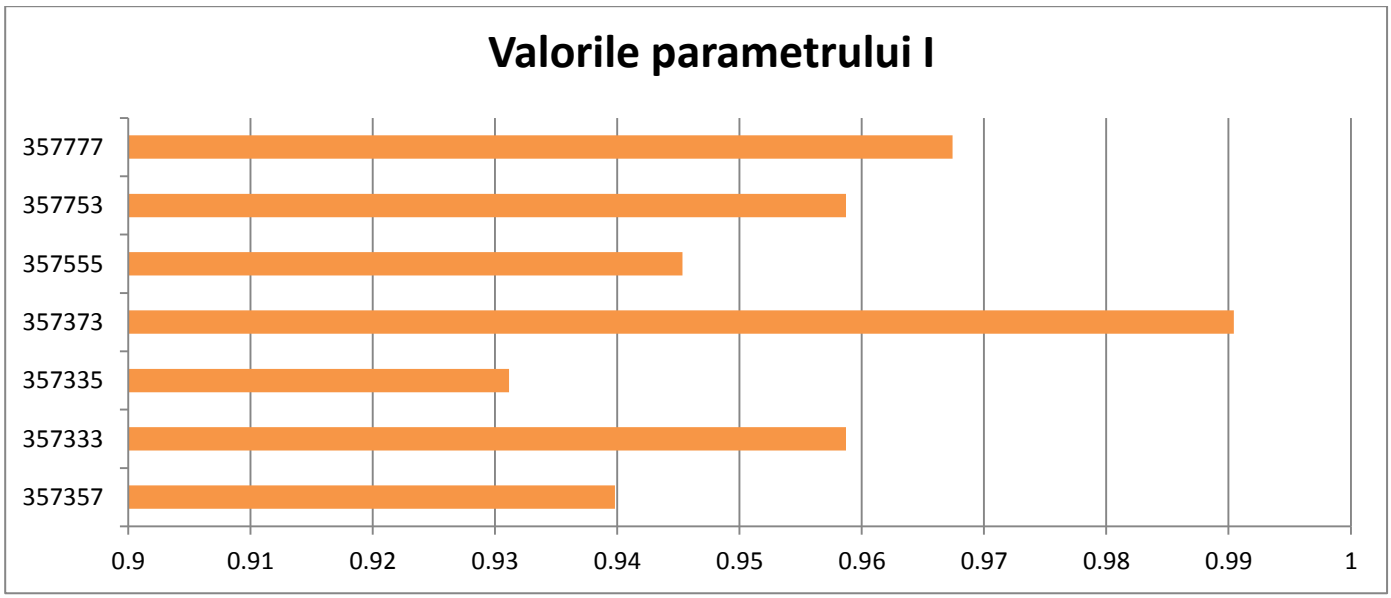


Figura 6.15

Urmărind datele din tabelul prezentat mai sus, precum și *Figura 6.14*, se observă că, din punct de vedere al frecvenței de aruncare a pachetelor, algoritmul care folosește configurația 357555 are frecvența de aruncare cu 7% mai mică decât algoritmul RED original. În concluzie, urmărind doar acest parametru, configurația 357555 este optimă, însă trebuie urmărit și parametrul care indică dimensiunea cozii medii. Așadar vom urmări variația parametrului integrator I, care depinde atât de frecvența de aruncare a pachetelor, cât și de dimensiunea medie a cozii. Din *Figura 6.15*, care arată variația parametrului I, rezultă că algoritmul optim este determinat de traseul funcției cu configurația 357335, prezentat în *Figura 6.1*.

CAPITOLUL 7

Concluzii

Detecția aleatoare timpurie (RED) reprezintă un mecanism eficient pentru evitarea congestiei la nivelul gateway-ului, în colaborare cu protocolul de transport din rețea. Gateway-urile RED aruncă pachetele atunci când dimensiunea medie a cozii depășește pragul maxim $\boldsymbol{max_{th}}$.

Probabilitatea cu care gateway-ul RED alege să notifice conexiunile despre apariția congestiei, este aproximativ proporțională cu cota cu care conexiunile respective participă la lățimea de bandă totală. Rata cu care ruterul marchează pachetele, depinde de nivelul congestiei. Cu ajutorul algoritmului RED se pot lua decizii conștiente referitoare la dimensiunea medie a cozii și a dimensiunii maxime a cozii de pachete, de la nivelul gateway-ului.

S-a studiat fenomenul combaterii congestiei cu strategii de tip RED, pornind de la soluțiile inițiale ale lui Jacobson, trecând prin soluțiile ulterioare prezentate [3]. A fost examinat în special articolul prezentat în Capitolul 4 şi s-a experimentat o metodă originală de calcul a celei mai bune strategii. S-au facut modificări in simulatorul Network Simulator în codurile sursă şi în programul TCL, prin care s-au efectuat experimentări şi s-au obținut performanțe de valori medii pentru coada de aşteptare şi pentru frecvența de aruncare a pachetelor.

S-a calculat un parametru integrat de performanță şi s-a obținut în raport cu acesta o strategie nouă de calcul a probabilității de aruncare a pachetelor care să minimizeze parametrul integrator. Rezultatul a fost prezentat în tabele şi grafice care arată o îmbunătățire de 6,9% față de algoritmul RED.

8.Bibliografie și referințe

[1] Andrew S. Tanenbaum, ”Computer Networks”, ediția a 4a
[2] Stefan Kohler, Michel Menth, Norbert Vicari, „Analytic Performance Evaluation of the RED Algorithm for QoS in TCP/IP Networks”, 9th IFIP Conference on Performance Modelling and Evaluation of ATM & IP Networks 2001, Budapest
[3] http://www.roman10.net/modification-of-red-aqm-in-ns2/
[4] Sally Floyd, Van Jacobson, „Random Early Detection Gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, August 1993
[5] http://books.google.ro/books?id=UIGR-5RiurQC&pg=PA223&lpg=PA223&dq=random+early+modification&source=bl&ots=TPrRuAWcFa&sag=uiQowWdKuSnk57PSDfz-7xT_5H8&hl=ro&ei=uUwuSoiJEI-fsgbCxMC8CQ&sa=X&oi=book_result&ct=result&resnum=9#PPA223,M1
[6] http://nsnam.isi.edu/nsnam/index.php/Installing_ns2.31_on_Ubuntu7.04
[7] Floyd, Sally; Jacobson, Van (August 1993). "Random Early Detection (RED) gateways for Congestion Avoidance".
[8] G.A. Carleciuc, Ștefan Stăncescu, „Congestion avoidance using modified Random Early Detection algorithm, and a comparison between standard RED and modified RED”, București
[9] S.Floyd, “Congestion Control Principles”, ACIRI, September 2000
[10]Martin May, Jean Bolot, Christophe Diot, Bryan Lyles, “ Reasons not to deploy RED” , Proc. of 7th. International Workshop on Quality of Service, 1999
[11] Dong Lin, Robert Morris, “ Dynamics of RED” , Harvard University , 2003
[12] Kai Jiang, Xiao Fan Wang, Yugeng Xi, “ A Robust RED Algorithm Based on Time delayed Feedback Control”, Shanghai Jioatong University, 2004
[13] Robert Shorten, Fabian Wirth, Douglas Leith, “A positive systems model of TCP- like congestion control: Asymptotic results” , April 7, 2004
[14]http://en.wikipedia.org
[15]http://www.tcpipguide.com/
[16]http://www.cs.cmu.edu/~mihaib/articole/csfq/csfq-html.html
[17]http://www.icir.org/floyd/papers/red/red.html
[18]http://www.faqs.org/rfcs/rfc2581.html
[19]http://www.isi.edu/nsnam/ns/
[20]http://www.ici.ro/SIC/sic2005_2/art06.pdf
[21]www.ietf.org/rfc
[22]http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html
[23]http://www.isi.edu/nsnam/ns/
[24]http://www.cse.msu.edu/~wangbo1/ns2/
[25]http://www.isi.edu/nsnam/nam/index.html
[26]http://www.ensc.sfu.ca/~ljilja/cnl/presentations/grace/modeling_TCP/sld016.htm
[27]http://www.isoc.org/inet2000/cdproceedings/2d/2d_2.htm#s1
[28]http://www.isi.edu/nsnam/ns/tutorial/index.html
[29]http://citeseerx.ist.psu.edu/

[30]http://nile.wpi.edu/NS/
[31]http://nsnam.isi.edu/nsnam/index.php/Installing_ns2.31_on_Ubuntu7.04
[32]http://www.isi.edu/nsnam/ns/tutorial/
[33]http://www.opalsoft.net/qos/DS.htm
[34]http://www.nabble.com/Network-Simulator-ns-2-f15582.html
[35]www.ieee.org
[36]www.acm.org

Anexa 1. Codul TCL

```
set ns [new Simulator]

set tf [open outm.tr w]
set windowVsTime [open winm w]
$ns trace-all $tf

set nf [open out.nam w]
$ns namtrace-all $nf

set p1 [lindex $argv 0]
set p2 [lindex $argv 1]
set p3 [lindex $argv 2]
set p4 [lindex $argv 3]
set p5 [lindex $argv 4]
set p6 [lindex $argv 5]
puts "p1: $p1; p2: $p2; p3: $p3; p4: $p4; p5: $p5; p6: $p6;"

Queue/RED set thresh_ 5
Queue/RED set maxthresh_ 15
Queue/RED set q_weight_  0.002
Queue/RED set p1_ $p1
Queue/RED set p2_ $p2
Queue/RED set p3_ $p3
Queue/RED set p4_ $p4
Queue/RED set p5_ $p5
Queue/RED set p6_ $p6

set NumbSrc 7
set Duration 50

for {set j 1} {$j<=$NumbSrc} {incr j} {
set S($j) [$ns node]
}


$ns duplex-link $S(1) $S(3) 10Mb 3ms DropTail
$ns duplex-link $S(2) $S(3) 10Mb 4ms DropTail
$ns duplex-link $S(3) $S(5) 10Mb 5ms DropTail
$ns duplex-link $S(4) $S(5) 10Mb 6ms DropTail
$ns duplex-link $S(5) $S(6) 10Mb 7ms DropTail
$ns duplex-link $S(6) $S(7) 1.5Mb 20ms RED
$ns queue-limit $S(6) $S(7) 25
$ns queue-limit $S(7) $S(6) 25




$ns duplex-link-op $S(1) $S(3) orient right-down
$ns duplex-link-op $S(2) $S(3) orient right-up
$ns duplex-link-op $S(3) $S(5) orient right-down
$ns duplex-link-op $S(4) $S(5) orient right-up
$ns duplex-link-op $S(5) $S(6) orient right
$ns duplex-link-op $S(6) $S(7) orient right
$ns duplex-link-op $S(6) $S(7) queuePos 0
$ns duplex-link-op $S(7) $S(6) queuePos 0
```

```
#$S(3) add-route $S(2) $S(6)

set tcp(1) [$ns create-connection TCP/Reno $S(1) TCPSink $S(7) 0]
$tcp(1) set window_ 30
set tcp(2) [$ns create-connection TCP/Reno $S(2) TCPSink $S(7) 1]
$tcp(2) set window_ 30
set tcp(3) [$ns create-connection TCP/Reno $S(3) TCPSink $S(7) 2]
$tcp(3) set window_ 30
set tcp(4) [$ns create-connection TCP/Reno $S(4) TCPSink $S(7) 3]
$tcp(4) set window_ 30
set tcp(5) [$ns create-connection TCP/Reno $S(5) TCPSink $S(7) 4]
$tcp(5) set window_ 30


#parametrizarea suselor tcp
for {set j 1} {$j<=5} {incr j} {
$tcp($j) set packetSize_ 552
}

set ftp1 [$tcp(1) attach-source FTP]
set ftp2 [$tcp(2) attach-source FTP]
set ftp3 [$tcp(3) attach-source FTP]
set ftp4 [$tcp(4) attach-source FTP]
set ftp5 [$tcp(5) attach-source FTP]


# Tracing a queue
set redq [[$ns link $S(6) $S(7)] queue]
set tchan_ [open all.q w]
$redq trace curq_
$redq trace ave_
$redq attach $tchan_

#set redq [[$ns link $S(1) $S(3)] queue]
#set traceq_ [open all2.q w]
#$redq trace curq_
#$redq trace ave_
#$redq attach $traceq_



$ns at 0.0 "$ftp1 start"
$ns at 1.0 "$ftp2 start"
$ns at 2.0 "$ftp3 start"
$ns at 3.0 "$ftp4 start"
$ns at 10 "finish"

proc finish {} {
    global nf , tchan_ , tf , ns
    $ns flush-trace
    close $tf
    close $nf
    #puts "running nam..."
    exec nam out.nam &
    exec grep "^d" outm.tr > aruncm.tr
    exec grep "^+" outm.tr > enqm.tr
    exec grep "^-" outm.tr > dequ.tr
    set awkCode {
```

```
      {
          if ($1 == "Q" && NF>2) {
            print $2, $3 >> "temp.q";
            set end $2
          }
          else if ($1 == "a" && NF>2)
          print $2, $3 >> "ave.tr";
      }
    }

   set awkCode3 {
      {

            print $2 >> "PKTARUNC.q";

            set end $2

      }
    }

    set f [open temp.queue w]
    puts $f "TitleText: red"
    puts $f "Device: Postscript"

    if { [info exists tchan_] } {
      close $tchan_
    }
    exec rm -f temp.q ave.tr
    exec touch ave.tr temp.q

    exec awk $awkCode all.q
    exec awk $awkCode3 aruncm.tr

    puts $f \"queue
    exec cat temp.q >@ $f
    puts $f \n\"ave_queue
    exec cat ave.tr >@ $f
    close $f
    exec xgraph -bb -tk -x time -y queue temp.queue &
    exit 0
}

set qfile [$ns monitor-queue $S(6) $S(7) [open queuem.tr w] 0.05]
[$ns link $S(6) $S(7)] queue-sample-timeout;


$ns run
```