Universitatea „Politehnica’’ Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Combaterea congestiei cu ajutorul algoritmilor RED

Proiect de licenţă

prezentat ca cerinţă parţială pentru obţinerea titlului de
Inginer în departamentul *Electronică Aplicată şi Tehnologia Informatiei*,
domeniul *Calculatoare şi Tehnologia Informaţiei*,
programul de studii *Ingineria Informaţiei*

Conducător ştiinţific

Conf. dr. ing. Ştefan Stăncescu

Absolvent

Damaschin Ionuţ

Anul 2014

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

Departamentul Electronică Aplicată si Ingineria Informației

Aprobat Director de Departament :

Prof. Dr. Ing. Sever PAŞCA

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Damaschin D. Ionuţ George, grupa 443A

1. Titlul temei: **Combaterea congestiei cu ajutorul algoritmilor RED.**

2. Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare):

Se vor face simulari cu simulatorul Network Simulator prin scripturi TCL; se genereaza fisiere de trace pe baza carora se vor realiza reprezentarile grafice si tabelele pentru compararea parametrilor de calitate si transfer in functie de algoritmii folositi; graficele vor fi realizate cu GNUPLOT; se va modifica sursa NS care implementeaza algoritmi RED, se vor experimenta transferuri cu algoritmii modificati si se vor compara performantele.

3. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline:
Retele de calculatoare, Sisteme de operare, Arhitecturi de reţea şi Internet

4. Realizarea practică/ proiectul rămân în proprietatea: *UPB*

5. Proprietatea intelectuală asupra proiectului aparţine: *UPB*

6. Locul de desfăşurare a activităţii: *UPB, ETTI*

7. Data eliberării temei: 12.12.2013

CONDUCĂTOR LUCRARE:

Prof.Dr. Ing. Conf. Ştefan Stancescu

STUDENT:

Ionuţ George Damaschin

DECLARAŢIE DE ORIGINALITATE A PROIECTULUI DE DIPLOMĂ/LUCRĂRII DE DISERTAŢIE[1)]

(MODEL)

Subsemnatul DAMASCHIN DIONUT GEORGE [2)], posesor al B.I./C.I./paşaport seria AS, nr. 643720, având CNP 1910716160134, am întocmit proiectul de absolvire/diplomă/lucrarea de disertaţie cu tema:[3)] COMBATEREA CONGESTIEI CU AJUTORUL ALGORITMILOR RED, în vederea susţinerii examenului de finalizare a studiilor universitare de licenţă/masterat, organizat de către Facultatea de ELECTRONICĂ, TELECOMUNICAŢII ŞI TEHNOLOGIA INFORMAŢIEI, Departamentul ELECTRONICĂ APLICATĂ ŞI INGINERIA INFORMAŢIEI din cadrul Universităţii POLITEHNICA din Bucureşti, sesiunea IULIE, anul universitar 2014.

Luând în considerare conţinutul art. 143 din Legea Educaţiei Naţionale nr. 1/2011, al. (4) „*Îndrumătorii proiectelor de diplomă/lucrărilor de disertaţie răspund în solidar cu autorii acestora de asigurarea originalităţii conţinutului acestora*" şi al. (5) „*Este interzisă comercializarea de lucrări ştiinţifice în vederea facilitării falsificării de către cumpărător a calităţii de autor al unui/unei proiect de diplomă/lucrări de disertaţie*", declar pe proprie răspundere, că acest/această proiect/lucrare este original(ă), fiind rezultatul propriei activităţi intelectuale, nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei în vigoare.

Cunosc faptul că plagiatul sau prezentarea unui/unei proiect/lucrări, elaborat(ă) de alt absolvent sau preluată de pe internet, din manuale şi cărţi, fără precizarea sursei constituie infracţiune (furt intelectual şi nerespectarea dreptului de autor şi a proprietăţii intelectuale) şi atrage după sine anularea examenului de diplomă/disertaţie, precum şi răspunderea penală.

Data: 23.06.2014 Semnătura: _____
(în original)

[1)] Declaraţia se completează „*de mână*" şi reprezintă un document care se depune la înscrierea pentru susţinerea examenului de finalizare a studiilor.
[2)] Numele, iniţiala prenumelui tatălui şi prenumele, cu majuscule.
[3)] Denumirea proiectului de diplomă/lucrării de disertaţie, cu majuscule.

Cuprins

Introducere

În aceasta lucrare mi-am propus sa studiez combaterea congestiei folosind algoritmul RED. Pe baza unor repere generale se va realiza un studiu al algoritmilor folosiţi pentru combaterea congestiei, urmând apoi să discut despre principiile de bază pentru combaterea congestiei. Se vor urmări mai mulţi algoritmi pentru abordarea congestiei în momentul în care aceasta a intervenit.

Am ales algoritmul RED (Random Early Detection), cunoscut şi ca Random Early Drop, care are avantajele de a controla congestia la nivel de transport de pachete şi că acesta funcţionează cu actualele protocoale de transport (TCP/IP). De asemenea nu este nevoie ca toate ruterele din reţea să folosească acelaşi mecanism de control al congestiei. RED calculează probabilitatea de aruncare a unui packet în două etape, atunci când dimensiunea cozii scade între th_min şi th_max, unde th_min reprezintă pragul de minim al cozii, th_max reprezintă pragul de maxim al cozii.

Simulatorul în care se va lucra în decursul acestui proiect este Network Simulator 2. Tot în cadrul acestui proiect, pornind de la algoritmul RED iniţial, cel inventat de Sally Floyd şi de Van Jacobson şi urmând modificările aduse acestui algoritm prin schimbarea codului sursa, am reuşit să compar anumite funcţii şi să obţin soluţii optime totodată pentru diferitele cazuri cum ar fi variatia cozii medie sau frecventa de aruncare a pachetelor chiar şi întreruperile din cadrul retelei. Totodata am reuşit sa compar performantele functiile introduse în codul sursa cu funcţia algoritmului RED nemodificat.

În implementarea şi dezvoltarea acestui proiect, un rol important l-a avut domnul profesor coordonator Ştefan Stancescu.

CAPITOLUL 1
Elemente de bază ce definesc congestia

1.1 Introducere

Definiţie

Performanţele unei subreţele se vor deteriora pentru cazul în care în subreţeaua respectivă sau o parte din ea este existent un număr foarte mare de pachete. O astfel de situaţie putem spune că se mai numeşte congestie a reţelei (congestion). În figura 1.1 este prezentat un grafic ce reflectă congestia în reţelele de telecomunicaţie. [1]

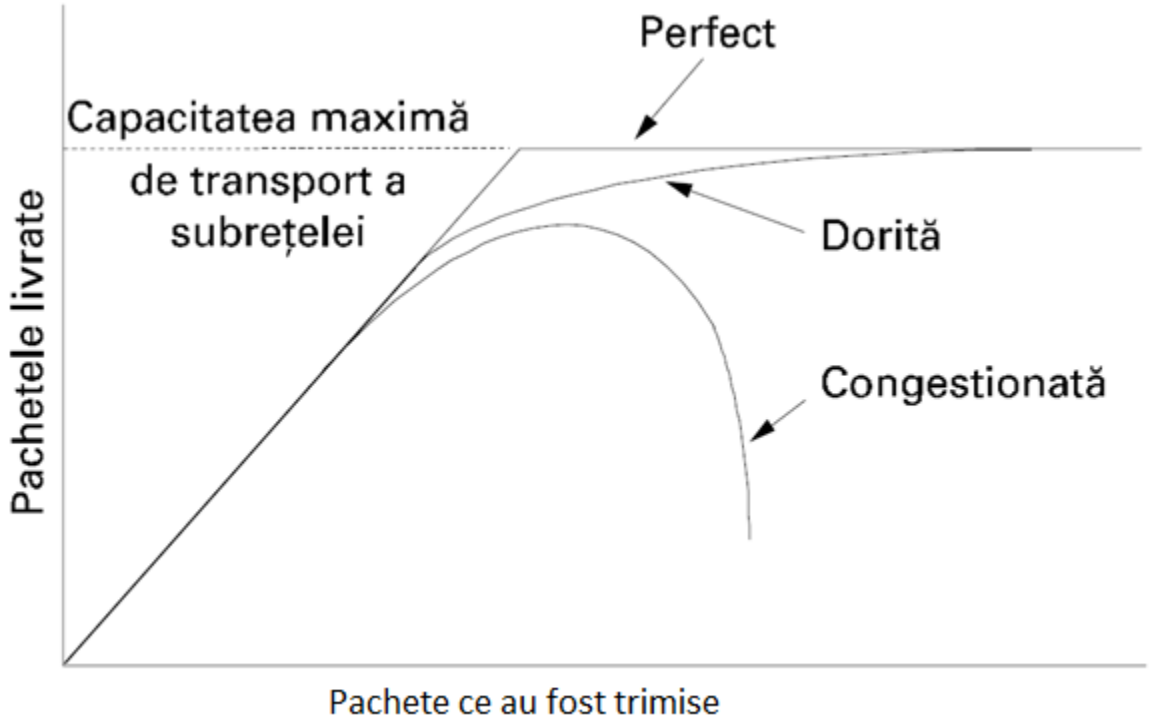


Figura 1.1 : Performanţele şi congestia scad radical pentru un trafic agresiv în reţea. [1]

1.2 Factori ce pot influenţa congestia

Pachetele care sunt trimise de-a lungul subreţelei de calculatoare vor fi expediate într-un mod integral, dar nu şi cele care conţin erori ale transmisiei şi doar pentru cazul în care numărul lor nu va putea depaşi capacitatea de transport din subreţeaua respectivă de calculatoare.

Numărul pachetelor care au fost expediate va trebui să fie direct proporţional cu numărul pachetelor ce sunt primite de către un nod.

Cu toate acestea dacă traficul are momente când devine din ce ăn ce mai agresiv, în acele momente routerele vor începe să nu poata să mai dea randament pentru toate pachetele ce se

trimit, unele dintre ele urmând sa se piardă, aceast lucru putând să fie din ce în ce mai rău, pe măsură ce traficul devine tot mai mare. Cu alte cuvinte, se poate ajunge la pierderea a tuturor pachetelor, nici un pachet neputând fi expediat, iar nivelul performanţelor să devină nul pentru un trafic extrem de agresiv.

În reţelele de calculatoare, există mai mulţi factori care pot produce congestia.

O coadă (sau queue) se va crea în momentul în care simultan mai multe grupuri de pachete pe trei ori patru linii de transmisie vor avea nevoie de o singura linie de ieşire.

În cazulîn care nu va exista îndeajuns spaţiu de memorie pentru a le păstra pe toate în coadă, atunci anumite pachete vor fi aruncate.

Neagle în anul 1987 a descoperit prin mai multe experimente că daca se doreşte a adăuga spaţiu memorie suplimentară, acest lucru se poate face însă între anumite limite. Adăugarea de spaţiu memorie suplimentară (în cantităţi enorme), infinită în routere nu ar duce la avantaje ale reducerii congestiei, ci dimpotrivă, congestia în reţeaua de calculatoare s-ar face din ce ăn ce mai mare ajungând să se deterioreze complet. Acest lucru se datorează faptului că pachetele au fost declarate ca fiind pierdute mai devreme ca ele să intre în coadă, iar din cauza unui timeout repetat, acestea au trimis de două ori aceeaşi informatie (duplicat).

Datorită dimensiunii duble ce va fi trimisă către următorul router (sau nod), acest lucru va încarca reţeaua respectivă din ce în ce mai mult. Cu alte cuvinte, pe măsură ce un pachet doreşte a fi transferat de-a lungul a unui anumit număr de routere cu memorie infinită, atunci când trece doar prin unul singur îşi va duplica dimensiunea; pentru următorul router îşi va duplica din nou dimensiunea, şi tot aşa până când reţeaua va fi încărcată total.

Un alt factor care poate duce la congestie sunt procesoarele lente care îngreunează transferul de pachete.

Pentru cazul în care CPU sau unitatea centrală a unui router prezintă performanţe scăzute, atunci aceasta va îngreuna viteza de rulare a target-urilor pe care trebuie să le îndeplinească, de exemplu updatarea tabelelor sau introducerea pachetelor în interiorul cozilor etc.

Cu toate că dacă liniile de comunicaţie nu vor fi utilizate pentru capacitate, se pot totodată crea cozi în retea.

Un cu totul alt factor care determină congestia este reprezentat de lărgimea benzii scăzute a liniilor reţelei. Acest lucru se poate remedia, dar nu foarte mult, şi doar dacă link-urile (sau liniile) vor fi schimbate cu altele mai performate iar procesorul să rămână acelaşi, sau pentru aceleaşi linii şi schimbarea procesorului cu unul performant. Aceste modalităţi de cele mai multe ori ajută reţeaua, însă fac doar ca punctual critic să se deplaseze. Pentru a îmbunătaţi într-un mod incomplet sistemul (şi nu in totalitate), va trebui ca punctul critic să fie mutat într-o altă parte, însă acest lucru nu este posibil uneori, deoarece deşi problema mutării punctului critic este des întâlnită, nu de fiecare dată părţile componente ale sistemului vor fi compatibile. Această problemă va rămâne neschimbată până când toate componentele vor avea un oarecare echilibru.

1.3 Diferenţe între controlul fluxului şi controlul congestiei

Între controlul fluxului şi controlul congestiei există o oarecare diferenţă pe care este important să o precizăm de-a lungul discuţiei.

- controlul fluxului face referire la transportul şi mai precis la traficul de pachete dintre locul de unde au fost trimise şi locul unde au ajuns. Principala acţiune ce trebuie îndeplinită de către

controlul fluxului este a nu da voie unui expeditor să trimita pachete în continuu cu viteze mai mari decât cele cu care cel care primeşte poate utiliza acele date.

De cele mai multe ori, destinatarul va trimite un mesaj (de feedback) către expeditor pentru a-l înştiinţa despre modul cum se desfăşoară transmisia în acel loc.

- controlul congestiei are rolul de a garanta ca reţeaua poate transporta tot traficul de pachete care o afectează. Aceasta putem spune că este o situaţie ce se desfăşoară la un nivel global şi totodată urmăreşte cum se comportă toate routerele, computerele gazdă, procesarea datelor ce sunt memorate şi apoi retransmise (modul store and forward) de către routerele din reţea, şi în acelaşi timp toţi factorii care minimizează capacitatea de transfer a pachetelor în reţea.

Diferenţa dintre controlul congestiei şi controlul fluxului nu este atât de usor sesizabilă, însă putem da un exemplu în aşa fel încât acest lucru să poată fi evidenţiat.

Putem astfel considera o reţea formată din fibre optice cu o capacitate de transmisie de un TerraBit/secunda şi un supercalculator care transmite un pachet către un simplu calculator cu viteza de 1 GB/s.

Pentru a vedea diferenţa dintre aceste două concepte, să considerăm o reţea cu fibre optice cu o capacitate de 1000 gigabiţi/sec pe care un supercalculator încearcă să transfere un fişier către un calculator personal la 1 Gbps.

Controlul fluxului este cel care va trebui să forţeze acel supercalculator să facă opriri dese cu scopul de a lăsa calculatorul obişnuit să se poată odihni chiar dacă în reţea nu există congestive şi nu este nici un fel de problemă în acest sens. Mai precis, dispositivele nu sunt compatibile din punct de vedere al performanţelor şi acest lucru îngreunează transmisia.

Pentru un alt caz ce poate evidenţia diferenţele între flux şi trafic, putem exemplifica o reţea de tipul store and forward (sau memorează şi retransmite) cu un număr de linii ce au viteze care ajung la 1 MB/secundă şi un număr de 1000 de calculatoare performante dintre care 500 trimit pachete cu viteza de 100 KB/secundă către celelalte 500 de calculatoare. În acest caz, problema se datorează traficului, deoarece acesta va depăşi limitele reţelei, nefiind vorba despre cazul unui destinatar lent şi a unui expeditor rapid.

De cele mai multe ori controlul fluxului şi controlul congestiei sunt confundate, iar acest lucru se face datorită faptului că există algoritmi ce sunt rulaţi pentru a controla congestia, care trimit mesaje de feed-back pentru diferiţi emiţători, pe care îi mai informează să nu mai trimită pachete, sau să trimită mai puţine atunci când reţeaua este congestionată. Mai putem spune că datorită faptului că un destinatar nu poate suporta încărcarea informaţiilor sau datorită faptului că reţeaua respectiă este încărcată la maxim, atunci un calculator care primeşte acele informaţii va primi şi un mesaj pentru a încetini transmiterea. Se va mai reveni la acest aspect în cele ce vor urma.

În continuare, ne vom ocupa pe baza unor repere generale de studiul algoritmilor folosiţi pentru combaterea congestiei, urmând apoi să discutăm despre modurile de bază pentru combaterea congestiei. În final, se vor urmări mai mulţi algoritmi pentru abordarea congestiei în momentul în care aceasta a intervenit.

1.4 Caracteristicile de bază ale controlului congestiei

Majoritatea situaţiilor ce sunt întâlnite la (sisteme) reţelele complexe de calculatoare se pot aborda urmărind teoria controlului. Urmârind acest mod de a vedea lucrurile, atunci toate soluţiile de rezolvare a situaţiilor respective vor fi împărţite în două cazuri:

- În buclă închisă
- În buclă deschisă.

Urmărind aceste soluţii, se constată că cele în buclă deschisă au rolul de rezolvare a situaţiei urmărind ca proiectarea să se faca riguros, în acest fel va garanta că nu se va mai ivi nici o problemă.

Nu se vor mai putea face modificări din momentul în care reteaua se va porni şi va fi funcţionabilă.

Pentru ca funcţionarea reţelei să fie continuă, controlul congestiei se va realiza într-o buclă deschisă unde instrumentele utilizate vor seta când pachetele vor trebui să fie distruse şi care dintre ele, când traficul nou va trebui să fie acceptat cât şi celelalte decizii pe care vor trebui să le ia pentru diferite routere din reţeaua respectivă. Pentru realizarea acestor lucruri nu se va urmări starea prezentă pe care o are reţeaua, acest lucru reprezentând un factor comun pentru acele decizii.

Pentru cazul soluţiilor ce vor fi realizate în buclă închisă, acestea folosesc termenul de feed-back sau de reacţie inversă. Acest gen de tratare a problemelor conţine trei paşi importanţi care vor fi urmaţi succesiv:

1. Reţeaua este examinată pentru a vedea în ce loc şi timpul cînd congestia se va produce;
2. Informaţii care au fost obţinute la primul pas vor fi expediate ca mesaje către nodurile unde vor avea loc evenimente.
3. Pentru ca situaţia să se remedieze, funcţionarea sistemului va fi minimizată.

Se poate utiliza o numeroasă gamă de metrici pentru a putea examina sistemul de legături. Procentul din totalul de pachete care au fost aruncate datorită insuficienţei spaţiului de memorie din router, variaţia întârzierii pachetelor, delay-ul (întârzierea) medie a pachetelor, dimensiunea medie a cozii sau numărul de pachete care vor urma să fie retransmise din cauza că timpul de transmisie a fost încheiat (timeout) fac parte din cele mai folosite metrici de monitorizare. Pentru oricare dintre situaţii, atunci când valorile unui parametru cresc, atunci va creşte şi congestia din reţea.

Pentru cazul pasului cu numărul doi, în bucla de reacţie se va găsi transferul de informaţii referitoare la congestia din reţea de la depistarea sa la momentul în care se pot realiza acţiuni pentru remediere.

Aşadar, există varianta de a trimite pachetele de la un router în care se găseşte congestie, la alte surse în care se poate afla trafic de pachete, astfel putând face un raport pentru acea situaţie. Se mai poate spune că aceste pachete încarcă reţeaua, astfel acest lucru nefiind dorit, deoarece congestia va fi prezentă de-a lungul reţelei.

Mai există şi alte variante în afară de cele prezentate mai sus, ca exemplu util, pentru a se completa de către noduri (routere) atunci când congestia va deveni agresivă, se poate rezerva un câmp în fiecare dintre pachete, sau mai simplu doar un bit. Se vor completa acele câmpuri pentru

toate pachetele care au fost trimise cu scopul de a informa şi nodurile vecine, atunci când un router va găsi congestie pe legături.

Mai există o modalitate prin care routerele sau alte noduri în care se găsesc pachete să poată trimite din când în când pachete de test pentru a primi un timp de răspuns şi în funcţie de acesta să îşi poată da seama dacă legăturile sunt congestionate, totodată datele de acest tip sunt utilizate pentru a se evita zonele în care există congestie.

Astfel putem exemplifica elicopterele care deţin staţii radio şi care zboară de-a lungul oraşelor cu scopul de a depista congestiile pe drumuri şi a informa ascultătorii semnalului radio să evite călătoriile (sau trimiterea pachetelor) spre zonele congestionate.

Pentru toate cazurile în care se urmăreşte primirea unui mesaj de feed-back, de fapt se aşteaptă ca informaţiile primate legate de congestive vor face ca nodurile respective care primesc aceste mesaje să ia măsuri utile pentru a nu se produce congestie.

Reglarea atentă a duratelor reprezintă un factor ce trebuie luat în considerare, deoarece acest lucru face ca reţeaua să funcţioneze fără probleme.

Reţeaua nu converge niciodată şi oscilează într-un mod foarte radical pentru cazul în care un router va anunţa mesajul de Stop în momentele în care vor ajunge consecutive cîte două pachete, şi apoi anunţă mesajul Go (pleacă) în momentul în care coada este liberă mai mult de 20 de microsecunde.

Procesul de control al congestiei va reacţiona foarte greu şi nu va fi util în timp real pentru cazul în care va trebui să se aştepte de exemplu o jumătate de ora pentru a garanta înainte de a da un mesaj de Start sau de Go. Astfel, pentru a funcţiona într-o manieră normală, vor fi utile anumite medieri, dar totuşi, va fi un lucru greu a identifica acele constante de timp care pot fi cele mai potrivite pentru reţea.

Pentru controlul congestiei există foarte mulţi algoritmi ce pot trata această problemă.

Reddy si Yang în anul 1995 au stabilit numeroase reguli de clasificare pentru algoritmii de acest tip, astfel putând realiza o organizare.

Pentru realizarea acestor reguli, au împărţit algoritmii aşa cum am prezentat mai sus, în cei cu bucla închisă şi cei cu buclă deschisă.

Totodată, algoritmii în buclă deschisă i-au împărţit în algoritmi care au rol asupra recepţionării şi algoritmi ce au rol asupra emiterii.

Algoritmii în buclă închisă au fost şi ei separaţi la rândul lor tot în două tipuri şi anume: cu feed-back explicit şi cei cu feed-back implicit. Pentru algoritmii cu feed-back implicit, aşa cum este timpul suficient pentru a transmite mesajele de confirmare, emiţătorul poate deduce existenţa congestiei doar din informaţiile venite de la noduri vecine.

Pentru algoritmi cu feed-back explicit, pentru a avertiza sursa respectivă despre congestie, pachetele vor fi expediate înapoi către sursa din momentul în care congestia se va produce.

Se mai poate spune că încărcarea reţelei pentru un moment, va fi relativ mare în raport cu încărcarea suportată de către componentele unui nod al reţelei, acest lucru ducând tot la congestie.

Pentru această rezolvare a acestei probleme există două posibilităţi:

- încărcarea pachetelor în interiorul reţelei va trebui redusă;
- adăugarea unei mai mari capacităţi privind resursele

Putem folosi ca exemplu o reţea care să utilizeze linii de telefonie care să ajute la o urcare progresivă a lărgimii benzii dintre anumite noduri. Creşterea lărgimii de bandă este mărită de către creşterea puterii transmisiei în cazul sistemelor în care se folosesc sateliţi.

Atunci când traficul va fi separat în mai multe căi de legătură, în loc să se utilizeze cea mai bună cale, acest lucru va duce la o creştere a lărgimii de banda. Routerele ce sunt utilizate pentru backup (folosite ca rezervă atunci când altele cedează), pentru ca reţeaua să fie mai rezistentă la defecţiuni, se vor utiliza, asigurând astfel o capacitate ridicată atunci când apare congestia agresivă.

Nu întotdeauna se poate realiza o creştere a capacităţii, deoarece capacitatea a crescut până la valoarea sa maximă, pentru aceasta situaţie, există decât o modalitate şi anume reducerea încărcării cu pachete.

Pentru a putea reduce încărcarea de pachete din reţea există numeroase modalităţi, cum putem enumera: scăderea serviciilor pentru o parte din useri sau doar pentru anumiţi; userii care refuză serviciile; sau userii care planifică cererile într-un mod care este foarte uşor previzibil.

Dintre aceste metode enumerate, doar pe anumite dintre ele le vom explica mai pe scurt. Ele se pot utiliza mult mai bine pe reţelele virtuale, iar pentru unele subrete ce utilizează legături virtuale, acestea se vor folosi pentru nivelul reţelei, dupa care se va urmări nivelul de transport pentru a se realiza astfel controlul congestiei.

1.5 Politici folosite pentru combaterea congestiei

În cele ce vor urma se va face un studiu pe baza combaterii congestiei in reţele.
În primul rând se vor examina reţelele care folosesc sisteme cu buclă deschisă, putând astfel să determinăm modul prin care se poate realiza controlul congestiei. Aceste sisteme cu buclă închisă sunt realizate cu scopul de a micsora congestionarea traficului. Scopul principal este de a nu da voie congestiei sa se producă, aceste sisteme actionând înaintea producerii. Acest obiectiv pe care îl au se poate realiza utilizând anumite politici pentru numeroase niveluri.

În tabelul 1.1 vor fi enumerate numeroase politici utilizate pentru fiecare nivel, transport, reţea cât şi legaturi de date care au rolul influenţarii congestiei in sens pozitiv sau negativ. Aceste politici au fost realizate in anul 1990 de către Raj Jain.

Vom începe cu prezentarea nivelului de legatură dintre nodurile reţelei, care este o politică inferioară, apoi vom continua cu prezentarea celorlalte niveluri.

Politica retransmisiei are rolul de a determina intervalul de timp în care se produce un timeout la sursă, şi determină in acel interval de timp informatii despre pachetele ce vor fi trimise. O sursă rapidă ce realizează un timeout foarte repede şi apoi pune toate pachetele în coada de asteptare utilizând retransmiterea lor, va realiza supraîncarcarea în comparatie cu o sursă lentă care fososeste selecţia atunci când are loc retransmisia pachetelor.

Politica ce stă la baza acestora este cea de memorare din zona buffer-ului. În momentul în care destinatarii vor arunca toate pachetele in afară de secvenţa respectivă, atunci pachetele vor fi nevoie să fie retrimise utilizând o nouă încarcare.

Repetarea selectivă este o procedură mult mai sigură decat retransmiterea utimelor pachete. Congestia mai poate fi afectată şi de către politica utilizată pentru confirmare. În cazul în care oricare din pachete vor fi confirmate, atunci acestea vor produce un trafic mai mare. Se mai poate spune că dacă aceste confirmări vor fi preluate de către trafcul pentru răspuns, atunci se pot realiza suplimentar noi intervale de timeout şi retransmisii de pachete. Volumul de informatie poate fi redus atunci când este folosită pentru controlul fluxului o schemă mult prea stransă (ex : o fereastă prea mică).

Nivelul	Politică
Transportul	Politica retransmisiei de pachete Politica pentru memorarea temporară a pachetelor în afară de secvenţa respectiva sauout-of-order caching Politica utilizata pentru confirmare Politica pentru controlul fluxului Determinarea intervalului de timeout
Reţeaua	Circuitele virtuale contra datagrame în interiorul reţelei Punerea pachetelor în cozi de aşteptare şi politici de servire Politica distrugerii pachetelor Algoritmii de dirijare a controlului Gestiunea timpului de viaţă al pachetelor
Legăturile de date	Politica retransmiterii de pachete Politica pentru memorarea temporară a pachetelor în afară de secvenţă sau out-of-order caching Politica pentru confirmare Politica pentru controlul fluxului

Tabel 1.1 : Politici ce influenţează congestia. [1]

Datorită faptului că un numar mare de algoritmi de control al congestiei rulează doar în reţele cu circuite virtuale, atunci la nivelul de reţea, alegerea dintre folosirea circuitelor virtuale şi datagrame are rol de a influenţa congestia.

Totodata, mai este precizată ordinea prin care pachetele pot fi prelucrate (ex : metoda Round Robin). Metoda prin care se va alege care pachet sa fie aruncat în momentul în care nu mai există memorie în nod, foloseste politica prin care pachetele sunt distruse. Atunci când este folosita o politică buna, congestia va fi pe cale de a fi eliminată din retea, iar pentru caz contrar, congestia va fi accentuată.

Evitarea congestiei datorată cresterii traficului de-a lungul link-urilor mai poate fi realizată şi utilizând un algoritm bun ce dirijează procesele din reţea. Un algoritm mai puţin bun ar putea trimite pachetele pe aceeaşi linie ce este congestionată la rândul ei. Întretinerea timpului de viaţă a unui pachet va putea să determine cât va trăi un pachet pană sa fie aruncat. Daca timpul respectiv este unul mare, atunci pachetele ce au fost pierdute vor putea încurca celelalte procese din reţea, iar în cazul în care timpul este foarte mic se vor produce timeout-uri mai devreme ca pachetele sa ajungă la receptor, astfel urmând sa fie retransmise.

În cazul nivelului de transport vor fi aceleaşi situatii complicate ca la nivelul de legătură de date dar în plus la nivelul de transport aflarea intervalelor de timeout-uri va fi cu mult mai dificilă, din cauza faptului că timpul in care are loc tranzitul de pachete va fi mai greu de estimat decât timpul în care are loc tranzitul pe un link ce leagă două routere.

Atunci când intervalul de timeout va fi mult prea mic, pachete suplimentare se vor trimite fără nici un rezultat. Atunci când intervalul de timeout va fi mult prea mare, congestia din reţea va fi redusă, dar pierderea de pachete nu va afecta timpul de raspuns.

CAPITOLUL 2
Tehnici cunoscute de prevenire a pierderii pachetelor în reţele

2.1 Introducere

Prevenirea pierderilor de pachete este principalul scop al asigurării calităţii serviciilor (Quality of Service). [3]
Aceasta se realizează printr-un management adecvat al cozilor.

Pierderea pachetelor este cauzată din mai multe motive, dar motivele principale sunt:
-Din cauza erorilor de transmitere a pachetelor, care este un fenomen relativ rar;
-Din cauza congestiilor care apar în reţea.

Pentru evitarea si controlul congestiilor ce apar în reţea trebuiesc utilizate anumite mecanisme în punctele de acces şi în ruterele de legătură din reţea.

TCP-ul, sau aplicaţiile client se află la capătul reţelei şi trebuie să utilizeze un mecanism de adaptare la condiţiile reţelei. Ruterele din reţea sunt cele care asigură, cu ajutorul cozilor, un management al traficului. În arhitectura ruterelor, cozile sunt programate să absoarbă vârfurile de trafic. Atunci când dimensiunea cozilor este limitată au loc scăderi şi întârzieri ale pachetelor prin reţeaua respectivă. Atunci când cozile sunt pline, pachetele sunt eliminate. Tot ce este important într-o reţea este să nu se ajungă la încărcarea totală a cozilor.

Gateway-urile din reţele de mare viteză sunt proiectate cu cozi de lungime foarte mare pentru controlul congestiilor care apar. Protocolul TCP de transport va detecta o congestie numai dacă un pachet a fost aruncat de către gateway. Deşi cozile au dimensiuni foarte mari, care ating limite maxime din punct de vedere al arhitecturii lor, nu este necesar ca acestea să fie pline în majoritatea timpului

Dacă am avea de-a face cu cozi pline, atunci întârzierile medii din reţea ar creşte direct proporţional. Ca urmare, atunci când vitezei reţelei creste, este din ce în ce mai utilă folosirea unor mecanisme care păstrează throughput-ul ridicat, dar dimensiunea cozilor este mică pentru aceste mecanisme, ceea ce reprezintă un oarecare dezavantaj.

Dacă nu este detectat nici un mesaj de raspuns explicit din partea gateway-ului, protocolul de transport, de nivel 4 ar putea afla despre apariţia congestiei prin apariţia link-urilor bottlenecks sau gâtuiri, cu ajutorul aruncatii de pachete, cu ajutorul modificărilor care apar în throughput dar şi prin modificările întarzierilor point to point.

Modificările care apar în traficul conexiunii, lipsa cunoaşterii numărului de gateway-uri ce sunt afectate de congestii, schimbările de rutare posibile, modelul traficului, dar şi propagarea întârzierilor limitează perspectiva pe care o are conexiunea individuală.

2.1.1 Caracteristici ale gateway-ului:

Un gateway este un nod de reţea echipat pentru interfaţare cu o altă reţea care utilizează diferite protocoale de comunicare.

Gateway-ul este cel care garantează cea mai eficientă detecţie a apariţiei congestiei. Doar gateway-ul deţine o vedere unică şi individuală a modului în care are loc evoluţia cozilor în timp.

Gateway-ul este structurat în mai multe conexiuni cu timpi de răspuns care variază, având toleranţele faţă de întârzieri diferite şi cerinţe diferite legate de throughput.

Deciziile legate de durata şi dimensiunea congestiei permise la nivelul gateway-ului sunt luate cel mai bine chiar de către gateway.

Tot gateway-ul este cel care ia cel mai bine deciziile legate de dimensunea şi de durata congestiei, astfel oferind un support în acest caz.

2.2 Protocolul TCP

2.2.1 Generalităţi ale protocolului TCP:

Protocolul de control al transmisiei (sau TCP - Transmission Control Protocol) este un protocol utilizat în mare parte de aplicaţii ce au nevoie de confirmare de primire a informaţiilor. TCP creează o conectare virtuală full duplex între două puncte sau noduri terminale, fiecare punct fiind definit de o adresă IP şi de către un port (TCP). [2]

TCP foloseste noţiunea de numere de port sau altfel cunoscute ca port numbers pentru a identifica recepţia şi transmisia aplicaţiilor punctelor finale (end-points) de la o gazdă, sau a prizelor de Internet sau internet sockets.

Fiecare parte dintr-o conexiune TCP are asociată cu un număr de port de 16 biţi, fără semn rezervat pentru a expedia sau a primi date referitoare la aplicaţie. Sosirea pachetelor de date TCP este identificată ca făcând parte dintr-o anumită conexiune TCP, după priză sau *socket*, care este o combinaţie între adresa gazdei sursă sau portul sursă şi adresa gazdei destinaţie sau portul de destinaţie. Acest lucru înseamnă că un calculator server poate susţine pentru o mulţime de clienţi, în acelasi timp si mai multe servicii per client, atât timp cât fiecare client sau utilizator are în vedere a iniţia în acelaşi timp mai multe conexiuni către un port de destinaţie din porturi sursă diferite.

Transmission Control Protocol (TCP) este unul dintre protocoalele de bază din cadrul protocoalelor folosite în Internet. TCP este unul dintre cele două componente originale ale suitei sau totalitatea protocoalelor (celalalt fiind Protocolul Internet, sau IP), astfel încât întreaga suită mai este numită şi stiva TCP/IP. În special, TCP oferind încredere suficientă, asigură livrare ordonată a unui flux de octeţi de la un program de pe un calculator la un alt program de pe un alt calculator aflat în reţea. Pe lângă îndatoririle sale de oferire a siguranţei traficului, TCP mai controlează şi mărimea segmentului de date, rata la care se face schimbul de date, evitarea congestionării traficului de reţea precum şi debitul de informaţie. Printre aplicaţiile cele mai folosite, ce utilizează TCP putem enumera transferul de fişiere (FTP), World Wide Web(WWW) şi poşta electronică (e-mail).

2.2.2 Funcţii oferite reţelei

Acest protocol corespunde nivelului de transport din stiva TCP/IP. TCP oferă serviciu de comunicare la un nivel intermediar dintre Protocol Internet (IP) şi un program de aplicaţie. În momentul în care un program de aplicare doreşte să trimită o bucată mai mare de date în Internet, software-ul poate emite o cerere unică pentru TCP şi să lase protocolul TCP să se ocupe de detaliile de manipulare în loc să fragmenteze datele în pachete IP de dimensiuni mici şi să emită o serie de cereri pentru protocolul IP.

Protocolul Internet (IP) realizează schimbul de blocuri de informaţii numite pachete de date. Un pachet este reprezentat de o secvenţă de octeţi şi mai este formată dintr-un antet, urmată de o secţiune de date propriu-zise. Antetul furnizează informaţii despre destinaţia unde va trebui să ajunga pachetul şi, opţional, dacă este nevoie, de informaţii de rutare folosite pentru transmitere, până când acesta ajunge la destinaţia finală. Secţiunea de date conţine informaţia de care au nevoie pachetele, pe care IP trebuie să o transmită.

Din cauza congestiilor din cadrul reţelei, încărcarea traficului, sau alte comportamente imprevizibile ale pachetelor IP,acestea pot fi pierdute, duplicate, sau livrate în altă ordine până când ajung la destinaţie. TCP-ul detectează problemele de aceste tipuri, solicită retransmiterea pachetelor care au fost pierdute, rearanjează pachetele într-o anumită ordine şi ajută la minimizarea traficul ce are loc în reţea, în vederea reducerii apariţiei altor probleme.

Odată ce protocolul TCP la recepţie termină de reasamblat secvenţa de octeţi ce a fost transmisa iniţial, o va păstra mai sus către programul de aplicaţie. Prin urmare, protocolul TCP va ascunde nivelului aplicaţie detaliile legate de transmise specifice nivelului inferior, din reţea.

Protocolul TCP este optimizat, în aşa fel sa livreze exact, realizând o livrare la timp a datelor, şi totodata, TCP înregistrează rareori, întârzieri relativ mari de timp (de ordinul secundelor, ceea ce este foarte mult faţă de cazurile în care întârzierile sunt de ordinul milisecundelor cum ar fi normal), iar intervalul de timp de aşteptare pentru unele mesaje care sosesc într-o altă ordine sau pentru retransmisia mesajelor care au fost pierdute este oarecum mare. Acest lucru nu se potriveşte întotdeauna pentru aplicaţii în timp real, cum ar fi Voice over IP (VoIP). Astfel, pentru acest gen de aplicaţii, pot fi utilizate protocoalele cum ar fi Real-Time Transport Protocol (RTP), ce rulează peste User Datagram Protocol (UDP).

Protocolul TCP este un serviciu de încredere ce garantează transportul unui flux de date trimise de la o gazdă la alta fără multiplicarea sau pierderea de date. Atunci când transferul de pachete de date pe reţeaua respectivă, nu este sigur, o tehnică, care se mai numeşte şi confirmare pozitivă cu retransmitere este utilizată pentru garantarea fiabilităţii transferurilor dintre pachete. Această tehnică deosebit de importantă, constă în faptul ca receptorul răspunde cu un mesaj de confirmare sau ACK (acknowledgement) de fiecare dată când primeşte un pachet de date de la emiţător. Expeditorul va păstra o copie a fiecarui pachet care a fost trimis, aşteptând confirmarea destinatarului înainte de a trimite următorul pachet. Expeditorul va păstra, şi un timer, atunci când pachetul a fost trimis către destinatar, şi va relua retransmiterea pachetului în cazul în care

timer-ul urmează să expire iar confirmarea recepţiei întârzie să aibe loc. Contorul de timp folosit de timer este necesar pentru cazul în care un pachet se va pierde sau este degradat.

TCP se bazeaza pe un set de reguli ce constau mai mult în asocieri cu alte protocoale: pentru protocol, el este totdeauna asociat cu Protocolul Internet şi pentru Protocolul Internet IP, reprezintă o metodă pentru a trimite datele, folosind un "şir de unităţi de mesaj" între computere prin intermediul Internetului. În timp ce IP-ul se va ocupa doar de transmiterea efectivă a datelor, TCP-ul verifică dacă un mesaj a fost distribuit în unităţi individuale de date, numite segmente, şi să ţină evidenţa segmentelor ce au fost transmise, pentru dirijarea eficientă cu ajutorul reţelei. De exemplu, atunci când un fişier HTML va fi trimis de la un server web, stratul software TCP din acel server împarte secvenţa de octeţi al fişierului respectiv, în segmente şi le distribuie în mod individual către stratul software IP (Nivel Internet).

Nivelul Internet mai are rolul şi de a încapsula fiecare segment TCP din reţea într-un pachet IP prin adăugarea unui antet, care va adăuga adresa IP destinaţie printre alte date disponibile. Chiar dacă fiecare dintre pachete au aceeaşi adresă destinaţie, pachetele pot fi rutate prin căi diferite prin intermediul reţelei. În cazul în care programul client pe computerul destinaţie le primeşte, protocolul TCP (Nivelul Transport) reasamblează segmentele individuale asigurând în acelaşi timp ordonarea corecta şi lipsa de erori a acestora, înainte ca pachetele să fie livrate către nivelul aplicaţie sau aplication.

Modul de lucru pentru protocolului TCP implică existenţa a trei faze. În prima fază, conexiunea trebuie să fie stabilită (Connection establishement), urmând apoi un proces în care are loc o confirmare pe baza mai multor paşi. Imediat ce conexiunea este realizată, va urma transferul datelor sau data transfer.

Odată ce transferul datelor s-a încheiat, conexiunea trebuie terminată (Connection termination) în ideea de a închide calea virtuală şi de a elibera resursele hardware/software implicate în proces. [2]

O sursă TCP este considerată ca fiind saturată pentru cazul unei sesiuni FTP sau File Transfer Protocol, unde toate segmentele au dimensiunile maxime posibile, rezultând pachete de mărimi constante. Termenul de segmente înseamnă acelaşi lucru cu termenul pachete. TCP-ul conţine o fereastră de congestie şi o limită pentru „slow-start” sau început lent, ce sunt descrise de variabilele CWND sau congestion window şi SSTHRESH sau slow start threshold (pragul începutului lent).

Congestion window (CWND) are rolul de a determina numărul maxim admisibil de pachete ce sunt neconfirmate în reţeaua respectivă, iar slow start threshold (SSTHRESH) are rolul de a controla creşterea dimensiunii ferestrei de congestie.

Dacă fereastra de congestie este mai mică decât pragul inceputului lent, în faza de „start lent”, fereastra de congestie este incrementată cu câte un pachet pentru fiecare confirmare neduplicată ce a fost primită. Aceast lucru duce la o creştere exponenţială a ferestrei de congestie. Când SSTHRESH este mai mic sau egal cu CWND, adică în faza de evitare a

congestiei, CWND creşte cu câte un pachet la fiecare RTT (round trip time) sau timp dus-întors, ceea ce corespunde unei creşteri liniare a ferestrei de congestie (Figura 2.1). [3]

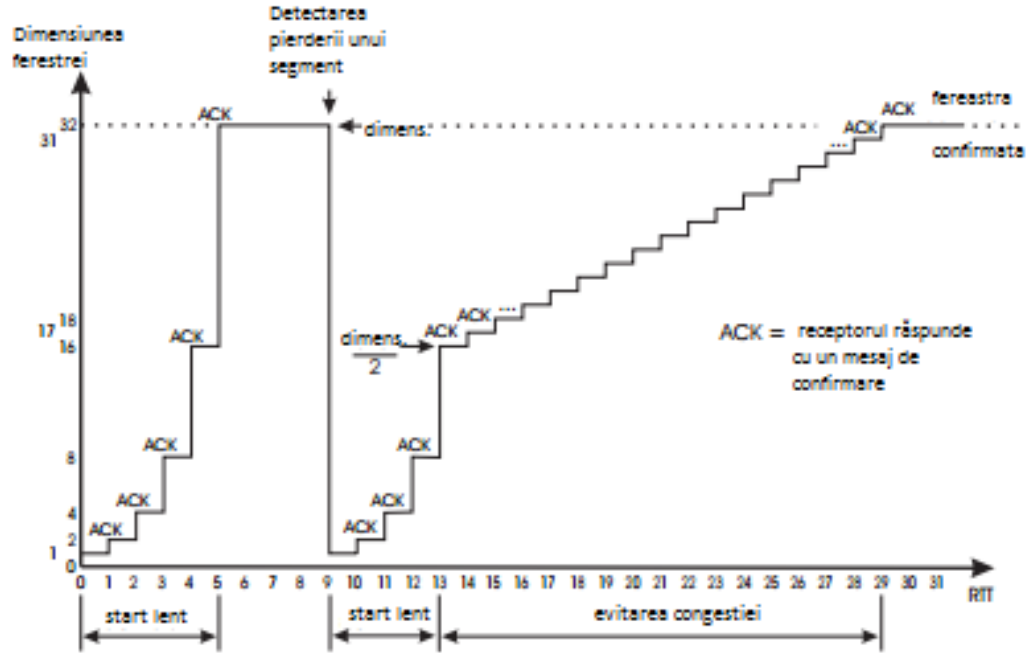


Figura 2.1 : Ilustrare simplificată a startului lent al protocolului TCP pentru algoritmul de evitare a congestiei [3]

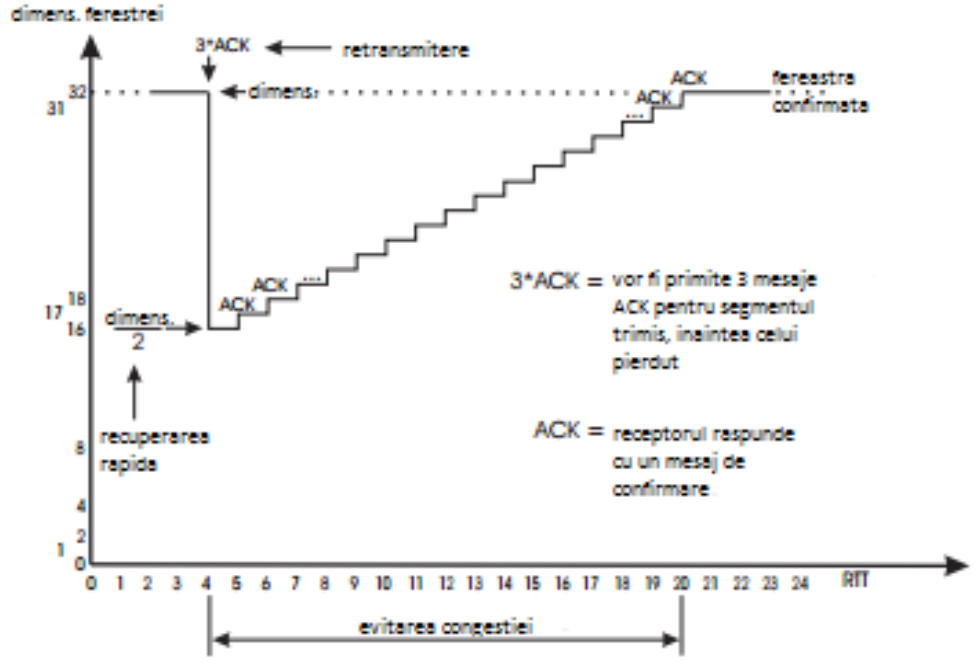


Figura 2.2 Ilustrare simplificată a startului lent al protocolului TCP pentru algoritmul de retransmitere rapida [3]

Protocolul TCP are doua moduri diferite de a reacţiona atunci când au loc pierderi de pachete în reţea. Acesta setează câte un cronometru pentru fiecare pachet trimis. Pentru cazul în care nu va fi primit un mesaj de confirmare pentru acel pachet, înainte de a expira timpul, atunci fereastra de congestie îşi reduce dimensiunea la mărimea maximă a segmentului (sau pachetului) iar pragul începutului lent este setat la valoarea maximă (adica SSTHRESH = flight_size / 2, 2 * dimensiunea segmentului). Flight size este cantitatea de informaţie care a fost neconfirmată în reţea.

Dacă s-ar presupune că emiţătorul este saturat, valoarea ferestrei de congestie ar fi egală cu valoarea cantităţii de informaţie care a fost neconfirmată.

Atunci când emiţătorul recepţionează trei ACK-uri ce sunt duplicate, înainte ca timpul să expire, va fi realizată o recuperare rapidă sau fast recovery, mai precis într-un mod mai simplificat, protocolul TCP va trimite pachetul pierdut din nou, iar valoarea pragului începutului lent se va seta la valoare maximă (flight_size/2, 2 * dimensiunea segmentului) şi astfel se va seta şi valoarea ferestrei de congestie la noua valoare calculată (Figura 2.2) [3].

2.3 Algoritmi TCP pentru evitarea congestiei

Pentru a evita colapsul congestiei (atunci când buffer-ul este plin şi are loc pierderea de pachete în şir), TCP foloseşte o strategie multi-feţe de control al congestiei. Pentru fiecare conexiune, TCP menţine o fereastră de congestie, limitând numărul total de pachete nerecunoscute care pot fi în tranzit end-to-end. Acest lucru este oarecum analog cu fereastră glisantă a TCP-ului folosită pentru controlul fluxului. [4]

Cu alte cuvinte, TCP foloseşte mecanismul numit start lent pentru a mări fereastra de congestie, după ce o conexiune este iniţializată şi după un timeout. Acesta începe cu o fereastră de două ori mai mare decat dimensiunea maximă a segmentului. Deşi rata iniţială este scăzută, rata de creştere este una foarte rapidă: pentru fiecare pachet recunoscut, fereastra de congestie creşte cu lungimea maximă a segmentului, astfel încât fereastra de congestie se dublează într-un mod eficient de fiecare dată (dus-întors).

Când fereastra de congestie depăşeşte pragul de început lent, algoritmul intră într-o stare nouă, numită evitare a congestiei. În unele implementări (de exemplu, Linux), pragul de început lent, iniţial este mare, şi astfel primul start lent, de obicei, se termină după o pierdere. Cu toate acestea, pragul de început lent este actualizat la sfârşitul fiecărui start lent, şi va afecta de multe ori start-urile lente ulterioare declanşate de timeout.

Controlul congestiei face referire de fapt la controlarea traficului ce pătrunde într-o reţea de telecomunicaţii, acest lucru presupunând evitarea supraîncărcării legăturilor dintre nodurile intermediare ale reţelei, numite şi link-uri, prin reducerea pachetelor ce vor fi trimise. Aşa cum a fost prezentat anterior, controlul congestiei nu trebuie sa fie confundat cu controlul fluxului (flow control) ce evită supraîncarcarea receptorului cu date de către emiţător.

Există mai mulţi algoritmi care previn şi în acelaşi timp evită congestia, prevăzuţi de protocolul TCP. Primii algoritmi apăruţi în domeniul telecomunicaţiilor sunt TCP Tahoe, şi TCP Reno.

Ambii algoritmi au la bază regulile de „start lent” şi „retransmitere rapidă” prezentate mai sus. Cei doi algoritmi, Tahoe şi Reno, diferă datorită modului în care reacţionează atunci când au loc pierderi de pachete:

2.3.1 Algoritmul Reno

Reno înjumătăţeşte fereastra de congestie, efectuând o retransmitere rapidă (fast retransmit), şi intrând totodată în recuparare rapidă (fast recovery) pentru cazul în care sunt recepţionate 3 ACK (acknowledged) duplicat.

2.3.2 Algoritmul Tahoe

Atunci când timpul de timeout urmează să expire înainte ca o confirmare să fie recepţionată, pierderea este detectată. Din acel moment, algoritmul Tahoe va reduce fereastra de congestie la valoarea maximă a dimensiunii unui segment sau pachet (MSS – maximum segment size) şi se resetează la faza de „slow-start" sau început lent.

2.3.3 Algoritmul TCP Vegas

În jurul anilor 1990, toate întarzierile dus-intors şi time-outurile erau bazate doar pe ultimul pachet din buffer.

Cercetătorii Lawrence Brakmo si Larry Peterson de la Universitatea din Arizona, au introdus algoritmul TCP Vegas în care întarzierea dus-întors este măsurată pentru fiecare pachet din buffer-ul respectiv şi timeout-ul este setat. Totodată, acest algoritm foloseşte şi o creştere aditivă a CWND. Această variantă de algoritm nu a fost atât de mult improvizată în afara laboratorului cercetătorilor.

2.3.4 Algoritmul TCP New Reno

Acest algoritm vine ca un upgrade pentru algoritmul simplu Reno,oferind o îmbunătăţire a retransmisiei în timpul fazei de recuperare rapidă sau fast recovery pentru TCP Reno. Pe parcursul fast recovery, pentru fiecare acknowledgement duplicat, va fi transmis un nou pachet care nu a fost trimis, de la sfârşitul ferestrei de congestie CWND, pentru a menţine fereastra de congestie plină. Pentru fiecare acknowledgement parţial din spaţiul de secvenţe numerice, emiţătorul va trimite pachetul următor cu o secvenţa numerică următoare acknowledge-ului.

Datorită faptului că acel cronometru de timeout va fi resetat ori de cate ori va există un anumit progres în buffer-ul de transmisie, algoritmul New Reno poate umple găuri oarecum mari sau mai multe găuri din spaţiul de secvenţe.

Datorită faptului ca algoritmul New Reno poate trimite pachete noi pana la sfârşitul ferestrei de congestie CWND în timpul fast recovery, este menţinut un throughput relativ mare pe durata procesului de umplere a găurilor, chiar dacă vor exista mai multe găuri, având mai multe pachete fiecare. Atunci când TCP intră în fast recovery, acesta va înregistra cea mai mare secvenţă numerică neconfirmată. Dacă această secvenţă numerică este confirmată, atunci TCP va reveni la starea de evitare a congestiei.

Dacă nu au loc pierderi de pachete, dar cu toate acestea, pachetele vor fi reordonate cu mai mult de trei secvenţe numerice, va apare o problemă în algoritmul New Reno. Atunci când acest fenomen are loc, New Reno intră greşit în fast recovery, însă când pachetul reordonat va fi livrat, va avea loc progresul secvenţei numerice acknowledged iar din acel moment până la sfârşitul fast recovery, fiecare bit din secvenţa numerică se va duplica şi va oferi retransmisia necesară.

2.3.5 Algoritmul TCP Hybla

Are rolul de eliminare a penalizării conexiunilor TCP care încorporează legături terestre cu latenţă de dimensiuni foarte mari, sau de legături radio, datorită timpilor de dus-întors care deţin

valori relativ mari. Algoritmul este provenit dintr-o evaluare analitică a dinamicii ferestrei de congestie CWND, care recomandă înlăturarea dependenţei performanţei faţă de timpul dus-întors (RTT – round trip time).

2.3.6 Algoritmul TCP BIC (binary increase congestion control)

Acest algoritm este o implementare a TCP-ului cu un algoritm de evitare a congestiei, optimizat pentru reţelele de viteze mari, având latenţe mari (numite long fat networkssau LFN). Algoritmul BIC este folosit decât în kernel-urile Linux, de la 2.6.8 pana la 2.6.18.

2.3.7 Algoritmul TCP Cubic

Algoritmul prezintă o versiune mai puţin agresivă a algoritmului BIC, în care fereastra de congestie este o funcţie cubică de timp, de la momentul ultimei congestii, având punctul de inflexiune setat la fereastra anterioară a evenimentului respectiv.

2.3.8 Algoritmul de prevenire a congestiei DECbit

Pentru cazul folosirii algoritmului DECbit, gateway-ul utilizează un bit în antetul pachetului pentru a furniza un mesaj de feedback reţelei legat de apariţia congestiei. În momentul în care un pachet ajunge la gateway, aceasta va calcula lungimea medie a cozii pentru ultima perioadă (adica ocupat+inactiv) plus perioada de "ocupat" din momentul respectiv (gateway-ul este ocupat atunci când sunt transmise pachete, daca nu, el va fi inactiv). [5]

Atunci când lungimea medie a cozii depăşeşte valoarea de unu, gateway-ul va seta bitul de indicare a congestiei în antetul pachetelor ce urmează să sosească.

Sunt folosite ferestre pentru controlul fluxului de catre sursă, pe care le updatează o dată pentru fiecare două perioade dus-întors. Pentru cazul in care cel puţin jumătate dintre pachetele din ultima fereastră ar fi avut bitul de identificare al congestiei setat, atunci fereastra ar fi fost scăzută exponenţial. În mod contrar, fereastra este crescută liniar.

2.4 Diferenţe între gateway-urile DECbit şi RED

Între gateway-urile care au implementat algoritmul DECbit şi gateway-urile care au implementat algoritmul RED sunt câteva diferenţe semnificative [5]:

-Prima diferenţă face referire la metoda de calcul a dimensiunii medii a cozii. Din cauza faptului că DECbit foloseşte ultimul ciclu (ocupat+inactiv) plus perioada de "ocupat" curentă pentru a calculul dimensiunii medie a cozii, dimensiunea cozii va putea fi mediată după o perioadă de timp destul de scurtă, ceea ce reprezintă un avantaj.

Pentru reţele de viteză mare, cu buffere foarte mari la nivel de gateway, se doreşte ca utilizarea constantei de timp folosite pentru calculul dimensiunii medii a cozii, să fie făcută explicit. Acest lucru nu este posibil pentru algoritmul DECbit, însă este realizat în cazul gateway-urilor RED. În reţele DECbit, sursele urmaresc, astfel monitorizând fracţiunea de pachete ce au fost marcate în ultimul interval, adica cel de dus-întors. Pentru cazul reţelelor cu gateway RED, sursa îşi va reduce fereastra chiar daca există un singur pachet care a fost marcat.

-Cea de-a doua diferenţă dintre cei doi algoritmi, este metoda prin care gateway-ul îşi alege conexiunile pe care le va înştiinţa despre apariţia congestiei ce va urma. La DECbit nu este

prezentată nici o diferenţă conceptuală dintre algoritmul pentru setarea bitului de congestie şi algoritmul pentru detecţia congestiei.

Dacă un pachet ajunge în gateway şi dimensiunea medie a cozii este mult prea mare, atunci bitul pentru identificarea congestiei este setat în antetul pachetului respectiv. Datorită acestei metode de marcare a pachetelor, reţele care folosesc algoritmul DECbit pot întâlni probleme în faţa traficului fluent, acest lucru fiind evitat pentru cazul gateway-urilor RED, aici marcarea pachetelor făcându-se într-un mod aleator.

Pentru gateway-urile care evită congestia în reţele, destinate pentru folosirea protocolului TCP, motivaţia în plus pentru ca marcarea să se facă aleator este evitarea sincronizării globale care poate să rezulte în urma unei reduceri în acelaşi timp a mai multor conexiuni TCP şi a ferestrei de comunicaţie. Aspectul acesta nu este important decât pentru cazul folosirii algoritmului DECbit, datorită faptului că fiecare sursă îşi micşorează moderat fereastra ca răspuns la apariţia congestiei în reţea.

CAPITOLUL 3
Random Early Drop

3.1 Introducere

Algoritmul RED (Random Early Detection), cunoscut şi ca Random Early Drop, Random Early Discard, sau chiar Random Early Delete este un mecanism de prevenire şi în acelaşi timp de evitare a congestiei. [6]

Acest algoritm deţine câteva metode care folosesc pentru alegerea conexiunii care să notifice această congestie şi pentru detectarea congestiei. Ruterele folosite de algoritmul RED sunt utilizate în cea mai mare parte cu reţele TCP/IP, fiind proiectate pentru reţele ăn care un singur pachet aruncat sau marcat este de ajuns pentru a semnala congestia la nivel de transport. [5]
Sally Floyd şi Van Jacobson sunt inventatorii acestui algoritm. Ei au venit în anul 1993 cu o lucrarea despre "detecţia congestiei în reţele folosind ruterele, folosind un mecanism de detecţie aleatoare timpurie".

Algoritmul Random Early Detection (RED) reduce congestia în cozi aruncând pachetele astfel încât unele conexiuni TCP pe o anumită perioadă limitată de timp vor transmite mai puţine pachete prin reţea. RED în mod intenţionat va arunca o parte din pachete înainte ca bufferul se umple total, în loc să aştepte până când o coadă se supraîncarcă, cauzând un număr mare de pachete aruncate. Această acţiune este destinată să facă emiţătorii care generează traficul să reducă volumul de pachete transmis. Se poate spune că acesta este avantajul de baza al utilizării acestui algoritm. [7]

Numele acestui algoritm Random Early Detection sau detecţie aleatoare timpurie descrie modul de utilizare al algoritmului. Atunci când a luat decizia de descărcare de pachete, algoritmul RED selectează într-un mod aleator pachete pe care le descarcă. [5]

Există două aspecte generale care descriu logica de execuţie a acestui algoritm:
- RED va trebui în primul rând să detecteze congestia propriu-zisă înainte ca aceasta să se întâmple, sau mai bine spus înainte ca buffer-ul să se umple. Altfel spus, RED trebuie să ia decizia pentru alegerea condiţiilor ce duc la aruncarea de pachete.

Atunci când se ia această decizie, va trebui să fie bine definit numărul de pachete care va trebui aruncat. În primul rând RED va trebui să verifice cât de mare este coada dispozitivului.
- algoritmul RED calculează lungimea medie pentru interfaţa respectivă, urmând apoi să decidă dacă va avea loc fenomenul de congestie sau nu. RED foloseşte încărcarea medie pentru că încărcarea reală îşi schimbă valoarea mult mai des. Datorită faptului că RED evită efectele sincronizării dintre pachete, este util ca aruncarea pachetelor să se faca într-un mod proporţional în funcţie de cât de încărcat este buffer-ul, evitând astfel aruncarea pachetelor de număr extrem de mare la un moment dat şi extrem de mic într-un alt moment.

Avantajul acestui algoritm de control al congestiei la nivelul de transport de pachete este că acesta funcţionează cu actualele protocoale de transport (TCP/IP), şi în al doilea rând nu este nevoie ca toate ruterele din reţea să folosească acelaşi mecanism de control al congestiei. Mecanismul acesta este usor din punct de vedere al implementarii, de evitare a congestiei poate fi folosit în actualele reţele TCP/IP nefiind nevoie de schimbarea protocolului de transport.

Prevenirea apariţiei congestiei într-o reţea cu ajutorul controlului dimensiunii medii a cozii pachetetelor respective reprezintă scopul principal pe care ruterele RED vor trebui să îl îndeplinească.

Algoritmul RED este cel care calculează dimensiunea medie a cozii, utilizând un filtru trece-jos, acesta având o medie ponderată exponenţială, dinamică sau mobilă.

Putem compara aceasta dimensiune medie a cozii folosind două praguri şi anume un prag care este minim (minimum thresh) şi un prag care este maxim (maximum thresh). În acest caz al folosirii pragurilor, când dimensiunea medie a cozii deţine o valoare mai mică decât minimum thresh, nu este marcat nici un pachet. Dacă în cazul în care dimensiunea medie a cozii deţine o valoare mai mare decât maximum thresh, atunci oricare dintre pachetele care sosesc va fi marcat. Atunci când pachetele care au fost marcate sunt de fapt aruncate, sau cu alte cuvinte dacă toate nodurile principale cooperează între ele, aceasta metodă poate să asigure faptul că dimensiunea medie a cozii să nu poata să deţină o valoare mult mai mare decât maximum thresh.

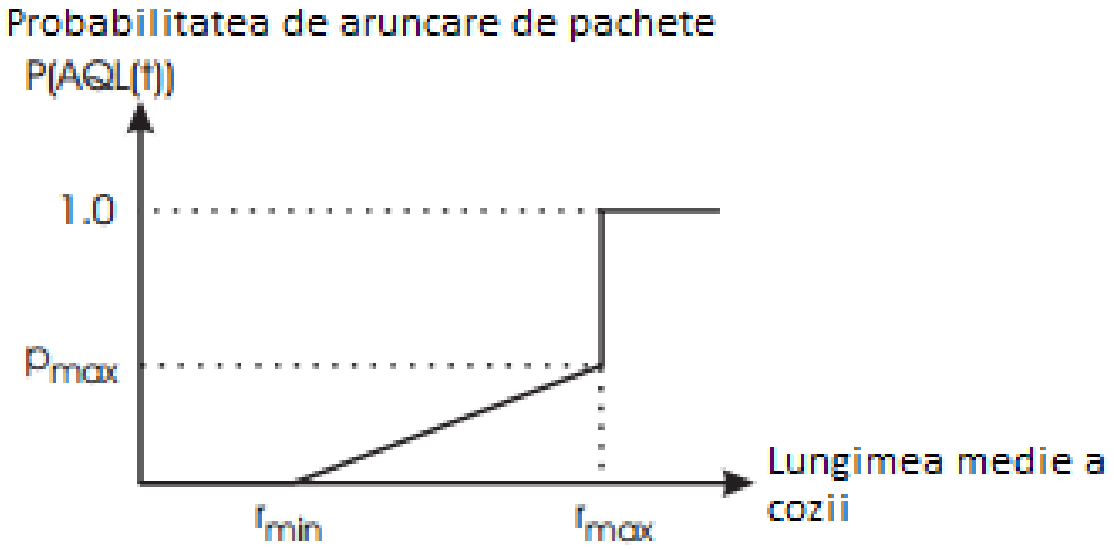


Figura 3.1: Creşterea probabilităţii de aruncare de pachete folosită de algoritmul RED [3]

Dacă dimensiunea medie a cozii se situează între valoarea de minimum thresh şi maximum thresh, atunci fiecare dintre pachetele ce ajung la destinaţie vor fi marcate cu probabilitatea *Pa,* acest parametru reprezentând o funcţie ce este dependentă de dimensiunea medie a cozii *avg*.

Ori de câte ori un pachet este marcat, atunci probabilitatea ca pachetul respectiv să facă parte dintr-o conexiune anume, va fi proporţională cu rata pe care conexiunea respectivă o are din lungimea totală de bandă care trece prin gateway. [5]

În cele ce urmează va fi prezentat algoritmul general RED pentru gateway-uri:

Pentru fiecare pachet care a sosit
Se va calcula dimensiunea medie a cozii (*avg)*
Dacă th_max ≤ avg ≤ th_min
Calculează probabilitatea P_a

Marchează pachetul care a sosit cu probabilitatea P_a

Altfel dacă th_max ≤ avg

Marchează pachetul care a sosit

Putem spune ca acest algoritm RED pentru gateway-uri este format din 2 algoritmi separaţi unul faţă de celălalt:

- Gradul maxim de trafic de pachete sau mai bine spus de pachete care sosesc simultan, care va fi permis în coada gateway-ului este determinat de algoritmul care are rolul de a calcula dimensiunea medie a cozii.
- La un anumit nivel al congestiei algoritmul ce calculează probabilitatea ca un pachet sa fie marcat, va determina cât de des gateway-ul poate să marcheaze pachetele . Rolul este ca gateway-ul să poată marca pachetele sosite în perioade de timp egale, cu ajutorul acestui lucru reuşind să evite marcarea pachetelor îndeajuns de des şi sincronizarea globală, putând astfel să obţină controlul dimensiunii medie a cozii.

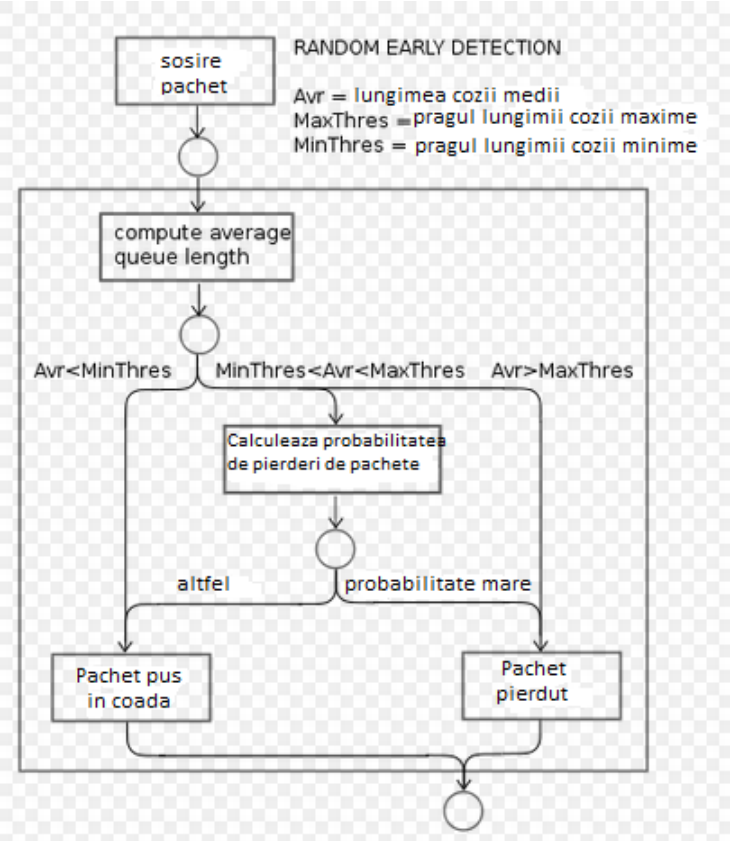


Figura 3.2: Schema algoritmului RED [6]

Intervalul de timp cât coada are valoare nulă (perioadă inactivă) foloseşte pentru a calcula dimensiunea medie a cozii, totodată estimând numărul de pachete m ce ar fi putut fi transmise de către gateway tot pe durata acestui interval. După perioada inactivă sau idle, gateway-ul calculează dimensiunea medie a cozii presupunând ca m pachete ar fi ajuns pe durata acestei perioade idle într-o coadă nulă. [5]

Valoarea cozii medii variază în funcţie de cum sunt alese pragurile de minim (minimum thresh) şi de maxim (maximum thresh), iar probabilitatea cu care pachetele sunt marcate (P_b) variază între valoarea nulă (0) şi o valoare maximă a probabilităţii (max_p):

$$P_b = \max\_p * \frac{\text{avg} - \text{th}_{\min}}{\text{th}_{\max} - \text{th}_{\min}}$$

P_a, care este probabilitatea finală care marchează pachetele poate creşte treptat, pe măsură ce numărătoarea pachetelor creşte în raport cu pachetul care a fost marcat cel mai devreme:

$$P_a = \frac{P_b}{1 - \text{count} * P_b}$$

Folosind aceasta modalitate, putem garanta că gateway-ul va aştepta puţin timp pentru marcarea unui pachet nou.

În momentul în care dimensiunea medie a cozii va depăşi valoarea pragului maxim, atunci gateway-ul marchează fiecare pachet care ajunge în coadă .

Mai exista o altă modalitate pentru algoritmul RED de măsurarea a cozii în biţi, renunţând astfel la măsurarea cozii în pachete. Totodata, avg sau dimensiunea medie a cozii va indica într-un mod precis valoarea întârzii medie care apare în gateway. Algoritmul va fi modificat pentru a asigura că probabilitatea cu care un pachet va fi marcat este proporţională cu dimensiunea pachetului în bytes, acest lucru întâmplându-se doar pentru cazul în care va fi folosită acea modalitate.

$$P_b = \max\_p * \frac{\text{avg} - \text{th}_{\min}}{\text{th}_{\max} - \text{th}_{\min}}$$

$$P_a = \frac{P_b}{1 - \text{count} * P_b}$$

$$P_b = P_b * \frac{\text{dimensiune\_pachet}}{\text{dimensiune\_maxima\_pachet}}$$

Iniţializare:

avg ← 0

count ← -1

Pentru fiecare pachet ce a fost utilizat

Calculează dimensiunea nouă a cozii medie avg:

if avg =! 0

avg←$(1-w_q)$*avg +w_q*q

else

m←f_time – q_time

avg←1-w_q^m *avg

if $th_{min} \leq avg < th_{max}$

count va fi incrementat

Probabilitatea P_a va fi calculată

$P_b = \max\_p * \frac{avg - th_{min}}{th_{max} - th_{min}}$

$P_a = \frac{P_b}{1 - count * P_b}$

Pachetul care a ajuns la destinaţie va fi marcat cu probabilitatea P_a

count ← 0

else $th_{max} \leq avg$

Pachetul care a sosit este marcat:

count← 0

else count ← -1

Dacă nu mai sunt pachete în coadă:

q_time ← time

Aceasta este o variantă mai detaliată a algoritmului RED [5] .Totodată putem observa că în acest mod, ca un pachet FTP (File Transfer Protocol) faţă de un pachet TELNET va putea avea şanse mult mai mari să fie marcat.

Variabilele care au fost salvate:

q_time: intervalul de timp al începerii perioadei inactiv (idle)
avg: dimensiune medie a cozii
count: pachetele provenite de la ultimul pachet care a fost marcat

Parametrii fixaţi:

th_{min}: pragul de minim al dimensiunii cozii
th_{max}: pragul de maxim al dimensiunii cozii
max_p: valoarea maximă pe care o poate avea P_b
w_q: ponderea cozii

Alţi parametrii:

q: mărimea curentă a cozii
p_a: probabilitatea curentă de marcare a pachetelor
f(t): funcţie liniară de timp t
time: timp curent

În functie de de mărimea traficului pachetelor din coadă ce este permis de către gateway şi pentru cazul în care există un trafic agresiv, şi de durata acestuia, se poate determina ponderea cozii (w_q). Dimensiunea medie a cozii dorite este cea care determină valorile pentru pragurile de minim al dimensiunii cozii şi de maxim al dimensiunii cozii. Dimensiunea medie a cozii care poate stabili

compromisuri dorite (de exemplu cum este compromisul dintre minimizarea intervalului de timp al întârzierii şi maximizarea throughput-ului) este dependent de atribuţiile reţelei.

3.2 Identificarea unui user ce are comportament necorespunzător

Detecţia congestiei cu ajutorul marcării de pachete este realizată cu ajutorul routerelor RED care pot notifica toate conexiunile dintre noduri. Există un anumit procent din lărgimea benzii pe care il foloseşte conexiunea respectivă în momentul respectiv, care este direct proporţional cu probabilitatea ca un pachet dintr-o anumită conexiune să fie marcat, spre deosebire de Tail Drop unde nu sunt respectate aceste reguli.

Routerele care folosesc algoritmul RED pot oferi o implementare eficientă pentru a se putea realiza identificarea legăturilor dintre noduri care folosesc un procentaj mare din lărgimea de bandă în momentul când se produce congestia. Aceste routere au rolul de a identifica foarte usor care din legăturile disponibile în reţea a primit o mare parte din ultimele pachetele ce au fost marcate, din cauza faptului că ruterele RED vor alege într-un mod random pachetele ce vor urma să fie marcate în momentul în care are loc congestia.

Din momentul în care numărul de pachete ce au fost marcate va fi îndeajuns de mare pentru acea legatură, atunci conexiunea care a acceptat un procent mare de pachete marcate va fi cel mai posibil să fie conexiunea ce a utilizat cel mai mult lungimea de bandă disponibilă. Putem spune ca această indicaţie ar putea fi folosită de nivele foarte avansate cu scopul restricţionării lărgimii de bandă a legăturilor în timpul în care congestia are loc.

Totodată, ruterele RED au posibilitatea păstrarii cu uşurinţă a unei liste ce conţine cele n pachete dintre cele care au fost marcate recent. Pentru cazul în care o parte foarte mare din pachetele ce au fost marcate fac parte dintr-o conexiune anume, va fi cel mai probabil ca acea legatură să aibă un procentaj mare din lărgimea medie a benzii.

În momentul în care unele dintre conexiunile TCP vor primi procente relativ mari din lărgimea de bandă, atunci este mult posibil ca acele conexiuni să fie surse ce nu respectă protocoalele TCP actuale, ori mai poate exista o conexiune ce are un interval de dus-întors mai scurt sau o fereastră mult mai mare decât cele folosite pentru alte conexiuni.

Urmărind fiecare dintre cazuri, pentru toate, ruterul folosit de algoritmul RED are posibilitatea de a fi programat astfel încât să poată oferi o prioritate mai mică acelor conexiuni care acceptă un procent mai mare din lărgimea de bandă în intervalul de timp al producerii congestiei în retea.

3.3 Calculul probabilităţii ca pachetele să fie marcate

Probabilitatea ca pachetele sa fie marcate P_b poate fi calculată ca o funcţie liniară de către lărgimea cozii medie. În cele ce vor urma vor fi comparate două metode de calcul al probabilităţii finale cu care pachetele sunt marcate cât şi avantajele folosirii metodei a doua.

Pentru cazul folosirii primei metode, atunci când largimea medie a cozii are valori constante, numărul de pachete ajunse la destinaţie între 2 pachete ce au fost marcate,va fi de fapt o variabilă geometrică aleatoare.

Pentru cazul utilizarii celei de-a doua metodă, numărul de pachete care au ajuns la destinaţie între 2 pachete marcate va fi o variabilă aleatoare uniformă.

Probabilitatea iniţială de marcare a pachetelor este calculată cu ajutorul relaţiei

$$P_b = \max\_p * \frac{avg - th_{min}}{th_{max} - th_{min}}$$

Unde max_p este valoarea maximă pentru probabilitatea cu care pachetele sunt marcate (P_b) când largimea medie a cozii ajunge la th_{max}.

3.3.1 Metoda 1 de calcul: Variabilă aleatoare geometrică

Dacă s-ar face o presupunere cum că fiecare pachet va fi marcat cu o probabilitate Pb considerând că timpul dintre marcări (notat X) fiind dat de numărul de pachete care ajung la destinaţie după ce un pachet a fost marcat, până când urmatorul pachet este marcat. Cu toate acestea fiecare dintre pachete vor fi marcate cu probabilitatea P_b:

Prob X = n = $1 - P_b^{n-1} * P_b$

Scopul acestei metode este ca pachetele să fie marcate la intervale îndeajuns de regulate folosind o lungime medie a cozii constantă . Nu este necesar ca intervalele de timp dintre marcările ce au avut loc să fie foarte mari si totodata nu este de dorit să avem atât de multe pachete marcate foarte apropiate. Cele doua variante pot avea loc atunci când X este o variabilă aleatoare geometrică, iar totodată cele doua variante enumerate pot duce la sincronizări globale, având mai multe conexiuni, astfel reducând ferestrele în acelaşi interval de timp.

Parametrul X este o variabilă aleatoare geometrică, având la rândul sau ca parametrii Pb şi E[X]=$\frac{1}{P_b}$.

3.3.2 Metoda 2 de calcul: Variabilă aleatoare uniformă

Folosind aceasta metodă care este o variantă mai bună, X fiind o variabilă aleatoare uniformă din mulţimea{1, 2, ...,$\frac{1}{P_b}$} , presupunând că $\frac{1}{P_b}$ este un întreg.

X va fi o variabilă aleatoare uniformă doar atunci când probabilitatea ca oricare din pachetele ce au sosit să fie marcate să fie $\frac{P_b}{1-count*P_b}$, count fiind numărul de pachete ce nu au fost marcate care au ajuns la destinaţie de când ultimul pachet a fost marcat.

Pentru aceasta situaţie:

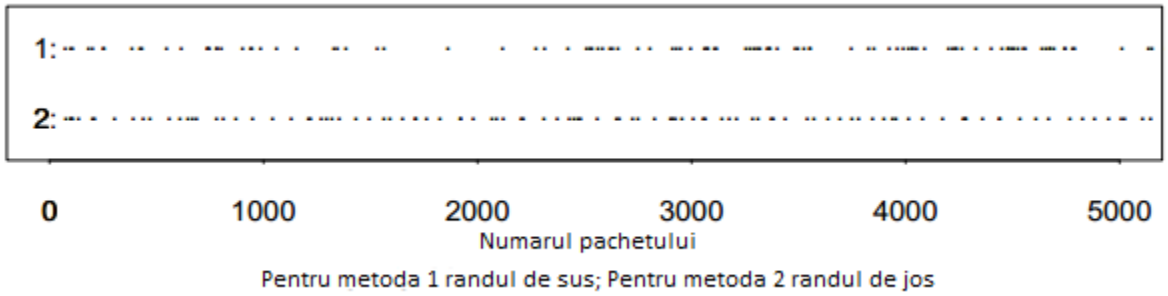
Prob X = n = $\frac{P_b}{1-(n-1)*P\_b} * \prod_{i=0}^{n-2}(1 - \frac{P_b}{1-i*P_b})$

Prob X = P_b pentru $1 \leq n \leq \frac{1}{P_b}$

Prob X = 0 oentru n > $\frac{1}{P_b}$

Pentru metoda a doua $E[X] = \frac{1}{2*P_b} + \frac{1}{2}$

Va fi prezentat un experiment in figura ce va urma in care vor fi comparate cele două metode unde pachetele sunt marcate:



Pentru metoda 1 randul de sus; Pentru metoda 2 randul de jos

Figura 3.3: Comparaţie între metodele de marcare (aleator) [5]

Pentru linia de sus ce reprezintă prima metodă, oricare pachet va fi marcat cu o probabilitate P, unde P ia valoarea 0.02. Pentru linia de jos ce reprezintă a doua metodă, oricare pachet va fi marcat cu o probabilitatea $\frac{P}{1+i*P}$, unde P va lua valoarea 0.01 iar numărul de pachete nemarcate i de la ultimul pachet care a fost marcat.

Fiecare din cele doua metode au marcat aproximativ 100 de pachete din 5000 care au sosit la destinaţie. Pe axa x este reprezentat numărul de pachete, iar pentru ambele metode este reprezentat pe figură câte un punct pentru fiecare pachet ce a fost marcat. Totodată, pachetele marcate sunt mult mai grupate pentru prima metodă, decât pentru cazul folosirii metodei a doua.

3.4 Calculul dimensiunii medie a cozii

Se va folosi un filtru trece jos pentru a calcula dimensiunea medie a cozii. Cu toate acestea, creşterile pe durată scurtă ale lărgimii cozii datorate traficului agresiv nu duc la creşterea foarte mare a dimensiunii medie a cozii.
Mai putem spune că filtrul trece jos are rolul de medie mobila exponenţială ponderată:

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

Constanta funcţie de timp a filtrului trece jos este determinată de către ponderea wq. În cele ce vor urma se va discuta despre limitele posibile pentru ponderea w_q .

3.5 Alegerea pragurilor minth şi maxth

Pragurile de minim şi de maxim au valorile optime care depind în principal de avg, lungimea medie a cozii. Pentru cazul în care traficul este unul agresiv, atunci valoarea minima a pragului

va trebui să fie suficient de mare pentru a putea permite menţinerea folosirii unei legături la un nivel îndeajuns de ridicat. Pentru cazul în care traficul este unul oarecum normal, cu întârzieri destul de mari, atunci utilizarea unui prag minim pachete ar putea duce la o legătură foarte proastă. [5]

Cea mai buna valoare pentru pragul maxim depinde în cea mai mare parte de întârzierea medie maximă permisă de gateway.

Când th_{min} - th_{max} este mai mare decât creşterea lungimii medie a cozii într-o anumită perioadă de timp, atunci algoritmul RED va da un randament maxim. În majoritatea cazurilor, pragul maxim este cel puţin de 2 ori mai mare decât pragul minim.

3.6 Alegerea valorilor pentru pondere

Metoda care realizează medierea nu va reuşi să filtreze congestia în urma tranzitului de pachete de la nivelul gateway-ului pentru valori ale ponderii w_q prea mari.

Astfel, se va presupune că în faza iniţială coada va fi goală, cu o largime medie a cozii de valoare nulă, urmând apoi o creştere considerabilă a dimensiunii cozii de la 0 la L pachete, după ce L pachete au ajuns la destinaţie. După ce ultimul pachet a sosit, adica cel numerotat cu litera L, atunci lungimea medie a cozii (avg) va fi [5]:

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

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

$$avg_L = L+1+\frac{(1-w_q)^{L+1}-1}{w_q}$$

Lungimea medie a cozii avgL pentru anumite valori ale ponderii w_q şi L vor fi prezentate în figura ce va urma . Pe axa x va fi reprezentată ponderea wq, care ia valori în intervalul 0.001 și 0.005.

Pe axa y va fi reprezentat L care ia valori cuprinse în intervalul 10 şi 100. Pentru w_q = 0.001, de exemplu, lungimea medie a cozii avg100 va fi de 4.88 de pachete , după o creştere a cozii de la 0 până la 100 de pachete.

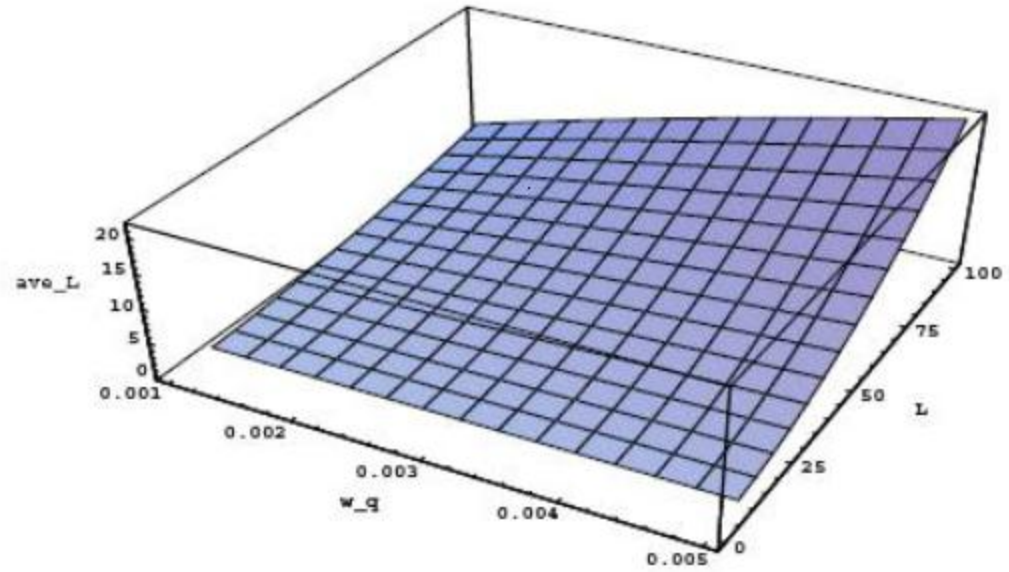


Figura 3.4: Lungimea medie a cozii în funcţie de L si de w_q [5]

Avem de asemenea folosit un prag minim th_{min} şi se ştie că se doreste permiterea traficului agresiv utilizându-se astfel L pachete, atunci va rezulta că ponderea va trebui aleasă în aşa fel încât ecuaţia scrisă mai jos să fie satisfacută pentru ca lungimea medie a cozii să fie mai mică decât pragul de minim al dimensiunii cozii:

$$L+1+\frac{1-w_q{}^{L+1}-1}{w_q} < th_{min}$$

Putem exemplifica pentru L=50, iar th_{min}=5 rezultă că ponderea va trebui aleasă mai mică sau egală cu valoarea de 0,0042.

Valoarea calculată a dimensiunii medie a cozii (avg) cu valori mai mici decât un anume prag va trebui să fie menţinută de către gateway-urile care au implementat algoritmul RED. Totodată, acest lucru nu va avea nici un sens pentru cazul in care dimensiunea medie a cozii calculată (avg) nu va avea valori oarecum asemănătoare cu valorile lungimii medii curente. Pentru cazul în care ponderea este setată la un nivel scăzut, atunci dimensiunea cozii medii va răspunde foarte încet la schimbările lungimii cozii ce au fost făcute. Aşadar, gateway-ul nu va putea să observe apariţia congestiei.

Dacă s-ar presupune trecerea cozii de la nici un pachet la un pachet, şi că în timp ce pachetele sosesc şi se întorc având aceeaşi rată, atunci coada va rămâne la un pachet. Totodată dacă am presupune că în faza iniţială lungimea medie a cozii era nulă, în cazul acesta, vor fi

nevoie de sosiri de pachete într-un număr egal cu $\frac{-1}{\ln(1-w_q)}$, cu lungimea cozii ce va rămane la valoarea 1, până în momentul când aceasta va deveni 0,63 = 1- $\frac{1}{e}$.

De exemplu pentru o valoare a ponderii w_q de 0,001 vor fi necesare 1000 de sosiri de pachete. Dacă am dubla valoarea ponderii, adica la 0,002 vor fi utile 500 de sosiri de pachete, iar pentru valoarea ponderii de 0,003 vor fi utile 333 sosiri de pachete.

3.7 Moduri de evaluare ale algoritmului RED

Există mai multe moduri prin care algoritmul RED poate fi caracterizat, cum ar fi [5]:

Evitarea congestiei

RED face ca dimensiunea medie a cozii ce a fost calculată să nu depăşeasca pragul maxim th_{max} pentru cazul in care pachetele sunt aruncate in momentul in care ajung, atunci când lungimea maximă a cozii va atinge pragul maxim.

Când valoarea ponderii w_q va fi setată într-un mod corespunzător, atunci RED va putea să controleze şi largimea medie a cozii curente.

Scalele de timp potrivite

Intervalul de timp suficient pentru ca gateway-ul să poată observa o scădere a ratei de primire de pachete va fi de mai mult de o perioadă dus şi întors după ce conexiunea va fi înştiinţată despre apariţia congestiei prin marcarea unui pachet anume.

Pentru cazul gateway-urilor folosite de algoritmul RED, scalele de timp folosite pentru observarea congestiei sunt aproximativ echivalente cu scalele de timp utilizate de conexiuni pentru a da un răspuns la congestie. Gateway-urile RED au rolul de a înştiinţa conexiunile pentru a-şi reduce ferastra.

Inexistenţa unei sincronizari globale

Nivelul de congestie este cel de care va depinde rata la care algoritmul RED va realiza marcarea pachetelor. În momentul in care are loc o congestie scăzută, cu alte cuvinte, pentru puţine pachete în coadă, probabilitatea de marcare a oricarui pachet din acea coadă va fi foarte mică şi va creşte direct proporţional cu creşterea congestiei din reţea.

Algoritmul RED are rolul de a evita sincronizarea globală folosind marcarea de pachete şi o rată de marcare a pachetelor cât se poate de mică.

Simplitatea

Algoritmul RED pentru gateway de marcare a pachetelor mai poate fi implementat utilizând un overhead moderat în reţele actuale.

Maximizarea puterii globale

Algoritmul RED poate controla într-un mod explicit dimensiunea medie a cozii, iar totodată se mai poate vedea că puterea globală este mai mare decât pentru cazul gateway-urilor de tip Drop Tail.

Pentru noile cercetari ce vor urma va fi nevoie să se determine valoarea optimă a dimensiunii medie a cozii pentru diferite reţele şi pentru diferite situaţii ale traficului.

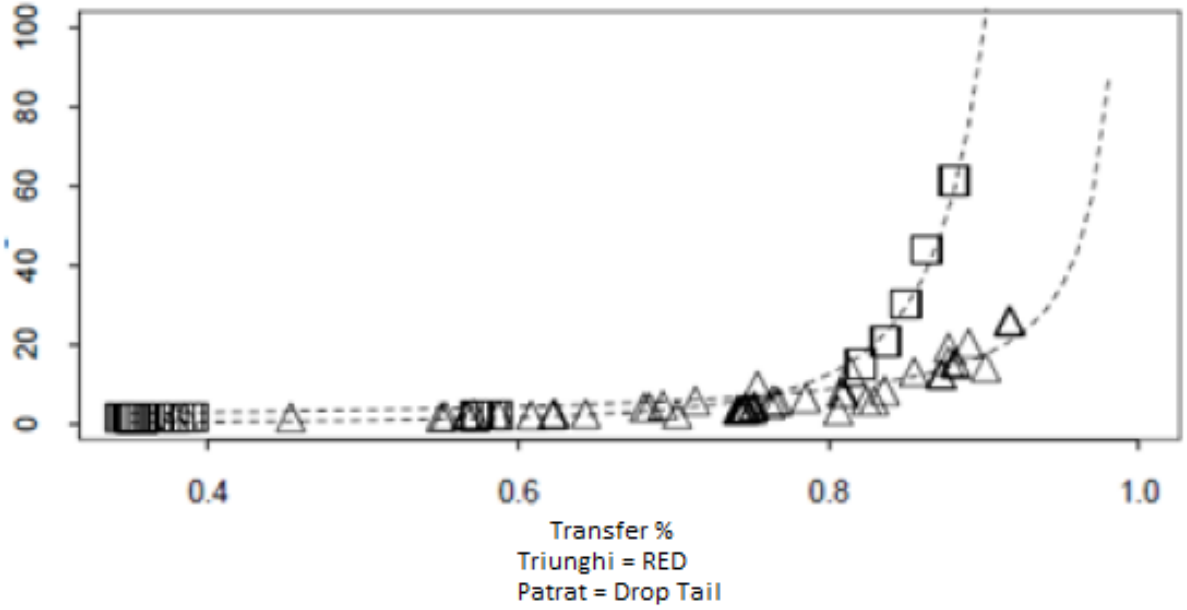


Figura 3.5: Diferenţe între transferul făcut de către gateway-urile RED şi Drop Tail [5]

Corectitudinea

Corectitudinea reprezintă un scop al mecanismului folosit pentru a evita congestia, însa ea nu este bine definită. Routerele care folosesc algoritmul RED nu pot face diferenţa între conexiunile private sau alte clase de conexiuni.

Fracţia de pachete marcate folosită la algoritmul RED pentru oricare conexiune este direct proporţională cu intervalul de timp când conexiunea respectivă ţine banda ocupată.

Potrivit pentru o gamă largă de medii

Procedeul aleator de a marca pachetele este utilizat pentru reţele unde conexiunile sunt caracterizate folosind o gamă foarte largă interval de timp (rountrip sau troughtput – transfer); Totodată mai este necesar acest procedeu pentru un număr foarte mare de conexiuni active în acelaşi timp. Cu ajutorul modificărilor lungimii medii a cozii pot fi detectate schimbări ale încărcării reţelei, rata de marcare a pachetelor fiind minimizată corespunzător.

Pentru cazul folosirii unei reţele în care ruterul RED detectează congestia aruncând pachetele marcate, astfel vor fi mai multe cazuri în care pierderea unui pachet nu va produce o descreştere a încărcăturii de pachete a routerului. Aruncarea unui pachet ce a aparţinut unei conexiuni TCP va fi usor semnalată de către sursă atunci când routerul pierde pachetul respectiv şi cel mai posibil după ce timpul de retransmisie expiră.

Pierderea unui pachet ar putea avea loc fără să fie observată de către sursă pentru cazul în care ruterul aruncă un pachet ACK sau acknowledged aparţinând unei conexiuni TCP sau non-TCP.

Cu toate acestea şi chiar pentru cazul utilizprii unei reţele congestionate care este caracterizată de trafic alcătuit din conexiuni TCP sau non-TCP, routerul va avea control asupra

lungimii medie a cozii deoarece va arunca toate pachetele pe care le primeşte din momentul în care pragul de maxim este depăşit.

3.8 Senzitivitatea parametrilor

Ruterele folosite de algoritmul RED conţin parametri suplimentari şi opţionali în acelaşi timp, care au rolul de a determina pragul superior sau th_{min} pentru lungimea medie a cozii, totodată determinând intervalul de timp pe care se va calcula această lungime şi frecvenţa maximă cu care se vor marca pachetele. [5] [8]

Tot pentru utilizarea ruterelor RED, parametrii pondere (wq) pragul de minim (th_{min}) şi pragul de maxim (th_{max}) sunt utili pentru ca acea persoana care realizează reţeaua să ia cele mai bune decizii în legătură cu lungimea medie a cozii dorită şi totodată în legătură cu dimensiunea şi durata traficului agresiv care va fi acceptat de către buffer.

Deoarece reprezintă limita superioară pentru probabilitatea de marcare a pachetelor P_b, parametrul max_p poate să ia valori dintr-o gamă suficient de largă. Atunci când nivelul congestiei este unul foarte ridicat, astfel încât routerul nu va putea să deţină controlul lungimii medie a cozii cu ajutorul marcării la maxim unei fracţiuni de max_p din pachete, va rezulta că lungimea medie a cozii trece peste pragul de maximum, ruterul urmând să marcheze toate pachetele până în momentul când congestia va putea să fie controlabilă.

La ruterele de tip Drop Tail, singurul parametru care variază este chiar dimensiunea bufferului. Procedeul cu care congestia va fi evitată ar fi normal şi necesar să aibă o sensibilitate scăzută pentru parametri, iar aceşti parametrii ar trebui să poată fi utilizaţi pentru reţelele cu o gamă largă de variaţie a lungimii de bandă.

În cele ce vor urma au fost descrise câteva reguli care ar trebui urmate pentru ca RED să poata oferi o performanţă acceptabilă pentru cazul unei game diversificate de parametrii ai ruterelor şi de numeroase condiţii de trafic pe reţea.

Asigurarea calculului corect al lungimii medie a cozii

Încă de la început ponderea w_q va fi aleasă mai mare sau egală cu valoarea de 0,001

Lungimea medie a cozii folosită de router va fi limitată de pragul maxim th_{max} pentru intervalul de timp în care lungimea medie a cozii avg ce a fost calculată va reflecta dimensiunea reală.

Pentru ca dimensiunea medie a cozii să nu aibă întârzieri prea mari atunci când lungimile reale ale cozii sunt determinate, valoarea ponderii sau w_q nu ar fi necesară pentru ca reţeaua să ia valori foarte mici.

Marimea pragului minim va trebui să fie indeajuns de mare astfel încât să se poată maximiza puterea din reţea

Valorile pragurilor de minim (th_{min}), respectiv de maxim (th_{max}) va fi nevoie să fie îndeajuns de mari pentru ca puterea prezentată în reţea să poată fi maximizată.

Valorile lungimii reale a cozii pot fi totodată destul de variabile datorită faptului că traficul din reţea care va fi de cele mai multe ori agresiv.

Legătura de ieşire sau bottleneck-ul va fi utilizat fără a da un randament foarte bun dacă dimensiunea medie a cozii va fi stabilită la un nivel foarte mic.

Diferenţa dintre pragul de maxim şi cel de minim th_{max}-th_{min} va trebui să fie indeajuns de mare în aşa fel să nu se producă sincronizarea globală

Diferenţa th_{max}-th_{min} va trebui să fie mult mai mare decât variaţia uzuală a lungimii medie a cozii pe durata transportului dus-întors, pentru ca sincronizarea globală să nu se poata produce. Lungimea medie calculată a cozii poate să oscileze cu regular până la valoarea maximă a pragului th_{max}, având totodată acelaşi comportament ca Drop Tail, acest lucru având loc doar pentru cazul în care acea diferenţă va fi prea mică.

3.9 Alte versiuni de implementare ale algoritmului RED

3.9.1 WRED (Weighted Random Early Detection)

WRED este utilizat ca o posibilitate pentru modalitatea de aruncare de pachete care se află la finalul unei cozi atunci când coada (sau buffer-ul) este încărcata la maxim (comportament tail-drop). Totodată reprezintă o implementare Cisco pentru algoritmul Random Early Detection (RED) fiind o metodă ce foloseşte la evitarea congestiei. [9] [11]

Folosind acest algoritm, avem de-a face cu problema sincronizării globale, în momentele în care mai multe transmisii de tip TCP parcurg routerul respectiv ieşind apoi prin aceeaşi interfaţă prin care au intrat.

Viteza este oarecum mică atunci când comunicaţia TCP are loc iar pentru cazul în care banda momentan nu este ocupată cu pachete, atunci oricare dintre comunicaţiile din reţea vor avea viteză de transfer mai mare. Pachetele vor începe să fie aruncate din coadă în momentul în care suma tututror transferurilor va trece peste lăţimea de bandă a interfeţei utilizată pentru ieşire.

Folosind aceasta modalitate, va fi nevoie ca toate comunicaţiile să îşi minimizeze viteza de transfer simultan, acest lucru ducând apoi la o scădere foarte mare a vitezei globale. Datorită faptului că sesiunile protocolului TCP din link-uri au o viteză relativ mică, vor reveni astfel la cazul iniţial atunci când lărgimea de bandă este îndeajuns pentru toate transferurile ce au loc şi viteza oricărei dintre sesiuni este foarte mică. Aşa cum putem observa, acest proces se va relua ori de câte ori va fi nevoie.

Se poate observa că folosind WRED, procentul din banda ce a fost ocupată este cu mult sub 100%. Totodată utilizând acest algoritm, procentul va putea creşte constant până la valori oarecum mari. Algoritmul WRED nu dă voie scăderii vitezei de transfer pentru toate sesiunile TCP ce au loc în acelaşi timp.

WRED mai poate să utilizeze praguri pentru oricare coadă care a fost creată de către sistem Aceste praguri vor fi monitorizate până în momentul în care un pachet va fi pus la coadă. Pachetul va fi transferat în coadă atunci când numărul de pachete care se află în coadă este unul mai mic decât pragul inferior utilizat, iar pentru cazul în care depaşeşte limita pragului superior atunci pachetul respectiv va fi aruncat. Aici mai există un caz, pentru care dacă se va afla între cele 2 praguri de minim şi de maxim, atunci va fi utilizată o funcţie care stabileşte dacă pachetul va fi aruncat sau pus în coadă.

Algoritmul WRED poate construi un profil de tip WRED pentru oricare valoare de clasificare a fiecărui transfer, cu alte cuvinte putem spune că acest lucru este de fapt diferenţa

dintre algoritmul RED şi Weighted RED. Un profil de tip WRED este reprezentat de anumite valori pe care le pot lua pragurile maxime şi minime. Pragurile pot lua valori care vor fi definite ca numărul pachetelor ce se află în coadă.

Weighted RED are posibilitatea de a trata într-un mod diferit unele tipuri de pachete din reţea utilizând aceste profiluri pentru cazul în care va avea loc congestia în traficul de pachete.

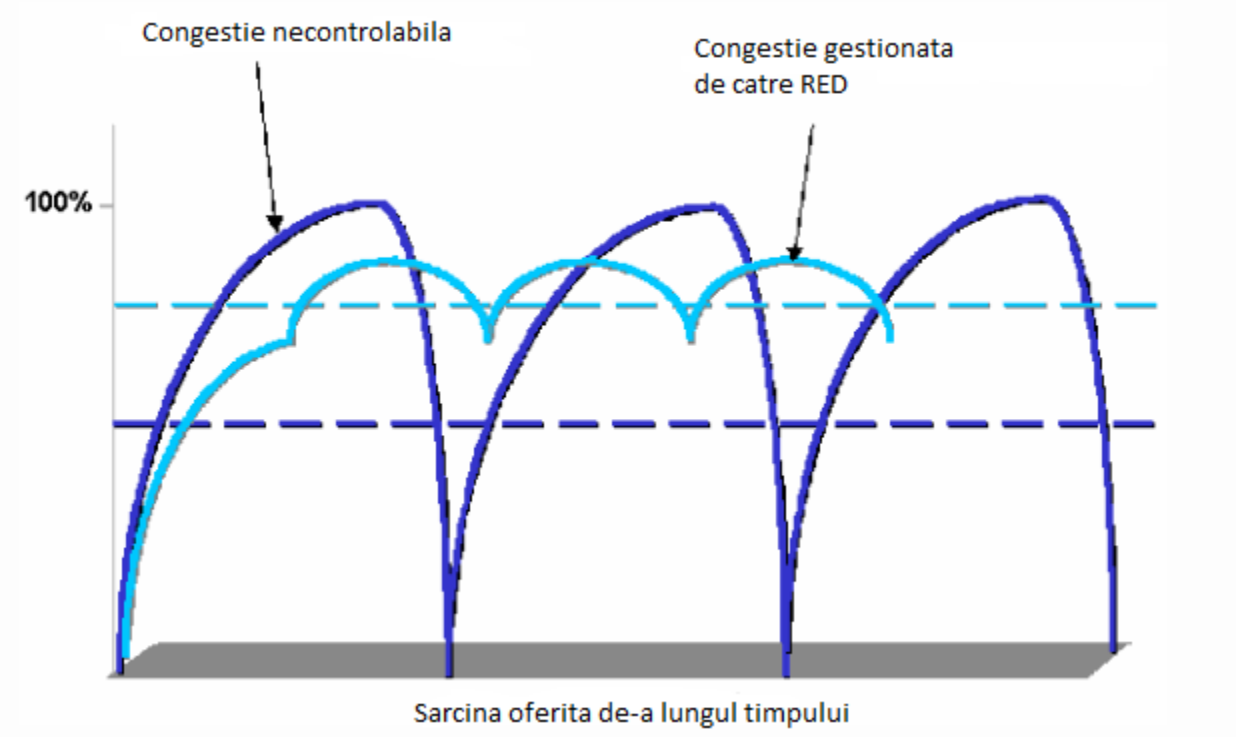


Figura 3.6: Diferenţele rezultate în urma implementării alguritmului WRED [10]

Algoritmul RED spre deosebire de algoritmul WRED nu poate realiza o diferenţiere QoS (Quality of Service) deoarece în cazul WRED sunt prezentate mai multe avantaje.

Un alt concept de importanţă majoră ce trebuie tratat, înainte de a discuta configuraţia WRED, este referitor la locul unde poate fi folosit WRED şi cum interacţionează cu uneltele de management al cozilor.

Înainte de implementarea configuraţiei folosite la algoritmul WRED, va trebui să se ţină cont de modul cum acesta interacţionează cu parametrii care gestionează coada şi de locul unde se poate folosi Weighted RED .

Lungimea medie a cozii depinde de lungimea medie anterioară şi de lungimea curentă a cozii. Urmârind formula, se va observa că dimensiunea medie a cozii va depinde de lungimea curentă şi de lungimea medie precedentă [11] :

$avg = o * (1-2^{-n}) + c * (2^{-n})$

Unde o este coada cea veche, iar c este coada curentă.

Algoritmul WRED deţine un comportament aproximativ la fel ca şi algoritmul RED, astfel calculând lungimea medie a cozii şi în acelaşi timp poate preciza dacă pachetele vor fi aruncate,

iar dacă acest lucru va fi afirmativ, care este procentul din pachetele disponibile care se vor arunca. Aceasta decizie va fi luată utilizând aceiaşi parametrii ce au fost descrişi în capitolele precedente pentru RED.

WRED, tot la fel ca şi RED foloseşte anumite criterii pentru a lua o decizie pentru câte pachete şi când acestea să poată fi descărcate, cum sunt următoarele:

-Dimenisunea medie a cozii
-Pragul de maxim
-Pragul de minim

WRED calculează lungimea medie a cozii în prima fază, exact acelaşi lucru fiind făcut şi de algoritmul RED, urmând în continuare să se compare această lungime cu pragul de maxim şi de minim pentru pachetele ce au fost marcate.

Pentru cazul în care valoarea medie a cozii se află între valorile limită ale pragurilor alese, atunci algoritmul WRED va descărca un anumit procent din pachetele disponibile în reţea iar când valoarea medie a cozii va depăşi th_{max}, atunci algoritmul va descărca toate pachetele ce au ajuns recent.

3.9.2 FRED (Fair Random Early Detection)

Algoritmul Fair RED utilizează ca baza algoritmul RED unde va mai adăuga noi parmetrii q_min şi q_max, reprezentând numărul minim şi numărul maxim de pachete ce pot fi adăugate în coadă de către oricare din legăturile disponibile.

Totodată, algoritmul Fair RED adaugă o nouă variabilă (globală) notată avgcq şi reprezintă o modalitate prin care se poate estima o medie a pachetelor ce vor ajunge în buffer-ul reţelei per link-ul respectiv. Legăturile care vor trimite pachete într-un număr mai mic decât valoarea parametrului avgcq, vor avea prioritate.

În variabila qlen va fi reţinut numărul de pachete actuale din coadă pentru oricare din conexiunile cu routerul. Totodată, FRED mai poate sa reţină şi numărul de tentative nereuşite ale fiecărei legături de a trimite un mesaj răspuns atunci când congestia va urma. Acest număr poate fi găsit în variabila strike.

Cu toate acestea, algoritmul FRED poate atribui legăturilor valori ale variabilei strike foarte mari. [7]

3.9.3 ARED (Adaptive Random Early Detection)

Principalul obiectiv al acestui algoritm este acela de a preveni dacă algoritmul RED este necesar să devină mai puţin sau mai mult agresiv privind modul de examinare a dimensiunii medie a cozii de pachete din reţea. Pentru cazul în care această dimensiune medie a cozii va depăşi într-un mod repetat th_{min} sau pragul de minim, algoritmul este prea agresiv sau mai precis examinarea avg va fi una riguroasă.

Algoritmul ARED nu va fi îndeajuns de agresiv pentru cazul ăn care valorile dimensiunii medie a cozii vor trece într-un mod constant peste th_{max} sau pragul de maxim. [8]

Algoritmul Adaptive Random Early Detect are posibilitatea de a-şi putea configura parametrii în funcţie de care este nivelul traficului din reţeaua respectivă. Cu toate acestea, dacă dimensiunea medie a cozii va lua valori ce se se vor afla între pragurile de minim, respectiv de maxim, atunci valoarea parametrului p_max va creşte într-un mod constant şi în acelaşi timp creşterea fiind una scăzută din punct de vedere al valorilor în funcţie de cât de încărcat este traficul din reţea.

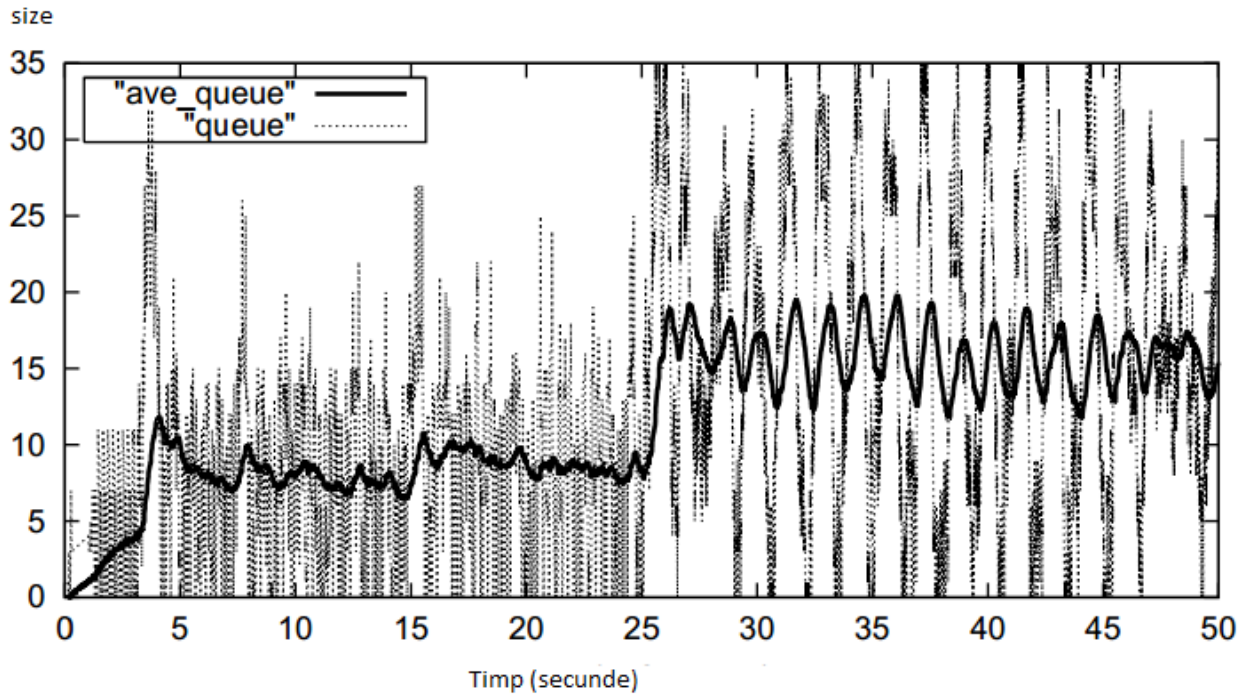


Figura 3.7: Algoritmul RED folosind o congestie ridicată [8]

Am putea spune că principalul dezavantaj la ARED este ca parametrii optimi folosiţi pentru a obţine rezultate contradictorii nu sunt stabiliţi clar in momentul iniţial.

Avantajul de bază pentru ARED este că îşi poate schimba parametrii în funcţie de cât de agresiv este traficul.

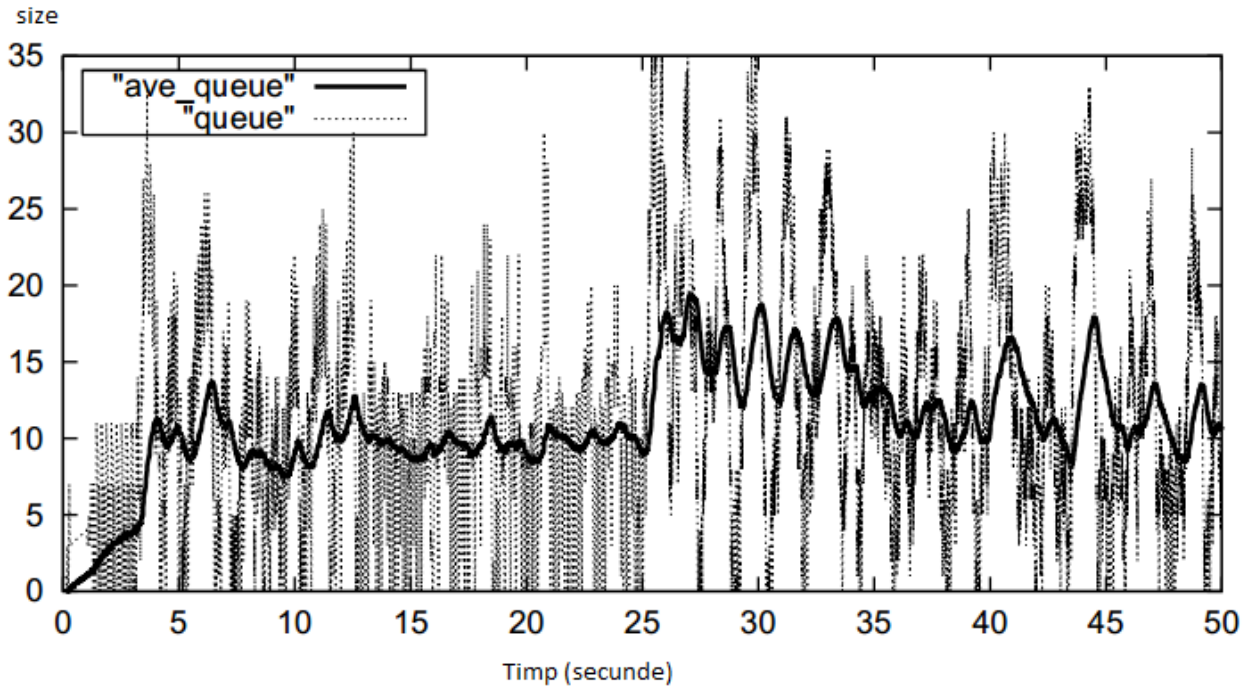


Figura 3.8: Algoritmul ARED folosind o congestie ridicată [8]

În functie de valorile lungimii medie a cozii, ARED descreşte valoarea parametrului p_max. În funcţie de pragul ce va fi depaşit (th_{min} sau th_{max}), în prezentarea algoritmului ce va urma mai jos, p_max va fi scalat utilizând astfel unul din factorii de creştere (α) sau descreştere (β).

Implementarea algoritmului ARED:
Pentru fiecare interval de timp:
If (avg > target si p_max ≤ 0,5)
Creşte valoarea p_max:
p_max ← p_max+α;
else if (avg<target si p_max ≥0,001)
Micşorează valoarea p_max:
p_max ← p_max * β ;
Variabile:
avg: lungimea cozii medie
Parametrii fixaţi, adică am exemplificat valori pentru aceşti parametrii:
Timp = 0,5 secunde
Obiectiv (target) pentru avg: [th_{min} + 0,4 *(th_{max}-th_{min}), th_{min} + 0,6 *(th_{max}-th_{min})]
α : parametru de creştere = min (0.01, $\frac{P\_max}{4}$)
β : parametru de descreştere = 0,9

3.9.4 RRED (Robust Random Early Detection)

Algoritmul Robust Random Early Detection constă în adăugarea unui bloc de filtrare şi în acelaşi timp de detectie, ce au loc pe un router înaintea blocului RED. [12] [13]

În momentul în care un pachet va fi expediat la un interval mic de timp după ce alt pachet din reţea a fost aruncat din coadă, se poate spune că cel de-al doilea pachet ar putea fi un pachet de atac. Astfel se poate realiza o diferenţiere între pachetele de tip normal (TCP) şi pachetele de atac.

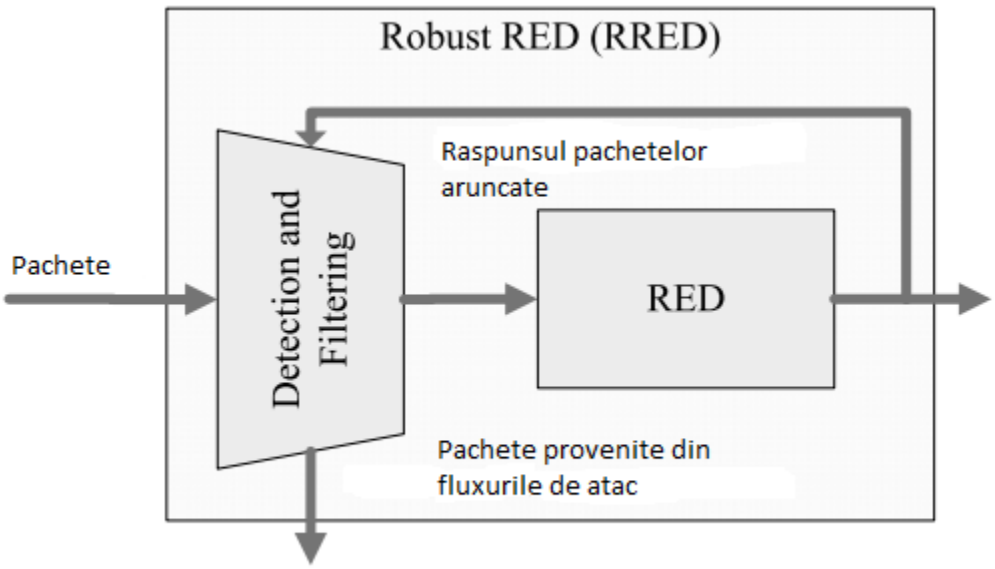


Figura 3.9: Schema arhitecturii algoritmului Robust RED [12]

Scopul acestui algoritm practic este de a realiza o îmbunatăţire privind modul de abordare al conexiunilor TCP din reţea împotriva pachetelor de atac LDoS (sau Low-rate Denial-of-Service attacks).

Aşadar RRED este o variantă a algoritmului RED simplu, iar modelul pe care îl urmează va fi acela de filtrare şi de monitorizare a pachetelor provenite din fluxurile de atac, înaintea aplicării algoritmul RED normal. [13]

CAPITOLUL 4
Programe si limbaje utilizate

4.1 Network Simulator

4.1.1 Istoric

Simulatorul în care se va lucra în decursul acestui proiect este Network Simulator 2.

Unii dintre cei care au ajutat la construirea acestui algoritm au fost chiar inventatorii algoritmului RED Sally Floyd, Kevin Fall, Steve McCanne. NS2 a fost construit intre anii 1996-1997, ca o variantă a simulatorului de reţele REAL.

Simulatorul numit REAL a fost inventat de catre Srinivasan Keshav în anul 1989 şi iniţial acesta avea decât datoria de a monitoriza atitudinea dinamică a schemelor pentru controlul congestiei sau a fluxurilor ce apăreau în reţeaua respectivă.[14]

Există numeroase sisteme de operare pe care Network Simulator 2 poate fi disponibil cum putem exemplifica : Linux/GNU, Solaris, FreeBSD, Mac OS X chiar şi versiuni Windows care suportă mediul Cygwin.

Singurul domeniu in care poate fi folosit Network Simulator este domeniul cercetării.

Totodata, NS poate oferi un suport foarte mare pentru a realiza simularea comunicaţiei dintre legăturile TCP ale reţelei, protocoalelor multicast şi a routerelor ce realizează reţeaua atât pentru reţelele ce utilizează wi-fy sau cele care utilizează cabluri.

4.1.2 Instalarea simulatorului

Simulatorul pe care l-am folosit este Network Simulator 2 cu versiunea v2.35, pe sistemul de operare Ubuntu 14.04, care a fost instalat pe computerul meu HP Compaq Pressario A900 Notebook cu următoarele performanţe:

- memorie RAM de 3GB
- processor Intel Pentium Dual 1,73 GHz
- memoria internă de 180 de GB

Pentru a instala programul NS2, am urmat următorii paşi [21] :

Înainte de a instala NS2, trebuie să instalaţi unele software-uri esenţiale:

```
$sudo apt-get install tcl8.5-dev tk8.5-dev
$sudo apt-get install build-essential autoconf automake
$sudo apt-get install perl xgraph libxt-dev libx11-dev libxmu-dev
```

Va trebui downloadată sursa NS2 de pe site-ul NS. Aşadar fişierul va fi de forma “ns-allinone-2.35.tar.gz“. Odată downloadată această arhivă trebuie despachetată în /home/):

```
$tar -zxvf ns-allinone-2.35.tar.gz -C /home/
```

Următorul pas este instalarea propriu-zisă:
$cd /home/ns-allinone-2.35
$sudo ./install
În continuare va trebui modificat fişierul .bahrc:
$vi /home/.bashrc
Pe ultima linie de comandă din acest fişier va trebui introdus scriptul:
export PATH=$PATH:/home/ns-allinone-2.35/bin:/home/ns-allinone-2.35/tcl8.5.10/unix:/home/ns-allinone-2.35/tk8.5.10/unix
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/ns-allinone-2.35/otcl-1.14:/home/ns-allinone-2.35/lib
export TCL_LIBRARY=$TCL_LIBRARY:/home/ns-allinone-2.35/tcl8.5.10/library
Se mai poate face o eventuală verficare a instalării:
$cd ns-2.35; ./validate
Se poate accesa programul executând comenzile:
$nam
$ns fisier.tcl

4.2 Limbajul OTcl

Network Simulator este un simulator scris în limbajul C++ (sau cu alte cuvinte, Network Simulator este compatibil unei ierarhii pentru clasele utilizate de acel interpretor OTcl şi unei alte ierarhii pentru clasele utilizate in C++) şi este orientat pe obiecte (OO). [14][15][20]

Aceste ierarhii pe care stau la baza simulatorului necesita dependent uneia fata de cealaltă ierarhie.

Între o clasă din cadrul ierarhiei ce a fost compilată şi o clasa din cadrul ierarhiei interpretorului există o legatură foarte stransă din punct de vedere al utilizatorilor care folosesc acest simulator. Tcl Object reprezintă o rădăcină pentru oricare dintre aceste ierarhii utilizate. [16] Simularile ce se doresc a fi realizate se fac cu ajutorul interpretorului de către useri.

Interpretorul este cel care face legătura la aceste obiecte (Tcl Object) şi în acelaşi timp mai are rolul de a reflecta un obiect din cealalta ierarhie a compilării. Ierarhia claselor interpretorului este realizată automat prin metode definite în clasa TclClass.

Cu ajutorul procedurilor ce sunt bine determinate in clasa Tcl Class, se va crea automat ierarhia claselor simulatorului.[17]

Cu alte cuvinte, obiectele care au fost reflectate de catre user vor fi referite prin modalităţile ce se găsesc în clasa Tcl Object. Mai sunt şi alte ierarhii care sunt compatibile cu limbajele OTcl, respectivC++ însă, nu sunt prezentate în clasa TclObject.

4.3 Network Animator

Nam reprezintă un tool al simulatorului NS ce se foloseşte pentru animaţiile reţelelor propriu-zise. Nam se bazează pe Tcl/TK (TookKit) , utilizat pentru a vizualiza informaţiile obţinute în urma simulărilor reţelei. Modul de gândire cu care a fost implement acest tool Nam este de fapt inventarea unui animator ce poate citi fragmente imense de informaţie folosite pentru animaţiile respective şi in acelasi timp să aibă posibilitatea de a putea observa foarte multe posibilităţi pentru reţelele propriu-zise. [18][19][20]

Nam a fost realizat cu scopul de a citi comenzile simple, însă cu o încărcătură mare de informaţie utilizând astfel fişiere mari în care anumiţi parametrii sunt monitorizaţi şi de a anima rezultatele.

Pentru ca tool-ul să aibă posibilitatea de a utiliza un volum de informaţie foarte mare, va trebui ca pentru realizarea animaţiilor, cantitatea de date ce se va afla în memoria respectivă să ocupe dimensiuni foarte mici. În acelaşi timp, acest lucru poate uşura procesul de realizare a animaţiilor, deoarece comenzile respective pot fi citite de fiecare dată când este nevoie oferindu-se un acces simplu.

În primul rând, pentru a putea utiliza acest tool, va fi nevoie de un fişier de monitorizare în care vor fi adăugate toate comenzile şi informaţiile despre parametrii de care este nevoie. În fişierul de monitorizare putem găsi detalii legate de nodurile reţelei, legăturile dintre noduri, pachete dar şi topologiile dintre legături. Network Simulator este cel care crează automat acest fişier de monitorizare.

Atunci când este pornit, NAM va citi din fişierul din monitorizare informaţia respectivă, după care va fi creată fereastra de tip pop-up, crează schema simulării apoi se va opri atunci când momentul de timp va atinge valoare nulă.

Cu alte cuvinte, putem spune ca NAM este o interfaţă pentru useri care are posibilitatea controlului asupra animaţiilor.

După ce a fost creat acel fişier de monitorizare, atunci simulatorul va fi pregătit pentru transmiterea informaţiilor şi totodată vor fi realizate şi animaţiile.

4.4 Concluzii despre simulator

Datorita faptului ca simulatorul NS2 va trebui să realizeze două obiective pentru realizarea simulării (pentru obţinerea datelor simularii şi pentru orientarea pe obiecte), atunci acesta va utiliza două limbaje pentru realizarea acestui lucru şi anume OTcl si C++.

Totusi, există cazuri în care se va avea de-a face cu simulări ce necesită aspecte pentru protocoalele utilizate în detaliu, acest lucru ducând la folosirea unui limbaj de programare ce deţine avantajul de a controla biţii ăi pachetele într-un mod mai eficient, totodată putând astfel să încarce seturi de informaţie cu mult mai mari.

Pentru astfel de limbaje, nu este un lucru foarte important determinarea şi remedierea defecţiunilor atunci când simularea este în decurs de rulare, însă o importanţă majora este pusa pe viteza de lucru a simularii.

Limbajele folosite de Network simulator deţin avantaje în funcţie de scopul în care sunt folosite.

De exemplu limbajul OTcl prezintă un mod de a rula mai greoi, pe când acesta prezintă o flexibiliate mai mare, putând fi realizate schimbările într-un mod mai rapid, pe când limbajul C++ nu prezintă o flexibilitate atât de bună, însă este mult mai rapid atunci când este stabilită rularea respectivă.

Pentru cazurile unde sunt utilizate scenarii apropiate din punct de vedere al valorilor, sau parametrii reţelei având dimensiunile apropiate,atunci timpul de care are nevoie iteraţia prezintă o importaţă deosebită, ca şi pentru viteza de lucru a simulării, deoarece acele valori fiind asemănătoare, va fi mult mai dificil a se realiza iteraţia respectivă. Dar totodată, acel timp în care are loc simularea nu va fi atât de important pentru cazul în care simularea are loc o singură dată.

CAPITOLUL 5
Modificări aduse algoritmului RED

5.1 Introducere

Aşa cum am descris în capitolele precedente, algoritmul RED calculează probabilitatea de aruncare a unui packet în două etape, atunci când dimensiunea cozii scade între th_min si th_max, unde th_min reprezintă pragul de minim al cozii, th_max reprezintă pragul de maxim al cozii.

Prima etapă reprezintă calculul probabilităţii cu care pachetele sunt marcate (P_b).

Cu ajutorul acestor valori se calculează probabilitatea P_b pe baza unei funcţii liniare:

$P_b = P_{max} * (avg-th\_min)/(th\_max-th\_min)$

Se observă în ecuaţia anterioară că P_b este dependent de valoarea dimensiunii medii a cozii.

În cea de-a doua etapă, RED contorizează numărul de packete care au trecut prin gateway până când ultimul pachet va fi aruncat, aplicându-se formula:

$P_a = P_b / (1 – count*P_b)$

Aşa cum am observat în articolul [22], am modificat primul pas, adică probabilitatea P_b aplicând 4 funcţii. Fiecare funcţie, în algoritm este apelată de o sintaxă de tipul edp_.func == x , unde x poate lua valori de la 1 la 4.

Aceste 4 funcţii sunt reprezentate mai jos de valorile v1,v2,v3,v4:

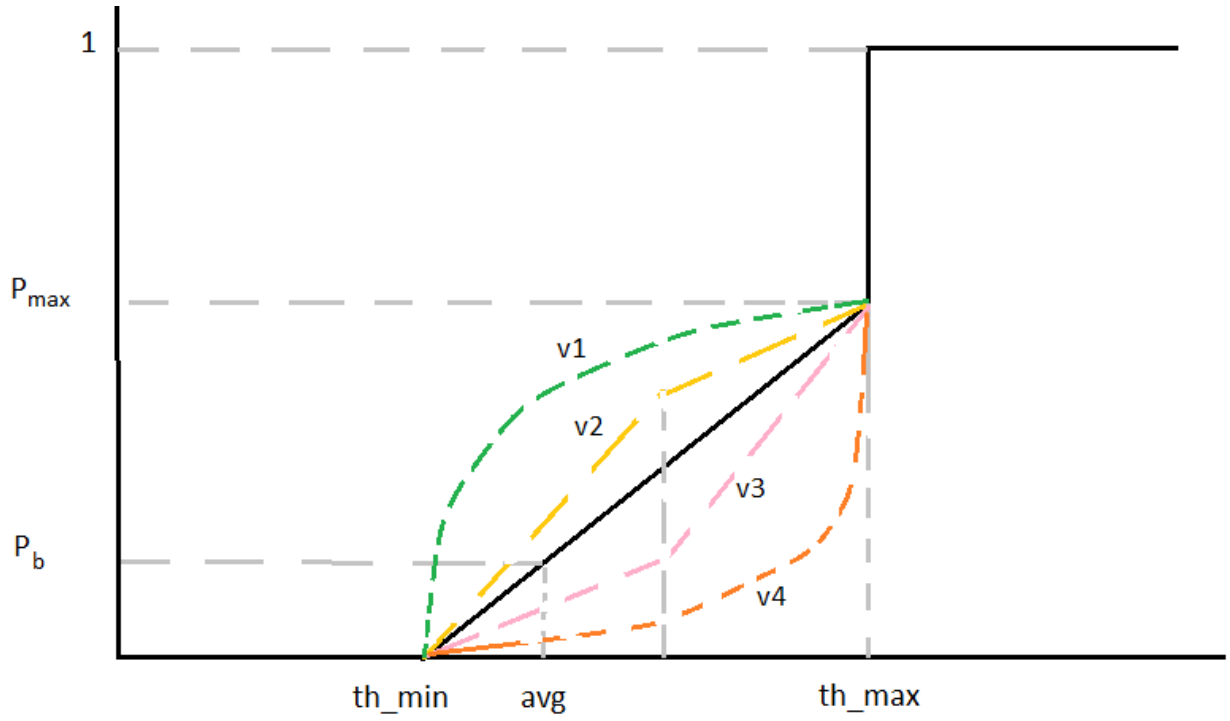


Figura 5.1: Diferite funcţii pentru a evalua probabilitatea de aruncare a unui pachet [23]

Putem observa în figura 5.1 de mai sus cum lungimea medie a cozii este dependentă de pragul cozii minime şi de pragul cozii maxime.

În implementarea cozii RED vom utiliza formula:
Pb = Pmax*(v_a * v_ave + v_b) , unde v_a = 1.0 / (th_max – th_min), v_b = -th_min / (th_max – th_min)
Această formulă este echivalentă cu prima formulă a P_b
Pentru funcţia v1,care va fi concavă vom folosi funcţia de logaritm:
Pb = Pmax*(m*log(v_ave) + n)

Pentru ca acele functii curbe să unească punctele de th_min si th_max avem următoarele ecuaţii:
0 = Pmax*(m*log(th_min) + n)
m = 1/(log(th_max) – log(-v_b/v_a))
n = -m*log(-v_b/v_a)
Pmax = Pmax*(m*log(th_max + n))
De asemenea, funcţia liniară iniţială ne dă funcţia:
0 = Pmax*(v_a*th_min + v_b)

În mod similar, ne putem exprima formula pentru funcţia de v4 funcţia convexă (am ales funcţie exponenţială), folosind th_max, v_b şi v_a :
Pb = Pmax*(m*exp(v_ave)+n) , unde pentru acest caz:
m = 1/(exp(th_max) – exp(-v_b/v_a))
n = -m*exp(-v_b/v_a)

În ceea ce privesc funcţiile v2 şi v3, care sunt funcţii liniare pe porţiuni:
Pb = Pmax*(m*v_ave + p) when v_ave in [th_min, (th_min+th_max)/2]
Pb = Pmax*(n*ave + q) when v_ave in [(th_min+th_max)/2, th_max]
Mai putem obţne următoarele ecuaţii cu care vom obţine relaţii dintre m si n.
m*th_min + p = 0
m*[(th_min+th_max)/2] + p = n*[(th_min+th_max)/2] + q
n*th_max + q = 1
Va rezulta:
n = [m*(th_max+th_min)/2 - m*th_min] / [(th_max+th_min)/2 - th_max]

Pentru funcţia v2 am setat m=1,4*v_a :
p = -1.4*v_a*th_min
n = [1.4*v_a*(th_max+th_min)/2 - 1.4*v_a*th_min] / [(th_max+th_min)/2-th_max]
q = 1 – n*th_max

Pentru funcţia v3 am setat m=0,6*v_a :
p = -0.6*v_a*th_min
n = [0.6*v_a*(th_max+th_min)/2 - 0.6*v_a*th_min] / [(th_max+th_min)/2-th_max]
q = 1 – n*th_max

Modificarea făcută algoritmului RED este practic făcută în funcţia calculată_p_new unde au fost introduse toate ecuaţiile necesare pentru a realiza aceste funcţii. Totodată, după cum se observa în anexa 2 şi în ecuaţiile de mai sus, am modificat funcţiile v2 si v3, cărora le-am introdus valori diferite (respectiv 1,4 pentru v2 şi 0,6 pentru v3) pentru parametrul m pe care îl conţin.

5.2 Reprezentarea grafică a funcţiilor

Pentru cazul iniţial unde nu a fost facută nici o modificare a codului sursă, atunci graficul unei funcţii va arăta ca în figura 5.2

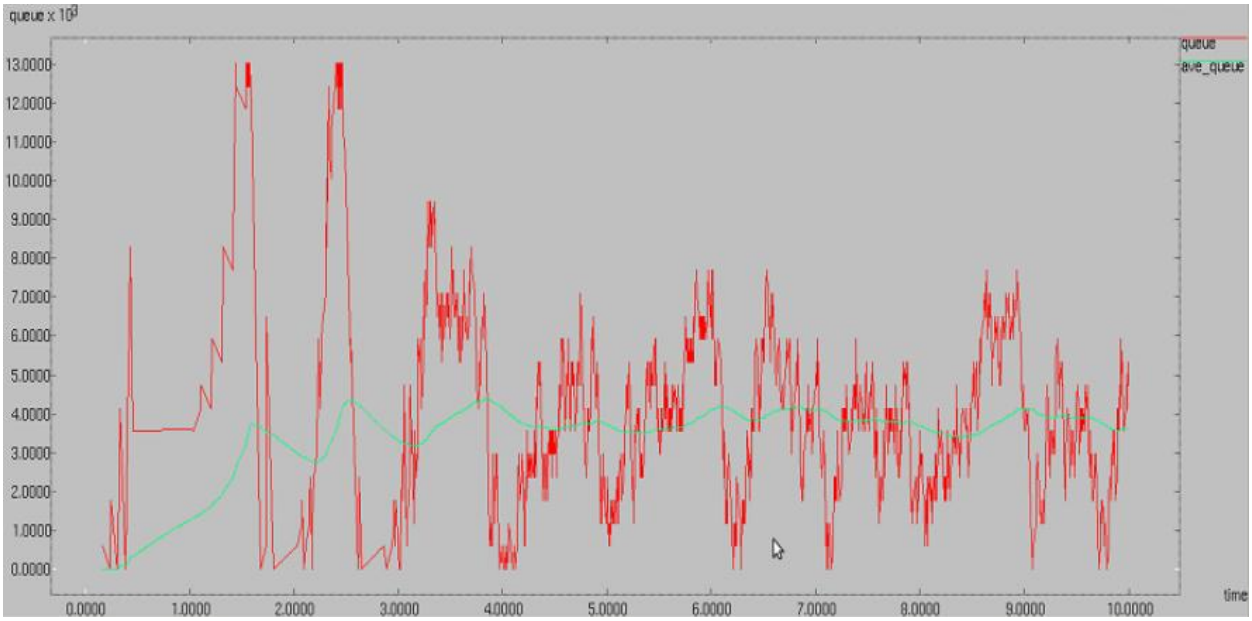


Figura 5.2 : Graficul algoritmului RED fără nici o modificare

Rulând simulările pentru fiecare din cele 4 funcţii s-au obţinut următoarele grafice, unde ave_queue (valoare reprezentată cu verde) este dimensiunea cozii medii şi queue (valoare reprezentată cu rosu) este valoarea curentă a cozii.

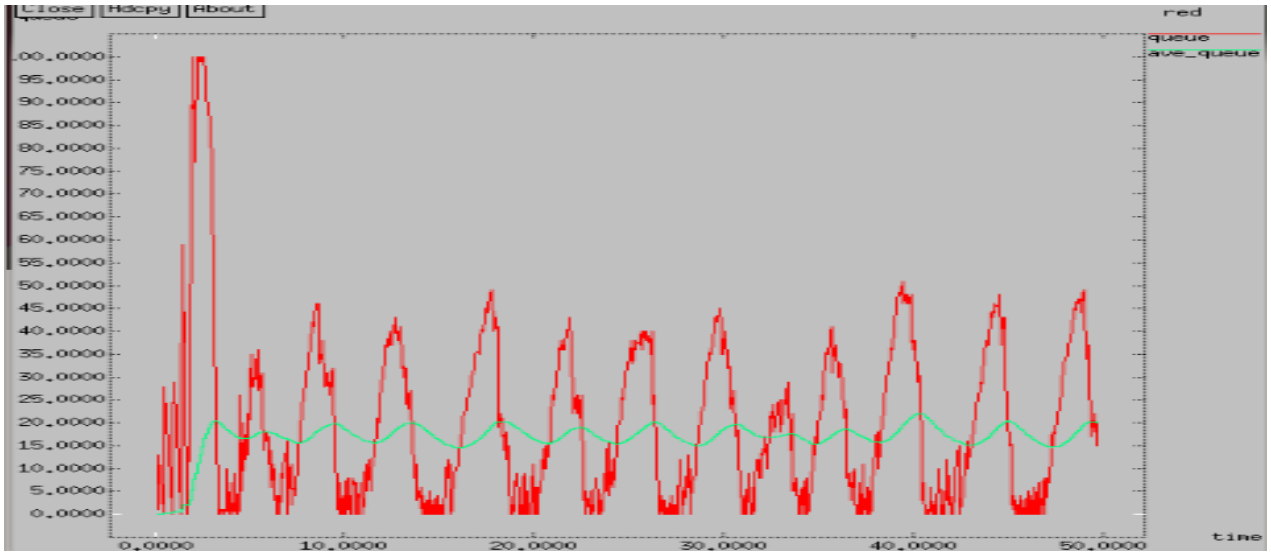


Figura 5.3 : Grafic folosind funcţia din algoritm – v1

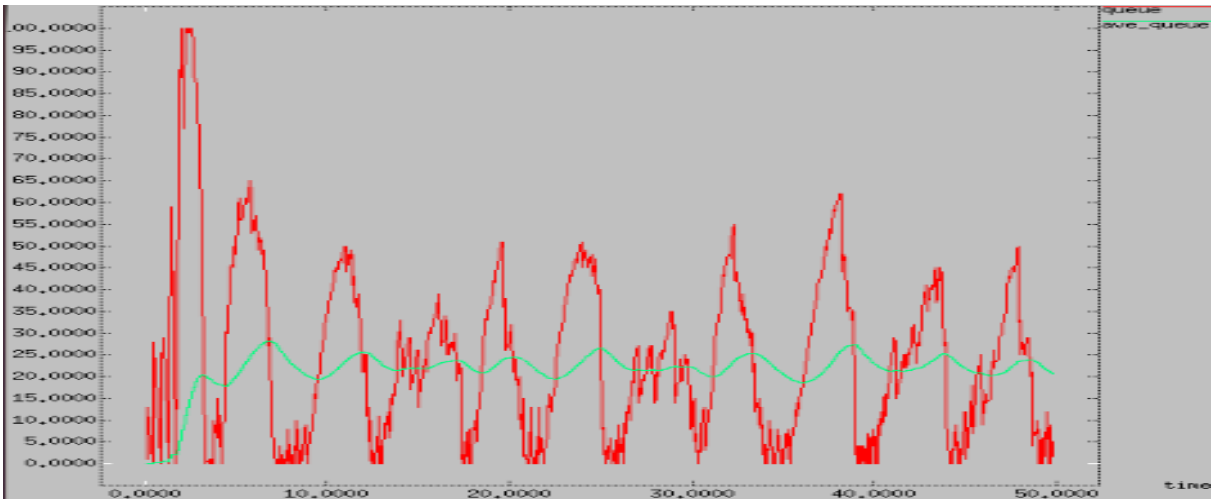


Figura 5.4 : Grafic folosind funcţia din algoritm – v2

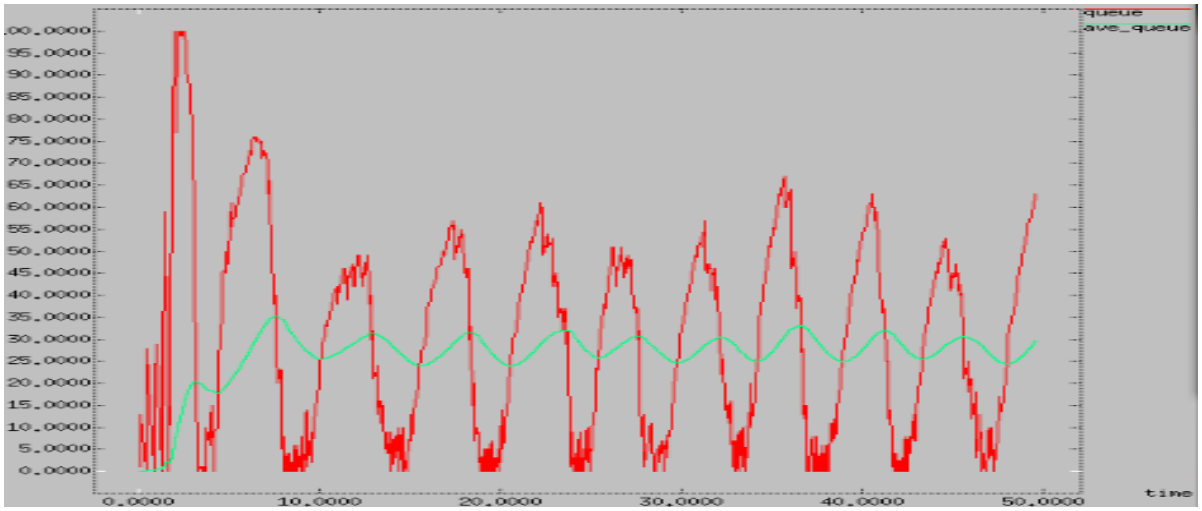


Figura 5.5 : Grafic folosind funcţia din algoritm – v3

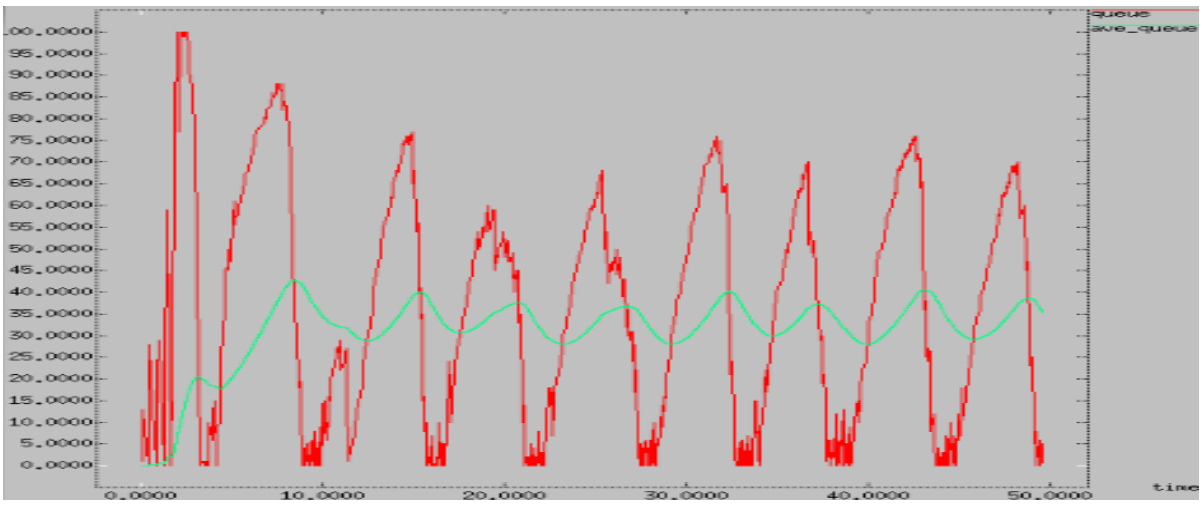


Figura 5.6 : Grafic folosind funcţia din algoritm – v4

Din aceste funcţii putem observa că valorile cozii medii variază mai mult pentru funcţia v4 faţă de celelalte trei funcţii, la v4 valorile cozii curente trecând de la valori foarte mari la valori foarte mici. În cazul primelor 3 functii, acestea oferă o probabilitate mai mică la aruncarea pachetelor, aşa cum vom vedea în simulările ce vor urma. Funcţia v4 pare a fi cea mai agresivă din acest punct de vedere.

Pentru aceste 4 funcţii am folosit script utilizând limbajul TCL pe care l-am rulat cu comanda :
$ns RED.tcl x y z
Unde x,y,z sunt parametrii ce descriu funcţia:
- x este valoarea uneia din cele 4 funcţii;
- y este unul dintre scenariile folosite.

Am utilizat o topologie formată din 16 noduri, acestea fiind legate între ele prin link-uri aşa cum urmează:

- Folosirea a 14 link-uri cu lăţimea de bandă de 10 Mbps cu delay-ul setat la de 100 ms.

Aceste 14 link-uri practice leagă 15 noduri. 14 dintre noduri trimit pachete către un singur nod, acesta numindu-se nod de acumulare al pachetelor.

Din momentul în care pachetele sosesc în nodul de acumulare, atunci se creează congestia până când o parte din pachetele pe care algoritmul le consideră utile ajung la destinaţie, adică în nodul terminal.

- Folosirea a unui link bottleneck cu lăţimea de bandă de 3 Mbps,cu delay-ul setat la 100 ms.

Am ales ca valoarea lăţimii de bandă pentru acest bottleneck să fie una mică, deoarece acesta poate fi luat ca un factor ce duce la congestie, astfel am putut mai uşor evidenţia acest aspect.

Dacă aş fi folosit o lăţime de bandă mai mare, pachetele sosite în nodul de acumulare ar fi avut suficient spaţiu pentru ca buffer-ul să nu se încarce.

Aşa cum am prezentat în algoritm, trimiterea pachetelor pe link-ul de bottleneck diferă, transmisia pachetelor către nodul superior fiind diferită faţă de transmisia pachetelor dinspre nodul superior spre nodul de acumulare al pachetelor.

Cele 14 noduri care trimit pachete către nodul de acumulare folosesc conexiuni FTP care utilizează protocolul TCP prezentat în capitolele anterioare. Nodurile trimit traffic FTP pe parcursul întregii simulări, adică în algoritm am prezentat acest interval de timp ca fiind durata de utilizare a conexiunilor FTP plus o secundă pentru a garanta ca procesele FTP au timp suficient penţru executie.

Dimensiunea cozii de bottleneck am ales-o ca fiind 100.

Pentru pachetul TCP am ales dimensiunea de 1024 biti, dar practic atunci când pachetul va fi creat si are loc simularea, acesta va fi de 1064 de biţi, adică se vor adăuga încă 40 ce sunt utilizaţi pentru header.

Valorile utilizate pentru parametrii RED în funcţie de scenariile folosite la rularea algoritmului din terminal:
Scenariul 1:
minth=15, maxth=45, wq=0,002, unde wq reprezintă încărcarea cozii.

Scenariul 2:
minth=20, maxth=60
Scenariul 3:
minth=25, maxth=75
Scenariul 4:
minth=30, maxth=90

Schema topologiei se găseşte în următoarea figură, dar poziţia nodurilor este aleatoare atunci când simularea are loc, acest lucru neavând nici o importanţă. În exemplul de simulare de mai jos link-ul de bottleneck este cel care leagî nodurile 15 si 16.

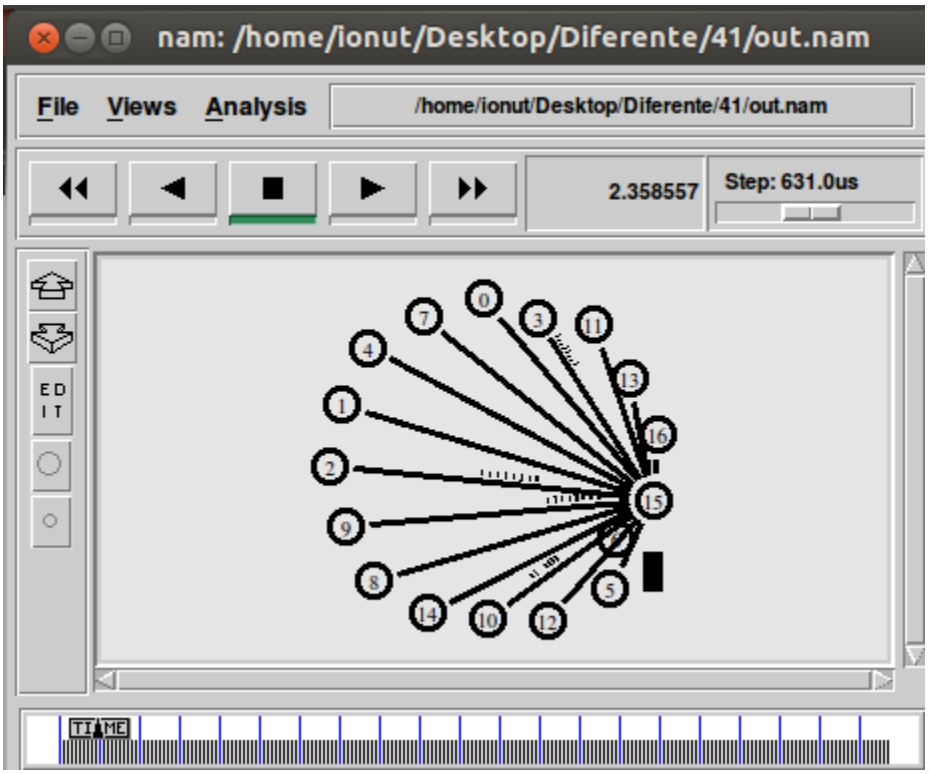


Figura 5.7 : Topologia folosită

Algoritmul TCL poate fi urmărit în Anexa 1.

5.3 Analiza fişierelor de trace

În urma simulărilor acestor funcţii utilizând limbajul TCL s-au generat mai multe fişiere de trace, în acestea aflându-se valorile parametrilor ce descriu funcţiile.

Atunci când NS rulează un script tcl, toate evenimentele de tipul pachete primite (receive, r), pachete pierdute (drop, d), pachete puse în capul cozii (enqueue, +) şi pachete puse în josul

cozii (dequeue, -) sunt puse într-un fişier de trace (out.tr), care este organizat pe 12 coloane în funcţie de anumiţi parametrii rezultaţi, ca în figura următoare:

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id
Ex: +	160.012933	1	0	tcp	60	-------	2	1.0.1.0	0.0.0.0	0	64

Figura 5.8: Cele 11 detalii despre fiecare tip de eveniment din fişierul principal de trace [23]

5.3.1 Structura fişierelor de trace

Pierderea de pachete (packet loss) apare atunci când unul sau mai multe pachete de date care călătoresc într-o reţea de calculatoare nu reuşesc să ajungă la destinaţia lor. [23] Pierderea de pachete se distinge ca unul dintre cele trei tipuri principale de erori întâlnite în format digital în comunicaţii; celelalte două fiind bitul de eroare şi pachetele false cauzate din cauza zgomotului.

Pachetele pot fi pierdute într-o reţea, deoarece acestea pot fi aruncate atunci când o coadă este supraîncarcată.

Valoarea pierderii de pachete în timpul stării de echilibru este o altă proprietate importantă a unui sistem de control al congestiei. Cu cât valoarea pierderii de pachete este mai mare, cu atât mai dificil este pentru protocoale de transport-layer pentru a menţine lăţimi de bandă ridicate, sensibilitate la pierderea de pachete individuale, precum şi a frecvenţelor şi modelor de pierderi în rândul secvenţelor de pachete puternic dependente de aplicaţii.

Debitul de transfer (Throughput)

Aceasta este caracteristica principală de măsurare a performanţei, şi cel mai des utilizată. În reţelele de comunicaţii, cum ar fi Ethernet sau pachet radio, debitul de transfer de reţea este rata medie de livrare a mesajului de succes de peste un canal de comunicare. Debitul este de obicei se măsoară în biţi pe secundă (bit / s sau bps), şi, uneori, în pachete de date pe secundă sau pachete de date pe un interval de timp.

Aceasta măsoară cât de curând receptorul este capabil de a obţine o anumită cantitate de date trimise de către expeditor. Este determinat ca raportul dintre totalul datelor primite la un capăt la altul cu întârzieri. Debitul este un factor important care are un impact direct asupra performanţei reţelei.

Întârzierea (Delay)

Întârzierea este timpul suplimentar scurs în timp ce un pachet călătoreşte de la un punct la altul. Cu cât valorile întarzierilor sunt mai mari, cu atât este mult mai dificil pentru protocoalele layer de transport să menţină lăţimi de bandă mari.

Lungimea cozii (Queue Length)

Un sistem de aşteptare în reţele poate fi descris de pachetele sosesc pentru serviciu, de aşteptare pentru serviciu în cazul în care aceasta nu este imediat, iar în cazul în care au aşteptat serviciul, părăsesc sistemul după ce au fost servite. Aceasta lungime a cozii este o caracteristică foarte importantă pentru a determina cât de bine lucrează algoritmul controlului congestiei.

Routerele renunţă la unul sau mai multe pachete înainte ca buffer-ul să devină complet plin. De fiecare dată când un pachet ajunge în buffer, algoritmul RED calculează lungimea cozii medii (avg).

Dacă lungimea cozii medii este mai mică decât un anumit prag inferior, congestia se presupune a fi minimă sau inexistentă, iar pachetul este pus în aşteptare.

Dacă lungimea cozii medii este mai mare decât un anumit prag superior, congestia se presupune a fi de un nivel ridicat şi pachetul este aruncat.

Dacă valoarea lungimii cozii medii este între doua praguri, acest lucru ar putea indica debutul de congestie. Probabilitatea de congestie este apoi calculată.

În scriptul TCL realizat, am adăugat în funcţia Finish linii de cod care duc la segmentarea fiţierului de trace în mai multe fişiere, fiecare reprezentînd unul dintre acei parametrii care descriu funcţia (ex.: ave.tr=lungimea medie a cozii; temp.q=momentele de timp când sunt aruncate pachete etc.).

Cu ajutorul acestei segmentari se poate face mai uţor identificarea şi contorizarea valorilor acestor parametrii pentru realizarea graficelor.

[24] Cu ajutorul limbajului AWK am extras pe rând evenimentele (+,-,r,d) din fişierul principal de trace (out.tr) folosind următoarea comandă din terminal:
$awk –f awk.awk out.tr

În urma acestei comenzi am obţinut numărul evenimentelor de acel tip.
Algoritmul folosit pentru determinarea numărului de pachete puse în capul cozii (enqueue, +) :

```
BEGIN{
              sum=0;
               }
               {
               if($1=="+"){
                   sum++;
                   }
               }
END
{
printf("The number of enqueue is: %d\n",sum);
}
```

5.4 Rezultate şi observaţii

Folosind cele 4 funcţii disponibile se vor realiza comparaţii între ele privind anumiţi factori cum ar fi: variaţia pachetelor aruncate, frecvenţa de aruncare a pachetelor, numărul pachetelor aruncate, întreruperi, cât şi jitter.

În continuare am constatat care este numărul evenimentelor de fiecare tip şi am salvat rezultatele într-un fişier de tip .txt pentru fiecare eveniment.

Totodată am constatat că atunci când folosesc aceeaşi funcţie pentru două scenarii diferite obţin acelaşi număr de evenimente, ceea ce înseamnă că tipul scenariului (sau alegerea pragurilor de minim şi de maxim) folosit nu influenţează evenimentele.

În urma simulărilor am introdus rezultatele în tabelul 1:

Functie\Eveniment	Enqueue (+)	Dequeue (-)	Reveived (r)	Drop (d)
1	65904	65713	65583	191
2	67311	67124	66981	172
3	67584	67350	67207	166
4	66855	66690	66562	324

Tabelul 5.1 : Tabel al funcţiilor în raport cu evenimentele simulării

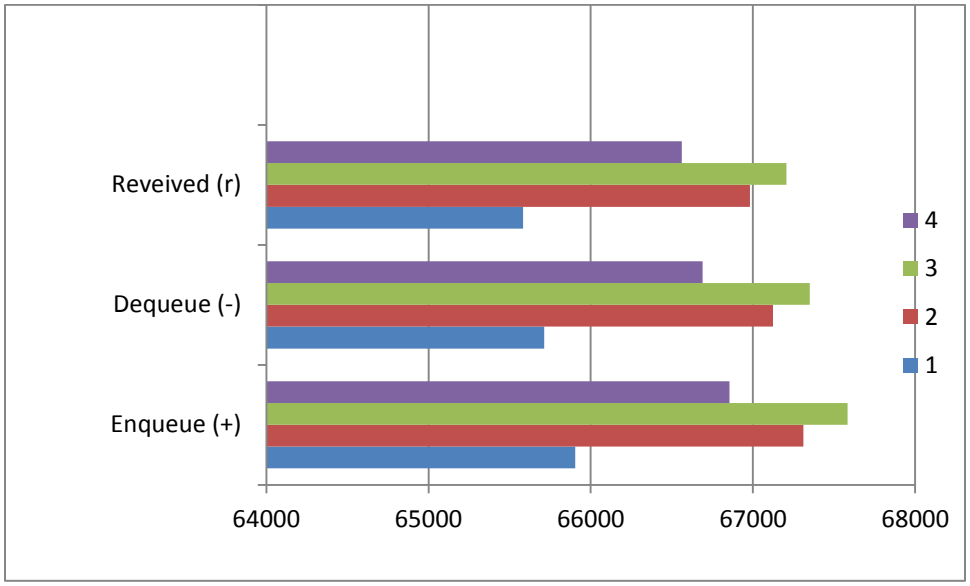


Figura 5.9: Grafic al evenimentelor în funcţie de activitatea pachetelor

Se observă din tabel că numărul pachetelor pierdute este mai mic pentru funcţiile v1,v2,v3, iar la funcţia v4 este mult mai mare (aproape dublu). Acest fapt se datorează şi valorilor cozii medii care variază mai mult pentru funcţia v4, aceasta fiind cea mai agresivă din acest punct de vedere.

Se mai poate observa că funcţiile v2 si v3 deşi au cel mai mic număr de pachete pierdute, în acelaşi timp au şi cel mai mare număr de acţiuni de alt tip ale pachetelor. O altă observaţie este că numărul de acţiuni ale pachetelor este mult mai mic pentru v1 (pentru oricare din evenimente), deoarece funcţia v1 nu prezintă o variatie atât de mare precum v4.

În continuare se vor realiza comparaţii privind variaţia pachetelor aruncate, cât şi frecvenţa de aruncare a pachetelor.

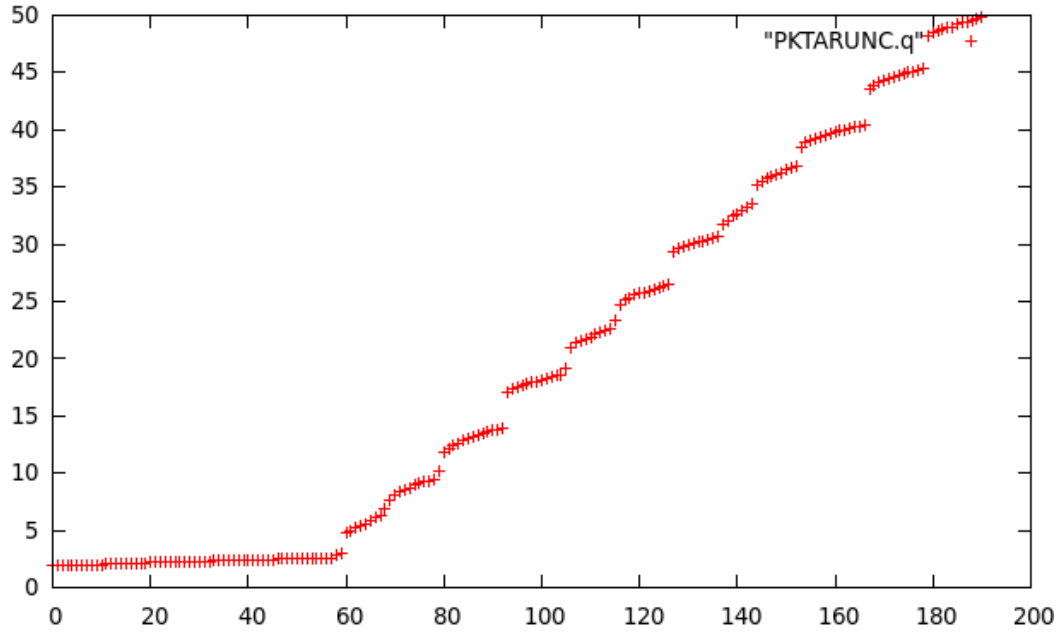


Figura 5.10: Variaţia pachetelor aruncate în funcţie de timp pentru funcţia v1

Din grafic se observă că în faza iniţială pachetele nu se pierd, urmând după un moment să crească numărul pachetelor aruncate, din cauza încărcării cozii.

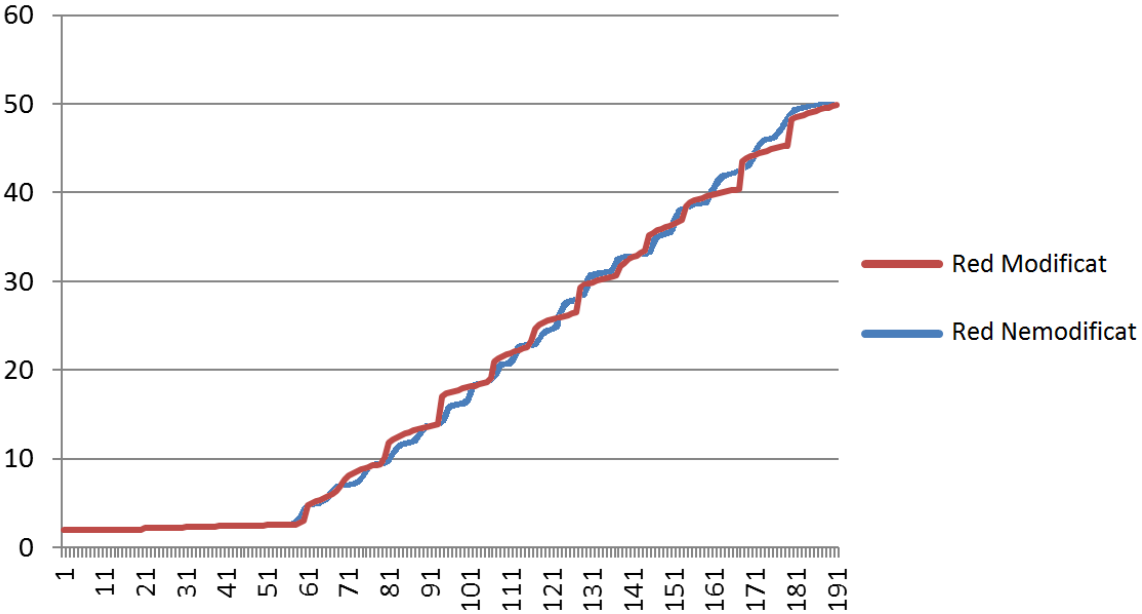


Figura 5.11: Variaţia pachetelor aruncate pentru algoritmul RED modificat şi cel nemodificat

Putem observa din aceasta comparaţie în figura 5.11 că aruncarea pachetelor este mult mai frecventă pentru RED nemodificat, lucru constatat şi în figura 5.13.

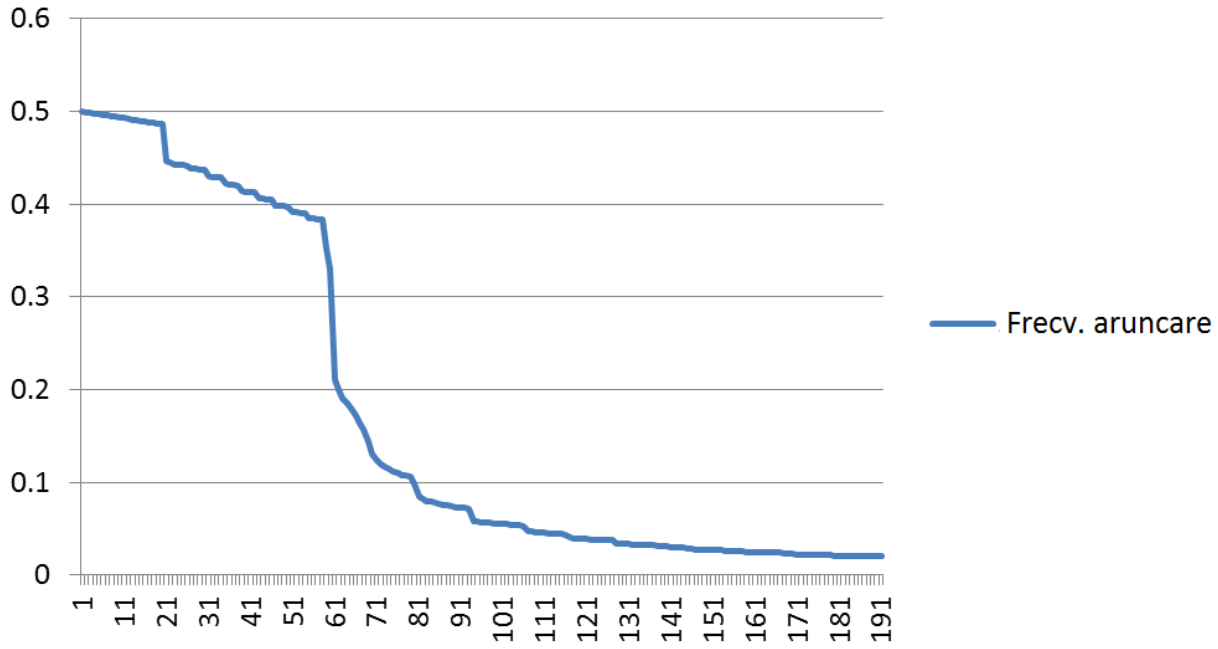


Figura 5.12: Frecvenţa de aruncare a pachetelor pentru funcţia v1

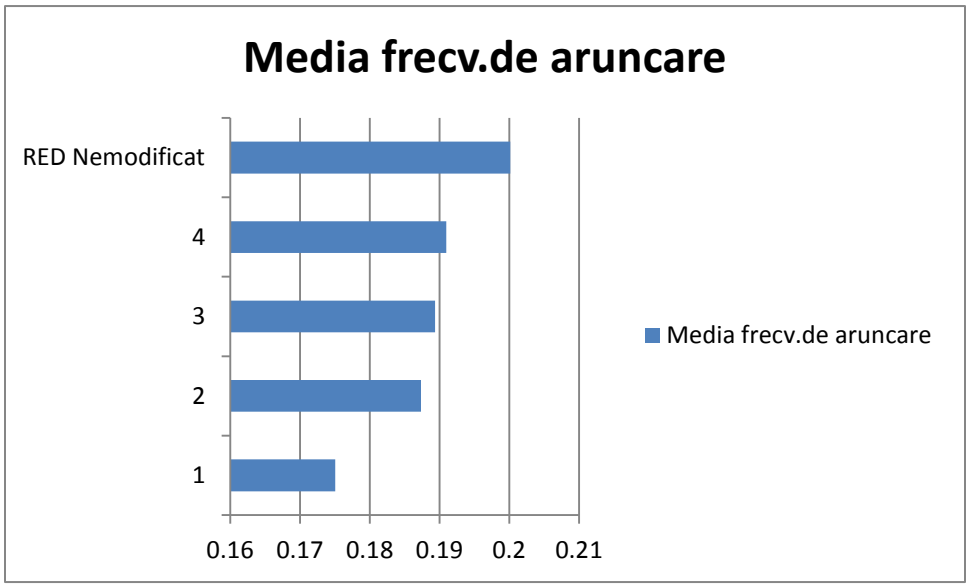


Figura 5.13: Grafic în funcţie de media frecvenţei de aruncare pentru cele 4 funcţii modificate şi pentru RED nemodificat

Se observă din figura 5.13 că dintre cele 4 funcţii, valoarea cea mai mare a mediei frecvenţei de aruncare o deţine funcţia v4. Mai putem preciza că media frecvenţei de aruncare a pachetelor pentru v4 este cu 4,8 % mai mică (daca procentajul este mic, atunci funcţia nu este

potrivită) decât cea a algoritmului nemodificat. Funcţia optimă pentru media frecvenţei de aruncare a pachetelor este funcţia v1, deoarece este cu 8% mai mică decât cea pentru algoritmul RED nemodificat.

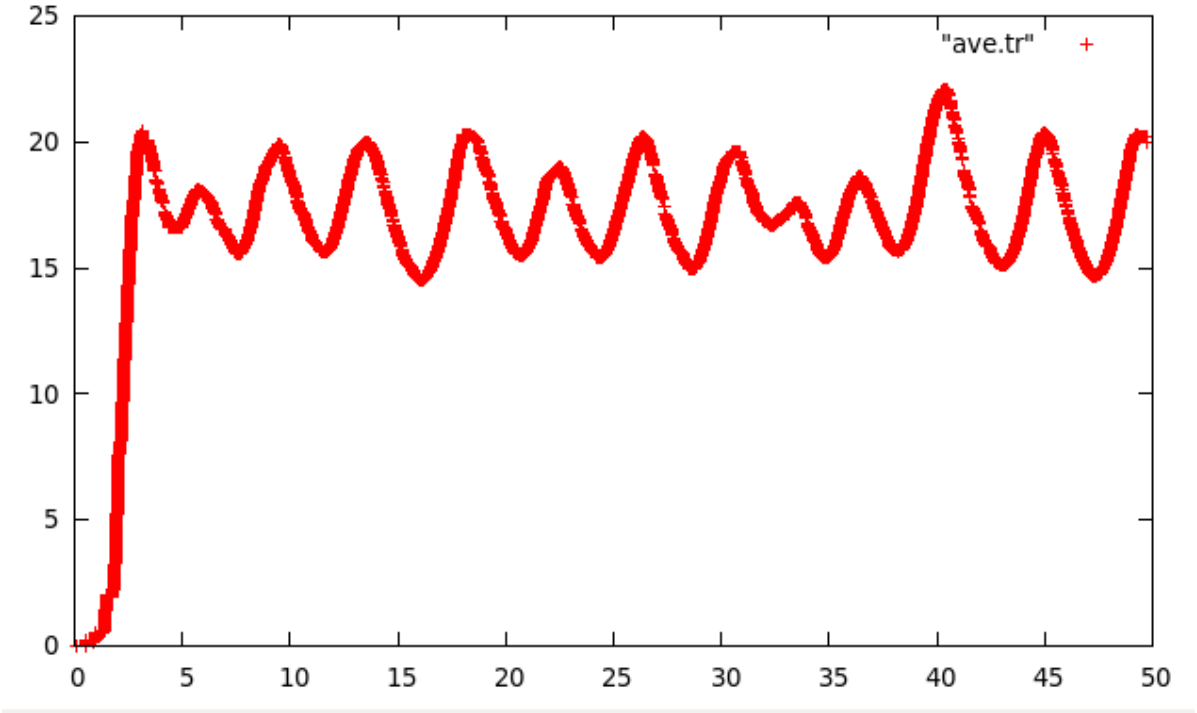


Figura 5.14: Variaţia cozii medii în funcţie de timp pentru funcţia v1.

Se observă că valorile cozii medii sunt minime în faza iniţială, urmând o creştere bruscă. Acest lucru se datorează timpului până când pachetele vor sosi simultan pe bottleneck.

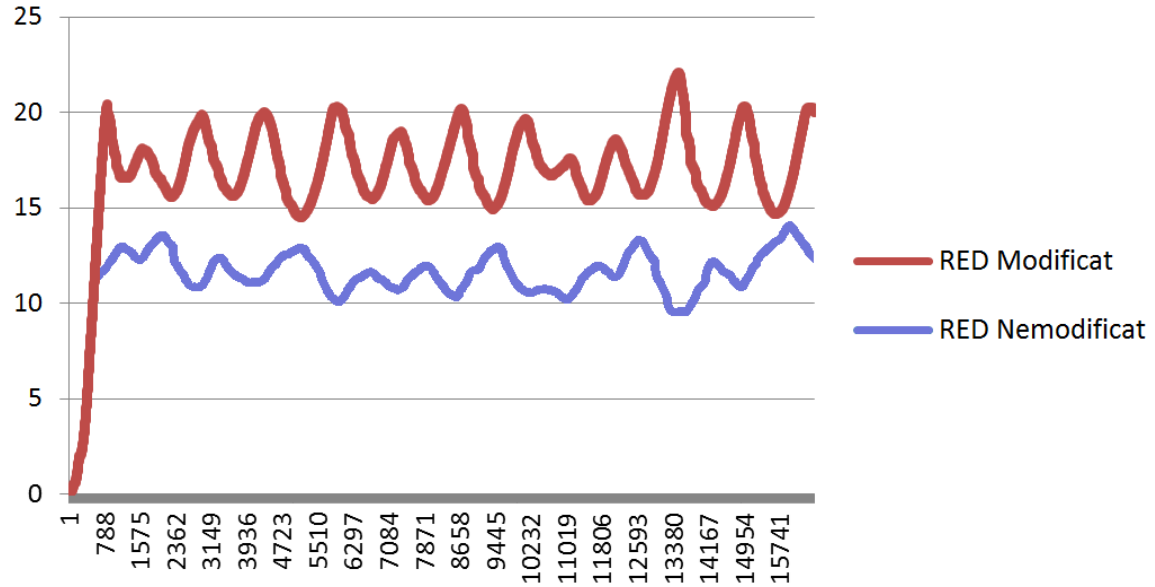


Figura 5.15: Variatia cozii medii in functie de timp pentru RED modificat (v1) si nemodificat

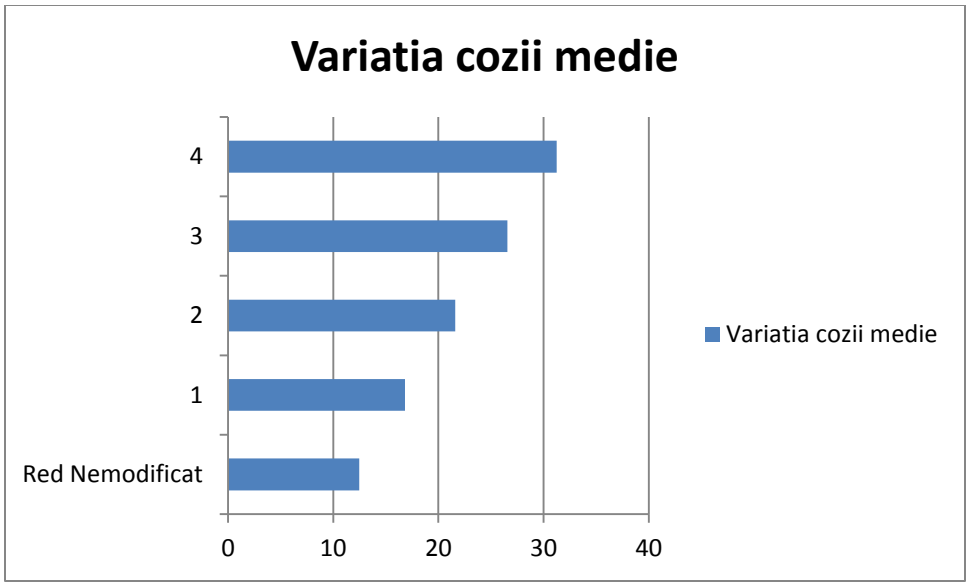


Figura 5.16: Grafic pentru variaţia medie a cozii (pentru cele 4 funcţii)

Urmărind figura 5.16, se observă că funcţia v4 variază în amplitudini mult mai mari (de peste două ori) decât cele ale algoritmului nemodificat, chiar şi mult mai mari decât celelalte funcţii. Pot spune că v4 este funcţia cea mai optimă din punct de vedere al mărimilor amplitudinii la care variază.

Am executat cu ajutorul GNUPLOT următoarele grafice folosind fişierele de trace obţinute [24] :

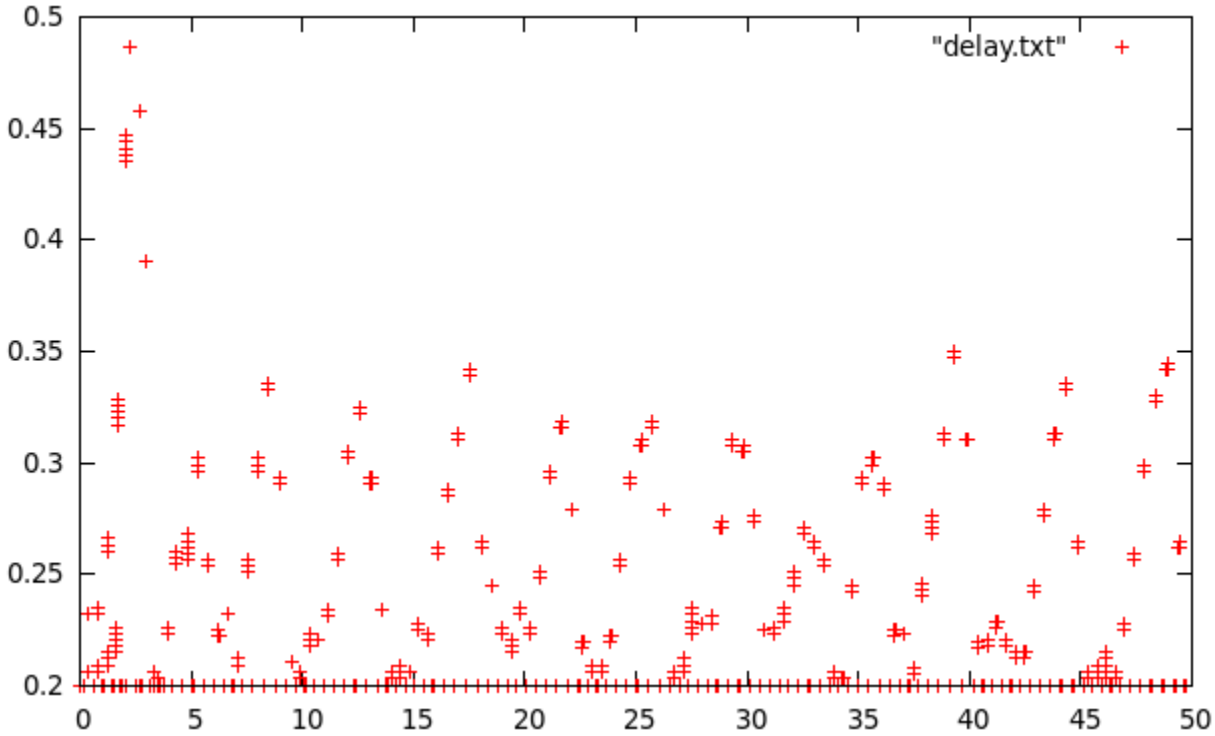


Figura 5.17: Variaţia întreruperilor (delay) în funcţie de timp pentru funcţia v1

	Numar Intreruperi	Timpul intreruperii (s)
1	1902	0.234738
2	2838	0.238602
3	2494	0.247863
4	2376	0.256565

Tabel 5.2: Detalii privind delay-ul în toate cele 4 funcţii

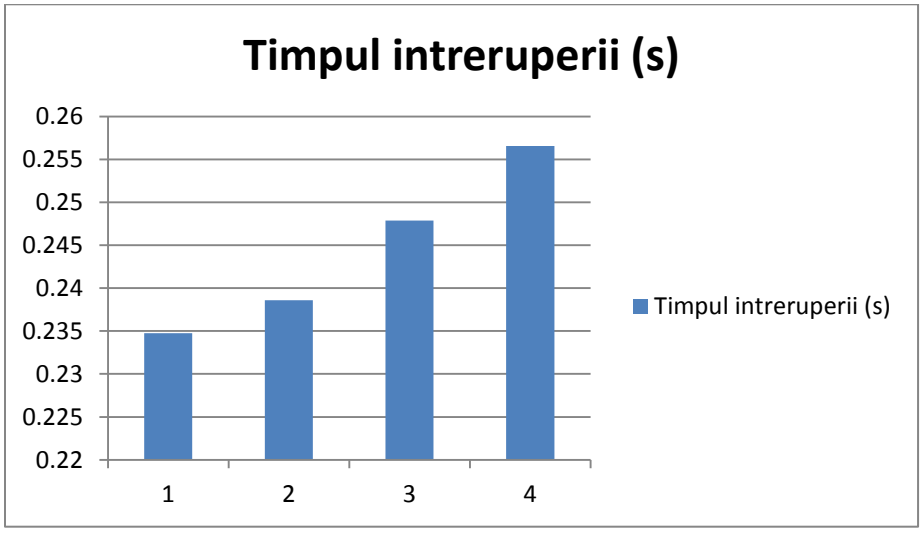


Figura 5.18: Media intervalelor de timp când au loc întreruperile pentru cele 4 funcţii

Din tabelul 5.2, cât şi din figura 5.18 se observă că funcţia v4 deţine cel mai mare număr de întreruperi, cât şi cele mai mari intervale de timp pentru acestea, ceea ce nu este un lucru bun pentru reţea. Cea mai optimă funcţie este v1 privind întreruperile.

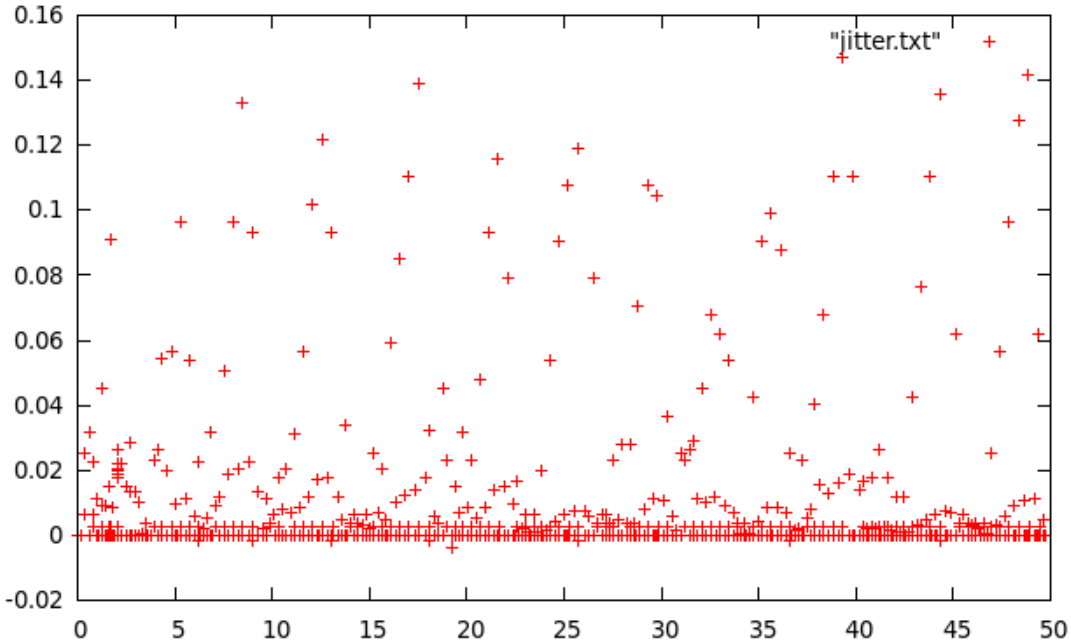


Figura 5.19: Jitter-ul (variaţia de întârziere) în funcţie de timp (pentru funcţia v1)

Jitter = ((EndTime(j)-StartTime(j)) - (EndTime(i)-StartTime(i))) / (j – i) [24]

În acest exemplu, am calculat bruiajul pentru traficul CBR prin transmisie UDP

	Valoare Jitter	Numar Jittere
1	0.003747	1902
2	0.002689	2838
3	0.00371	2494
4	0.003853	2376

Tabel 5.3 Detalii privind Jitter-ul în toate cele 4 funcţii

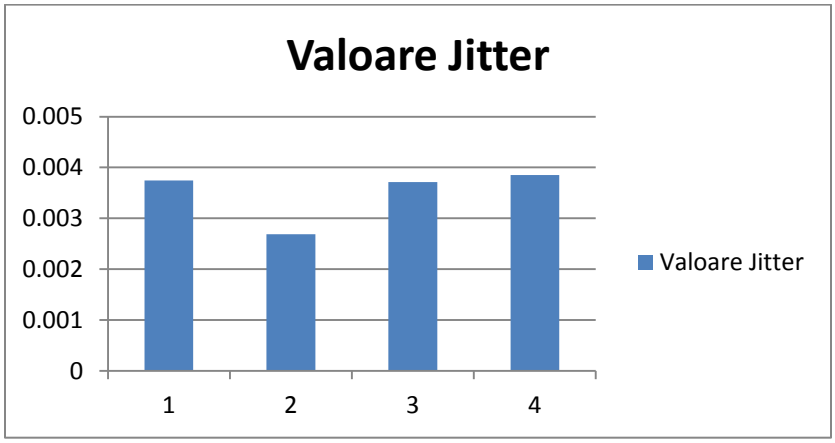


Figura 5.20: Media valorilor jitter-elor celor 4 funcţii

Din figura 5.20 se observă că valoarea medie cea mai mică a jiter-elor dintre cele 4 funcţii este cea a funcţiei v2, iar numărul de jittere cel mai mare este tot al funcţiei v2.

Concluzii

În concluzie, algoritmul Random Early Detection sau RED deţine o bună modalitate cu care congestia ce apare la nivel de gateway poate fi evitată prin intermediul protocoalelor de transport [5][3].

În momentul în care valoarea pragului maxim th_{max} va fi depăşită de către lungimea cozii medii, atunci geteway-ul va începe să ştearga pachetele din coadă. Cu alte cuvinte, folosind acest algoritm, la nivel de gateway congestia poate fi prevenită cu ajutorul informaţiilor care circulă în reţea referitoare la valoarea maximă a pragului respectiv sau a lungimii medie a cozii. Aceste lucruri se pot realiza utilizând marcarea pachetelor, aceasta făcându-se în funcţie de starea congestiei în acel moment, sau se mai pot realiza primind informaţii despre lărgimea de bandă din toată reţeaua. Cu ajutorul acestor informaţii routerele vor putea reduce numărul de pachete pe care vor să îl trimită, dacă este cazul.

Tot în cadrul acestui proiect, pornind de la algoritmul RED iniţial, cel inventat de Sally Floyd şi de Van Jacobson şi urmând modificările aduse acestui algoritm, am reuşit să compar anumite funcţii şi să obţin soluţii optime totodată pentru diferitele cazuri. Modificările au fost făcute în codul sursă al simulatorului Network Simulator şi totodată am modificat un script .tcl care avea rolul de a genera funcţiile respective, cât şi fişierele de trace cu valorile parametrilor.

Totodată, comparând funcţiile respective s-a ajuns la obţinerea performanţelor în ceea ce priveşte media de aruncare de pachete, variaţiile cozii medii sau întreruperile.

Fiecare din parametrii au fost examinaţi in detaliu, iar pentru cazul variaţiei cozii medii am obţinut performanţe de peste două ori mai bune decât algoritmul RED iniţial, iar pentru media de aruncare de pachete am obţinut performanţe cu 8% mai bune.

Bibliografie:

[1]Computer Networks 4th Edition - Andrew S. Tanenbaum
[2] http://ro.wikipedia.org/wiki/Protocol_de_control_al_transmisiei
[3] Analytic Performance Evaluation of the RED Algorithm for QoS in TCP/IP Networks - Stefan Kohler, Michael Menth and Norbert Vicari
http://www-info3.informatik.uni-wuerzburg.de/TR/tr259.pdf
[4] http://en.wikipedia.org/wiki/TCP_congestion-avoidance_algorithm
[5] Random early detection gateways for congestion avoidance - Sally Floyd and Van Jacobson
http://www.icir.org/floyd/papers/early.twocolumn.pdf
[6] http://en.wikipedia.org/wiki/Random_early_detection
[7] Quality of Service Control in High-Speed Networks - H. Jonathan Chao,Xiaolei Guo pag 215
[8] Adaptive RED: An Algorithm for Increasing the Robustness of RED's Active Queue Management - Sally Floyd, Ramakrishna Gummadi, Scott Shenker
[9]http://www.cisco.com/c/en/us/td/docs/ios/12_2/qos/configuration/guide/fqos_c/qcfconav.html#wp1003370
[10] http://networkengineering.stackexchange.com/questions/5316/how-are-weighted-fair-queuing-and-weighted-random-early-detection-related
[11] http://en.wikipedia.org/wiki/Weighted_random_early_detection
[12] RRED: Robust RED Algorithm to Counter Low-Rate Denial-of-Service Attacks - Changwang Zhang, Jianping Yin,Weifeng Chen, Zhiping Cai
[13] http://en.wikipedia.org/wiki/Robust_random_early_detection
[14] http://en.wikipedia.org/wiki/Ns_(simulator)
[15] http://en.wikipedia.org/wiki/OTcl
[16] Introduction to Network Simulator - Giovanni Neglia, Mouhamad Ibrahim
[17] Introduction to Network Simulator 2 - Noun Choi, Hai Vu, Ryan Burchfield, Sara Arbab Yazd
[18] http://www.isi.edu/nsnam/nam/index.html
[19] http://wiki.tcl.tk/16279
[20] http://www.isi.edu/nsnam/ns/tutorial/
[21] http://ajlinx.wordpress.com/2013/06/19/install-ns2-ns-allinone-2-35-on-ubuntu-12-04-for-beginners/
[22] http://www.roman10.net/modification-of-red-aqm-in-ns2/
[23] Congestion Control in Communication Network Using RED, SFQ and REM Algorithm - Dolly Kalav, Sudha Gupta
[24] School of Computer Science and Engineering Seoul National University - Min Chen

Pentru cazul referinţelor de tip link, acestea au fost verificate in data de 28.06.2014 şi toate sunt funcţionale.

Anexa 1

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

set NumSenders 15
set NumReceivers 1

#read scenario, seed and bottleneck bandwidth from the command
line input arguments
set Scenario [lindex $argv 0]
set function [lindex $argv 1]
set seed [lindex $argv 2]
puts "scenario: $Scenario; function: $function; seed: $seed"
ns-random $seed
set BufSize 100
set PktSize 1024
#set winSize 200
#in seconds
set Duration 50
#create all nodes: note that the order of creating the nodes
matter
for {set i 1} {$i <= $NumSenders} {incr i} {
    set s($i) [$ns node]
}
set r1 [$ns node]
set d1 [$ns node]
#open the nam trace file
set nf [open out.nam w]
$ns namtrace-all $nf
#open the traffic trace file to record all events
set nd [open out.tr w]
$ns trace-all $nd
#define a finish procedure
proc finish {} {
    global ns , nf , nd , qtf
    $ns flush-trace
    close $nf
    close $nd

    #puts "running nam..."
    exec nam out.nam &
    #exec xgraph out.tr -geometry 800x400 &
    exec grep "^d" out.tr > aruncm.tr
    exec grep "^+" out.tr > enqm.tr
    exec grep "^-" out.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 $qtf
}
exec rm -f temp.q ave.tr
exec touch ave.tr temp.q
exec awk $awkCode queue.txt
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
}

 #link the nodes
if { $Scenario == 1 } {
    Queue/RED set thresh_ 15
    Queue/RED set maxthresh_ 45
} elseif { $Scenario == 2 } {
    Queue/RED set thresh_ 20
    Queue/RED set maxthresh_ 60
} elseif { $Scenario == 3 } {
    Queue/RED set thresh_ 25
    Queue/RED set maxthresh_ 75
} elseif { $Scenario == 4 } {
    Queue/RED set thresh_ 30
```

```
    Queue/RED set maxthresh_ 90
}

Queue/RED set queue_in_bytes_ false
Queue/RED set gentle_ false
Queue/RED set function_ $function

for {set i 1} {$i <= $NumSenders} {incr i} {
    $ns duplex-link $s($i) $r1 10Mb 100ms DropTail
    $ns queue-limit $s($i) $r1 $BufSize
}
#r1 d1 and d1 r1 are different
$ns duplex-link $r1 $d1 3Mb 100ms RED
$ns queue-limit $r1 $d1 $BufSize
 #trace the queue: note that link r1 d1 is different from d1 r1
set redq [[$ns link $r1 $d1] queue]
set qtf [open queue.txt w]

$redq trace curq_
$redq trace ave_
$redq attach $qtf

 #set up TCP connections
for {set i 1} {$i <= $NumSenders} {incr i} {
    set tcp($i) [new Agent/TCP]
    $ns attach-agent $s($i) $tcp($i)
    set sink($i) [new Agent/TCPSink]
    $ns attach-agent $d1 $sink($i)
    $ns connect $tcp($i) $sink($i)
    $tcp($i) set fid_ $i
    $tcp($i) set packetSize_ $PktSize

    #$tcp($i) set window_ $winSize
    #set up FTP over TCP connection as traffic source
    set ftp($i) [new Application/FTP]
    $ftp($i) attach-agent $tcp($i)
    $ftp($i) set type_ FTP
}
#schedule events for the FTP agents
set StartTime [expr [ns-random]  / 2147483647.0 / 100]
puts "starttime $StartTime"
#temporarily set to 2
for {set i 1} {$i <= $NumSenders} {incr i}  {
    $ns at $StartTime "$ftp($i) start"
    $ns at $Duration+$StartTime "$ftp($i) stop"
}
```

```
#ensure the ftp application have enough time to finish, so we +1
$ns at $Duration+$StartTime+1 "finish"
 #run the simulation
$ns run
```

Anexa 2:

```
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 lm, ln, lp, lq, Xmin;
    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 {
                // p ranges from 0 to max_p as the average queue
                // size ranges from th_min to th_max
        if (edp_.func == 1) {
            lm = 1.0/(log(th_max) - log(-v_b/v_a));
            ln = -lm*log(-v_b/v_a);
            p = lm*log(v_ave) + ln;
        } else if (edp_.func == 2) {
            Xmin = -v_b/v_a;
            if (v_ave < (th_max + Xmin)/2) {
                lm = 1.4*v_a;
                lp = -lm*Xmin;
                p = lm * v_ave + lp;
            } else {
            ln = (1.4*v_a*(th_max+Xmin)/2-
1.4*v_a*Xmin)/((th_max+Xmin)/2-th_max);
                lq = 1.0 - ln*th_max;
                p = ln * v_ave + lq;
            }
        } else if (edp_.func == 3) {
            if (v_ave < (th_max - v_b/v_a)/2) {
            lm = 0.6*v_a;
            lp = v_b*lm/v_a;
            p = lm*v_ave + lp;
```

```
            } else {
            ln = (0.6*v_a*(th_max+Xmin)/2-
0.6*v_a*Xmin)/((th_max+Xmin)/2-th_max);
            lq = 1.0 - ln*th_max;
                p = ln * v_ave + lq;
            }
        } else if (edp_.func == 4) {
            lm = 1.0/(exp(th_max) - exp(-v_b/v_a));
            ln = -lm*exp(-v_b/v_a);
            p = lm*exp(v_ave) + ln;
            //printf("%f, %f, %f, %f\n", exp(th_max), exp(-
v_b/v_a), lm, v_ave);
        } else {
                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;
}
```