Universitatea “Politehnica” din Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

Algoritmi AQM cu detecția/semnalarea congestiei

Proiect de Diplomă

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

Conducător ştiinţific
Conf. dr. ing. Ştefan Stăncescu

Absolvent
Muntean Andrada

2014

ACRONYMS

ACK Acknowledgment
AQM Active Queue Management
BDP Bandwidth Delay Product
CPU central processing unit
CRC cyclic redundancy check
CWND congestion window size
DCE Distributed Computing Environment
DNS Domain Name System
FTP File Transfer Protocol
ICMP Internet Control Message Protocol
IEEE Institute of Electrical and Electronics Engineers
IP Internet Protocol
FTP File Transfer Protocol
LAN local area network
LCP link control protocol
HTTP Hypertext Transfer Protocol
MSL maximum segment lifetime
MSS maximum segment size
NS2 Network Simulator 2
OSI open systems interconnection
PDU protocol data unit
PPP Point-to-Point Protocol
PSH push flag, TCP header
RED Random Early Detection
RFC Request for Comment
RTT round-trip time
SMTP Simple Mail Transfer Protocol
SMS Systems Management Server
SSTHRESH Slow-start threshhold Value
SACK selective acknowledgment
SYN synchronize packet
TCP Transmission Control Protocol
TTL time-to-live
UDP User Datagram Protocol
VOIP Voice over Internet Protocol.
QoS Quality of Service

Cuprins

Introducere....7

1. Congestia traficului 9

1.1 Definirea Congestiei....9

1.2 Principii generale ale controlului congestiei.... 15

1.3 Politici pentru prevenirea congestiei 17

2. Tehnici cunoscute de prevenire a pierderii pachetelor în rețele 21

2.1 Introducere.... 21

2.2 TCP.... 22

2.3Algoritmi de management al cozii la receptor (AQM).... 27

3 TCP NewReno.... 33

3.1 Introducere.... 33

3.2.TCP Reno 35

3.2 Caracteristici New Reno.... 39

4 . Modificare Algoritm.... 43

4.1 Introducere.... 43

4.2 Soluţia propusă pentru modificarea algoritmului 44

5. Simulări 45

Modificarea simulatorului 47

Scenariul Simulat : 49

6. Prezentarea rezultatelor 49

7. Concluzii.... 58

8 . Bibliografie și referințe.... 61

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

(MODEL)

Subsemnatul MUNTEAN FV ANDRADA [2)], posesor al B.I./C.I./paşaport seria UX, nr. 216665, având CNP 2910727385575, am întocmit proiectul de absolvire/diplomă/lucrarea de disertaţie cu tema:[3)] ALGORITMI AQM DE DETECTIE / SEMNALARE A CONGESTIEI, în vederea susţinerii examenului de finalizare a studiilor universitare de licenţă/masterat, organizat de către Facultatea de ELECTRONICĂ, TELECOMUNICAŢII ŞI TEHNOLOGIA INFORMAŢIEI Departamentul CALCULATOARE ŞI TEHNOLOGIA 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: 26.06.2014 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.

Introducere

În aceastã lucrare propun o modificare a algoritmului New Reno în etapa de “Fast recovery” a prafului de Slow Start threshold . Aceastã modificare urmãreste sa aduca o imbunãtire a ferestrei de congestive şi a lãţimii de bandã. Urmãresc comportamentul algoritmilor New Reno, Reno si Tahoe în comparaţe cu algoritmul New Reno modificat propus de mine. Deoarece New-Reno are la bază algortimul Reno, modificat la partea de Fast-recovery, iar Reno la rândul său are la bază algotimul Tahoe, imbunătăţind partea de fast Recovery, iar Tahoe este alcătuit din Slow start, congestion avoidance and fast retransmit, dacă se imbunataseste una din cele patru părţi ce ajută la funcţionarea algortimului (Slow start, congestion avoidance and fast retransmit, fast-recovery), acesta va reprezenta o versiune îmbunătăţit.

Problema atât cu Reno cât şi cu NewReno este că în algoritmul de recuperare rapidă, TCP înjumătăţeşte fereastra de congestie, indiferent de starea rețelei. O altă problemă cu NewReno este că, atunci când nu există pierderi de pachete, dar pachetele sunt reordonate cu mai mult de trei confirmări duplicat, NewReno intră în mod eronat în recuperare rapidă, iar acest aspect duce iar la înjumătăţirea fereastrei de congestie. Deoarece fereastra TCP de congestie controlează numărul de pachete pe care un expeditor TCP le poate trimite în rețea, în orice moment, prin urmare, procesul de stabilire a dimensiunii ferestrei de congestie la jumătate din valoarea face ca algoritmul NewReno TCP să fie ineficient în ceea ce privește utilizarea capacităţii de legătură a algoritmului ce stă la baza lui .

În lucrarea de faţã, propun o modificare a fazei de recuperare rapidă, pe care o sa o compar pe aceeaşi topologie de retea cu performanţele algoritmilor New Reno, Reno si Tahoe. Urmaresc comportamentul ferestrei de congestive , întârzierea pachetelor, laţimii de bandã , numãrul de pachete pierdute , Numarul de pachete pierdute , întârzierea intre pachete si întârziera end-to-end.

Scopul lucrării este cel de a îmbunatăţii modul de funcţionare a algoritmului New Reno .

1. Congestia traficului

1.1 Definirea Congestiei

Congestia se poate defini în raport cu supraîncărcarea reţelei sau reacţia reţelelor la încărcări mari sau foarte mari. Congestia apare atunci când pe o legătură se doreşte a fi transmis un volum de date mai mare decât capacitatea sa. Sarcina de a transmite un volum de date mai mare decât capacitatea legăturii pe care se va transmite , fenomenul cauzând pierderi de pachete transmise prin reţea, se numeşte congestie.

Deoarece se foloseşte un singur mijloc de comunicare, performanţele reţelei scad o dată cu creşterea pachetelor transferate în reţea, numărul de utilizatori şi a dimensiunilor aplicaţiilor .

Factorii care duc la producerea congestiei într-o reţea sunt :

- cantitatea traficului;
- dimensiunea pachetelor;
- dimensiunile fizice ale reţelei.

Timpul de rãspuns al reţelei depinde de încărcarea acesteia . În funcţie de numărul de coliziuni, care cresc o dată cu încărcarea, determinând retransmişii ce încarcã şi mai mult reţeaua, va descreşte sau va creşte performanţa reţelei .

Ceea ce duce ca o reţea să fie congestionată poate să fie atât supraîncărcarea bufferelor echipamentelor de reţea şi implicit a cozilor rezultând pierderi de pachete), ca şi cazul în care bufferele nu sunt supra-încărcate ci doar utilizatorul nu are o disponibilitatea a serviciilor reţelei ce îi poate satisface nevoile aplicaţiei folosite .

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

1) Eficienţă

2) Heterogenitate

3) Capacitatea de a funcţiona cu surse defecte

4) Stabilitate

5) Scalabilitate

6) Simplitate

7) Echitate

Eficienţa

Sunt două aspecte de menţionat în ceea ce priveşte eficienţa unei scheme de control a congestiei. În primul rând trebuie să se ţină cont de cantitatea de trafic suplimentar pe care schema de control a congestiei o aduce în reţea. Am dori desigur că o asemenea schemă să presupună o povară minimă pentru reţea. În al doilea rând trebuie să ne gândim dacă schema de control nu conduce la o subutilizare a resurselor critice ale reţelei. Scheme de control ineficiente pot "gâtui" sursele chiar dacă nu există pericol de congestie, conducând la o subutilizare a resurselor. Nu se doreşte să se lucreze suboptimal într-o reţea.

Heterogenitatea

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

Capacitatea de a funcţiona cu surse defecte

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

Stabilitatea

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

Scalabilitatea

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

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

Simplitatea

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

Echitatea

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

Când se vorbeşte despre echitate, sunt două aspecte de care trebuie ţinut cont: definirea şi implementarea. Numeroase criterii de definire a echităţii au fost propuse de-a lungul timpului dar nici unul nu este absolut. Implementarea unui crieteriu de echitate generează probleme care uneori depăşesc domeniul controlului congestiei. De exemplu, s-ar putea decide că este echitabil să se avantajeze anumite surse care sunt dispuse să plătească pentru acest lucru sau care trimit un anumit tip de pachete (mai importante - email). [1]

Toţi algoritmi de gestionare a cozii active inventaţi până acum au fost proiectati pentru a asigura alocarea de lățime de bandă corect între fluxurile TCP, care sunt proiectate să coopereze cu clasicul TCP de control al congestiei.

Este recunoscut faptul că combinaţia/combinarea mecanismului de combatere a congestiei împreună cu algoritmul de routara prin cozi nu suportă o împărţire corectă a benzii între cele două linkuri. În particular, conexiunile cu RTT-uri nu primesc raţia cuvenită în ceea ce priveşte lungimea de bandă. Mai mult, mecanismele menţionate mai sus nu se supun normelor de control al congestiei. Problema va fi abordată în două perspective. Prima, a fost propus un algoritm de tratare a congestiei cu ajutorul mai multor variante corecte de TCP (Hybla TCP – ce realizează alocarea de lăţime de bandă justă prin intermediul supracompensarii ferestrei de congestie pentru conexiunile cu RTT-uri lungi.)

Dacă debitul cu care intră pachete într-un nod depăşeşte fie capacitatea de prelucrare a nodului,fie capacitatea legăturii prin care pachetele trebuie să iasă, nodul memorează pachetele într-o structură de coadă, de unde le extrage pe măsură ce pot transmise prin legătura de ieşire. Un efect imediat este creşterea timpului de propagare, din cauza staţionării pachetelor în coada de aşteptare. Dacă excesul de debit de intrare se păstrează mai mult timp, coada creşte până când memoria alocabilă cozii de aşteptare este epuizată; în acel moment, nodul va trebui fie să sacrifice pachete, fie să solicite, prin intermediul mecanismului de control al fuxului de la nivelul legăturii de date, micşorarea debitului de intrare. De notat că reducerea şi, în extremis, blocarea fuxului de date la intrarea într-un nod poate duce la acumularea de date de transmis în nodul vecin dinspre care vine acel ux, duciand mai departe la blocarea reciprocă a unui grup de noduri.

Principalele probleme ce apar în cazul în care capacitatea nodurilor sau legăturilor directe este depăşită sunt următoarele:

1) Într-o reţea aglomerată, pachetele sau datagramele întârzie mult sau chiar se pierd, lucru care poate declanşa retrimiteri intempestive de pachete, ducând la aglomerare şi mai mare a reţelei şi la performanţe şi mai scăzute. O astfel de situaţie, de degradare suplimentară a performanţelor în urma creşterii încărcării, se numeşte *congestie* şi trebuie evitată sau, cel puţin, ţinută sub control.
2) Capacitatea disponibilă a reţelei trebuie împărţită în mod echitabil între utilzatori.
3) Diferite aplicaţii au diferite priorităţi cu privire la caracteristicile necesare ale serviciului oferit de reţea. Reacţia unei reţele aglomerate trebuie să ţină cont de aceste priorităţi. De exemplu, un nod supraaglomerat poate să fie nevoit să arunce o parte dintre datagramele aflate în tranzit.

Dacă datagramele aparţin unei aplicaţii de transfer de fişiere, este preferabil să fie aruncate cele mai recente (acestea find retransmise mai târziu; dacă se aruncă datagramele mai vechi, este posibil ca destinatarul să nu aibă ce face cu cele mai noi şi să trebuiască retransmise toate).

Pentru exemplificare, să considerăm o reţea în care, dintr-un anumit motiv, rata de sosire a pachetelor la un ruter depăşeşte rata de deservire a acestuia. În acest moment, pachetele sunt introduse într-o coadă de aşteptare, ceea ce conduce la întârzieri. Întârzierea adiţională poate determina sursele să retransmită, crescând astfel şi mai mult încărcarea reţelei. Acest tip de feedback conduce la o deteriorare rapidă a situaţiei, unde traficul este dominat de retransmisii şi productivitatea efectivă – banda efectivă (throughput) scade brusc. Mai departe, dacă este implementat un mecanism de control al traficului „router-to-router", noi pachete pot să nu mai

fie admise în coadă, aşa că acestea pot inunda un ruter precedent de asemenea. Se poate ajunge astfel la o situaţie de tipul „**deadlock**”, în care întreg traficul este îngheţat.

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

Aceste definiţii nu sunt satisfăcătoare din mai multe motive. În primul rând, întârzierile şi pierderile de pachete sunt indicatori de performanţă ce sunt impropriu folosiţi drept indicatori pentru congestie, dat fiind că modificarea acestor indicatori se poate datora unor fenomene diferite de congestie. În al doilea rând, definiţia nu specifică în mod exact punctul de la care o reţea poate fi considerată congestionată. De exemplu, în timp ce o reţea care are timpi medii de întârziere în fiecare ruter de 1 până la 10 durate de deservire nu este cu siguranţă congestionată, nu este clar dacă o reţea ce are timpi medii de întârziere de 1000 durate de deservire este sau nu congestionată. Nu pare posibil să se stabilească o valoare de prag care să determine congestia! Nu în ultimul rând, o reţea congestionată din punctul de vedere al unui utilizator nu este neapărat congestionată din punctul de vedere al altuia. De exemplu, dacă utilizatorul A poate tolera o rată de pierdere a pachetelor de 1 la 1000, şi utilizatorul B poate tolera o rată de pierdere de 1 la 100, şi rata de pierdere efectivă este de 1 la 500, atunci A vă considera că reţeaua este congestionată, în timp ce pentru B reţeaua va funţiona în parametri normali. O reţea trebuie considerată necongestionată numai dacă toţi utilizatorii săi sunt de acord cu presupusa stare.

Din cele prezentate mai sus, deduce faptul cã starea de congestionare a unei reţele depinde de perspectiva utilizatorului. Un utilizator care cere puţin de la o reţea poate tolera o pierdere de performanţă mult mai bine decât un utilizator cu cerinţe mari. De exemplu, un utilizator care foloseşte reţeaua numai pentru e-mail va fi muţumit cu o întârziere de o oră, pe când acestă „performanţă” nu este acceptabilă pentru un utilizator care foloseşte reţeaua pentru transimiuni de voce în timp real. Punctul cheie este noţiunea de ***folos***, folos pe care un utilizator îl obţine de la reţea, şi modul în care acest folos se degradează odată cu creşterea încărcării reţelei.

Definiţie

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

Observaţii:

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

O reţea care controlează congestia trebuie să fie sensibilă la funcţiile de utilitate ale utilizatorilor şi să fie capabilă să îşi direcţioneze resursele astfel încât să nu existe pierderi de

utilitate odată cu creşterea încărcării. Aşadar, reţeaua trebuie să fie capabilă să facă diferenţa între conversaţii şi să prioritizeze conversaţiile în funcţie de stringenţa utilităţii participanţilor la o anumită conversaţie.

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

În figura 1.1 se prezintă simptomele unei reţele congestionate.

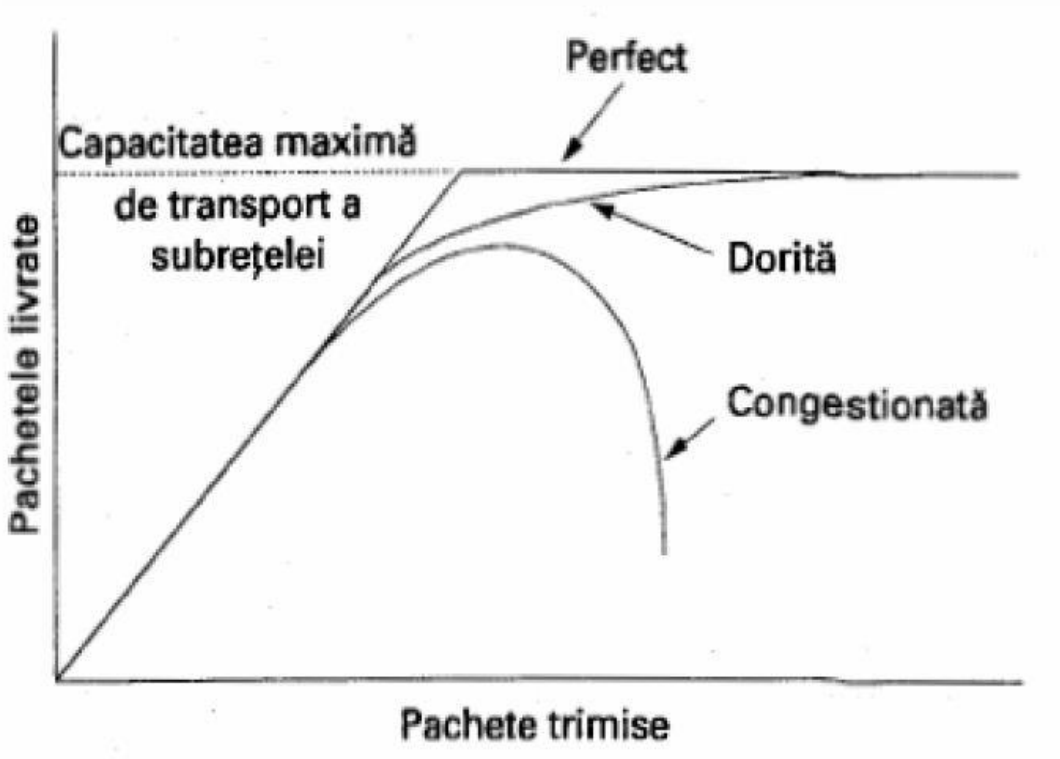


Figura 1.1: Dacă traficul este prea intens, apare congestia şi performaţele se degradează puternic.

O alta cauză sunt procesoarele lente care pot cauza congestia. Dacă unitatea centrală a ruter-ului (CPU) este lentă în execuţia funcţiilor sale (introducerea în cozi, actualizarea tabelelor etc.), se pot forma cozi, chiar dacă linia de comunicaţie nu este folosită la capacitate maximă. Similar şi liniile cu lăţime de bandă scăzută pot provoca congestia. Schimbarea liniilor cu unele mai performante şi păstrarea aceluiaşi procesor sau vice-versa de obicei ajută puţin, însă de cele

mai multe ori nu fac decât să deplaseze punctul critic. Adevărata problemă este de multe ori o incompatibilitate între părţi ale sistemului. Ea va persista până ce toate componentele sunt în echilibru. [1]

Este important de subliniat diferenţa dintre controlul congestiei şi controlul fluxului, deoarece relaţia este subtilă.

Controlul congestiei trebuie să asigure că subreţeaua este capabilă să transporte întreg traficul implicat. Este o problemă globală, implicând comportamentul tuturor calculatoarelor gazdă, al tuturor ruterelor, prelucrarea de tip store-and-forward din rutere şi toţi ceilalţi factori care tind să diminueze capacitatea de transport a subreţelei.

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

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

La cealaltă extremă, să considerăm o reţea de tip store-and-forward cu linii de 1 Mbps şi 1000 de calculatoare mari, din care jumătate încearcă să transfere la 100 kbps către cealaltă jumătate. Aici problema nu apare datorită unui emiţător rapid care surclasează un receptor lent, ci pentru că traficul cerut depăşeşte posibilităţile reţelei. Motivul pentru care controlul congestiei şi controlul fluxului sunt adesea confundate este acela că unii algoritmi pentru controlul congestiei funcţionează trimiţând mesaje înapoi către diferitele surse, spunându-le să încetinească atunci când reţeaua are probleme. Astfel, un calculator gazdă poate primi un mesaj de încetinire fie din cauză că receptorul nu suportă încărcarea, fie pentru că reţeaua este depăşită.[3]

1.2 Principii generale ale controlului congestiei

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

Multe din problemele care apar în sistemele complexe, cum ar fi reţelele de calculatoare, pot fi privite din punctul de vedere al unei teorii a controlului. Această abordare conduce la

împărţirea tuturor soluţiilor în două grupe: în buclă deschisă şi în buclă închisă. Soluţiile în buclă deschisă încearcă să rezolve problema printr-o proiectare atentă, în esenţă să se asigure că problema nu apare. După ce sistemul este pornit şi funcţionează, nu se mai fac nici un fel de corecţii.

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

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

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

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

Al doilea pas în bucla de reacţie este transferul informaţiei legate de congestie de la punctul în care a fost depistată la punctul în care se poate face ceva. Varianta imediată presupune trimiterea unor pachete de la ruterul care a detectat congestia către sursa sau sursele de trafic, pentru a raporta problema. Evident, aceste pachete suplimentare cresc încărcarea reţelei exact la momentul în care acest lucru era cel mai puţin dorit, subreţeaua fiind congestionată.

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

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

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

Se cunosc numeroşi algoritmi pentru controlul congestiei. Pentru a oferi o modalitate de organizare a lor, Yang şi Reddy (1995) au dezvoltat o taxonomie pentru algoritmii de control al congestiei. Ei încep prin a împărţi algoritmii în cei în buclă deschisă şi cei în buclă închisă, aşa

cum s-a precizat anterior. În continuare împart algoritmii cu buclă deschisă în unii care acţionează asupra sursei şi alţii care acţionează asupra destinaţiei. Algoritmii în buclă închisă sunt de asemenea împărţiţi în două subcategorii, cu feedback implicit şi cu feedback explicit. În algoritmii cu feedback explicit, pachetele sunt trimise înapoi de la punctul unde s-a produs congestia către sursă, pentru a o avertiza. În algoritmii impliciţi, sursa deduce existenţa congestiei din observaţii locale, cum ar fi timpul necesar pentru întoarcerea confirmărilor.

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

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

1.3 Politici pentru prevenirea congestiei

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

Prevenirea congestiei se bazează pe un set de mecanisme ce ajută cozile echipamentelor să evite congestia. Cozile se supraîncărca atunci când traficul sosit pe interfeţele de intrare depăşeşte capacitatea interfeţelor de ieşire. Atunci când mai mult trafic soseşte pe interfeţele de ieşire decât acestea pot suporta, cozile (memoriile && bufferele) routerelor se încarcă . Mecanismele de management al cozilor ne pot ajuta să evităm supraîncărcarea cozilor şi respectiv congestia. Practic, aceste mecanisme ne oferă posibilitatea de a alege între pierderea de pachete şi întârzierea sau jitterul – variaţiile în timp ale întârzierilor (parametri critici pe reţea).

Să începem cu nivelul legătură de date şi să ne continuăm apoi drumul spre niveluri superioare. Politica de retransmisie stabileşte cât de repede se produce timeout la un emiţător şi ce transmite acesta la producerea timeout-ului. Un emiţător foarte activ, care produce repede timeout şi retransmite toate pachetele în aşteptare folosind retransmiterea ultimelor *n,* va produce o încărcare mai mare decât un emiţător calm, care foloseşte retransmiterea selectivă. Strâns legată de acestea este politica de memorare în zonă tampon. Dacă receptorii distrug toate pachetele în afara secvenţei, acestea vor trebui retransmise ulterior, introducând o încărcare

suplimentară. În ceea ce priveşte controlul congestiei, repetarea selectivă este, în mod sigur, mai bună decât retrimiterea ultimelor *n*.

În Tabelul 1.1. sunt prezentate diferite politici pentru nivelul legătură de date, reţea şi transport, care pot influenţa congestia

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

Tabelul 1.1

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

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

Un algoritm de dirijare bun poate ajuta la evitarea congestiei prin răspândirea traficului de-a lungul tuturor liniilor, în timp ce un algoritm neperformant ar putea trimite toate pachetele

pe aceeaşi linie, care deja este congestionată. Gestiunea timpului de viaţă asociat pachetelor stabileşte cât de mult poate trăi un pachet înainte de a fi distrus. Dacă acest timp este prea mare, pachetele pierdute vor încurca pentru mult timp activitatea, iar dacă este prea mic, este posibil să se producă timeout înainte de a ajunge la destinaţie, provocând astfel retransmisii.

La nivelul transport apar aceleaşi probleme ca la nivelul legăturii de date dar, în plus, determinarea intervalului de timeout este mai dificil de realizat, deoarece timpul de tranzit prin reţea este mai greu de prezis decât timpul de tranzit pe un fir între două rutere. Dacă intervalul de timeout este prea mic, vor fi trimise inutil pachete suplimentare. Dacă este prea mare, congestia se va reduce, însă timpul de răspuns va fi afectat de pierderea unui pachet. [4]

2. Tehnici cunoscute de prevenire a pierderii pachetelor în rețele

2.1 Introducere

TCP (Transmission Control Protocol) realizează fragmentarea mesajelor în pachete şi asigură transmiterea corectă a mesajelor între utilizatori. Pachetele unui mesaj sunt numerotate, putându-se verifica primirea lor în forma în care au fost transmise şi reconstituirea mesajelor lungi, formate din mai multe pachete.

TCP este un protocol orientat pe conexiuni, care permite ca un flux de octeţi trimişi de la sursă să ajungă fără erori la destinaţie .

Principalele caracteristici ale TCP sunt:

- Transfer de date în flux continuu - datele circulă în acelaşi timp, în ambele sensuri ale conexiunii.

- Siguranţa transmisiei - recuperează pachetele transmise cu erori, pierdute sau cu număr de secvenţă eronat.

- Controlul fluxului de date – în transferul de date dintre două procese, când aplicaţia destinaţie trimite o confirmare către emitent, se indică şi numărul permis de octeţi ce se pot recepţiona, pentru a se asigura că transmiterea rapidă de mesaje de către un emiţător, nu face ca un receptor lent să primească mai multe mesaje decât poate prelucra. În urma unui astfel de mesaj, emiţătorul îşi va dimensiona pachetele transmise la lungimea indicată de receptor.

- Multiplexarea - permite mai multor procese, care rulează pe acelaşi host, să utilizeze facilităţile protocolului TCP simultan.

- Controlul conexiunii - presupune stabilirea numărului de secvenţă şi a dimensiunii ferestrei, pentru fiecare pachet TCP.

Pentru a realiza transferul de date, TCP fragmentează fluxul de octeţi în pachete şi transmite fiecare pachet la nivelul Internet. La destinaţie, procesul TCP receptor reasamblează pachetele primite într-un flux de ieşire. Înainte de a începe transferul datelor, între cele două procese utilizator (programe de aplicaţie) se stabileşte, prin intermediul reţelei, o conexiune logică numită circuit virtual.

Livrarea la destinaţie a fluxului de octeţi în ordinea în care au fost emişi, fără pierderi şi fără duplicate, se realizează prin folosirea unei tehnici de confirmare pozitivă cu retransmitere

(dacă confirmarea de primire nu a fost transmisă într-un timp prestabilit, pachetul este transmis din nou) , combinată cu fereastră glisantă (sistem de confirmare în espectativă) . Mecanismul ferestrei glisante, utilizat de TCP, operează la nivelul octeţilor şi nu al pachetelor, permiţând o transmisiune eficientă şi un control al fluxului.

2.2 TCP

TCP implementează un mecanism de control al fluxului pe bază de fereastra. Propriu zis vorbind, un protocol pe bază de fereastră înseamnă că dimensiunea ferestrei curente definește o limită superioară a cantităţii de date nerecunoscută, care poate fi în tranzit între o pereche dată expeditor-receptor. Inițial, fluxul de control al TCP-ului a fost guvernat doar de dimensiunea ferestrei maximă permisă anunțate de către receptor și politică care a permis expeditorului să trimită noi pachete numai după ce a primit unidirecționat pachetul anterior.

Principalul scop al TCP este de a asigura circuit logic sigur sau serviciu de conexiune intre doua procese pereche. El nu se bazeaza pe siguranta altor protocoale de nivel inferior (cum este IP), asa ca TCP trebuie sa garanteze el insusi siguranta transmisiei. TCP poate fi caracterizat prin urmatoarele facilitati pe care le asigura pentru aplicatiile care il utilizeaza:

- Transfer de date in flux (Stream Data Transfer)

Din punctul de vedere al aplicatiei, TCP transfera un sir (stream) continuu de octeti prin retea. Aplicatia nu trebuie sa-si faca griji cu segmentarea datelor in blocuri de baza sau datagrame. O face TCP prin gruparea octetilor in fragmente TCP, care sunt transferati IP pentru a fi transmisi la destinatie. De asemenea, insusi TCP decide cum sa segmenteze datele si poate trimite datele mai departe intr-un mod convenabil.Uneori, o aplicatie are nevoie sa fie sigura ca toate datele trasnsmise catre TCP au fost in realitate transmise la destinatie. Pentru aceasta, o functie `push` este definita. Ea va forta transmiterea tuturor segmentelor TCP ramase in memorie catre destinatie. Functia de inchidere (`close`) normala forteaza si ea transmiterea datelor la destinatie.

- Siguranta

TCP atribuie un numar de secventa fiecarui octet transmis si asteapta o confirmare (acknowledgment - ACK) de la aplicatia care receptioneaza datele TCP. Daca confirmarea nu vine intr-un interval de timp prestabilit, datele sunt transmise din nou. Deoarece datele sunt transmise in blocuri (segmente TCP) numai numarul de secventa al primului octet este trimis calculatorului destinatie. Aplicatia TCP destinatie foloseste numerele de secventa pentru a le ordona atunci cand sosesc neordonate si sa elimine segmentele duplicate.

- Controlul transmisiei (Flow Control)

Aplicatia TCP destinatie, cand trasmite o confirmare (ACK) catre emitent, indica de asemenea numarul de octeti pe care ii poate receptiona pe langa ultimul segment TCP primit, fara sa se

suprancarce sau sã aparã depasirea memoriilor tampon (internal buffers) ale sale. Aceasta este trimis in ACK sub forma numarului cel mai mare de secventa pe care-l poate receptiona fara probleme. Acest mecanism este cunoscut sub numele de fereastra glisanta (window-mechanism) si va fi discutat in detaliu in acest capitol.

- Multiplexarea

Este realizata prin utilizarea porturilor, la fel ca si la UDP.

- Conexiunile logice

Siguranţa si mecanismele de control ale transmisiei descrise mai înainte necesitã ca TCP sã iniţializeze si menţina cateva informaţii referitoare la starea fiecarui flux de date (data stream). Aceasta informaţie de stare ce include socketurile, numerele de secvenţã, dimensiunile ferestrelor se numeste conexiune logica. Fiecare conexiune este unic identificatã de cãtre o pereche de socketuri utilizate de cãtre procesele emitente si receptoare.

- Full Duplex

TCP asigurã doua fluxuri concurente de date in ambele sensuri.

În afară de advertismentul ferestrei receptorului, awnd, controlul congestiei TCP introduce două noi variabile pentru conexiune: fereastra de congestie "CWND" şi pragul de start lent. Dimensiunea ferestrei transmiţătorului, w, a fost definită ca:

$$w = \min(cwnd, awnd) \qquad (2.1)$$

în loc să fie egală cu *awnd*. Fereastra de congestie poate fi considerată ca fiind o contra-fereastră de avertizare, unde variabila "*awnd*" este folosită pentru a atenţiona pe transmiţător sa suprafolosească resursele receptorului îm timp ce scopul "*cwnd*-ului" este acela de a preveni transmiţătorul de a trimite mai multe pachete decât reţeaua poate suporta în condiţiile de încărcare curentă.

Controlul congestiei (reactivă) = se aplica după ce rețeaua este supraîncărcată

Congestie Evitarea (Proactive) = se aplica înainte sa devina rețeaua supraîncărcată

TCP (Transport Controll Protocol) este un protocol punct la punct, orientat pe conexiune, introdus pentru a proteja rețeaua împotriva supraîncărcării puternice și pentru a realiza o împărțire echitabilă a lățimii de bandă pentru diferite surse TCP. Protocolul TCP detectează erori precum pachete eronate, pierdute sau duplicate. Emițătorul atribuie o secvență numerică fiecărui pachet (segment) și solicită o confirmare pozitivă (ACK) de la receptor. Receptorul detectează

pierderea de pachete verificând secvența numerică și confirmând, la fiecare pachet primit, ultimul pachet recepționat în ordinea corectă. Pentru fiecare pachet care nu este în ordine, o confirmare duplicat este trimisă. Acest ACK duplicat are rolul de a notifica perechea de faptul că a fost detectată o pierdere, și pentru a îi transmite emițătorului următoarea secvență numerică așteptată.

Protocolul TCP asigură un flux de octeţi sigur, printr-o conexiune nesigură, cap la cap, a reţelei sau inter-reţelei. Inter-reţeaua este formată prin asocierea unor reţele care diferă ca: topologie, lăţime de bandă, întârzieri, dimensiuni de pachete, etc. TCP se adresează dinamic la reţeaua Internet şi este robust la mai multe tipuri de defecte. Entitatea de transport TCP gestionează fluxurile TCP si interfeţele către IP. Conexiunile TCP sunt:
-duplex;
-punct-la-punct (TCP nu suportă difuzarea);
-orientate pe fluxuri de octeţi nu de mesaje.

Fluxurile de date de la procesele utilizator sunt împărţite în segmente de maxim 64 kB, tipic de 1500 de octeţi. Apoi fiecare segment este expediat ca o datagramă IP separată. Datagramele IP recepţionate sunt furnizate entităţii TCP care reconstruieşte fluxul original de octeţi, în ordine, deoarece IP nu asigură livrarea sau transportul ordonat al datagramelor. Serviciul TCP este asigurat prin crearea de către emiţător şi receptor a unor puncte finale numite socluri sau socket-uri. Portul este un punct de acces la servicii de nivel transport, TSAP (Transport Service Acces Point). Numerele de port mai mici decât 256 sunt alocate porturilor general cunoscute, rezervate serviciilor standard.(telnet-23, http -80).

Serviciile TCP principale asigurate de TCP sunt:
- **expedierea datelor (SEND),** atunci când e convenabilpentru TCP,
- **urgentarea expedierii (PUSH),** cere transmitereaimediată dacă e posibil,
- **urgentarea recepţiei (URGENT DATA):** transmiterea şi recepţia imediată pentru a întrerupe o prelucrare distantă, deoarece PUSH nu are efect la distanţă, dar URGENT DATA are.

Formatul antetului TCP

Antetul TCP are 20 octeţi plus o parte opţională. Segmentul TCP, format din antet şi date, are lungimea maximă de 64kB, adică 65.536 B, dar e limitat de **MTU (Maximum Transfer Unit)** al fiecărei reţele traversate. Eventual reţelele din cale mai fragmentează segmentul TCP. Fiecare fragment va avea propriile antete TCP şi IP.

<table>
<tr><td colspan="7">Port SursA</td><td>Port destinatie</td></tr>
<tr><td colspan="8">Numar de secventa</td></tr>
<tr><td colspan="8">Numar de confirmare</td></tr>
<tr><td>Lungime Antet (4)</td><td>Rezervati (6)</td><td>URG</td><td>ACK</td><td>PSH</td><td>RST</td><td>SYN</td><td>FIN</td><td>Dimensiunea ferestrei</td></tr>
<tr><td colspan="7">Suma de Control</td><td>Indicator URGENT</td></tr>
<tr><td colspan="8">Optiuni (1-n) cuvinte pe 32 biti</td></tr>
<tr><td colspan="8">Date (optional)</td></tr>
</table>

2.1 Formatul Antetului TCP

Unde:

- Portul sursa : Numãrul portului sursa pe 16 biti folosit de receptor pentru a raspunde.
- Portul destinatie: Numãrul portului destinatie pe 16 biti.
- Numarul de secventa: Numãrul de secventa al primului octate de date din acest segment. Daca bitul de control SYN este setat, numarul de secventa este numarul de secventa initial (n) si primul octet de date este n+1.
- Numarul de confirmare: Daca bitul de control ACK este setat, acest camp conţine valoarea urmatorului numãr de secventa pe care receptorul se asteaptã sã-l primeascã.
- Data Offset: Numarul de cuvinte de 32 biti din antetul TCP. El indica unde incep datele.
- Reserved: Şase biţi rezervaţi pentru utilizare in viitor; trebuie sa fie zero.
- URG: Precizeazã cã articolul pointer de urgenta are semnificatie in acest segment.
- ACK: Precizeazã cã articolul de "Numar de secventa confirmat" are semnificatie in acest segment.
- PSH: Functia push.
- RST: Reseteazã conexiunea.
- SYN: Sincronizeazã numerele de secventa.
- FIN: Nu mai sunt date de la emitent.
- Fereastra: Folosit in segmenteleACK. El specifica numarul de octeti de date incepand cu acela indicat in "numarul de secventa confirmat" pe care receptorul (adica emitentul acestui segment) doreste (si poate) sa accepte.
- Suma de control: Complementul faţã de unu al sumei in complement faţã de unu al tuturor cuvintelor de 16 biţi din pseudo- antet, antetul TCP si datele TCP. La calcularea sumei de control, campul "Suma de control" este considerat zero.
- indicatorul de urgenţă – permite identificarea poziţiei unor date de urgenţă, în cadrul protocolului TCP;

- date – datele protocolului nivelului superior

Mecanismul de comunicare TCP

Stabilirea conexiunii:
- înţelegerea în 3 paşi (hand-shake) sau comunicarea cu confirmare SYN = 1 arată o cerere de conectare **datelor Terminarea conexiunii**
- fluxul: fiecare octet e numerotat modulo 2
- antetul conţine numărul de secvenţă al primului octet

- controlul fluxului: prin credite (număr de octeţi)
- datele sunt transmise când vrea entitatea TCP: - PUSH - transmite acum
-urgent: transmite această dată din fluxul normal care are indicatorul urgent
-dacă se recepţionează un TPDU nepotrivit acestei conexiuni, se pune reset =1 în segmentul care pleacă

Terminarea conexiunii
- deconectarea normală cu FIN = 1;
- deconectarea bruscă (abort) cu pierdere de date.

Politici de implementare

- **emisia** trebuie aleasa **dimensiunea segmentului** şi a **ferestrei de transmisie**:
  - dacă segmentul este prea mic, antetul este o încărcare prea mare
  - dacă segmentul este prea mare, rezultă întârzieri prea mari
  - alegerea ferestrei este o problemă la liniile cu lăţime de bandă mare şi întârziere mare: cu fereastra de 64 kB şi linia T3 de 4,736 Mbps, transmiterea a 64 KB durează 12 msec. Dacă RTT (Round Trip Time), întîrzierea dus-întors este de 50 msec, atunci 75 % din timp emiţătorul este inactiv, aşteptând confirmări.
- **livrarea datelor**
  - datele pot fi memorate sau sunt livrate imediat, ordonat. Cu menţiunea urgent sunt livrate imediat, dacă e posibil
  - segmentele în afara secvenţei sunt acceptate sau rejectate
  - la expirarea timpului se retransmite numai primul segment, toate, sau individual, (se menţin cronometre separate per segment)
  -confirmarea se face imediat sau cumulat (se aşteaptă date în sens opus sau expirarea timpului)
- **controlul fluxului şi al congestiei**

Există trei ferestre, pentru emisie W_t, pentru receptive W_{re}, si pentru congestive . Se alege fereastra de emisie

$$W_t = \min\ (\ W_{re}, Wc)\ \ (2.2)$$

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

TCP reacționează la pierderea pachetelor în două feluri diferite. Setează câte un cronometru pentru fiecare pachet trimis. În cazul în care nu este primită o confirmare pentru acel

pachet, înainte de expirarea timpului, CWND își reduce dimensiunea la dimensiunea maximă a segmentului iar SSTHRESH este setat la *max:*

$$\text{SSTHRESH} = \frac{flightsize}{2.2 * \text{dimsegment}} \quad (2.3)$$

Flight size reprezintă cantitatea de informație neconfirmată în rețea. Presupunând că emițătorul este saturat, valoarea CWND este egală cu valoarea flight size. Când emițătorul recepționează trei ACK-uri duplicat, înainte de expirarea timpului, este realizată o recuparare rapidă (fast recovery): într-o manieră simplificată, TCP trimite pachetul pierdut din nou, reduce SSTHRESH la *max* și setează CWND la noua valoare calculată SSTHRESH.

Congestia reţelelor este o cauză a distribuției inegală a resurselor de rețea și a fluxului de rețea, iar algoritmii de control al congestiei pot fi împărţiţi în algoritmi „Source” şi „link”. Algoritmul de control al congestiei TCP este cel mai utilizat algoritm „Source”, dar utilizarea TCP-ului nu a putut evita posibilitatea de apariție a congestiei în rețele. Algoritmii „ Link” ar putea depăși în mod eficient impasul și problemele de sincronizare la nivelul de strategie drop-coadă,iar cercetarea se concentrează pe algoritmii de AQM (Active Queue Management).Esita trei tipuri de algoritmi Aqm : Red (Random early Decetion), Algoritm AQM bazat pe teoria controlului şi Algortm AQM bazate pe teoria optimizării.

2.3 Algoritmi de management al cozii la receptor (AQM)

Principalele caracteristici de performanță ale AQM includ :

• Utilizarea eficientă a cozii: coada ar trebui să evite supraîncărcarea rezultată în pierderea de pachete şi retransmisii nedorite sau pachete goale= goliciuni au rezultat într-un link subincarcat

• Întârzierea cozii: este de dorit să se păstreze întârzierea şi micile sale variații.

• Robustețea: Schema AQM are nevoie de a menținerea un comportament robust, în ciuda diferitelor condițiile de rețea, cum ar fi variații în numărul de sesiuni de TCP sau variaţii în întârzierea propagarii și capacitatea legăturii.

TCP a fost optimizat pentru rețele cu fir. Orice pierdere de pachete este considerată a fi rezultatul unei congestie a rețelei și dimensiunea ferestrei de congestie este dramatic redusă ca o măsură de precauție. O serie de algoritmi alternativi de control al congestiei au fost propusi pentru a ajuta la rezolvarea problemei fara fir : cum ar fi Las Vegas, Westwood, Veno și Santa Cruz.

TCP îndeplinește două sarcini de control fundamentale:

1) De control end-to-end al debitului, pentru a evita supraincarcarii buffer-ului receptorului ;
2) Adaptarea controlului congestiei la conectare, adaptarea cantitatii de informații transferata în rețeaua la starea congestionarea rețelei. Această funcție este bazata pe recuperarea congestiei, mai degrabă decât de prevenire. Crește traficul TCP oferit rețelei până când rețeaua este obligata să renunțe la unele pachete; apoi se strange pentru a da timp rețelei de a reveni din congestie, și începe din nou să crească traficul oferit pentru a forța un alt episod de congestie. Controlul fluxului TCP este pus în aplicare prin intermediul receptorului, „*advertised window" (rcvwnd)* W_{re}, stabilit de către receptorul în antetul de confirmare a segmentelor (ACK). Controlul congestiei este pus în aplicare prin intermediul ferestrei de congestie (CNWD) W_c, calculata de către transmițător după algoritmul de control al congestiei TCP. În final cantitatea de date care ii este permis TCP-ului să o transmită la un moment dat este calculat ca

$$W_{tr} = \min\{ W_c , W_{re} \} - W_v (2.4)$$

Wv este cantitatea de date restante, adică, a ferestrei de transmisie de alunecare, care este umplută cu datele transmise deja, dar încă nerecunoscute.

Controlul congestiei se referă la controlarea traficului ce pătrunde într-o rețea (de telecomunicații sau de calculatoare), ceea ce presupune evitarea supraîncărcării legăturilor dintre nodurile intermediare ale rețelei, prin reducerea numarului de pachete trimise. Așa cum s-a prezentat anterior, controlul congestiei nu trebuie confundat cu controlul fluxului care protejează receptorul de a fi „copleșit" de către emițător.

Slow-start este parte a strategiei de control al congestiei folosit de TCP, protocolul de transmitere a datelor utilizate de mai multe aplicații pe Internet. Slow Start este folosit în combinație cu alti algoritmi pentru a evita trimiterea de mai multe date decât rețeaua este capabilă să transmită, scopul este de a evita congestionarea rețelei. Algoritmul este specificată de RFC 5681.

Slow-start este unul dintre algoritmii care TCP ii foloseste pentru a controla congestie în interiorul rețelei. De asemenea, este cunoscut sub numele de faza de creștere exponențială. În timpul fazei de creștere exponențială, începe creșterea in regim încet a creșterii ferestrei de congestie TCP de fiecare dată cand o confirmarea este primită. Aceasta crește dimensiunea ferestrei cu numărul de segmente recunoscute. Acest lucru se întâmplă până când o confirmare nu este primită pentru se ajunge la unele segment sau o valoare de prag predeterminată. În cazul în care are loc un eveniment de pierdere, TCP presupune că este din cauza congestiei rețelei și ia măsuri pentru a reduce sarcina oferite pe rețea. Odată ce pragul a fost atins, TCP intră în fază de creștere liniară (de evitare a congestiei). In acest moment, fereastra creşte cu un segment pentru fiecare RTT. Acest lucru se întâmplă până când se produce un eveniment de pierdere.

Cu toate că strategia este denumit "Slow-Start", creșterea fereastrei de congestie este destul de agresivă, mai agresivă decât faza de evitare a congestiei. [29] Înainte de „Slow Start" a fost introdus în TCP, faza inițială de evitare pre-congestie fost chiar mai rapid.

Algoritmul de bază slow-start : Algoritmul începe în faza de creștere exponențială, inițial cu o dimensiune a ferestrei de congestie (cwnd) de 1, 2 sau 10 [30] segmente și crește prin

dimensiunea unui segment (SS) pentru fiecare nou ACK primit. În cazul în care receptorul trimite un ACK pentru fiecare segment, acest comportament se dublează în mod eficient dimensiunea ferestrei fiecare călătorie dus-întors a rețelei. În cazul în care receptorul acceptă un ACK întârziate, rata de creștere este mai mică, dar totuși crește cu un minimum de un SMS de fiecare dată pentru un drum dus-întors. Acest comportament continuă până când dimensiunea ferestrei congestiei (cwnd) atinge dimensiunea ferestrei de avertizare a receptorului sau până când se produce o pierdere.

Atunci când are loc o pierdere, jumătate din cwnd curent este salvat ca un prag de pornire lentă (SSThresh) și Slow Start începe din nou de la dimensiunea ferestrei cwnd inițială. Odată ce cwnd ajunge la SSThresh, TCP trece în modul de evitare a congestiei în cazul în care fiecare nou ACK crește cwnd de SS × SS / cwnd. Acest lucru duce la o creștere lineară a cwnd.

Fast Restransmit : Un expeditor TCP foloseste un cronometru pentru a recunoaște segmente pierdute. În cazul în care o confirmare nu este primită pentru un anumit segment într-un timp specificat (în funcție de estimat timpul de întârziere tur-retur), expeditorul va asuma segmentul a fost pierdut în rețea, și va retransmite segmentul. Confirmarea Duplicate este baza pentru mecanismul Fast Restransmit , care funcționează după cum urmează: după ce a primit un pachet (de exemplu, cu numărul de ordine 1), receptorul trimite o confirmare prin adăugarea de la 1 la numărul de ordine (de exemplu, numărul de ordine 2), ceea ce înseamnă că receptor primește numărul de pachete de 1 și se așteaptă ca numărul de pachete de 2 de la expeditor. Să presupunem că cele trei pachete ulterioare s-au pierdut. Intre timp receptorul primește un număr de pachete de 5 și 6. După ce a primit numărul de pachete de 5, receptorul trimite o confirmare, dar numai pentru numărul de ordine 2. Când receptorul primește numărul de pachete de 6, acesta trimite un alt valoare confirmare de 2. Deoarece expeditorul primeste mai mult de o confirmare cu același număr de ordine (2 în acest exemplu), aceasta se numește duplicat confirmare. [31]

Recuperarea rapidă : Există o variație a algoritmului slow-start, cunoscut sub numele de recuperare rapidă, care utilizează retransmiterea rapidã, urmată de evitarea congestiei . În algoritmul de recuperare rapidă, în modul de evitare a congestiei, atunci când pachetele nu sunt primite (ceea ce ar insemna detectarea a trei ACK duplicat), dimensiunea ferestrei de congestie este redusã la pragul de Slow Start , mai degrabă decât valoarea inițială.

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

- ***Tahoe:*** pierderea este detectată atunci când timpul de timeout expiră înainte ca o confirmare să fie recepționată. În acel moment, Tahoe reduce fereastra de congestie la valoarea maximă a dimensiunii unui segment (MSS – maximum segment size) și se resetează la faza de „slow-start”.
- ***Reno:*** în cazul în care sunt recepționate trei confirmări duplicate (ACK duplicat), Reno înjumătățește fereastra de congestie, efectuează o retransmitere rapidă (fast retransmit) și intră în recuparare rapidă (fast recovery). Până la mijlocul anilor 1990, toate timeout-urile și întarzierile dus-intors erau bazate numai pe ultimul pachet din buffer. Cercetătorii Larry Peterson și Lawrence Brakmo de la Universitatea din Arizona, au introdus ***TCP Vegas*** în care timeout-ul este setat iar întarzierea dus-întors este măsurată pentru fiecare pachet din buffer. De asemenea, acest algoritm folosește o creștere aditivă a ferestrei de

congestie. Această varianta nu a cunoscut o răspândire foarte mare în afara laboratorului cercetătorilor.

- ***TCP New Reno*** îmbunătățește retransmisia în timpul fazei de recuperare rapidă (fast recovery) a TCP Reno. Pe parcursul recuperării rapide, pentru fiecare ACK duplicat, este transmis un nou pachet netrimis de la sfârșitul ferestrei de congestie, pentru a menține fereastra plină. Pentru fiecare ACK parțial din spațiul de secvențe numerice, emițătorul trimite următorul pachet cu secvența numerică următoare ACK-ului. Întrucât cronometrul de timeout este resetat ori de cate ori există un progres în buffer-ul de transmisie, New Reno poate umple găuri mari sau mai multe găuri în spațiul de secvențe. Deoarece New Reno poate trimite pachete noi la sfârșitul ferestrei de congestie în timpul recuperării rapide, este menținut un throughput mare pe durata procesului de umplere a găurilor, chiar și atunci când există mai multe găuri, de mai multe pachete fiecare. Când TCP intră în recuperare rapidă, acesta înregistrează cea mai mare secvență numerică neconfirmată. Când această secvență numerică este confirmată, TCP revine la stare de evitare a congestiei. Atunci când nu au loc pierderi de pachete, dar in schimb pachetele sunt reordonate cu mai mult de 3 secvențe numerice, apare o problemă în New Reno. Când acest fenomen are loc, New Reno intră greșit în recuperare rapidă, însă când pachetul reordonat este livrat, are loc progresul secvenței numerice ACK și din acel moment până la sfârșitul recuperării rapide, fiecare bit al secvenței numerice produce un duplicat și retransmisia nu este necesară.

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

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

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

În rutare , management activ al cozii (AQM) reordoneaza arbitrar sau pierde pachetele de rețea în interiorul bufferul de transmisie a interfației unui controler de rețea. Sarcina este efectuată de către designerul de rețea. Un router de Internet menține de obicei un set de cozi, câte unul pe fiecare interfață. Sarcina activa a cozii este de a pierde sau marca pachetele înainte ca ea sa fie plina. De obicei, acestea operează prin menținerea uneia sau mai multor probabilități de pierdere sau de marcare .

Pentru a evita colapsul congestie, TCP foloseste o strategie multi-fațete de control a congestie. Pentru fiecare conexiune, TCP menține 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 TCP folosită pentru controlul fluxului. TCP folosește un mecanism numit start lent [29] pentru a mări fereastra de congestie, după o conexiune este initializatã si dupa un timeout. Acesta începe cu o fereastră de două ori dimensiunea maximă a segmentului (MSS). Deși rata inițială este scăzută, rata de creștere este foarte rapidã: pentru fiecare pachet recunoscut, fereastra de congestie crește cu 1 MSS, astfel încât fereastra de congestie dublează în mod eficient de fiecare dată dus-întors (RTT). Când fereastra de congestie depășește un prag ssthresh algoritmul intră într-o stare nouă, numită evitare a congestiei. În unele implementari (de exemplu, Linux), ssthresh inițială este mare, și astfel primul start lent, de obicei, se termină după o pierdere. Cu toate acestea, ssthresh este actualizat la sfârșitul fiecărui start lent, și va afecta de multe ori începe lent ulterioare declanșate de timeout.

Evitare a congestiei. Atâta timp cât ACK-uri non-duplicat sunt primite, fereastra de congestie este aditiv a crescut cu un SMS de fiecare dată dus-intors. Când un pachet este pierdut, probabilitatea de a fi primit ACK duplicat este foarte mare (este posibil, deși puțin probabil ca fluxul tocmai a suferit reordonare pachete de extremă, care ar putea, de asemenea, ACK duplicat prompte). Comportamentul de Tahoe si Reno diferă în modul în care detectează și reacționează la pierderea de pachete:

- Tahoe: ACK triple duplicat sunt tratate la fel ca un timeout. Tahoe va efectua "Fast Restransmit", a stabilit pragul de start lent la jumătate ferestrei de congestie curent, reduce fereastra de congestie la 1 MSS, și a reseta pentru faza de Slow Start. [29].
- Reno: În cazul în care sunt primește trei ACK duplicat (de exemplu, patru ACK recunoscând în același pachet, care nu sunt piggybacked pe date, și nu se schimba fereastra de publicitate receptorului), Reno va înjumătăți fereastra de congestie (în loc de a stabili o la 1 MSS ca Tahoe), a stabilit pragul de pornire lent egal cu noul ferestrei de congestie, efectua o retransmite rapid, și intră într-o fază numită de recuperare rapidă. În cazul în care un ori ACK din, început lent este utilizat ca acesta este cu Tahoe. [28]

Recuperarea rapidă. (În cazul algoritmului New Reno) În această stare, TCP retransmite pachetul lipsește, care a fost semnalat prin trei ACK duplicat, și așteaptă o confirmare a întregii ferestrei de transmisie înainte de a reveni la evitarea congestiei. Dacă nu există nici o confirmare, TCP Reno experimentează un timeout și intră în starea de slow-start. Ambii algoritmi reduc fereastra de congestie la 1 MSS pe un eveniment de timeout.

TCP Cubic îmbunătățește corectitudinea împărţirii prin utilizarea în timp real pentru evoluția din fereastră de congestie. S-au făcut multe mai multe cercetări pentru a se găsi o metodă de tratare a congestiei şi pentru împărţirea corectă a lăţimii de bandă la nivel de reţea. Au fost propuşi mai mulţi algoritmi AQM bazaţi pe echitate. Ei folosesc o gamă largă de tehnici pentru a detecta care link mai multă de lățime de bandă decât ar trebuie şi aruncă în reţea mai multe pachete ca urmare a ocupării acestui flux mai mare. În unii algoritmi (de exemplu CHOKE), nu există o detectare explicită a ocupării de prea multă lățime de bandă - echilibrul între consumul de bandă este menținut în mod automat şi datorită algoritmului ingenios de gestionare a cozii.

Acum trebuie subliniat faptul că cele mai multe studii anterioare au fost concentrate pe un singur tip de soluție și anume, fie pe nivelul rețea (AQM) sau nivelul transport (controlul congestiei TCP). Când noul AQM a fost propus, a fost testat pentru controlul congestiei clasice (New Reno) . Similar, după ce a fost inventată noua variantă de TCP, a fost evaluată cu managementul clasic al cozii.

Schemele pentru gestionarea cozii pot fi clasificate în mai multe moduri. Din punct devedere al performanţei se iau în considerare mai mulţi parametrii (de exemplu debit, coadă, dimensiune, corectitudine). Diferitele tipuri de algoritmi AQM sunt axate pentru optimizarea unuia sau poate chiar mai multora dintre aceşti parametrii.

3 TCP NewReno

3.1 Introducere

Performanţa TCP este afectată de congestia în reţea şi de selecţia unui format de date adecvat pentru capacitatea canalului. Congestie apare mai ales atunci când o cantitate mare de flux (tranzacţiei FTP) este trimis. Scopul acestei lucrări este de a rezolva acest tip de problemă a congestiei , folosind abordări de simulare în algoritmi deja definiţi. Traficul de Internet de astăzi este în mare parte realizat de către protocolul de control al transmisiei (TCP). TCP în co-relaţie cu UDP reprezintă nucleul stratului curent de transport internet. Studii recente arată mai multe eforturi de eficientizare în direcţia de control a congestiei şi de selectatie a ratei de trimitere potrivită pentru tranzacţii în TCP. Implementări moderne, TCP, sugerează, de asemenea, diferite scheme de control a congestiei folosind fereastră de congestie (cwnd) şi tehnici de întârziere. Cu toate acestea, scopul este de a creşte dimensiuna maxim posibil a cwnd , fără pierderi de pachete şi selectaţie a ratei de transmisie adecvată înainte de a apărea congestia . Mai multe variante de TCP sunt disponibile , dintre care s-a recunoscut că standardul de transfer TCP New Reno se deteriorează în reţele de mare viteză cu întârzieri de lungimi de bandă mari (BDP) . Au fost propuşi pentru a aborda această deteriorare noi algoritmi de control a congestie.

Cererile utilizatorilor de internet legate de calitatea reţelei au crescut ca urmare a serviciilor ce au deveni treptat diversificate şi mai sofisticate, din cauza gradului remarcabil în care internetul a crescut, ceea ce se datorează în parte de accesul şi tehnologii de reţea. Aplicaţii care implică timp real servicii de livrare mass-media, cum ar fi VoIP, sisteme video streaming şi de conferinţă TV, toate au experimentat un nivel dramatic de dezvoltare care necesită cantităţi mari şi stabile de resurse de reţea, în scopul de a menţine calitatea serviciului (QoS). De exemplu; calitatea în timp real a aplicaţiilor de streaming este foarte dependentă de întârziere de propagare şi întârzierea bruiaj. Lăţimea de bandă disponibilă pe calea reţelei end-to-end este de asemenea un factor important pentru asigurarea lin conţinut bogat, inclusiv de voce şi video. Există o serie de tehnologii de layere de reţea, cum ar fi IntServ şi DiffServ , care furnizează astfel de servicii de reţea de înaltă calitate pe Internet. Cu toate acestea, punerea în aplicare a IntServ sau arhitecturi DiffServ ar avea nevoie de mecanisme suplimentare pentru a fi utilizate la toate routerele prin care fluxurile de trafic traversează în scopul de a beneficia în mod suficient de introducerea IntServ sau DiffServ în reţea. Pe de altă parte, un număr de aplicaţii video streaming folosesc UDP ca protocol de transport , şi UDP controlează viteza de transmitere a datelor în funcţie de starea reţelei. Cu toate acestea, aceste mecanisme au un cost mare atunci când se modifică programul de aplicaţie pentru realizarea cerinţelor QoS-aplicaţii specifice, precum şi setarea parametrilor sunt foarte sensibile la diverşi factori de reţea.

Mai mult decât atât, atunci când co-exista astfel de aplicaţii în reţea şi împărtăşesc resursele în reţeaua congestionată , nu putem estima performanţa reţelei sau a cererilor, pentru că

mecanismele de control pentru astfel de aplicații sunt concepute și puse în aplicare în mod independent, fără a ține seama de efectul de interacțiuni cu alte aplicații. Deoarece TCP controlează viteza de transmitere a datelor în funcție de starea rețelei.

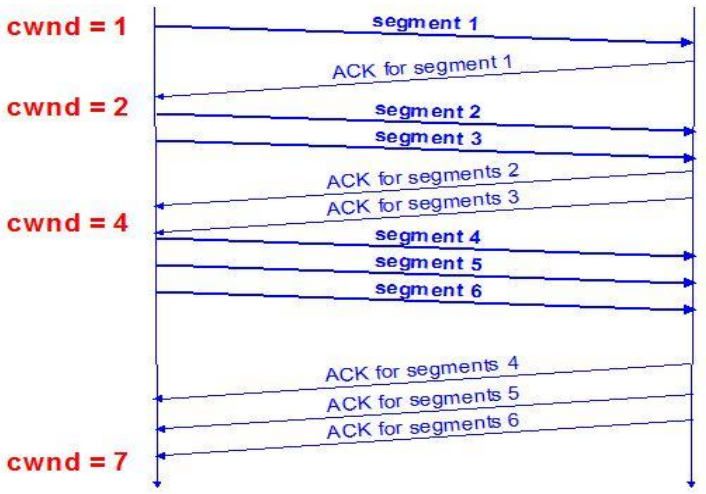
- Controlul congestiei TCP este guvernat de doi parametrii:
  - Congestion Window (cwnd)
  - Slow-start threshhold Value (ssthresh)

  Valorile inițiale sunt : 2^{16} -1
- Controlul congestiei are două etape:
  - slow start (cwnd < ssthresh)
  - congestion avoidance (cwnd ≥ ssthresh)

În cazul „Slow Start" valoarea inițială a cwnd este 1 . (Notă: Unitatea este o dimensiune segment. TCP de fapt se bazează pe bytes şi suplimente de 1 SMS (dimensiunea maximă a segmentului) . Receptorul trimite o confirmare (ACK) pentru fiecare segment (Notă: În general, un receptor de transport trimite un ACK pentru fiecare alt segment.) De fiecare dată când un ACK este recepționat de către expeditor, fereastra de congestie este majorat cu 1 segment:

$$cwnd = cwnd + 1 \quad (3.1)$$

Dacă un ACK recunoaște două segmente, cwnd este incrementat cu doar 1 segment. Chiar dacă ACK recunoaște un segment care este mai mic decât un MSS bytes lung, cwnd este crescut cu 1. Dimensiunea ferestrei congestie crește foarte rapid. Pentru fiecare ACK, vom incrementa cwnd cu 1 , indiferent de numărul de segmente ACK-uri. TCP încetinește creșterea cwnd când Cwnd> ssthresh.



Faza de evitare a congestiei este pornită dacă cwnd a atins valoarea de prag a slow-start-ului.

Dacă cwnd ≥ ssthresh apoi de fiecare dată când este recepționat un ACK, cwnd va creștere după cum urmează:

$$cwnd = cwnd + \frac{1}{cwnd} \qquad (3.2)$$

Astfel cwnd va fi incrementat numai dacă toate segmentele cwnd au fost recunoscute.

3.2.TCP Reno

Performanța la starea de echilibru a unui flux TCP de transfer (de exemplu, un flux cu o cantitate mare de date de trimitere, cum ar fi transferurile FTP) poate fi caracterizata prin rata de trimitere, care reprezinta cantitatea de date transmise de către expeditor în unitatea de timp. O cantitate semnificativă de trafic pe Internet, inclusiv WWW (HTTP), transfer de fișiere (FTP), e-mail (SMTP), și accesul de la distanță (Telnet) de trafic, se efectuează de către protocol de transport TCP . TCP împreună cu UDP formează miezul de layer de transport pe Internet de astăzi. În mod tradițional, simularea și punerea în aplicare si măsurarea au fost instrumentele de alegere pentru examinarea performanțelor diferitelor aspecte ale TCP-ului. Recent, atentia a fost îndreptata catre caracterizarea rata de trimitere a unui flux TCP ca o funcție de pierdere de pachete și de întârziere dus-întors.

Pentru a evita colapsul retelei cauzat de congestie, TCP foloseste o strategie de control al congestiei multi-fațete. Pentru fiecare conexiune, TCP menține 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 TCP folositã pentru controlul fluxului. TCP folosește un mecanism numit „slow start” *(* start lent) pentru a mări fereastra de congestie, dacă o conexiune este initializat si dupa un timeout. Acesta începe cu o fereastră de două ori mai mare decat dimensiunea maximă a segmentului (MSS). Deși rata inițială este scăzută, rata de creștere este foarte rapid: pentru fiecare pachet recunoscut, fereastra de congestie crește cu 1 MSS, astfel încât fereastra de congestie se dublează în mod eficient de fiecare dată dus-întors (RTT). Când fereastra de congestie depășește un prag „ssthresh” algoritmul intră într-o stare nouă, numită evitare a congestiei. În unele implementari (de exemplu, Linux), „ssthresh” inițial este mare, și astfel primul start lent, de obicei, se termină după o pierdere. Cu toate acestea, ssthresh este actualizat la sfârșitul fiecărui start lent, și va afecta de cele mai multe ori un început lent ulterioar declanșat de timeout.

Evitare a congestiei. Atâta timp cât ACK-uri non-duplicat sunt primite, fereastra de congestie este aditivã, crescand cu un SMS de fiecare dată cand este efectuat un ciclu dus-intors. Când un pachet este pierdut, probabilitatea de a fi primit ACK duplicat este foarte mare (este posibil, deși puțin probabil ca fluxul sa fi suferit reordonarea pachetelor). Comportamentul algoritmilor Tahoe si Reno diferă în modul în care detectează și reacționează la pierderea de pachete:

Tahoe: ACK triple duplicat sunt tratate la fel ca un timeout. Tahoe va efectua ” fast

retransmit", a stabilit pragul de start lent la jumătate ferestrei de congestie curentă, reduce fereastra de congestie la 1 MSS, și a reseta pentru slow-start state.

Reno: În cazul în care sunt primite trei ACK duplicat (de exemplu, patru ACK recunoscute în același pachet, dar ale caror date nu sunt intorase , și nu se schimba fereastra de avertizare a receptorului), Reno va înjumătăți fereastra de congestie (în loc de a stabili-o la 1 MSS ca Tahoe), a stabilit pragul de pornire lent egal cu noua ferestră de congestie, efectuand o retransmitere rapidă, și intră într-o fază de recuperare rapidă.

Recuperarea rapidă. Reno in această stare, TCP retransmite pachetul care lipsește, semnalat in urma cu trei ACK duplicat, și așteaptă o confirmare a întregii ferestrei de transmisie înainte de a reveni la evitarea congestiei. Dacă nu există nici o confirmare, TCP Reno experimentează un timeout și intră în starea de slow-start.

Ca o definire putem spune că:

Pentru controlul congestiei TCP se definesc următorii parametri:

• Dimensiunea maximă a segmentului expeditor (SMS-uri) reprezintă suma maximă de date care pot fi trimise într-un singur segment TCP, fără a include antetul.

• Fereastra expeditorului (swnd) reprezintă numărul maxim de bytes care poate fi trimisi. Valoarea sa are valoarea cea mai mică dintre fereastra receptor și fereastra de congestie.

• Fereastra destinatarului (rwnd) este cel mai nou fereastra de publicitate de către receptor.

• fereastra de congestie (cwnd) este o variabilă de stare TCP, care limitează cantitatea de date care pot fi trimise.

• Fereastra pentru pierderi (lw) este valoarea ferestrei de congestie după o pierdere de pachete a fost detectatã.

• prag *slow-start(* lent-start) (ssthresh) este o altă variabilă TCP, care determină ca algoritm de control al congestiei sa fie declansat sau nu : fie de pornirea a unui *slow-start* (dacă cwnd ≤ ssthresh), fie de evitare a congestiei (dacă cwnd ≥ ssthresh).

Slow-Start și evitarea congestiei : Slow-Start, denumit la propriu , crește exponențial dimensiunea ferestrei de congestie. Acesta este utilizat de către o entitate TCP la începutul unei transmiteri sau după detectarea unei pierderi de pachete. Scopul Slow-Start este de a umple cât mai curând posibil, un canal de transmisie. După ce fereastra de congestie a atins valoarea de prag, algoritmul de evitare a congestiei este activat . Fereastra de congestie continuă să crească liniar, adăugând până la un SMSS, dar nu mai puțin de un octet. În ambele cazuri un timer de retransmisie este utilizat pentru fiecare pachet. Timeout-ul semnalează pierderea de pachete. Acest lucru duce la retransmiterea și înjumătățirea pragului Slow-Start . Fereastra de congestie este setată la valoarea pierderii ferestrei.

Expeditorul păstrează două variabile pentru controlul congestiei: o fereastră slow-start/congestion, cwnd, și o dimensiune prag, ssthresh, pentru a comuta între cele două algoritmi.

Expeditorul trimite întotdeauna un minim de cwnd și advertismentul fereastrei receptorului. Pe un timeout, jumătate din dimensiunea ferestrei curente este înregistrată în ssthresh, apoi cwnd este setat la 1 pachet (acesta inițiază Slow-Start-ul). Când date noi sunt acked, expeditorul va face urmatorul algoritm:

If (cwnd<ssthresh)

/* daca in continuare se face procesul de Slow-start , se va deschide fereastra exponential*/ Cwnd+=1;

Else /* altfel se va face congestia evitand cresterea cu -1*/

Cwnd+=1/cwnd;

Astfel, Slow-Start deschide repede fereastra la ceea ce avertismentul de congestiei crede că este un punct de funcționare în condiții de siguranță (jumătate fereastra care ne-a băgat în bucluc), apoi avertismentul de congestie este eliminat și se încet crește dimensiunii ferestrei pentru a sonda pentru mai multă lățime de bandă disponibilă in retea.

Recuperarea rapidă și retransmiterea rapidă: După punerea în aplicare a algoritmului , noi probleme apar.Primul este legat la detectarea pierderii de pachete. În mod normal, o pierdere de pachete se deduce pe baza de expirare a timer-ului de retransmisie. Aceasta poate duce la întârzieri semnificative în transmisii de date, astfel încât un alt mod de a determina pierderea de pachete a fost adăugat la TCP. In condiții normale, o entitate TCP trebuie să trimită un ACK duplicat pentru fiecare pachet care vine din secvență. Un pachet poate fi primit din secvență ca urmare a dublării de pachete de rețea, întârzieri de pachete sau pierdere. Algoritmul de retransmisie rapida consideră că un pachet a fost pierdut atunci când primește 3 ACK duplicat, înainte de expirare a timer de retransmisie. În acest fel timp prețios este salvat.

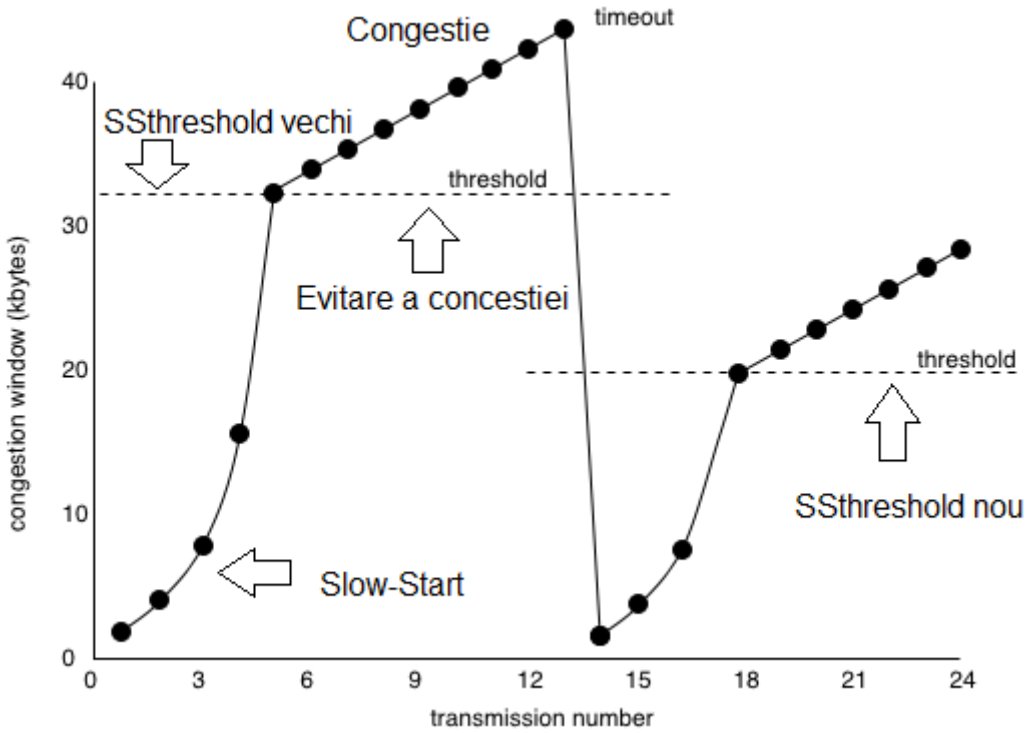


Figura 3.1 : Slow-Start și evitarea congestiei

A doua problemă este legată de scăderea drastică a ferestrei de congestie după o detectare a pierderilor de pachete. Dacă un pachet este pierdut în timpul Slow-Start sau eitarea congestiei , valoarea ferestrei de congestie este setată la valoarea pierderii ferestrei (1 SMSS). Algoritmul de recuperare rapidă abordează această problemă. Noua valoare a fereastrei de congestie, după o pierdere de pachete este detectată de retransmisia rapida si este setată la ssthresh + 3 SMSS. Aceasta se numește "inflație artificială" a ferestrei de congestie. Pe lângă faptul că, pentru fiecare nou duplicat ACK primit , fereastra de congestie va crescte cu un SMSS. Mecanismul de evitare a congestiei, în cazul în care dimensiunea ferestrei de control a congestiei este W, atunci dimensiunea sa creste cu 1 / W de fiecare dată când este recepționat un ACK. Invers, fereastra este redusă de fiecare dată când un pachet pierdut este detectat.

Exista un comportament de evitare a congestiei TCP în termeni de "runde". O rundă începe cu transmisia de pachete de W, în care W este dimensiunea actuală a ferestrei de congestie TCP. După ce toate pachetele care intră în fereastra de congestie au fost trimise, nu mai este trimis nici un alt pachet până cand primul ACK este recepționat de unul dintre aceste pachete W. Acest ACK marchează sfârșitul etapei actuale și începutul runda următoare. In acest model, durata unei runde este egal cu RTT și este considerată independentă față de dimensiunea

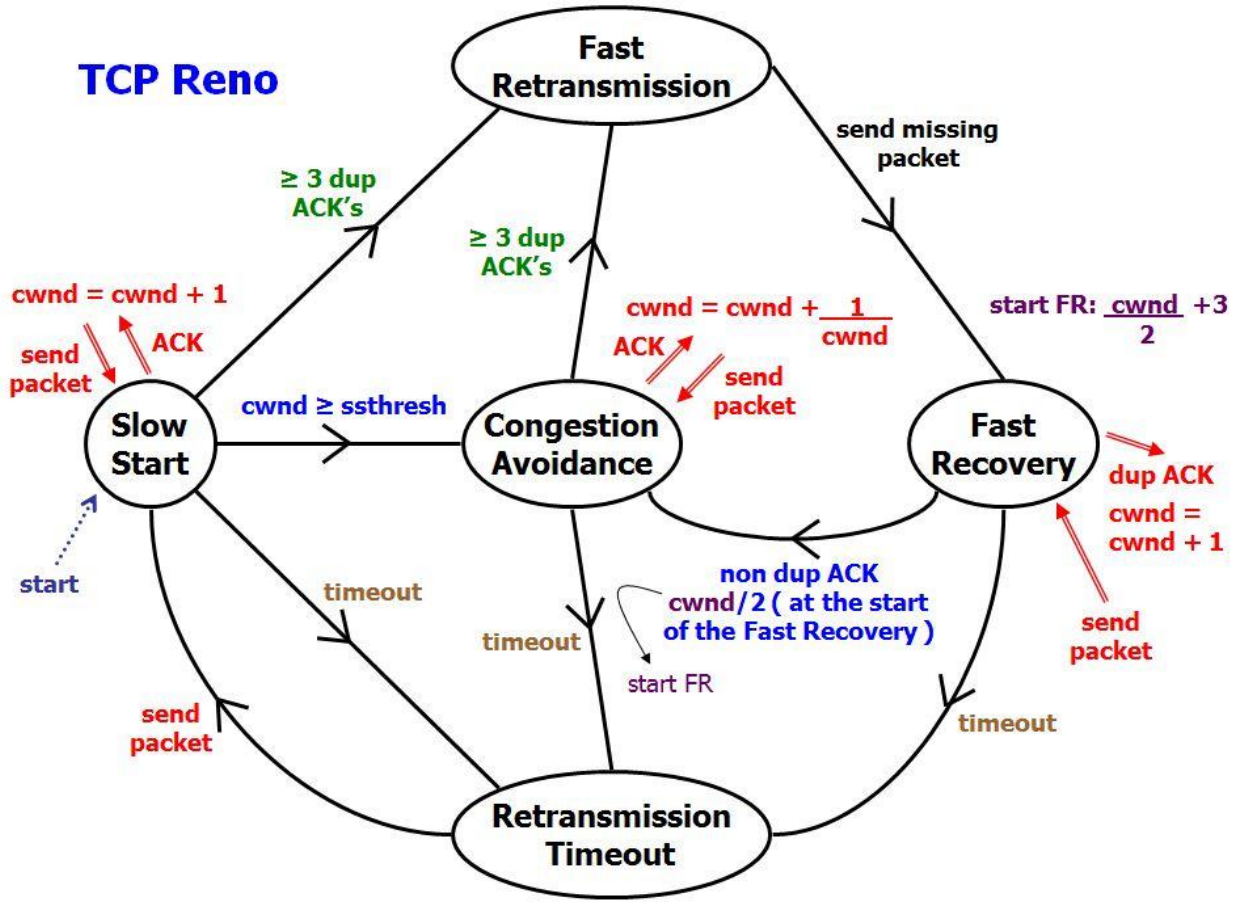


Figura 3.2 : Diagrama Algoritm TCP Reno

ferestrei. Conceptul nostru de runde este similar cu conceptul de "mini-cicluri.

3.2 Caracteristici New Reno

New Reno, definit de RFC 6582 (care a devenit comun definițiile anterioare în RFC 3782 și RFC 2582), îmbunătățește retransmisia în timpul fazei de Fast recovery a algoritmului TCP Reno. Timpul de recuperare rapidă, pentru fiecare ACK duplicat care este returnat la TCP New Reno, un nou pachet netrimise de la sfârșitul ferestrei de congestie este trimis, pentru a menține fereastra de transmisie integrală. Pentru fiecare ACK care face progrese parțiale în spațiul secvență, expeditorul presupune că punctele de ACK la o nou nod , iar următorul pachet aflat dupa numărul de ordine al ultimului ACK este trimis.

Nou RENO reprezinta o ușoară modificare a algoritmului AQM TCP-RENO. Acesta este capabil să detecteze multiple pierderi de pachete și este mult mai eficient decât RENO in ceea ce priveste pierderia de pachete. Ca Reno, New-Reno intră de asemenea în rapidă retransmisie în care primește mai multe pachete duplicate, însă in acest caz , New reno diferă de RENO prin faptul că nu iese din Fast-recovery cât atunci toate datele care au fost pierdute în momentul în intrarii in faza de recuperare rapida sunt ackwnoleg. Astfel, se depășește problema cu care se confruntă Reno , de reducere a CWD de mai multe ori .

Faza de fast-transmit are aceleaşi etape ca la Reno. Diferența constă ca în faza de fast-recovery trebuie să se retransmită toate pachetele pierdute. Ori de câte ori New-Reno intră în faza de recuperare rapidă se constată segmentul maxime, care este foarte important . Desi aceasta faza se desfasoara ca la Reno, cu toate acestea, atunci când un ACK este proaspăt primit, atunci există două cazuri:

- Dacă sunt ACK toate segmentele care au fost retransmise , am intrat în faza de fast-recovery , atunci se iese din această și se setează CWD la ssthresh și se continuă transmisia
- Dacă ACK este un ACK parțială atunci se deduce că următorul segment în linie a fost pierdut și se re-transmite segmentul și se stabilește numărul de ACK duplicat primite de la zero. Se iese de recuperare rapidă, atunci când toate datele din fereastra sunt recunoscute.

Deoarece cronometrul timeout este resetat ori de câte ori există un progres în bufferul de transmisie, aceasta permite New Reno sa umple arii mari, sau mai multe arii , în spatiul de secventa - la fel ca TCP Sack . Deoarece New Reno poate trimite noi pachete de la sfârșitul ferestrei de congestie în timpul de recuperare rapidă, randamentul ridicat se menține în timpul procesului de umplere a ariei , chiar și atunci când există mai multe arii, de mai multe pachete fiecare. Când TCP intră in recuperare rapidă se înregistrează cel mai mare număr de secvențe de pachete nerecunoscut. Când acest număr de ordine este recunoscut, TCP revine la starea de evitare a congestiei.

O problemă apare la New Reno, atunci când nu există pierderi de pachete, dar în schimb, pachetele sunt reordonate cu mai mult de 3 numere de secvență de pachete. Când se întâmplă acest lucru, New Reno intră din greșeală in recuperare rapidă, dar în cazul în care pachetul reordonat este livrat, ACK-secvență de numărare a progres apare și de acolo până la sfârșitul recuperarii rapide, fiecare bit de secvență a numărului de progres produce un duplicat care este imediat ACKed.

În TCP Reno, dimensiunea ferestrei este schimbată continuu într-o situație tipică. Dimensiunea ferestrei continua să crescă până se produc pierderi de pachete. TCP Reno are două faze în creșterea dimensiunii de fereastră: Etapa de start lent și de evitare a congestiei. Când un pachet ACK (confirmare) este primit de TCP la expeditor la momentul t + t ` [sec], actuala dimensiune a ferestrei cwnd (t + t`) este actualizată de la cwnd (t), după cum urmează:

- Faza de Slow-Start :

  If cwnd(t)<ssthreash(t)

  Cwnd(t+t`)=cwnd(t)+1;

- Faza de evitare a congestiei:

  If cwnd(t)> ssthreash(t)

  Cwnd(t+t`)=cwnd(t)+1/ cwnd(t)

În cazul în care, ssthresh (t) este o valoare de prag la care TCP se schimbă de la faza de început lent la faza de evitare a congestiei. Când sunt detectate pierderi de pachete prin expirarea retransmisie timeout, cwnd (t) și ssthresh (t) sunt actualizate ca:

$$Cwnd(t)=1; \quad (3.3)$$

$$ssthreash(t)= cwnd(t)/2; \quad (3.4)$$

Pe de altă parte, atunci când se detectează pierderea de pachete TCP de către un algoritm de retransmisie rapid, se schimbă cwnd (t) și ssthresh (t) ca;

$$Cwnd(t)=ssthreash(t) ; \quad (3.5)$$

$$ssthreash(t)= cwnd(t)/2; \quad (3.6)$$

TCP Reno intră apoi într-o fază de recuperare rapidă în cazul în care pierderea de pachete este găsită de către algoritmul retransmis rapid. In această fază, dimensiunea ferestrei este mărită cu un pachet atunci când un pachet ACK duplicat este primit. Pe de altă parte, cwnd (t) este readus la SSTH (t) atunci când pachetul ACK non-duplicat corespunzând pachetul retransmis este primit.

Conexiunea TCP pornește în modul de pornire lent și crește exponențial fereastra de congestie (cwnd) până când se trece pragul start lent (ssthresh). Odată ce cwnd este mai mare decât ssthresh, TCP se mută la faza de evitare a congestiei. În acest mod, obiectivul principal este de a menține debitul mare, fără a provoca congestie. Dacă TCP detectează orice pierdere a

unui segment, acest lucru este un indiciu puternic că rețeaua este saturată, astfel ca o acțiune corectivă TCP reduce rata de flux de date prin reducerea cwnd după care din nou se întoarce pentru a încetini faza de început.

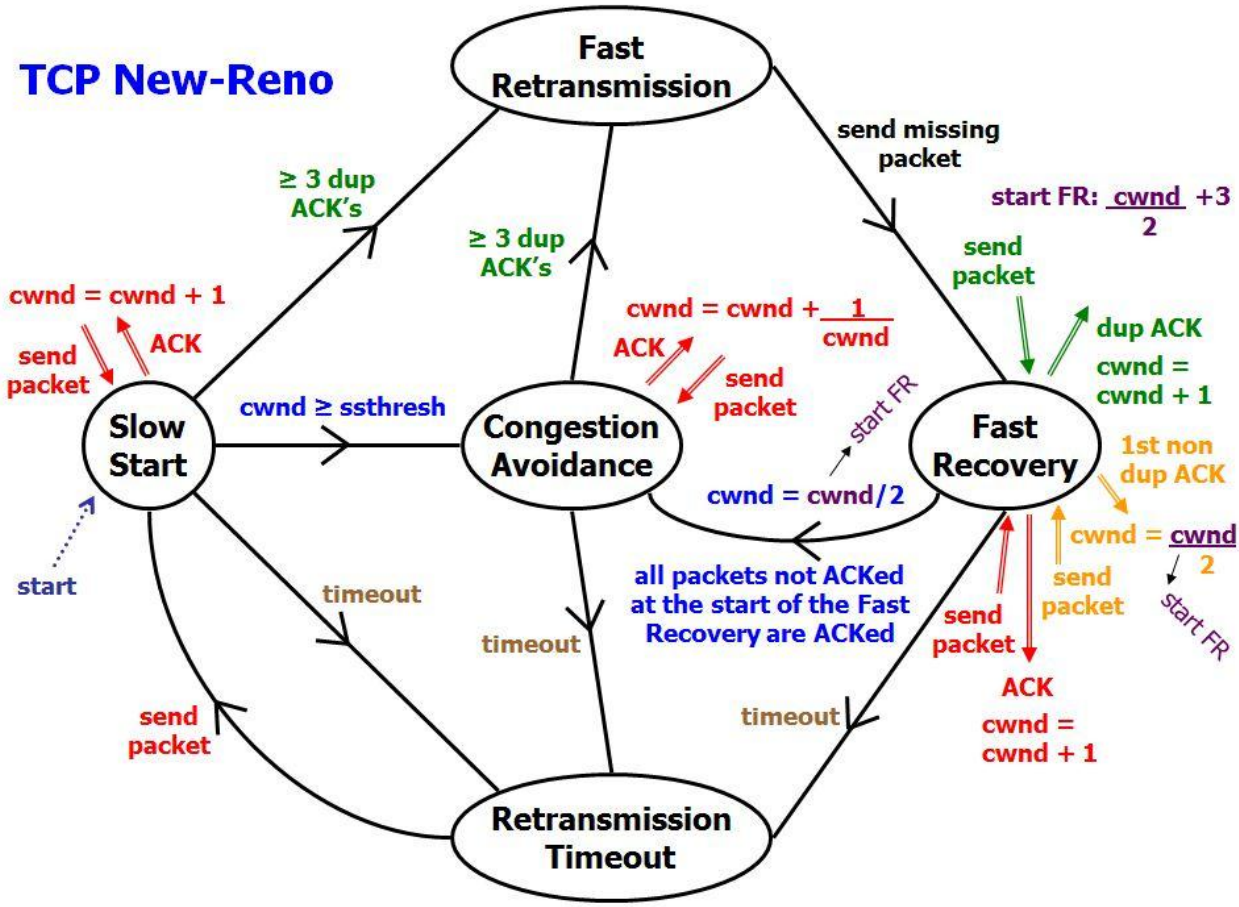


Figura 3.3 Diagrama Algoritm TCP New Reno

Principiu de funcţionare a algoritmul New Reno este urmatorul:

- Algorlitmul Slow Start
  Cwnd=1;
  Executa pentru fiecare pachet în parte : cwnd =cwnd+1'
  Pana cand (evenimentul de congestie sau cwnd > ssthresh
- Algoritmul de evitare a congestiei:

// atunci cand se iese din Slow Start si cwnd > ssthresh

Executa pentru fiecare Ack: cwnd=cwnd+ (1/cwnd)

Pana cand exipra timpul (Timeout) sau trec 3 Dupacks

- Algoritmul de retransmisie rapidã:
  // dupã ce s-au transmis 3 dupacks
  Retrimite pachetele pierdute;
  Apeleazã algoritmul de recuperare rapidã
- Algoritmul de recuperare rapidã
  // dupã retransmisia rapidã nu se intra in Slow Start
  Ssthresh=cwnd/2
  Cwnd=ssthresh+3;
  Pentru fiecare DACK primit;
  Cwnd= cwnd + 1
  Trimite un packet nou daca este permis;
  Dup ace s-a primit un ACK:
    Daca ACK este partial :
      Rãmãi in recuperare rapida;
      Retransmite urmãtorul pachet pierdut ;
    Daca ACK este complet:
      Cwnd= ssthresh;
      Iesi din recuperare rapida
      Alepeazã algoritmul de evitare a congestiei;

- Cand se termina timpul de Timeout:
  Ssthresh = cwnd/2;
  Cwnd =1;
  Apeleaza algoritmul de Slow Start

4 . Modificare Algoritm

4.1 Introducere

Problema atât cu Reno cât şi cu NewReno este că în algoritmul de recuperare rapidă, TCP înjumătăţeşte fereastra de congestie, indiferent de starea reţelei. O altă problemă cu NewReno este că, atunci când nu există pierderi de pachete, dar pachetele sunt reordonate cu mai mult de trei confirmări duplicat, NewReno intră în mod eronat în recuperare rapidă, iar acest aspect duce iar la înjumătăţirea fereastrei de congestie. Deoarece fereastra TCP de congestie controlează numărul de pachete pe care un expeditor TCP le poate trimite în reţea, în orice moment, prin urmare, procesul de stabilire a dimensiunii ferestrei de congestie la jumătate din valoarea face ca algoritmul NewReno TCP să fie ineficient în ceea ce priveşte utilizarea capacităţii de legătură.

Reno şi New Reno retransmite cel mult un pachet pierdut în unitatea de timp pentru un drum dus-întors. Deoarece se urmăreşte următoarele seturi de metrici: rata de transmisie, network fairness, comportamentul ferestrei de transmisie şi rata de pierdere a pachetelor analizăm comporamentul algoritmului TCP NEW Reno cu Tahoe, Reno. Toţi algoritmii AQM menţionaţi, în momentul transmisiei, au fereastra de transmisie cu o valoare scăzută, ulterior crescând în mod constant până la un vârf, unde după un interval de timp fereastra va scade , rămânând constantă până la stabilirea unei noi conexiuni. Pentru fiecare conexiune nouă a existat o scădere a ratei de transmisie, urmată de o fază de stabilizare . Algortimul New Reno îşi păstrează comportamentul oscilant la fel şi Reno şi Tahoe.

În ceea ce priveşte împărţirea corectă a fluxurilor în reţea (Network Fairness) TCP New Reno are performanţe foarte bune, fluxurile cu debit mai mare nu consumă întreaga bandă , lăsând fluxurile cu debit mai mic, au o transmisie echitabilă .

Numărul de pachete pierdute în reţea este un factor important atunci când se analizează calitatea metodei de control al congestiei, deoarece pierderea de pachete, poate avea ca efect creşterea congestie. Chiar şi timeout, primirea de pachete triple este considerat un semn de congestionarea a reţelei. În funcţie de numărul de pachete triple, New Reno a înregistrat performanţele cele mai bune conform studiilor .

Deoarece New-Reno are la bază algortimul Reno, modificat la partea de Fast-recovery, iar Reno la rândul său are la bază algotimul Tahoe, imbunătăţind partea de fast Recovery, iar Tahoe este alcătuit din Slow start, congestion avoidance and fast retransmit, dacă se imbunataseste una din cele patru părţi ce ajută la funcţionarea algortimului (Slow start, congestion avoidance and fast retransmit, fast-recovery), acesta va reprezenta o versiune îmbunătăţită a algoritmului ce stă sub el. Am ales să modific partea de recuperare rapidă .

4.2 Soluţia propusă pentru modificarea algoritmului

Propun o modificare a algoritmul de recuperare rapidă . În timpul evitării congestiei , în cazul în care expeditorul TCP primeşte trei ACK duplicat, se trece în modul Fast Retransmit să Propun o modificare a algoritmul de recuperare rapidă . În timpul evitării congestiei , în cazul în care expeditorul TCP primeşte trei ACK duplicat, se trece în modul Fast Retransmit să retransmită pachete ce pare a fi pierdute fără a aştepta să expire cronometrul retransmisiei. Apoi se calculează fereastra nouă de congestie , se stabileşte ssthresh la maxim două segmente de cwnd , iar cwnd se stabileşte la valoarea ssthresh plus numărul de recunoaşteri duplicate primite şi continuă cu faza de recuperare rapidă. Expeditorul TCP incrementează cwnd cu unul pentru fiecare confirmare duplicat primită, şi trimite segmentul nou dacă este permis. Cu ACK parţială, se retransmit segmentele confirmate şi se procesează. Cu ACK complet, se stabileşte cwnd la valoarea ssthresh şi se apelează algoritmul de evitare a congestiei. În cazul în care expeditorul TCP detectează pierderile de expirare a timeout-ului , se stabileşte ssthresh la maxim de două segmente şi cwnd, şi seturile de cwnd la un segment, şi intră în algoritmul Slow Start.

Mecanismul propus evită aceste probleme prin modificarea algoritmului de revenire rapidă a TCP - NewReno. Ideea de bază este de a ajusta ferestrei de congestie (cwnd) a expeditorului TCP în funcţie de nivelul de congestie în reţea, pentru a permite transferul mai multor pachete la destinaţie. Ajustarea ferestrei de congestie are trei obiective principale. Prima dintre ele este de a utiliza resursele de reţea disponibile, al doilea este de a minimiza probabilitatea de congestie, iar al treilea este de a oferi o lăţime de bandă echitabil între mai multe conexiuni. Aceste obiective ar putea fi realizat prin adoptarea dimensiunii ferestrei de congestiei bazat pe starea reţelei.

5. Simulări

Ns2 este un simulator de evenimente , orientat pe simularea in rețea care să permită imitarea intr-o varietate de rețele locale și largi. El pune în aplicare diferite protocoale de retea (TCP, UDP), surse de trafic (FTP, web, CBR, exponențiale on / off), mecanisme de gestionare coadă (RED, DropTail), protocoale de rutare (Dijkstra), etc NS2 este scris în C + + și Otcl la separa de control și implementări cale de date. Simulatorul are o ierarhie de clase în C + + (ierarhia compilat) și o ierarhie corespunzătoare în interpretul Otcl (ierarhie interpretat) Motivul pentru care ns2 folosește două limbi este că sarcini diferite au cerințe diferite: Un exemplu de simulare de protocoale necesită manipularea eficientă de bytes și antete de pachete de a avea viteza . Pe de altă parte, în studiile de rețea în cazul în care obiectivul este de a varia parametri și a examina rapid o serie de scenarii de timp pentru a schimba modelul și rulați-l din nou, este mult mai important. În NS2, C + + este utilizat pentru a aplica detaliat protocol și, în general, pentru astfel de cazuri în care fiecare pachet de un flux trebuie să fie prelucrate. De exemplu, dacă doriți să pună în aplicare o nouă disciplină de așteptare, apoi C + + este limbajul de alegere. Otcl, pe de altă parte, este adecvat pentru configurarea și configurare. Otcl ruleaza destul de încet, dar poate fi schimbat foarte repede face construcția simulări mai ușor. În NS2, a compilate C + + a obiectelor poate fi pusa la dispoziția interpretului Otcl. În acest fel, obiectele gata făcute C + + obiecte pot fi controlate de la nivelul OTcl. Există destul de multe tutoriale de înțeles disponibile pentru utilizatorii NS-noi. Trecând prin, pentru exemplu, următoarele tutoriale ar trebui să vă dau o imagine destul de bună a modului de a crea scenarii simple, de simulare cu NS2:http://nile.wpi.edu/NS/ sau http://www.isi.edu/nsnam/ns/tutorial/index.html

OTcl

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

De ce două limbaje?

Network Simulator folosește două limbaje deoarece simulatorul are două tipuri de

obiective pe care trebuie să le realizeze. Pe de o parte, simulări detaliate ale protocoalelor necesită un limbaj de programare al sistemului, care poate manipula eficient bytes, pachete, anteturi și care poate implementa algoritmi ce rulează cu seturi foarte mari de date. Pentru aceste obiective, viteza la rulare este importantă, în timp ce rularea simulării, găsirea bug-urilor și repararea acestora este mai puțin importantă. Pe de altă parte, o mare parte a cercetărilor în rețele implică variații mici ale parametrilor sau configurațiilor sau numărului de scenarii. În aceste cazuri, timpul necesar iterației (schimbarea modelului și rularea acestui din nou) este mai importantă. Din moment ce configurația rulează o singură dată (la începutul simulării), timpul de rulare pentru această parte este mai puțin important.
NS împacă cele două cerințe cu cele două limbaje, C++ și OTcl. C++ este rapid la rulare, însă mai greoi la schimbare, făcându-l potrivit pentru implementările detaliate ale protcoalelor. OTcl rulează mult mai greu, însă poate fi schimbat foarte rapid (și interactiv), făcându-l ideal pentru configurarea simulărilor.

Network Animator

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

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

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

Instalarea simulatorului

Simulările au fost realizate cu ajutorul simulatorului Network Simulator v2.35 rulând pe un sistem de operare Ubuntu 10 instalat pe o maşină virtuală creată pe un Notebook HP 4520s, având următoarele specificaţii: processor Intel Core i3, memorie RAM 4GB, memorie externa 400GB.
Pentru instalarea simulatorului am urmat următorii paşi:
-descărcarea şi instalarea fişierului ns-2 all-in-one cu următoarele comenzi:

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

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

-setarea variabilelor [6]

$ gedit ~/.bashrc
$ source ~/.bashrc
-validare
$ cd ns-2.34
$ ./validate

Modificarea simulatorului

Pentru implementarea modificarilor propuse in capitolul anterior, evitarea trecerii în modul Fast Retransmit care retransmite pachete ce par a fi pierdute fără a aștepta să expire cronometrul retransmisiei a trebuit sa modific algoritmului de revenire rapidă a TCP - NewReno. Aceasta modificare se realizeaza prin inlocuirea algoritmului sursa din NewReno.cc si NewReno.h ale simulatorului NS2.

În fişierul sursă a algoritmului NewReno, NewReno.cc, am făcut următoarea modificare: în algoritmul de recuperare rapidă , Ssthresh nu va mai lua valoarea cwnd/2 , ci va ii va fi atribuita jumătate din cwnd ştiuta ulima data plus 3 (numărul de ACK neconfirmate în faza de congestie , dar recuperate în urma algoritmului de retransmitere rapidă. Modificarea in codul sursa este urmatoarea:

```
reno_action:
        recover_ = maxseq_;
        reset_rtx_timer(1,0);
        if (!lossQuickStart()) {
              trace_event("NEWRENO_FAST_RETX");
            last_cwnd_action_ = CWND_ACTION_DUPACK;
            last_cwnd_action_ = CWND_ACTION_ECN;
            slowdown(CLOSE_SSTHRESH_HALF|CLOSE_CWND_HALF);
            output(last_ack_ + 1, TCP_REASON_DUPACK);       // from top
                        dupwnd_ = numdupacks_;
            }
        return;
}
```

```
void NewRenoTcpAgent::recv(Packet *pkt, Handler*)
{
        hdr_tcp *tcph = hdr_tcp::access(pkt);
        int valid_ack = 0;

        /* Use first packet to calculate the RTT  --contributed by Allman */

      if (qs_approved_ == 1 && tcph->seqno() > last_ack_)
                endQuickStart();
      if (qs_requested_ == 1)
          processQuickStart(pkt);
        if (++acked_ == 1)
                basertt_ = Scheduler::instance().clock() - firstsent_;

        /* Estimate ssthresh based on the calculated RTT and the estimated
          bandwidth (using ACKs 2 and 3).  */

        else if (acked_ == 2)
                ack2_ = Scheduler::instance().clock();
        else if (acked_ == 3) {
                ack3_ = Scheduler::instance().clock();
                new_ssthresh_ = int((basertt_ * (size_ / (ack3_ - ack2_))) / size_);
                if (newreno_changes_ > 0 && new_ssthresh_ < ssthresh_)
                        ssthresh_ = new_ssthresh_;
        }
```

Scenariul Simulat :

Topologia utilizată este alcătuită din:
- 7 link-uri cu o lăţime de bandă de 10 Mbps
- 3 link bottleneck cu o lăţime de bandă de 1 Mbps şi un delay de 50 ms

Considerăm că avem sapte conexiuni FTP ce utilizează TCP, şi setăm dimensiunea maximă a ferestrei la 30. Fiecare dintre cele sapte noduri trimit trafic FTP pe parcursul întregii perioade de simulare (50 de secunde). Dimensiunea cozii de bottleneck este 25. Se alege dimensiunea de 552 de bytes pentru pachetul TCP (pachetul efectiv creat în timpul simulării are 592 bytes deoarece se adaugă 40 bytes ce corespund headerului).

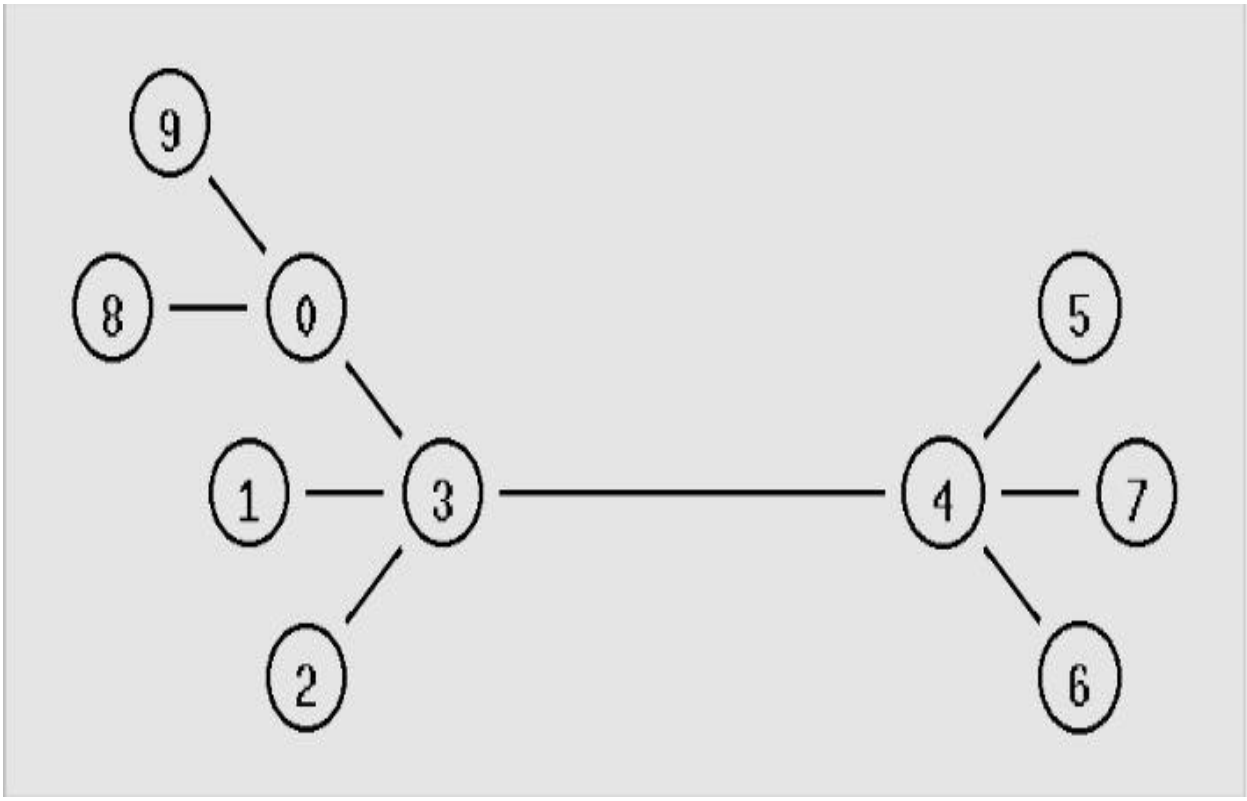


Figura 5.1 Topologia reţelei simulate

6. Prezentarea rezultatelor

Am rulati simularile pentru topologia descrisã in capitolul anterior , iar in continuare vom prezenta graficele care descriu variaţia ferestrei de congestive in cazul algoritmului modificat New Reno, New Reno, Reno si Tahoi , precum si graficele rezultate in urma comparatiei laţimii de bandã , numarul de pachete retrimise, pierdute si întârzierea între pachete în cazul fiecãrui algoritm din cei meţionaţi.

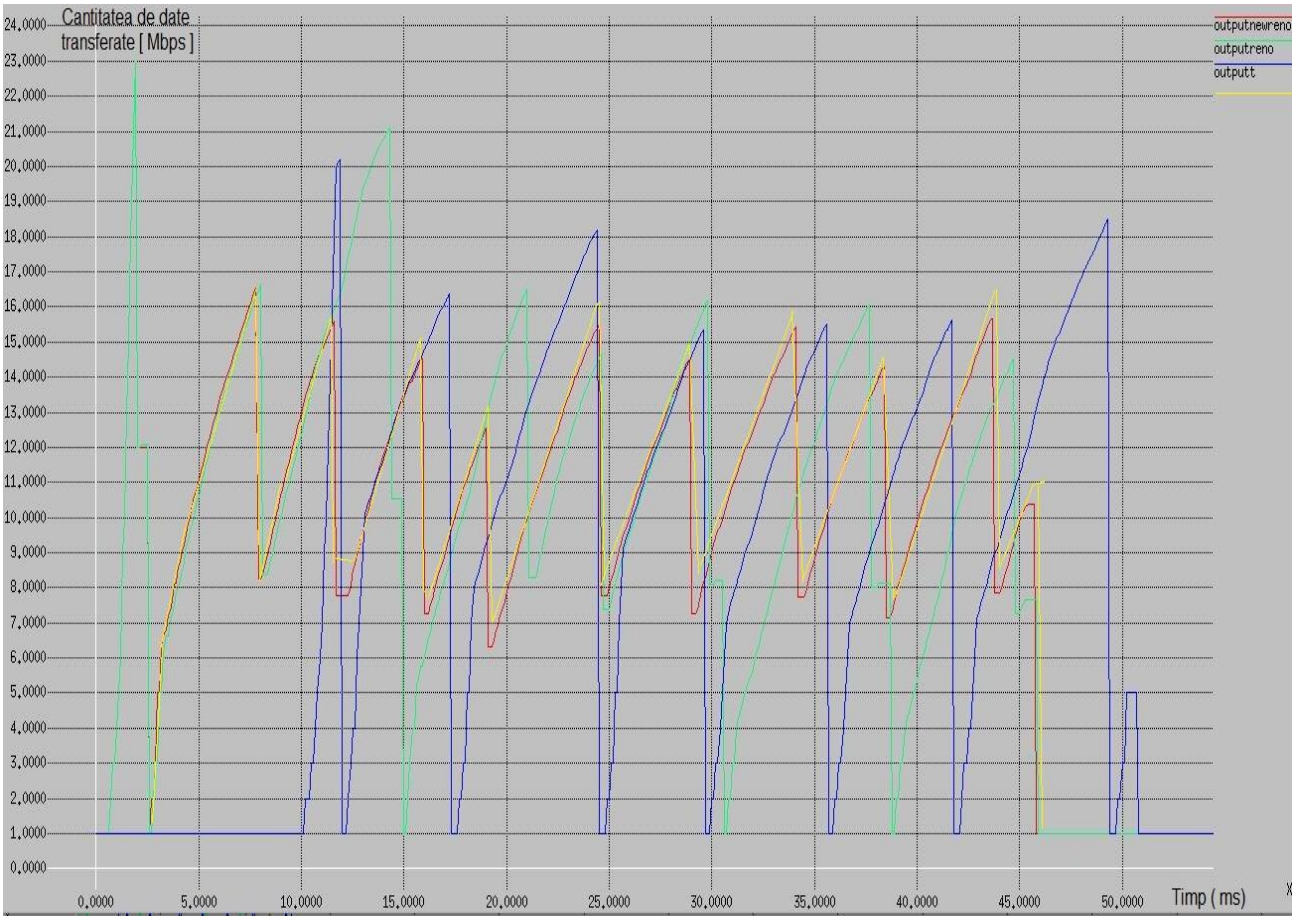
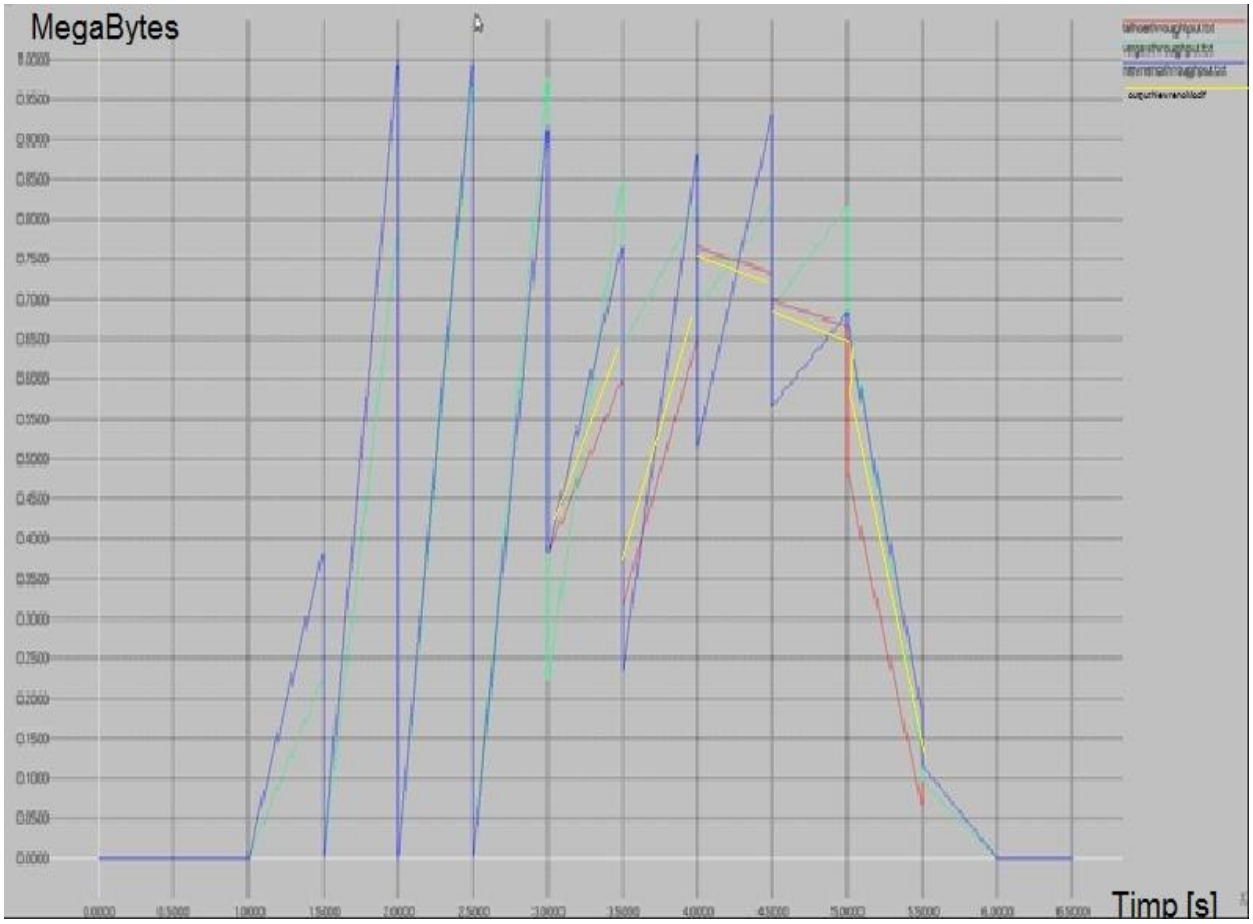


Figura 6.1. Variaţia ferestrei de congestie in cazul algoritmilor New Reno, Reno si Tahoe

Figura 6.1 prezintã comportamentul ferestrei de congestie (cwnd) pentru mecanismele de control al congestiei; TCP Reno, NewReno, Tahoe și NewRenoModificat . Figura arată schimbarea ferestrei de congestie cu timpul. În TCP Reno și NewReno, atunci când se constată pierderi de pachete; aceste protocoale selectează dimensiunea ferestrei la 50% din dimensiunea curentă. În TCP NewReno Modificat, de fiecare dată când pierderile de pachete sunt detectate, se modifică dimensiunea ferestrei de valori diferite în funcție de starea rețelei

Figura 6.2 Variaţia lãţimii de bandã in cazul algoritmilor New Reno, Reno si Tahoe, modificat New Reno



Se observa cã dintre cei patru algoritmi propuşi, în ceea ce priveşte lãţimea de bandã , perfotmanţle cele mai bune le are algoritmul New Reno , urmat de variant modificatã . Reno si Tahoe oscileaza foarte mult, ceea ce va duce la o transmisie destul de ineficientã , deoarece deşi uneori lãţimea de bandã are valori mari, aceasta nu se menţine , trecând de la valori mari la scãzute, aproape de nule chiar.

Variaţia numărului de pachete pierdute

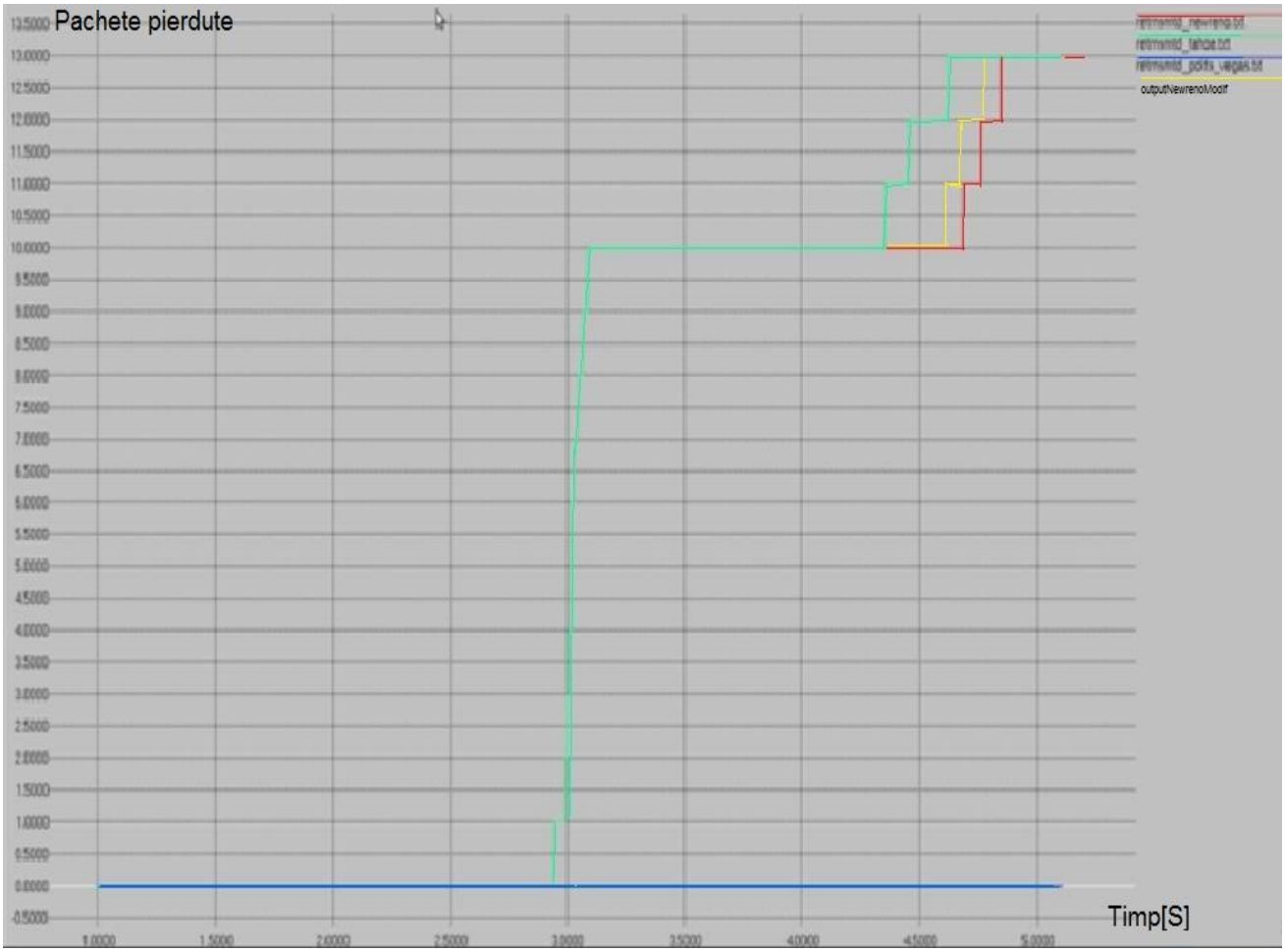


Figura 6.3 Variaţia numărului de pachete pierdute

Dupa cum se observa, cantitatea cea mai mica de pachete pierdute o are algoritmul New Reno urmat de New Reno modificat. Este de astepta ca algoritmul tahoe sa fie pe ultimul loc, deoarece pierderea este detectată atunci când timpul de timeout expiră înainte ca o confirmare să fie recepționată. În acel moment, Tahoe reduce fereastra de congestie la valoarea maximă a dimensiunii unui segment (MSS – maximum segment size) și se resetează la faza de „slow-start”.

Delay între pachete

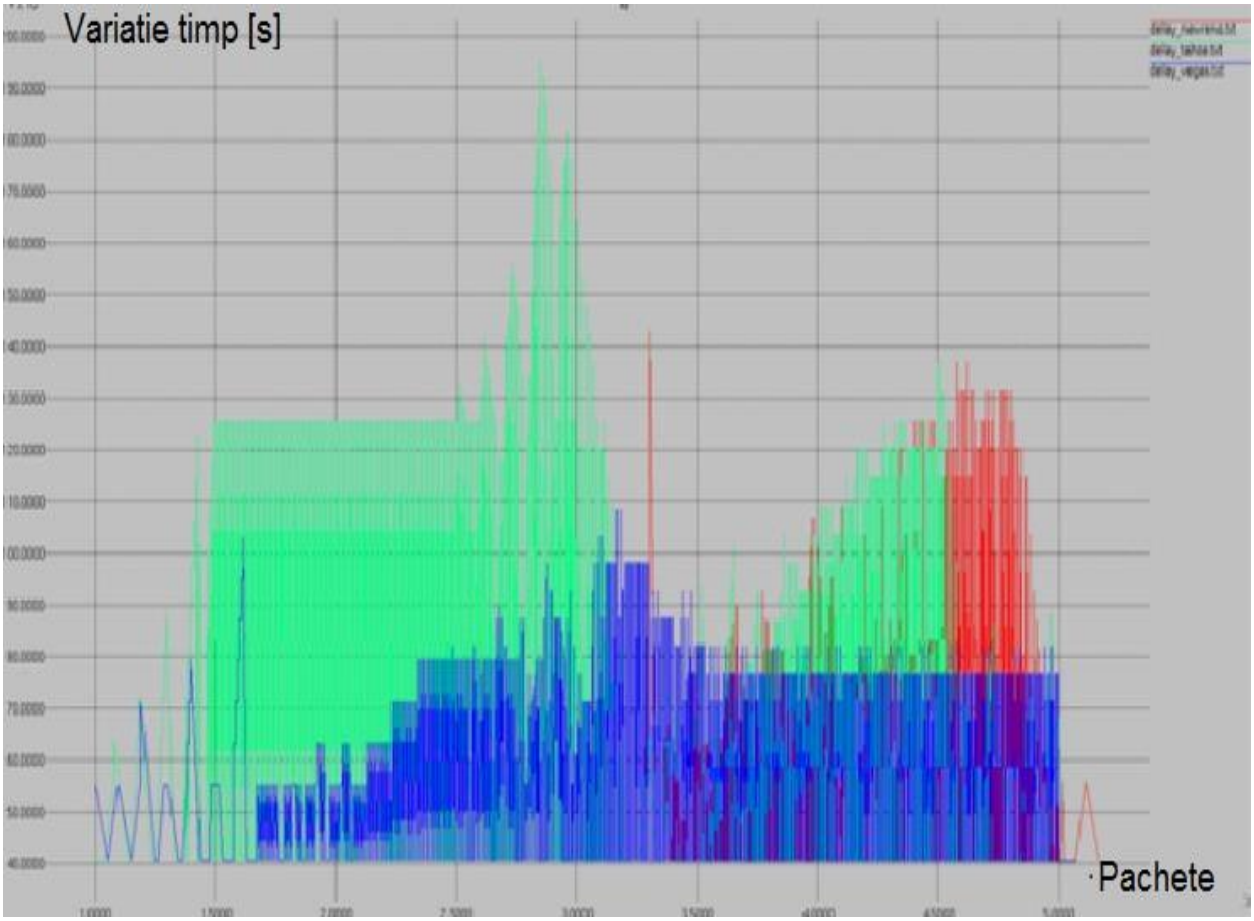


Figura 6.4 : Delay între pachete

Figura prezintã întârziere intre pachete în funcție de timp pentru Reno TCP, NewReno, și NewRenoModificat . Așa cum se arată în figură, spre deosebire NewReno care au întârzieri mai mari decât Reno, newRenoModificat reduce întârzierea de pachete. Acest lucru se datorează faptului că, în timpul algoritmul de recuperare rapidă, TCP NewReno așteaptă să recupereze toate pachetele pierdute și trimite câteva pachete noi, dar cu NewRenoModificat modificările ferestre congestie pe baza stării rețelei și ar putea trimite mai multe pachete noi, astfel se reduce întârzierea de pachete.

Tabelul 6.5 arată pierderile de pachete în funcție de timp pentru Reno TCP, NewReno, Tahoe și NewRenoModificat. Este clar din figură că cele patru protocoale au același comportament în timpul început lent și fazele de evitare a congestiei. Cu toate acestea, cu timpul, rata pierderilor de pachete de NewRenoModificat creste. Acest lucru se datorează faptului că, dimensiunea ferestrei congestie crește mai mult decât cea a Reno și NewReno, și mai multe pachete sunt transmise, astfel încât probabilitatea ca pierderile au crescut de asemenea. Acest lucru apare în figură, NewRenoModificat asigură pierderi mai mari decât NewReno. Acest comportament este observat, de asemenea, în NewReno TCP comparativ cu Reno.

Numarul de pachete pierdute in retea sunt:

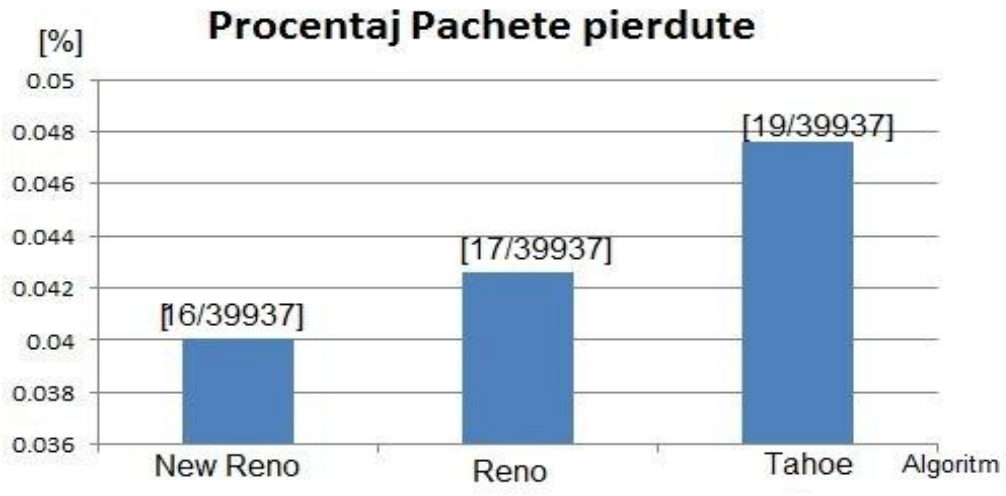


Figura 6.5 : Numãrul pachetelor pierdute in procente

Tip Algoritm	New Reno	Reno	Tahoe	NewrenoModificat
Numar pachete pierdute in retea	16	17	18	18

Tabelul 6.1 Numãrul pachetelor pierdute

Din totalul de 39937 de pachete trimise in retea.

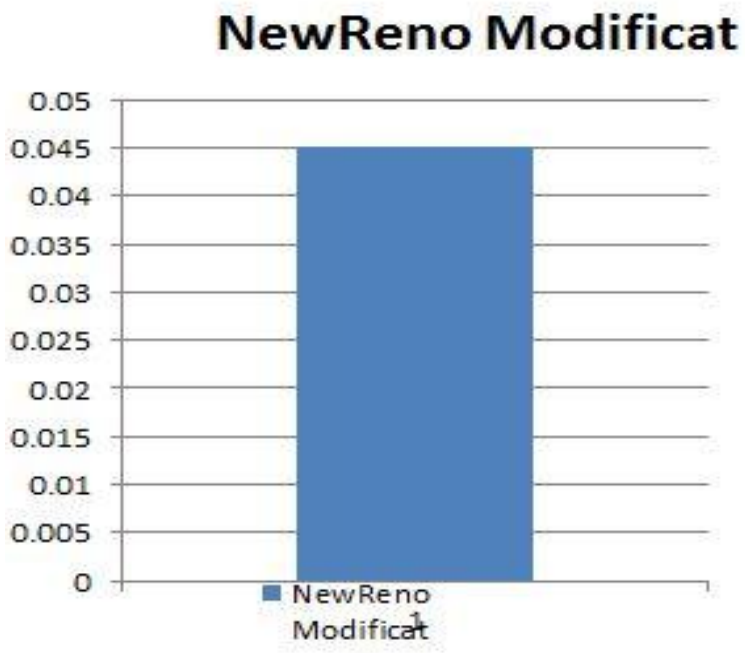


Figura 6.6 : Numãrul pachetelor pierdute in procente pentru algoritmul New Reno modificat

Pierderea de pachete poate fi cauzatã de o serie de factori, inclusiv degradarea semnalului peste medie a rețelei , pierderea de pachet se datoreaza canalului de congestie, pachetele sunt respinse în tranzit, fie din cauza unui hardware al rețelei defect, drivere de rețea defecte sau rutine de rutare corupte . În afară de aceasta, probabilitate de pierderea apachetelor este afectată și de raportul și distanța semnal-zgomot între emițător și receptor.

Figura6.6 arată pierderile de pachete în funcție de timp pentru New reno modificat . Este clar din figura6.5 si figura 6.6 că cele patru protocoale au același comportament în timpul Slow Start și fazele de evitare a congestiei. Cu toate acestea, cu timpul, rata pierderilor de pachete de NewReno modificat este crescut. Acest lucru se datorează faptului că, dimensiunea ferestrei congestie crește mai mult decât cea a Reno și NewReno, și mai multe pachete sunt transmise, astfel încât a crescut si probabilitatea pierderilor de asemenea. Acest lucru apare în figură6.6 , NewReno modificat cauzează pierderi mai mari decât NewReno. Acest comportament este observat, de asemenea, în NewReno TCP comparativ cu Reno.

End-to-End Delay : Întârziera end-to-end se referă la timpul necesar pentru un pachet să fie transmise într-o rețea de la sursă la destinație.

Tip Algoritm	New Reno	Reno	Tahoe	NewrenoModificat
End to End Delay	0,279	0,07	0,15	0,23

Tabelul 6.2 Intarzierea End-to-End

Durata medie a unui pachet de date pentru a ajunge la destinație include, de asemenea, întârzierea cauzată de procesul de descoperire a traseului și coada în transmiterea de pachete de date. Numai pachetele de date livrate cu succes la destinații sunt introduse in aceasta numărare.Se observa ca timpul cel mai mare de intre sursa si destinatie a unui pachet este inregistrat la algoritmul New Reno, la fel si in cazul New Reno modificat. Cel mai scurt timp de parcurgere a unui pachet în rețea este deținut de Reno.

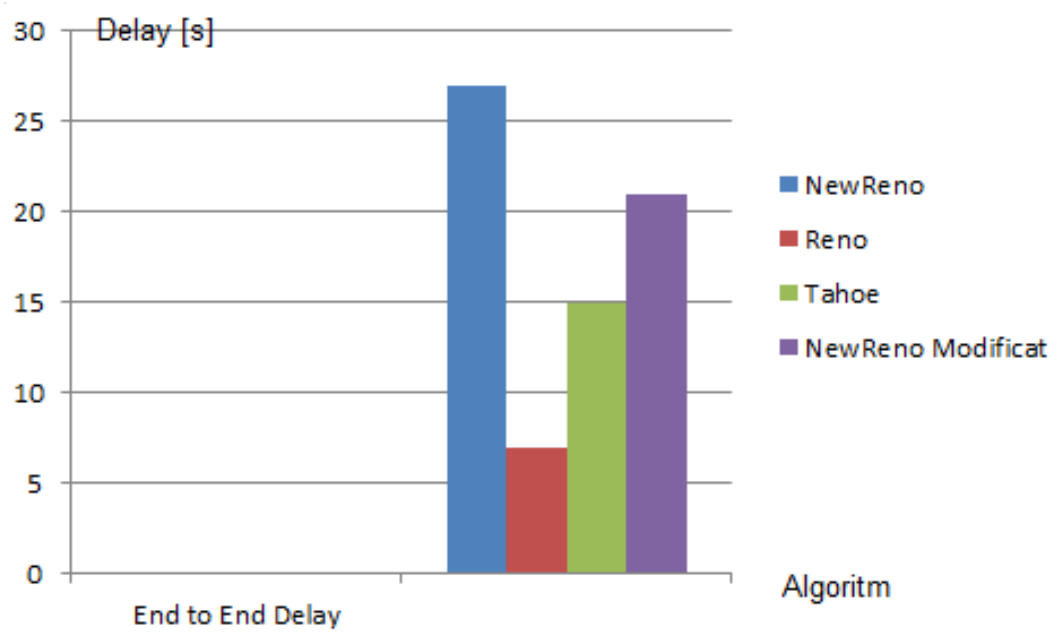


Figura 6.7 : End-to-End Delay

În cazul setarii paramerului de slow start threshold în faza de fast recovery , fereastra de congestie se schimba astfel:

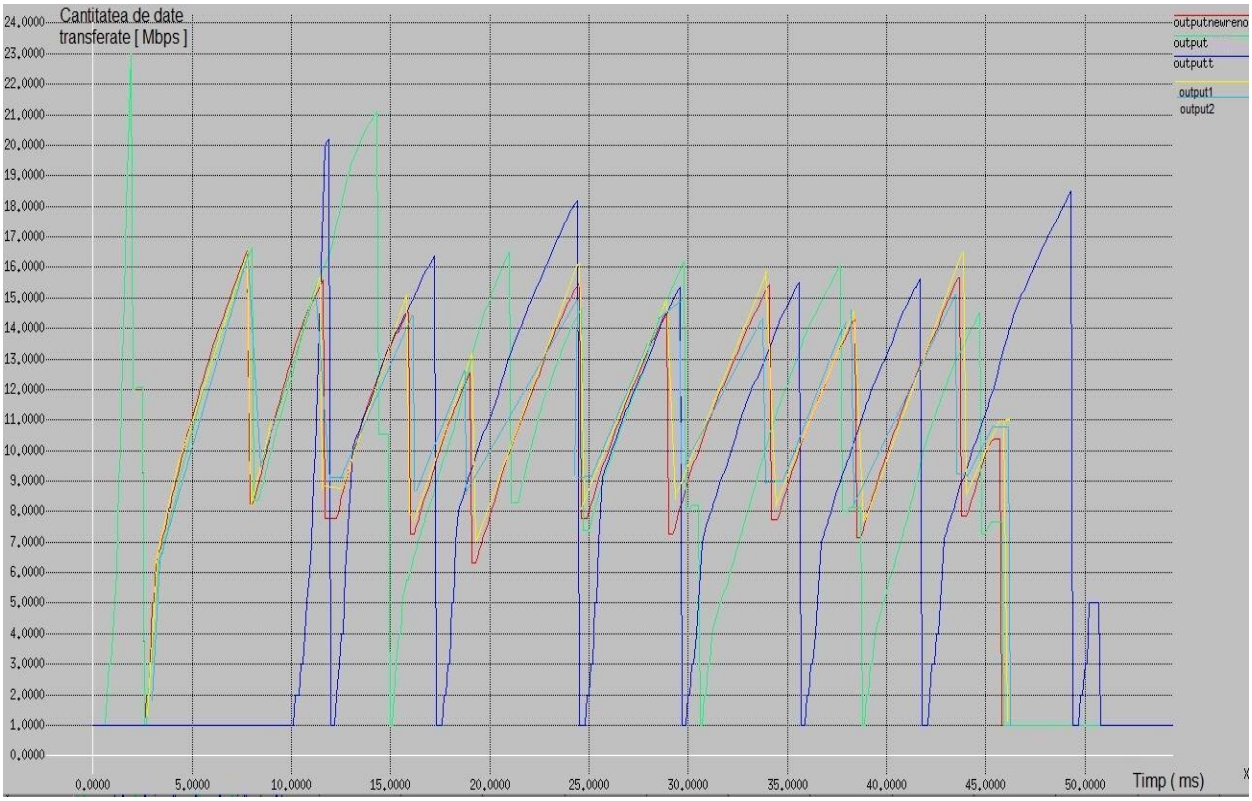
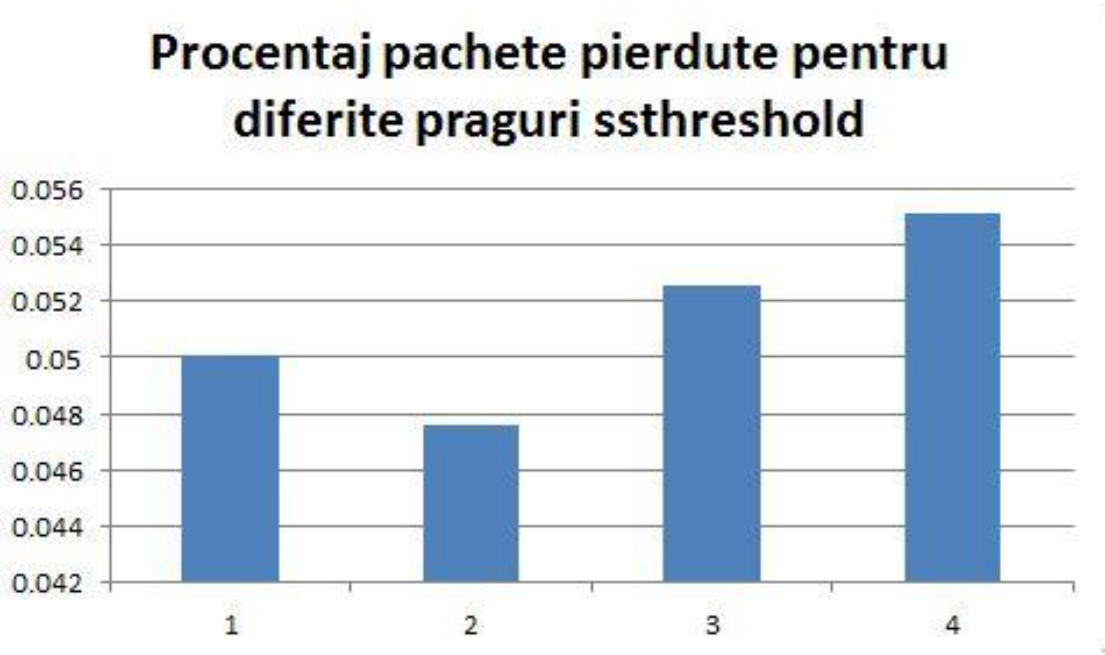


Figura 6.8 : Fereastra de congestive pentru diferite

Se observa o imbunataţire a ferestrei de congestive, deoarece aceasta nu mai prezinta oscilaţii atat de frecvente ca in cazul new Reno, fiind în jurul unui prag optim. Conform simulãrilor realizate, am observant cã pragul optim de setare a ssthresholt este egal cu patru sesimi din fereastra cwnd.

.

Dacã aleg pragul gasit ca optim, am ales sã analizez si numãrul de pachete pierdute



1 – ssthreshold= (cwnd/2)+15

2– ssthreshold= (3* cwnd)/5

3 – ssthreshold = (CWND *2)/3

4 – ssthresholt =(CWND * 3)/4

Pragul optim de 2/3 din fereastra cwnd are un dezavantaj. Acela produs de faptul ca procentajul numatului de pachete pierdute in retea are o usoara crestere, fata de New Reno, dar are totuşi performant mai bune decat Tahoe si Reno.

7. Concluzii

În această lucrare se propune un algoritm de recuperare rapidă modificat pentru creșterea performanțelor NewReno TCP. Mecanismul este dezvoltat prin adaptarea ferestrei de congestie a expeditorului TCP în funcție de nivelul de congestie în rețea. Acest nivel este determinat prin utilizarea Round Trip Time (RTT), care reprezintă un indicator pentru sarcinile de trafic pe rețea.Mecanismul propus este evaluată prin folosirea simulatorului NS2 și comparația cu NewReno TCP , Reno TCP si Tahoe TCP . Rezultatele simularilor arată că, în comporatie cu algoritmul de recuperare rapida modificat cu NewReno TCP, cel modificat îmbunătățește performanţele sale atât împotriva debitul și întârzierea pachet din cauza de a transfera mai multe pachete la destinație. Deși modificările suplimentare ale algoritmul de recuperare rapida a îmbunătățit performanța, mecanismul propus, NewRenoModificat, este ineficient în ceea ce privește pierderile de pachete, așa cum se arată în Figura . Îmbunătățiri suplimentare ar trebui să fie luate în considerare în activitatea viitoare pentru a îmbunătăți NewRenoModificat împotriva pierderilor de pachete.

New-Reno TCP este o variantă de Reno, cu o mica modificare în algoritmul de recuperare rapida. Acest lucru a fost făcut în scopul de a rezolva problema timeout atunci când mai multe pachete sunt pierdute formă aceeași fereastră. Rețineți că performanţele mai ridicate au fost obținute ca urmare a micii modificari Reno TCP. Deși New Reno rezolvă problema timeout atunci când mai multe pachete sunt pierdute formă aceeași fereastră, se poate retransmite doar un singur pachet pe Round Trip Time.

Se observã din capitolul 6 cã cele mai bune performanţe in ceea ce priveste lãţimea de bandã le are algoritmul New reno modificat , urmat de New Reno. Lãţimea de bandã nu oscileaza ca in cazul Reno si tahoe.

La capitolul numar de pachete pierdute, algoritmul New Reno modificat are performanţemai mici decat algoritmul original New Reno.

Varianta modificata a algoritmului New Reno desi prezintã îmbunãtãţiri ale ferestrei de congestie si a lãţimea de bandã, prezinta un deficit a performanţelor atunci cand vine vorba de numarul de pachete pierdute şi durata medie a unui pachet de a ajunge la destinatie.

8 . Bibliografie și referințe

1. W. Richard Stevens – “TCP/IP Illustrated” ,Volume 1The Protocols “ Editia a doua , Addison-Wesley, 1994.
2. Aleksander Malinowski , Bogdan M. Wilamowski - “Transmission Control Protocol—TCP”
3. Xiang Chen, Hongqiang Zhai, Jianfeng Wang, and Yuguang Fang – “ TCP Performance over Networks”
4. W. Stevens - TCP Slow Start, Congestion Avoidance,Fast Retransmit, and Fast Recovery Algorithms” Network Working Group , NoAO , 1997
5. B. Braden, ed., "Requirements for Internet Hosts --nCommunication Layers," RFC 1122, Oct. 1989.
6. V. Jacobson, "Modified TCP Congestion Avoidance Algorithm," April 30, 1990.
7. J. Padhye, and S. Floyd, “ On Inferring TCP Behavior”, Computer Communications Review ACM-SIGCOMM, Vol. 31, August 2001
8. M. Allman, V. Paxson, and W. Stevens, “TCP Congestion Control”, RFC 2581, IETF, April 1999
9. G. R. Wright, W. R. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1995.
10. V. Jacobson, "Congestion Avoidance and Control," Computer Communication Review, vol. 18, no. 4, pp. 314-329, Aug. 1988.
11. M. Miyake, and H. Inamura, "TCP Enhancement Using Recovery of Lost Retransmissions for NewReno TCP," Transactions of Information Processing Society Journal, Vol. 46 (9), pp. 2185-2195, September 2005.
12. Flavius Copaciu , Gabriel Lazar , Virgil Dobrota “Practical Analysis of TCP Implementations:Tahoe, Reno, NewReno”
13. K. Fall, and S. Floyd, “Simulation–Based Comparison of Tahoe, Reno and SACK TCP”, Computer Communications Review ACMSIGCOM , July 1996
14. HARDIK V. MIYANI, VISHV B. KUKADIYA, MR. KAPIL S. RAVIYA, MR.DHRUMIL SHETH - “PERFORMANCE BASED COMPARISON OF TCP VARIANTS ‘TAHOE, RENO, NEWRENO, SACK’ IN NS2 USING LINUX PLATFORM”
15. Celio Albuquerque, Brett J Vickers “Preventing Congestion Collapse and promoting fairness in Internet”IEEE/ACM Transactions, Volume (12)-01, Feb 2004.
16. T.V.Lakshman, U. Madhow, and B. Suter, “Window-based error recovery and flow control with
17. a slow acknowledgment channel: A study of TCP/IP performance,” in Proc. IEEE Infocom 1997.
18. Saleem-ullah Lar, Xiaofeng Liao and Songtao Guo - “Modeling TCP NewReno Slow Start and Congestion- Avoidance using Simulation Approach” IJCSNS International Journal of Computer Science and Network Security, VOL.11 No.1, January 2011
19. Jitendra Padhye, VictorFiroiu, “Modeling TCP Reno Performance A simple model & its Empirical Validation” IEEE/ACM Transactions Vol.ume(8)-02 April 2000.
20. Dongmin Kim, Beomjoon Kim, Jechan Han and Jaiyong Lee “Enhancement to the Fast Recovery Algorithm of TCP NewReno,ICOIN 2004, LNCS 3090, pp 332-341.”
21. Habibullah Jamal and Kiran Sultan, “Performance Analysis of TCP Congestion Control Algorithms”, International Journal of Computers and Communications, Volume (02)-01, 2008.
22. Jitendra Padhye, Victor Firoiu, Donald F. Towsley, Fellow, IEEE, and James F. Kurose, Fellow, IEEE
23. “Modeling TCP Reno Performance: A Simple Model and Its Empirical Validation” IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 8, NO. 2, APRIL 2000
24. J. Padhye, V. Firoiu, and D. Towsley, “A stochastic model of TCP Reno congestion avoidance and control,” Tech. Rep. UMASS-CS-TR-1999-02.
25. S.Floyd, “Congestion Control Principles”, ACIRI, September 2000 Robert Shorten, Fabian Wirth, Douglas Leith, “A positive systems model of TCP- like congestion control: Asymptotic results” , April 7, 2004

26. V. Jacobson, and M. J. Karels, "Congestion Avoidance and Control," Proceedings of ACM SIGCOMM, Vol.18 (4), pp. 314-329, August 1988.

27. Hanaa Torkey , Gamal Attiya , Ibrahim Z. Morsi - " Modified Fast Recovery Algorithm for Performance Enhancement of TCP-NewReno", International Journal of Computer Applications (0975 – 8887) Volume 40– No.12, February 2012
28. M. Niels, B. Chadi, A. Konstantin, and A. Eitan, "Inter-protocol fairness between TCP NewReno and TCP Westwood," The 3rd EuroNGI Conference on Next Generation Internet Networks, Vol.1, pp. 21-23, May 2007.
29. Jacobson, Van; Karels, MJ (1988). "Congestion avoidance and control". *ACM SIGCOMM Computer Communication Review* **18** (4): 314–329.
30. Corbet, Jonathan. "Increasing the TCP initial congestion window". LWN. Retrieved 10 October 2012.
31. Mark Allman, Vern Paxson, W. Richard Stevens (April 1999). "Fast Retransmit/Fast Recovery". *TCP Congestion Control*. IETF. sec. 3.2. RFC 2581. Retrieved 2010-05-01.
32. S. McCanne and S. Floyd, "ns Network Simulator",
33. http://www.isi.edu/nsnam/ns.
34. http://nsnam.isi.edu/nsnam/index.php/Installing_ns2.31_on_Ubuntu7.04
35. http://www.nabble.com/Network-Simulator-ns-2-f15582.html
36. http://nile.wpi.edu/NS/
37. www.acm.org
38. http://www.isi.edu/nsnam/ns/tutorial/
39. http://www.opalsoft.net/qos/DS.htm
40. www.ieee.org
41. http://www.netlab.ecnu.edu.cn/software/ns-class/tcp_8h.htm
42. http://www.unibuc.ro/prof/niculae_c_m/telecom/transm_ctrl_prot_tcp.htm
43.