Tema de casa

Retele de Calculatoare

Sliding window protocols

Studentii:

Bestea Vladut, 442 A

Gradinaru-Tascau Gabriel, 442 A

Pirvu Sorin, 442 A

Cuprins

Introducere (Bestea Vladut)

TCP este un protocol standard cu STD numarul 7. TCP este descris in RFC 793. Statutul sau este de recomandat, dar in practica orice implementare a TCP/IP care nu este folosita exclusiv pentru rutare va include TCP. TCP asigura considerabil mai multe facilitati pentru aplicatii decat UDP, mai ales recuperarea erorilor controlul transmisiei si siguranta. TCP este un protocol orientat pe conexiune spre deosebire de UDP care este fara conexiune. Cele mai multe aplicatii protocol, precum Telnet si FTP, utilizeaza TCP. Cele doua procese comunica unul cu celalalt printr-o conexiune TCP (InterProcess Communication - IPC).

TCP se poate compara cu o comunicatie telefonica. In aceasta analogie, un birou al unei firme este calculator host, numarul de telefon este portul, un apel telefonic este conexiunea, un angajat al firmei ce lucreaza in acest birou este o aplicatie (sau un protocol aplocatie). Mergand mai departe cu analogia, compania de telefonie este IP. O parte a telefoanelor din birou au numere publice (binecunoscute), iar alta parte sunt folosite pentru comunicatii curente. Firma realizeaza pentru public o serie de servicii, fiecarui serviciu fiindui alocate unul sau mai multe telefoane publice. La telefoanele publice destinate unui serviciu poate raspunde oricare dintre angajatii din birou care are activitati legate de acel serviciu. Atunci cand un angajat are de discutat mai mult cu un client, pentru a elibera linia telefonica publica, el comuta apelul pe o linie de comunicatie obisnuita. Analog procedeaza si aplicatiile server atunci cand accepta o conexiune cu o aplicatie client.

Principiul ferestrei glisante (Bestea Vladut)

Un concept suplimentar care sustine transmiterea de flux, cunoscut sub numele de *fereastra glisanta*(sliding window) face ca transmisia de flux sa fie eficienta.

Pentru a realiza siguranta, emitatorul transmite un pachet si apoi asteapta o confirmare inainte de a trimite altul. Datele calatoresc intre masini doar intr-o singura directie la un moment de timp chiar daca reteaua este capabila sa sustina o comunicatie simultana in ambele directii. Reteaua poate fi complet nefolosita in timpul in care masina

asteapta raspunsul (de exemplu in timpul in care masina receptoare calculeaza suma de control). Daca ne imaginam o retea ce prezinta intarzieri mari, problema devine clara: un protocol cu simpla confirmare pozitiva iroseste o cantitate substantiala din largimea de banda a unei retele deoarece el trebuie sa intarzie trimiterea unui nou pachet pana cand receptioneaza o confirmare pentru pachetul anterior.

Tehnica ferestrei glisante este o forma mult mai complexa a confirmarii pozitive si a retransmisiei fata de metoda simpla. Protocoalele cu fereastra glisanta folosesc mai bine largimea de banda deoarece ele permit utilizatorilor sa transmita multiple pachete inaintea primirii unei confirmari. Cea mai simpla cale de a observa operatia de alunecare a ferestrei este sa ne imaginam o secventa de pachete ce vor fi transmise. Protocolul produce o mica fereastra de dimensiune fixa si transmite toate pachetele care se afla in interiorul ei.

Emitentul grupeaza pachetele sale si utlilizeaza urmatoarele reguli:

- Emitentul poate transmite pachetele din fereastra fara sa receptioneze o confirmare (ACK), dar porneste cate un cronometru (care va semnaliza depasirea unui interval de tmp prestabilit) pentru fiecare dintre pachete.
- Receptorul trebuie sa confirme fiecare pachet primit, indicand numarul de secventa al ultimului pachet receptionat corect.
- Emitentul deplaseaza fereastra (o gliseaza) la fiecare mesaj de confirmare.

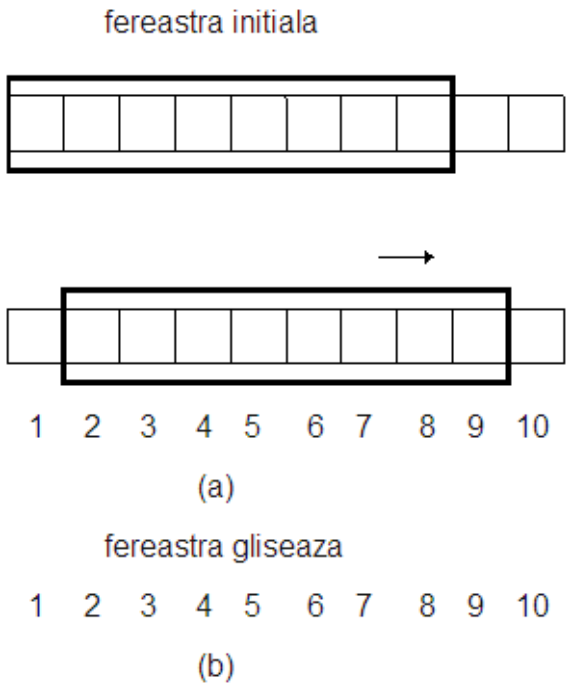


Figura (a): Un protocol cu fereastra glisanta avand 8 pachete in fereastra.

Figura (b): Fereastra gliseaza astfel incat pachetul 9 poate fi trimis doar cand a fost receptionata confirmarea pentru pachetul 1. Doar pachetele neconfirmate sunt retransmise.

O data ce emitatorul a primit o confirmare pentru primul pachet (figura b) fereastra poate aluneca si el trimite urmatorul pachet. Fereastra continua sa alunece atat timp cat confirmarile sunt receptionate.

Performantele protocoalelor cu fereastra glisanta depind de dimensiunea ferestrei si de viteza cu care reteaua accepta pachete.

Cu o fereastra de dimensiune unitara, protocolul cu fereastra glisanta devine un protocol cu simpla confirmare. Prin cresterea dimensiunii ferestrei este posibila eliminarea completa a timpului de neutilizare a retelei. Astfel in stare activa, emitatorul poate transmite pachete cu viteza cu care reteaua este capabila sa le transmita. Ideea de baza este: deoarece un protocol cu fereastra glisanta adecvat pastreaza reteaua complet incarcata cu pachete, el obtine o crestere substantiala de viteza fata de un protocol cu simpla confirmare pozitiva.

Teoretic, un protocol cu fereastra glisanta intotdeauna stie care pachete au fost confirmate si pastreaza un timer separat pentru fiecare

pachet neconfirmat. Daca un pachet este pierdut, timerul expira si emitatorul retransmite acel pachet. Cand un emitator gliseaza fereastra sa, ea trece peste toate pachetele confirmate. La receptorul final, protocolul software foloseste o fereastra identica cu cea a emitatorului, acceptand si confirmand pachetele sosite. Astfel, o fereastra imparte o secventa de pachete in trei grupe: acele pachete aflate in stanga ferestrei (toate sunt transmise, receptionate si confirmate), acelea din dreapta ce n-au fost inca trimise si cele aflate in interiorul ferestrei ce sunt transmise in prezent. Pachetul cu numarul cel mai mic din fereastra este primul pachet din acea secventa care nu a fost inca confirmat.

Probleme care pot sa apara

Pachetul al 2-lea s-a pierdut.

Emitentul nu va receptiona ACK 2, asa ca el va ramane pe pozitia 1 (ca in figura anterioara). De fapt, deoarece receptorul nu a primit pachetul 2, el va confirma pachetele 3, 4 si 5 cu un ACK 1, intrucat pachetul 1 a fost ultimul primit in secventa. La emitent, cronometrul pornit pentru pachetul 2 va semnaliza depasirea timpului de asteptare a confirmarii si acest pachet va fi retransmis. De observat ca receptia acestui pachet de catre receptor va determina emiterea confirmarii ACK 5, intrucat acum au fost receptionate cu succes toate pachetele de la 1 la 5 si fereastra emitentului se va deplasa cu patru pozitii dupa receptionarea ACK 5. Pachetul 2 a ajuns la destinatie, dar confirmarea s-a pierdut: Si acum emitentul nu primeste ACK 2, dar va primii ACK 3. ACK 3 este o confirmare pentru toate pachetele pana la 3 (inclusiv pachetul 2) si emitentul poate acum deplasa fereastra la pachetul 4.

Acest mecanism de fereastra glisanta asigura:

- transmisie sigura.
- folosire mai buna a latimii de banda a retelei (o mai mare viteza de transmisie).
- controlul transmisiei, deoarece receptorul poate intarzia confirmarea unui pachet, cunoscand memoria libera de care dispune si dimensiunea ferestrei de comunicare.

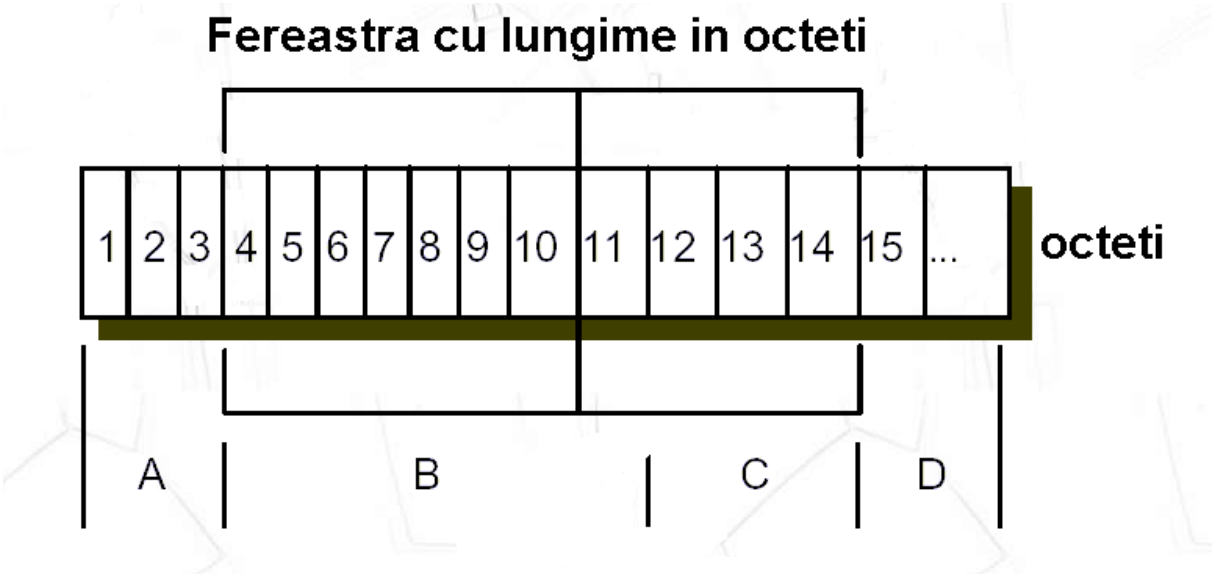
Principiul ferestrei glisante aplicat la TCP

(Gradinaru Gabriel)

Principiul ferestrei glisante este folosit de catre TCP, insa cu cateva diferente:

- Deoarece TCP asigura o conexiune de flux de octeti (byte-stream connection), numerele de secventa sunt atribuite fiecarui bit din stream. TCP imparte acest flux continuu de biti in segmente TCP pentru a le transmite. Principul ferestrei glisante este folosit la nivel de octet; adica, segmentele trimise si confirmarile vor transporta numere de secventa pentru octet si dimensiunea ferestrei va fi exprimata in numar de biti in loc de numar de pachete.
- Dimensiunea ferestrei este determinata de catre receptor cand conexiunea este stabilita si este variabila in timpul transferului de date. Fiecare mesaj ACK va include dimensiunea ferestrei pe care receptorul este apt sa opereze in acest moment particular.

Fluxul de date al emitentului poate fi reprezentat ca in figura urmatoare.



- Octetii A sunt octetii transmisi despre care s-a receptionat confirmarea de primire.
- Octetii B sunt octetii trimisi despre care nu s-a receptionat confirmarea de primire.
- Octetii C pot fi trimisi fara a se astepta nici o confirmare.
- Octetii D sunt octeti care nu pot fi inca transmisi.

Trebuie retinut ca TCP grupeaza octetii in segmente si numai un segment poarta numarul de secventa al primului octet din segment.

Ferestre cu dimensiuni variable si controlul fluxului

(Gradinaru Gabriel)

O diferenta intre protocolul TCP cu fereastra glisanta si protocolul simplificat cu fereastra glisanta, apare deoarece TCP-ul permite ca dimensiunea ferestrei sa varieze in timp. Fiecare confirmare care specifica cum au fost receptionati octetii contine o fereastra de avertisment care indica cati octeti suplimentari pot fi acceptati de receptor. Gandim fereastra de avertisment ca specificand dimensiunea curenta a bufferului receptorului. Ca raspuns la o fereastra de avertisment ce indica o valoare crescatoare, emitatorul creste dimensiunea ferestrei sale si trimite octeti care n-au fost confirmati. Ca raspuns la o fereastra care indica o valoare descrescatoare emitatorul isi scade dimensiunea ferestrei sale si opreste trimiterea octetilor care in urma modificarii se afla in acum in afara ferestrei. In schimb micile anunturi insotind confirmarile determina ferestrele sa-si modifice dimeniunea tot timpul determinand o distributie variabila (din punctul de vedere al cantitatii de informatie trimisa).

Avantajul folosirii unei ferestre cu dimensiune variabla este ca furnizeaza un control al fluxului precum si siguranta transportului. Daca bufferul receptorului incepe sa se umple, el nu poate accepta mai multe pachete, asa ca el trimite un avertisment de micsorare a ferestrei. In cazul extrem, receptorul anunta o dimensiune de fereastra zero pentru a opri intreaga transmisie. Mai tarziu cand bufferul incepe sa aiba spatiu disponibil, receptorul anunta o dimensiune de fereastra corespunzatoare pentru a reporni fluxul.

Existenta unui mecanism pentru controlul fluxului este esential intr-o retea ca Internetul, unde masinile au viteze variate si vehicularea informatiilor prin retele si rutere se face la viteze si capacitati diferite. Acolo sunt doua probleme reale. Prima, protocoalele de retea doresc un control al fluxului pe conexiuni, intre sursa si ultima destinatie. De exemplu, cand un minicalculator comunica cu un mainframe, minicalculatorul doreste sa se regleze fluxul de date catre el sau in caz contrar software-ul va fi rapid supraincarcat. Astfel, TCP-ul trebuie sa implementeze controlul fluxului pe o conexiune pentru a garanta siguranta distributiei. A doua, protocoalele de retea doresc un mecanism de control al fluxului care sa permita sistemelor intermediare (de exemplu ruterelor) sa controleze o sursa care transmite mai mult trafic decat poate accepta o masina.

Cand masinile intermediare devin supraincarcate, apare o problema numita *congestie*, iar mecanismul care o solutioneaza este numit *controlul congestiei*. TCP-ul foloseste schema sa cu fereastra glisanta pentru a solutiona problema controlului fluxului pe o conexiune.

Operatiile protocolului (Pirvu Sorin)

Emițatorul si receptorulau fiecare un numar de secventa curent n_t si n_r, respectiv. Fiecare dintre ele au, de asemenea o dimensiune a fereastrei w_t si w_r. Dimensiunile ferestrelor pot varia, dar in implementarile simple sunt fixe. Dimensiunea ferestrei trebuie sa fie mai mare decat zero pentru a se realize progres . In implementarile uzuale n_t este urmatorul pachet de transmis, adica numarul de ordine al primului pachet care nu a fost inca transmis. De asemenea, n_r este primul pachet care nu a fost primit inca. Ambele cifre sunt in crestere monotona in timp, si cresc tot timpul.

Receptorul poate pastra, de asemenea, evidenta celui mai mare numar de ordine care nu a fost inca primit; variabila n_s este numarul de ordine al numarului cel mai mare de secventa primit plus unu. Pentru receptoare simple, care accepta doar pachete in ordine (WR = 1), acest lucru este la fel ca nr, dar poate fi mai mare daca $w_r > 1$. Notam distinctia: toate pachetele mai jos de nr s-au primit, nici un pachet mai sus de ns nu a fost primit, intre n_r si n_s unele pachete au fost primite.

Cand receptorul primeste un pachet, aceasta actualizeaza variabilele sale in mod corespunzator si transmite o confirmare cu nr nou. Transmițatorul ține evidența a cea mai mare le-a primit confirmarea, n_a. Transmițatorul stie ca toate pachetele pana la, dar fara n_a incluse nu au fost primite, dar nu este sigur cu privire la pachetele intre n_a și n_s; adica $n_a \leq n_r \leq n_s$.

Numerele de secventa asculte intotdeauna regula care $n_a \leq n_r \leq n_s \leq n_t \leq n_a + w_t$. Asta este:

$n_a \leq n_r$: Cea mai mare confirmare primita de emițator nu poate fi mai mare decat cel mai mare n_r recunoscut de receptor.

$n_r \leq n_s$: Interval de pachete pe deplin primite nu se poate extinde dincolo de finalul de pachete primite parțial.

$n_s \leq n_t$: Cel mai mare pachet primit nu poate fi mai mare decat cel mai mare pachet trimis.

$n_t \leq n_a + w_t$: Cel mai mare pachet trimis este limitat de cea mai mare confirmare primita și dimensiunea ferestrei de transmisie

Operatiile transmitatorului (Pirvu Sorin)

Ori de cate ori operațiunea transmițatorul are date de transmis, el poate transmite pana la w_t pachete inainte de a primi cele mai recente confirmari na. Adica se poate transmite n_t pachete de , atata timp cat $n_t < n_a+w_t$... In absența unei erori de comunicare, emițatorul primește in curand o confirmare pentru toate pachetele pe care le-a trimis, lasand n_a egal cu n_t. Daca acest lucru nu se intampla, dupa o intarziere rezonabila, emițatorul trebuie sa retransmita pachetele intre n_a si n_t. Tehnici pentru definirea "intarziere rezonabila" pot fi extrem de elaborate, dar ele afecteaza numai eficiența, fiabilitatea de baza a protocolului ferestrei glisante nu depinde de detalii.

Operatiile receptorului (Gradinaru Gabriel)

De fiecare data cand un pachet cu numarul x este primit, receptorul verifica pentru a vedea daca se incadreaza in fereastra de primire, $n_r \leq x < n_s+w_r$. (Cele mai simple receptoare trebuie doar sa urmareasca o valoare $n_r = n_s$.) In cazul in care se incadreaza in fereastra, receptorul le accepta. In cazul in care este numerotat n_r, numarul de ordine la primire este crescut cu 1, și, eventual, mai mult, daca pachetele suplimentare consecutive au fost primite anterior și stocate. Daca $x > n_r$, pachetul este stocat pana cand toate pachetele anterioare au fost primite. Daca $x \geq n_s$, acesta din urma este actualizat la $n_s=x+1$. Daca numarul de pachete nu este in fereastra de receptie, se debaraseaza receptorul de ea si nu se modifica n_r sau n_s. Daca pachetul a fost acceptat sau nu, receptorul transmite o confirmare care conține n_r curent. (Confirmarea poate include, de asemenea, informații despre pachetele suplimentare primite intre n_r sau n_s, dar nu mai ajuta la eficiența.) Retineti ca nu exista nici un punct avand in fereastra de primire w_r mai mare decat fereastra de transmisie w_t , pentru ca nu este nevoie sa va faceți griji cu privire la primirea un pachet care nu va fi transmis, gama utila este $1 \leq w_r \leq w_t$.

Raspunsul la congestie (Pirvu Sorin)

In practica, TCP-ul trebuie sa raspunda la congestiile din retea. Congestia este o conditie de intarziere grava determinata de supraincarcarea cu datagrame a unuia sau a mai multor puncte de comutare (cum ar fi de pilda ruterele). Cand apare o congestie, intarzierile cresc si ruterul incepe sa puna datagramele intr-o coada pana ce va putea sa le ruteze. Trebuie reamintit faptul ca fiecare ruter are o capacitate finita de stocare si ca datagramele concureaza pentru acea stocare (de exemplu, intr-o retea bazata pe datagrame nu exista o prealocare de resurse pentru o conexiune TCP individuala). In cazul cel mai defavorabil, numarul total al datagramelor venite la ruterul congestionat creste pana cand ruterul atinge capacitatea maxima de stocare dupa care incepe sa piarda datagrame care mai apar.

In mod obisnuit un punct final nu cunoaste detaliile locului in care a aparut congestia si nici de ce a aparut. Din nefericire, majoritatea protocoalelor de transport folosesc timpul de asteptare si retransmisia, asa ca ele contribuie la cresterea intarzierilor prin retransmisia datagramelor. Aceasta agraveaza congestia in loc s-o usureze. Daca nu se verifica incarcarea cu trafic a retelei si se continua retransmisia, intazierile cresc pana cand reteaua devine nefolosibila. Aceasta problema este cunoscuta sub numele de *colapsul congestiei*.

Ca sa evite colapsul congestiei, TCP-ul trebuie sa reduca rata de transfer cand congestia apare. Ruterele supravegheaza lungimea cozilor si folosesc o tehnica asemanatoare mesajului ICMP care indica o sursa prea rapida, pentru a informa gazdele ca o congestie este pe cale sa apara; protocoalele de nivel transport precum TCP-ul pot ajuta la evitarea congestiei prin reducerea automata a ratei de transfer ori de cate ori apar intarzieri.

Ca sa evite congestia, standardul TCP recomanda acum doua tehnici: *pornire usoara* si *descresterea multiplicativa*. Am spus ca pentru fiecare conexiune, TCP-ul trebuie sa stie dimensiunea ferestrei receptorului (de exmplu utilizand fereastra de anunt). Ca sa controleze congestia TCP-ul foloseste o a doua limita, numita limita de congestie a ferestrei sau fereastra de congestie. La orice moment de timp, TCP-ul reactioneaza ca si cum marimea ferestrei este:

Fereastra_permisa = min(dimensiune_fereastra_receptor, freastra_congestie)

In starea normala a unei conexiuni necongestionate fereastra de congestie are aceeasi dimensiune ca si fereastra receptorului. Reducand fereastra de congestie se reduce traficul pe care TCP-ul il injecteaza in conexiune. Pentru estimarea dimensiunii ferestrei de congestie, TCP-ul presupune ca majoritatea datagramelor se pierd intr-o congestie si foloseste urmatoarea strategie: evitarea congestiei prin descrestere multiplicativa: dupa pierderea unui segment, reduce la jumatate dimensiunea ferestrei de congestie (poate cobora pana la cel mult un segment). Pentru toate segmentele care raman in fereastra permisa, reduce exponential timpul de retransmisie.

Deoarece TCP-ul reduce fereastra de congestie la jumatate pentru fiecare pierdere, el va descreste exponential fereastra daca aceastea continua. Cu alte cuvinte, daca congestia este posibila, TCP-ul reduce exponential volumul traficului si rata retransmisiei. Daca piederile continua, TCP-ul limiteaza transmisia la o singura datagrama si continua sa dubleze timpul de asteptare inaintea retransmisiei. Ideea este sa furnizezi o reducere rapida si consistenta a traficului pentru ca ruterele sa aiba suficient timp pentru a sterge datagramele exitente in cozile lor.

Cum sesizeaza TCP-ul cand s-a terminat congestia? O ideea ar fi ca TCP-ul sa inverseze descresterea multiplicativa si sa dubleze fereastra de congestie cand traficul reporneste. O astfel de actiuni sunt nerecomandate deoarece au ca rezultat un sistem instabil ce oscileaza aiurea intre un trafic oprit si congestie. In schimb TCP-ul foloseste o tehnica numita *pornire usoara* ce gradeaza cresterea transmisiei.

Redresarea prin pornire usoara (de tip aditiv): ori de cate ori se porneste traficul intr-o conexiune noua sau traficul creste dupa o perioada de congestie, se porneste cu o fereastra de congestie avand dimensiunea unui singur segment si creste aceasta dimensiune cu o unitate (segment) la fiecare moment de timp in care soseste o confirmare.

Acest mecanism evita inundarea retelei cu un trafic suplimentar imediat dupa eliminarea unei congestii sau cand noi conexiuni pornesc deodata.

Termenul de pornire usoara poate fi impropriu deoarece fata de conditiile ideale, pornirea nu este foarte lenta. TCP-ul initializeaza fereastra de congestie cu valoarea 1, trimite un segment initial si

asteapta. Cand confirmarea soseste, el creste dimensiunea ferestrei de congestie la 2 si asteapta. Cand cele 2 confirmari sosesc fiecare dintre ele va incrementa fereastra de congestie cu 1 astfel incat TCP-ul va trimite 4 segmente. Confirmarile sosite pentru acele ferestre vor creste la 8 dimensiunea ferestrei de congestie si asa mai departe ajungandu-se foarte aproape de limita ferestrei receptorului. Chiar pentru ferestre foarte mari, el necesita doar $\log_2 N$ calatorii complete inainte ca TCP-ul sa poata trimite N segmente.

Ca sa se evite cresterea prea rapida a dimensiunii ferestrei si sa se determine o congestie suplimentara, TCP-ul aduga o restrictie suplimentara. Odata ce fereastra de congestie ajunge la jumatate din dimeniunea ferestrei avute inaintea aparitiei congestiei, TCP-ul intra intr-o faza de *evitare a congestiei* si incetineste rata de incrementare. In timpul fazei de *evitare a congestiei*, el creste dimensiunea ferestrei de congestie cu 1 doar daca toate segmentele din fereastra au fost confirmate.

Sindromul “silly window” si pachetele mici

(Bestea Vladut)

Probleme serioase apar cand aplicatiile receptoare si emitoare functioneaza la viteze diferite. Ca sa intelegem problema reamiontim ca TCP-ul buffereaza datele, si sa vedem ce se intampla daca aplicatia receptoare alege sa citeasca datele sosite cu viteva de un octet la un moment de timp. Cand o conexiune este prima data stabilita, TCP-ul receptor aloca un buffer de K octeti si foloseste campul WINDOW din segmentele de confirmare pentru a avertiza emitatorul de dimensiunea bufferului sau. Daca programul de aplicatie genereaza rapid datele, TCP-ul va transmite segmentul de date avand o dimensiune egala cu dimensiunea maxima a ferestrei. Eventual emitatorul poate receptiona o confirmare care indica ca intregul continut al ferestrei a fost receptionat dar bufferul nu mai are spatiu suplimentar.

Cand aplicatia receptoare citeste un octet de date din buffer, un octet din spatiul bufferului va deveni liber. Deoarece TCP-ul de pe masina receptoare genereaza o confirmare care foloseste campul WINDOW pentru a informa emitatorul ca a fost eliberat spatiu in bufferul sau, receptorul va anunta in cazul nostru o fereastra de un octet. In consecinta TCP-ul emitator va raspunde prin trimiterea unui segment care contine un singur octet de date.

Desi fereastra de avertisment de un octet lucreaza corect (pentru a pastra bufferul plin) rezultatul va fi o serie de segmente mici de date. TCP-ul emitator trebuie sa formeze un segment ce contine un singur octet de date plasandu-l intr-o datagrama IP. Cand aplicatia receptoare citeste un alt octet, TCP-ul va genera o alta confirmare ce va determina emitatorul sa trimita un alt segment care sa contina doar un octet de date. Rezultatul acestei interactiuni poate fi o stare stabila in care TCP-ul trimite cate un singur segment pentru fiecare octet de date.

Un astfel de tip de transfer ocupa inutil din largimea de banda a retelei deoarece fiecare datagrama contine doar un octet de date generand si o supraincarcare inutila cu calcul intrucat atat emitatorul cat si receptorul trebuie sa proceseze fiecare segment (cu tot ceea ce implica: incapsulare, verificarea sumei de control, examinarea numarului de secventa, etc).

Am vazut cum o mica fereastra disponibila determina segmente mici, emitatorul fiind obligat sa trimita segmente se contin doar mici cantitati de date. Sa ne imaginam o implementare TCP agresiva, care trimite date ori de cate ori exista spatiu disponibil in bufferul receptorului, si sa vedem ce se intampla daca o aplicatie genereaza un singur octet de date la un moment de timp. Dupa ce genereaza un octet de date, TCP-ul creaza si transmite un segment. Deasemenea, TCP-ul poate sa trimita un mic segment daca o aplicatie genereaza date in blocuri de dimensiune fixa formate din B octeti, iar TCP-ul emitator extrage date din buffer in blocuri avand dimensiunea maxima a unui segment (M), unde M□B, deoarece ultimul bloc din buffer poate fi mai mic.

Cunoscut sub numele de sindromul “silly window” (SDS), problema prezentata chinuie implementarile TCP. Ea poate fi sitetizata astfel: implementarile initiale ale TCP-ului prezinta o problema cunoscuta sub numele de sindromul “silly window” in care ficare confirmare anunta un mic spatiu disponibil si fiecare segment trasporta o mica cantitate de date.

Evitare aparitiei sindromului “silly window”

(Gradinaru Gabriel)

Specificatiile actuale ale TCP-ului includ metode heuristice de prevenire a sindromului “silly window”. Una se foloseste pe masina emitatoare pentru a evita transmiterea unor cantitati mici de date in fiecare segment in timp ce alta se foloseste pe masina receptoare pentru a evita transmiterea de valori mici prin fereastra de avertisment ceea ce

ar detrmina transmiterea de pachete mici de date. Metodele lucreza bine impreuna, asa ca este bine ca atat receptorul cat si emitatorul sa aiba o implementare care ca ajute la evitarea aparitiei sindromului deoarece oricand este posibil ca una dintre masini sa greseasca implementarea corecta a metodei avand ca rezultat compromiterea performantelor.

In practica, software-ul TCP de pe ambele masini (receptoare si emitatoare) trebuie sa contina un cod de evitare a sindromului. Ca sa intelegem de ce, reamintim ca o conexiune TCP este full-duplex. Astfel, o implementare a TCP-ului contine un cod pentru trimiterea datelor precum si un cod pentru receptia lor.

Evitarea sindromului de catre partea receptoare

In general, un receptor intretine o inregistrare interne a dimensiunii ferestrei disponibile, dar intarzierile indica pentru emitator necesitate unei crestere in dimensiunea ferestrei pana cand fereastra poate injecta o cantitate de date mai mare. Definitia “cantitatii importante” depinde de dimensiunea maxima a segmentului. TCP-ul o defineste ca fiind minim o jumatate din dimensiunea bufferului receptorului sau numarul octetilor de date dintr-un segment de dimensiune maxima.

Masina receptoare previne aparitia sindromului impiedicand anuntul unei ferestre de dimeniuni mici in cazul in care aplicatia receptoare extrege incet octetii de datde. De exemplu, cand un buffer receptor este plin, el trimite o confirmare ce indica o fereastra de dimensiune zero. Aplicatia receptoare extrage octetii din buffer iar TCP-ul receptor calculeaza spatiul de curand disponibilizat in buffer. In loc sa trimita imediat o fereastra de avertisment, receptorul asteapta pana cend spatiul disponibil ajunge la jumatate din capacitatea intregului buffer sau la dimensiunea maxima a unui segment. Astfel, emitatorul receptioneaza intotdeauna o valoare mare pentru fereastra curenta, permitandu-i sa transfere segmente mari. Metoda poate fi sintetizata astfel: evitarea sindromului “silly window” de catre partea receptoare: inainte sa trimiti o fereastra de avertizare modificata dupa anuntarea unei ferestre de dimensiune zero, asteapta ca spatiul disponibil sa devina cel putin egal cu 50% din dimesiunea totala a bufferului sau cu dimensiunea maxima a segmentului.

Intarzierea confirmarilor

Doua metode de abordare pot fi prezentate pentru implementarea metodei de evitare a sindromului de catre partea receptoare. In prima, TCP-ul confirma fiecare segment care vine, dar nu anunta o crestere a

ferestrei sale pana cand fereastra atinge limita specificata prin metoda heuristica. In a doua abordare, TCP-ul intarzie trimiterea unei confirmari cand metoda indica ca fereastra nu este suficient de mare pentru a fi anuntata. Standardele recomanda intarzierea confirmarilor.

Pentru evitarea potentialelor probleme, standardele TCP introduc o limita pentru timpul cat poate fi intarziata o confirmare. Astfel, implementarile nu pot intarzia o confirmare mai mult de 500 ms. Mai mult ca sa garanteze acelui TCP receptionarea unui numar suficient de estimari de timp complet de calatorie, standardul recomanda acelui receptor sa confirme obligatoriu fiecare segment diferit de date.

Evitarea sindromului de catre partea emitatoare

Metoda folosita de TCP-ul emitator pentru evitarea aparitiei sindromului este deopotriva surprinzatoare si eleganta. Reamintim ca scopul este evitarea trimiterii pachetelor mici. Totodata reamintim ca aplicatia emitatoare poate genera date in mici blocuri (de exemplu un octet la un moment de timp). Astfel, pentru a atinge scopul, un TCP emitator trebuie sa permita aplicatiei sa faca apeluri multiple de tip *write* si sa colecteze data tranferata prin fiecare apel inainte de a le trimite intr-un singur segment de dimensiune mare. Deci, TCP-ul emitator trebuie sa intarzie transmiterea unui segment pana cand el strange o cantitate rezonabila de date, tehnica cunoscuta sub numele de *clumping*.

Intrebarea care apare este: “Cat de mult trebuie sa astepte TCP-ul inainte sa trimita datele?” Daca TCP-ul asteapta prea mult aplicatia va functiona cu mari intarzieri, iar daca nu asteapta un timp suficient de lung, segmentele vor fi mici si productivitatea scazuta. Mai mult, TCP-ul nu stie daca sa astepte deoarece nu stie daca aplicatia ca genera mai multe date in viitorul apropiat.

Ca si algoritmul TCP folosit pentru retransmisie si algoritmul de pornire usoara folosit pentru evitarea congestiei, tehnica folosita de TCP pentru a evita transmiterea pachetelor mici este adaptiva - intarzierea depinde de performantele curente ale retelei. Mecanismul de evitare a sindromului folosit de catre partea emitatoare este numit *self clocking* deoarece el nu face calculul intarzierilor. In schimb, TCP-ul foloseste sosirea unei confirmari pentru a cere transmiterea unor pachete suplimentare. Metoda poate fi sintetizata astfel: evitarea sindromului de catre partea emitatoare: cand o aplicatie emitatoare genereaza date suplimentare pentru a fi trimise printr-o conexiune prin care datele anterioare au fost transmise dar n-au fost confirmate, pune

datele intr-un buffer de iesire ca de obicei, dar nu trimite segmentele suplimentare pana cand nu sunt sufieciente date pentru a completa un segment de dimensiune maxima. Aplica regula chiar daca utilizatorul cere o operatie de tip push.

Daca aplicatia genereaza un octet de date la un moment, TCP-ul va trimite primul octet imediat. In orice caz, pana cand va sosi ACK, TCP-ul va acumula octeti suplimentari in bufferul sau. Astfel, daca aplicatia este mai rapida ca reteaua (cum ar fi de pilda transferul unui fisier), segmente succesive vor contine fiecare multi octeti. Daca aplicatia este mai lenta in comparatie cu reteaua (cum ar fi de pilda un utilizator ce trimite caractere de la tastatura), mici segmente vor fi trimise fara mari intarzieri.

Cunoscut sub numele de algoritmul lui Nagle dupa numele inventatorului sau, tehnica este foarte eleganta deoarece necesita un efort de calcul mic. O gazda nu doreste sa pastreze timere pentru fiecare conexiune si nici sa examineze un ceas cand o aplicatie genereaza date. Mult mai important este faptul ca, desi tehnica se adapteaza la combinatii arbitrare ale intarzierilor retelei, dimensiunilor maxime de segmente si vitezei aplicatiilor el nu face o scadere a productivitatii in cazurile conventionale.

Ca sa intelegem de ce productivitatea ramane mare pentru comunicatiile conventionale, observam ca aplicatiile optimizate pentru productivitate mare nu genereaza un octet de date la un moment de timp. In schimb, asemenea aplicatii scriu blocuri mari de date la fiecare apel. Astfel, bufferul de iesire TCP va fi umplut de la inceput cu date suficiente pentru cel putin un segment de dimensiune maxima. Mai mult, deoarece aplicatia produce date mai rapid decat TCP-ul poate tranfera, bufferul emitatorului ramane aproape plin in permanenta. Ca urmare, TCP-ul continua sa trimita pachete la orice rata pe care reteaua o permite in timp ce aplicatia continua sa incarce bufferul. Ca sa concluzionam: TCP-ul cere acum receptorului si emitatorului sa implenteze metode heuristice pentru a evita sindromul “silly window”. Un receptor evita anuntarea unei ferestre mici si un emitator foloseste o schema adaptiva pentru a intarzia transmisia astfel incat el poate strange date pentru a forma segemente de dimensiune mare.

Protocolul Go Back N (Bestea Vladut)

Nevoia pentru o fereastra mare la emitator apare atunci cand pentru produsul largime de banda x timpul de propagare dus-intors este mare. Daca largimea de banda este mare, chiar si pentru intarzieri moderate, emitatorul isi va termina repede fereastra. Daca intarzierea este mare (de exemplu, canal de satelit), emitatorul isi va termina fereastra chiar si pentru largimi de banda moderate. Produsul acestor doi factori spune de fapt care este capacitatea canalului, iar pentru a opera la eficienta maxima, emitatorul trebuie sa fie capabil sa o umple fara sa se opreasca.

Aceasta tehnica este cunoscuta ca **banda de asamblare** (**pipelining**). Considerand capacitatea canalului de *b* biti pe secunda, dimensiunea cadrului de *l* biti si timpul de propagare dus-intors *R* secunde, timpul necesar pentru a transmite un singur cadru este *l/b* secunde. Dupa ce a fost transmis ultimul bit al unui cadru de date, apare o intarziere de *R/2* inainte ca bitii sa ajunga la receptor si o alta intarziere de cel putin *R/2* pentru sosirea confirmarii, rezultand o intarziere totala de *R*. In cazul protocoalelor pas-cu-pas, linia este ocupata pentru un timp egal cu *l/b* si in asteptare pentru un timp egal cu *R*, rezultand:

$$\textit{utilizarea liniei} = l/(l+bR).$$

Daca $l < bR$, eficienta va fi mai mica de 50%. Deoarece pana la intoarcerea confirmarii exista intotdeauna o intarziere nenula, in principiu poate fi folosita banda de asamblare, pentru a tine linia ocupata tot acest interval, dar daca intervalul este mic, complexitatea suplimentara face efortul inutil.

Utilizarea benzii de asamblare in cazul unui canal de comunicatie nesigur ridica probleme serioase. Mai intai sa vedem ce se intampla daca un cadru din mijlocul unui sir lung este modificat sau pierdut. Multe cadre succesive vor ajunge la receptor inainte ca emitatorul sa observe ca ceva este gresit. Atunci cand un cadru modificat ajunge la receptor este evident ca el trebuie eliminat, dar ce trebuie sa faca receptorul cu toate cadrele corecte care urmeaza? Sa reamintim ca nivelul legatura de date receptor este obligat sa livreze pachete catre nivelul retea in secventa.

Exista doua moduri de baza de tratare a erorilor in prezenta benzii de asamblare. Un mod, numit **revenire cu n pasi** (**go back n**), este ca receptorul sa elimine pur si simplu cadrele care urmeaza, netrimitand confirmari pentru cadrele eliminate. Aceasta strategie corespunde unei ferestre de receptie de dimensiune 1. Cu alte cuvinte, nivelul legatura de

date refuza sa accepte orice cadru exceptandu-l pe urmatorul care trebuie livrat catre nivelul retea. Daca fereastra emitatorului se umple inaintea expirarii contorului de timp, banda de asamblare va incepe sa se goleasca. In cele din urma, timpul emitatorului va expira si se vor retransmite toate cadrele neconfirmate, in ordine, incepand cu cadrul pierdut sau modificat. Daca rata erorilor este mare, aceasta abordare poate risipi o mare parte din largimea de banda.

Cealalta strategie generala de tratare a erorilor atunci cand este folosita banda de asamblare se numeste **repetare selectiva** (**selective repeat**). Cand aceasta este utilizata, un cadru incorect este respins, dar toate cadrele corecte care il urmeaza sunt memorate. Cand contorul de timp al emitatorului expira, cel mai vechi cadru neconfirmat este retransmis. Daca acest cadru ajunge corect, receptorul poate transmite catre nivelul retea, in ordine, cadrele pe care le-a memorat. Repetarea selectiva este deseori combinata cu utilizarea confirmarilor negative (NAK), care sunt trimise atunci cand se detecteaza o eroare, de exemplu cand se primeste un cadru cu suma de control incorecta sau cu numar de secventa necorespunzator. Confirmarile negative simuleaza retransmisia inainte de expirarea contorului de timp corespunzator, imbunatatind astfel performanta.

Strategia de repetare selectiva corespunde unei ferestre a receptorului mai mare ca 1. Orice cadru din interiorul ferestrei poate fi acceptat si memorat pana cand toate cele precedente vor fi trimise nivelului retea. Daca fereastra este mare, aceasta abordare poate necesita un spatiu mare de memorie pentru nivelul legatura de date.

Aceste doua alternative reprezinta compromisuri intre largimea de banda si spatiul ocupat de tampoane la nivelul legatura de date. In functie de care resursa este mai deficitara, poate fi utilizata una sau cealalta.

In pseudocod, protocolul de revenire cu n pasi arata astfel:

N = window size
Rn = request number
Sn = sequence number
Sb = sequence base
Sm = sequence max

Receiver:

Rn = 0

Do the following forever:

If the packet received = Rn && the packet is error free
Accept the packet and send it to a higher layer
Rn = Rn + 1
Send a Request for Rn
Else
Refuse packet
Send a Request for Rn

Sender:

Sb = 0
Sm = N − 1

Repeat the following steps forever:

1. If you receive a request number where Rn > Sb
Sm = Sm + (Rn − Sb)
Sb = Rn

2. If no packet is in transmission,
Transmit a packet where Sb <= Sn <= Sm.
Packets are transmitted in order.

Atunci cand alegem o valoare N pentru dimensiunea ferestrei trebuie sa tinem cont de unele lucruri:

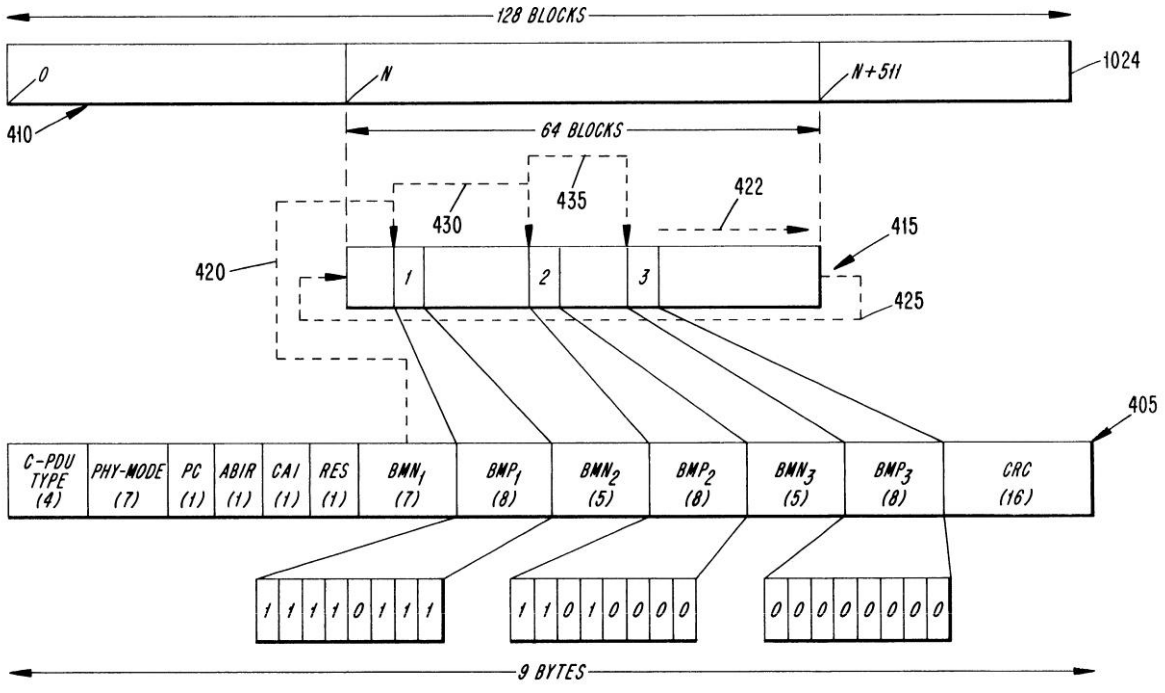
- Emitatorul nu trebuie sa transmita prea rapid. N trebuie sa fie limitat de posibilitatea receptorului de a procesa pachetele.
- N trebuie sa fie mai mic decat numarul de elemente din secventa (care sunt numerotate de la 0 la N) pentru a verifica transmisiunea in cazul in care unele pachete nu sunt acceptate.
- N trebuie sa fie cat mai mare posibil fara a contrazice primele doua puncte.

Protocolul Selective repeat (Pirvu Sorin)

Protocolul Go Back N functioneaza bine daca erorile sunt rare, dar daca linia este slaba, se pierde mult din largimea de banda cu retransmiterea cadrelor. O alta strategie de tratare a erorilor este ca

receptorul sa accepte si sa numeroteze cadrele care urmeaza dupa un cadru deteriorat sau pierdut. Un astfel de protocol nu elimina cadre doar pentru ca un cadru anterior a fost deteriorat sau pierdut. In acest protocol, atat emitatorul cat si receptorul mentin o fereastra de numere de secventa acceptabile.

Dimensiunea ferestrei emitatorului incepe de la 0 si creste pana la un maxim predefinit *MAX_SEQ*. Spre deosebire de aceasta, fereastra receptorului are dimensiunea fixa *MAX_SEQ*. Receptorul are un tampon rezervat pentru fiecare numar de secventa din cadrul ferestrei. Fiecare tampon are un bit asociat (*arrived - sosit*) care ne spune daca tamponul este plin sau gol. De fiecare data cand soseste un cadru, numarul sau de secventa este verificat de functia *between,* pentru a vedea daca face parte din fereastra. Daca da, si daca nu a fost deja receptionat, este acceptat si memorat. Aceasta actiune are loc fara sa se verifice daca contine sau nu urmatorul pachet asteptat de nivelul retea. Desigur cadrul trebuie pastrat la nivelul legatura de date si nu trebuie trimis catre nivelul retea decat atunci cand toate cadrele cu numere mai mici au fost deja livrate nivelului retea in ordinea corecta. Un protocol utilizand acest algoritm este prezentat in figura urmatoare:



Receptia nesecventiala introduce anumite probleme ce nu sunt prezente in protocoalele in care cadrele sunt receptionate numai in ordine.

In protocolul Go Back N, s-a presupus in mod implicit ca acel canal este puternic incarcat. Cand soseste un cadru, nu se trimite imediat o confirmare. Confirmarea este atasata la urmatorul cadru de date de iesire. Daca traficul invers este slab, confirmarea va fi retinuta o perioada mare de timp. Daca traficul este intens intr-o directie si inexistent in cealalta directie, atunci sunt trimise numai *MAX_SEQ* cadre si apoi protocolul se blocheaza, de aceea am presupus ca exista intotdeauna trafic in directia inversa.

Aceasta problema este rezolvata in protocolul selective repeat. Dupa sosirea unei secvente de cadre cu date, este pornit un contor de timp auxiliar, prin *start_ack_timer*. Daca pana la expirarea acestui contor nu a aparut trafic in sens invers, atunci este trimis un cadru de confirmare separat. O intrerupere datorata contorului auxiliar se numeste eveniment *ack_timeout*. Cu acest artificiu, fluxul de trafic unidirectional este acum posibil, deoarece absenta cadrelor de date in sens invers, pe care pot fi atasate confirmari, nu mai este un obstacol. Exista numai un contor auxiliar si daca *start_ack_timer* este apelata in timpul functionarii contorului, acesta este resetat la un interval complet de timp de confirmare.

Este esential ca timpul de expirare asociat contorului auxiliar sa fie mult mai scurt decat cel utilizat pentru cadrele de date de iesire. Aceasta conditie este impusa pentru a ne asigura ca o confirmare pentru un cadru corect receptionat soseste inainte ca timpul emitatorului sa expire si acesta sa retransmita cadrul.

Protocolul selective repeat utilizeaza pentru tratarea erorilor o strategie mai eficienta decat protocolul Go Back N. De fiecare data cand receptorul are motiv sa suspecteze ca a aparut o eroare, trimite inapoi la emitator un cadru cu o confirmare negativa (NAK). Un asemenea cadru reprezinta o cerere pentru retransmiterea cadrului specificat in NAK. Exista doua cazuri in care receptorul ar trebui sa fie suspicios: a sosit un cadru modificat sau a sosit un alt cadru decat cel asteptat (un posibil cadru pierdut). Pentru a preveni producerea cererilor multiple de retransmisie a aceluiasi cadru pierdut, receptorul va tine minte daca un NAK a fost deja trimis pentru un anumit cadru.

In unele situatii, timpul necesar pentru ca un cadru sa se propage la destinatie, sa fie prelucrat si sa se receptioneze confirmarea este (aproape) constant. In aceste conditii, emitatorul isi poate ajusta contorul de timp sa fie putin mai mare decat intervalul de timp normal asteptat intre emiterea unui cadru si receptionarea confirmarii sale. Totusi, daca acest timp variaza puternic, emitatorul trebuie sa aleaga intre fixarea intervalului la o valoare mica (riscand retransmisii inutile) si fixarea la o valoare mare (ramanand in asteptare timp indelungat dupa producerea unei erori).

Bibliografie

- Computer Networks - Andrew Tanenbaum
- http://computing.dcu.ie/
- http://publib.boulder.ibm.com
- Computer Networks: A Systems Approach - Larry L. Peterson & Bruce Peterson
- Computer Networking: A Top-Down Approach – James Kurose & Keith Ross