Retele de calculatoare -proiect

Aplicatie pentru transfer de fisiere – implementare printr-un mecanism de control al congestiei (TCP)

Studenti:

Petruca F.D. Marco-Alexandru 1306B

Chiuchiu D.D. Sorin-Alexandru 1307A

Profesor coordonator:

Prof. Nicolae-Alexandru Botezatu

Cuprins:

1. Descriere

* Transmiterea de pachete
* Congestia.
* Ce este si cum are loc
* Congestion control
* TCP pentru controlul congestiilor

1. TCP Tahoe

* Formatul pachetelor
* Despre algoritm
* Etapele acestuia

1. Solutia noastra
2. Bibliografie
3. **Descriere**

**Transmiterea de pachete**

Internetul este o reţea compusă din agregarea unui număr mare de reţele diferite într-o construcţie unică. Dacă două organizaţii au reţelele lor proprii, folosind tehnologii eventual diferite, aceste două reţele pot fi conectate între ele cu ajutorul unui ruter, care este cuplat la ambele reţele, rolul acestuia fiind de a prelua dintr-o reţea datele din sursa şi de a le retransmite. Un ruter este legat la fiecare reţea pe care o deserveşte printr-o interfaţă si va avea deci cel puţin două interfeţe. Cand un pachet apare prin una din interfeţe, ruterul pune acel pachet într-o memorie internă, il prelucreaza si decide in ce directie trebuie transmis. După aceasta prelucrare, trimite pachetul pe interfaţa care este cea mai potrivită pentru destinaţie.

**Congestia**

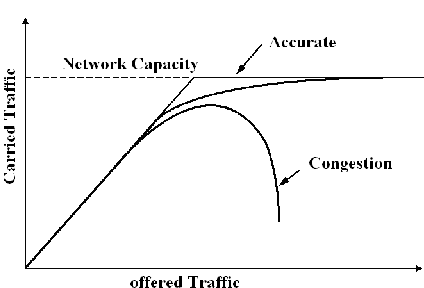
**Ce este si cum are loc**

Congestia reprezinta fenomenul de imposibilitate a transmiterii datelor pe un segment de retea. Aceasta apare atunci cand un segment al retelei primeste mai multe pachete decat capacitatea ei de transport, astfel dupa umplerea spatiului din ruter incepe sa piarda pachete sau timpul de raspuns este prea mare. Ca atare, in lipsa unei confirmari sursele retransmit pe acelasi traseu pachetele pierdute, ceea ce duce la o mai mare crestere a congestiei. O astfel de ingreunare a retelei poate duce la blocarea totala a acesteia.

**Controlul congestiei**

Teoria controlului congestiei a fost introdusă de Frank Kelly, care a aplicat teoria microeconomică și teoria optimizării convexe. Controlul congestiei fluidizeaza traficul de informatii intr-o retea pentru a evita colapsul acesteia rezultat din supraincarcare. Acest lucru se realizeaza de obicei prin reducerea ratei de pachete, impiedicand expeditorul sa copleseasca reteaua.

In prezent, cea mai larg folosita solutie pentru problema congestiilor este protocolul TCP care administreaza fluxuri intre 2 calculatoare.



**TCP**

**Ce este**

Soluţia folosită de majoritatea implementărilor este protocolul TCP ce a fost propus de Van Jacobson în 1988. TCP este un protocol folosit pentru a transmite informatii de la un calculator la altul, care garanteaza o transmitere totala a datelor. Modul in care este realizat acest lucru este prin faptul ca pentru fiecare segment de date trimis, receptorul va emite o confirmare a primirii segmentului. In cazul in care aceasta confirmare nu este receptionata, pachetele neconfirmate sunt pastrate de emitator si retransmise pana la receptionarea unui raspuns.

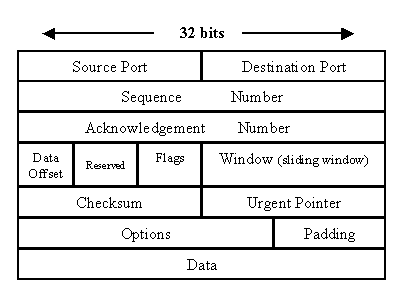
**Evitarea congestiei**

TCP mentine o „fereastra de congestie”(congestion window) numita CWND prin care limiteaza numarul total de pachete aflate in retea dar care nu au primit inca confirmare. In cazul aparitiei unei congestii, daca TCP nu primeste confirmarea pentru pachetele trimise, aceasta fereastra este readusa la un pachet (primul trimis care nu a primit confirmare), iar in urma reprimirii confirmarilor, fereastra se mareste din nou.

1. **TCP Tahoe**

Este primul algoritm creat pentru controlul congestiei , acesta fiind propus de Van Jacobson. Acesta merge pe principiul „conservarii pachetelor”, in sensul ca daca o conexiune ruleaza la maximul capacitatii benzii (bandwith) atunci un alt pachet nu este injectat in retea decat daca un alt pachet este scos. TCP Tahoe implementeaza acest principiu prin folosirea confirmarilor (ACKs), pentru ca o confirmare reprezinta faptul ca un pachet a fost scos ( off the wire ) de receptor. Totodata, acesta mentine si o fereastra a congestiei (CWND).

**Pachete TCP**



TCP funcționează în modul full-duplex (două fluxuri de octeți independenți care călătoresc în direcții opuse). Doar la începutul și la sfârșitul unei conexiuni datele vor fi transferate într-o direcție și nu în alta. TCP-ul de primire returnează un segment numit ACK pentru a confirma primirea cu succes a segmentului. TCP-ul expeditor trimite un alt segment ACK și apoi trece la trimiterea datelor.

**Alcatuirea pachetelor TCP:**

* Campurile **Source Port** si **Destination Port** (16 biti fiecare) au rolul de a identifica inceputul si finalul conexiunii.
* Campul **Sequence Number** (32 biti) specifică numărul atribuit primului octet de date din mesajul curent. În anumite circumstanțe, poate fi folosit și pentru a identifica un număr de secvență inițial care va fi utilizat în transmisia viitoare.
* Câmpul **Acknowledgement Number** (32 de biți) conține valoarea următorului număr de secvență pe care expeditorul segmentului se așteaptă să îl primească, dacă bitul de control **ACK** este setat. Rețineți că **Sequence Number** se referă la fluxul care curge în aceeași direcție cu segmentul, în timp ce **Acknowledgement Number** se referă la fluxul care curge în direcția opusă segmentului.
* Câmpul **data Offset (a.k.a. Header Length)** (lungime variabilă) indică câte cuvinte de 32 de biți sunt conținute în antetul TCP. Aceste informații sunt necesare deoarece câmpul **options** are lungime variabilă, astfel încât lungimea antetului este și ea variabilă.
* Câmpul **reserved** (6 biți) trebuie să fie zero. Acesta este pentru utilizare viitoare.
* Câmpul **Flags** (6 biți) conține diferite steaguri:

URG - Indică faptul că au fost plasate unele date urgente.

ACK - Indică faptul că numărul de confirmare este valid.

PSH - Indică faptul că datele trebuie transmise aplicației cât mai curând posibil.

RST - Resetează conexiunea.

SYN - Sincronizează numerele de secvență pentru a iniția o conexiune.

FIN - Înseamnă că expeditorul steagului a terminat de trimis date.

* Câmpul **Window** (16 biți) specifică dimensiunea ferestrei de primire a expeditorului (adică spațiul disponibil pentru datele primite).
* Câmpul **Checksum** (16 biți) indică dacă antetul a fost deteriorat în timpul transportului.
* Câmpul **Urgent pointer** (16 biți) indică primul octet de date urgente din pachet.
* Câmpul **Options** (lungime variabilă) specifică diverse opțiuni TCP.
* Câmpul **Data** (lungime variabilă) conține informații de nivel superior.

Pachetele TCP sunt foarte complexe și încorporează mai multe mecanisme pentru a asigura fiabilitatea și controlul fluxului pachetelor de date:

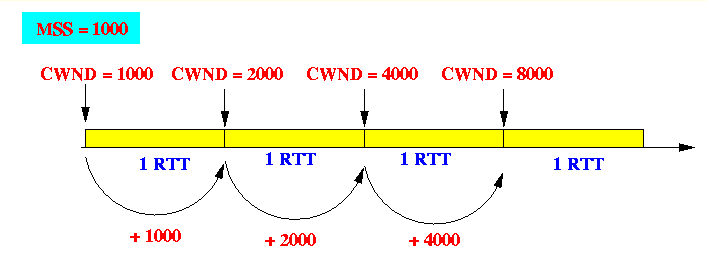
* **Streams**: datele TCP sunt organizate ca un flux de octeți, asemanator unui fișier.
* **Reliable delivery**: **Sequence Number** sunt folosite pentru a coordona datele care au fost transmise și primite. TCP va retransmite dacă stabilește că datele s-au pierdut.
* **Network adaptation**: TCP in cazul unei congestii va reduce fluxul intr-o retea si va relua transmiterea pentru a evita blocarea.
* **Round-trip time estimation**: TCP monitorizează schimbul de pachete de date, avand o estimare a cât timp ar trebui să dureze pentru a primi o confirmare și retransmite automat dacă acest timp este depășit.

**TCP Tahoe** foloseste 3 mecanisme pentru controlul congestiei : slow start , congestion avoidance si fast retransmit

1. **Slow start phase**

In aceasta faza, TCP Tahoe se foloseste de o crestere exponentiala a variabilei CWND (fereastra de congestie) pentru trimiterea de pachete. CWND creste pana ajunge la valoarea de threshold (SSThresHold) care reprezinta marimea ferestrei pe care TCP o considera **sigura**, iar dupa, incepe al 2-lea mecanism al acestui algoritm.

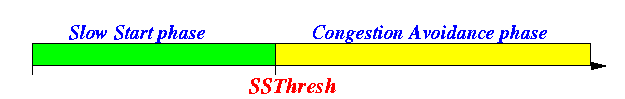
Detalii de implementare :

* SSThresHold = primeste o valoare considerata sigura (AWS);
* La inceputul unei noi conexiuni , CWND primeste dimensiunea 1 ;
* Dupa fiecare trimitere cu succes, in care emitatorul primeste confirmarea primirii pachetelor, numita si RTT( Round Trip Time ) , CWND creste in felul urmator : CWND = 2 \* CWND;
* Cu alte cuvinte , CWND se dubleaza pentru fiecare confirmare noua
* Reprezentare grafica a cresterii CWND dupa mai multe RTT : 
* CWND se opreste din crestere atunci cand depaseste valoarea sigura (SSThresHold).
* Procedura de Slow Start este folosita in urmatoarele 2 situatii :

1. Atunci cand o conexiune TCP este pornita pentru prima oara. Caz in care , SSThresHold = AWS;
2. Atunci cand TCP detecteaza un pachet pierdut. In acest caz , SSThresHold = CWND/2

* Faza de Slow Start poate fi terminata in 2 feluri :

1. Atunci cand CWND ajunge mai mare decat SSThresHold, ceea ce reprezinta o terminare „ normala „ .



2) Cand se termina abrubt , adica se detecteaza un pachet pierdut in timpul acestei proceduri. Caz in care SSThresHold = CWND / 2 , apoi TCP reintra in faza de slow start.

**2) Congestion Avoidance Phase**

TCP intra in faza de evitare a congestiei doar daca faza de slow start a fost terminata normal ( CWND > SSThresHold ) si nu a aparut o congestie.

Scopul acestui mecanism este de a incrementa CWND cu dimensiunea unui singur pachet atunci cand nu se detecheaza vreo pierdere de pachete.

Tot in aceasta faza se foloseste conceptul de AIMD pentru modificarea variabilei CWND

Additive Increase / Multiplicative Decrease (AIMD) este un algoritm care este cel mai mult folosit in controlul congestiei oferit de TCP. AIMD combina cresterea liniara a ferestrei de congestie atunci cand nu se detecteaza congestie cu reducerea exponentiala atunci cand congestia apare.

Algoritmul consta in cresterea ratiei de transmisie ( cresterea CWND ) pana cand un pachet este declarat pierdut. Notiunea de crestere aditiva se refera la adaugarea unei valori fixe la CWND pentru fiecare RTT , iar atunci cand se detecteaza congestie , CWND este scazut cu un factor multiplicativ.

Putem ajunge la o formula de calculare a CWND

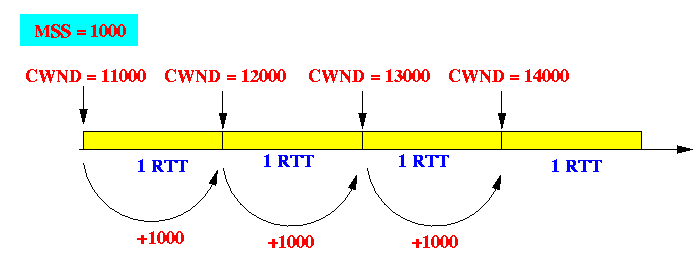


Unde :

W(t) – CWND ; a > 0 ; 0 < b < 1 ;

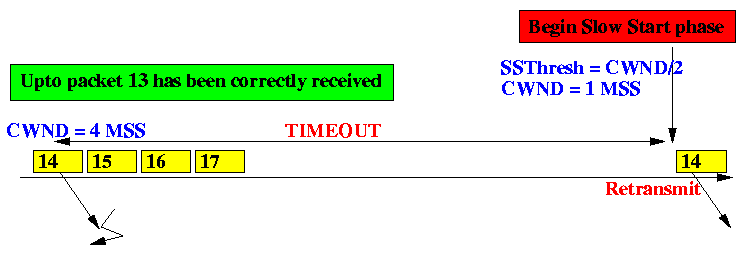
In cazul TCP Tahoe, in cazul aparitiei unei congestii, CWND va fi resetat la valoarea 1.

Reprezentare grafica a cresterii CWND in timpul fazei de evitare a congestiei :



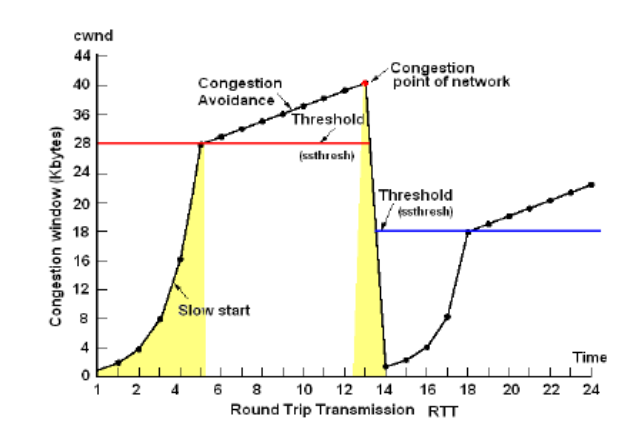
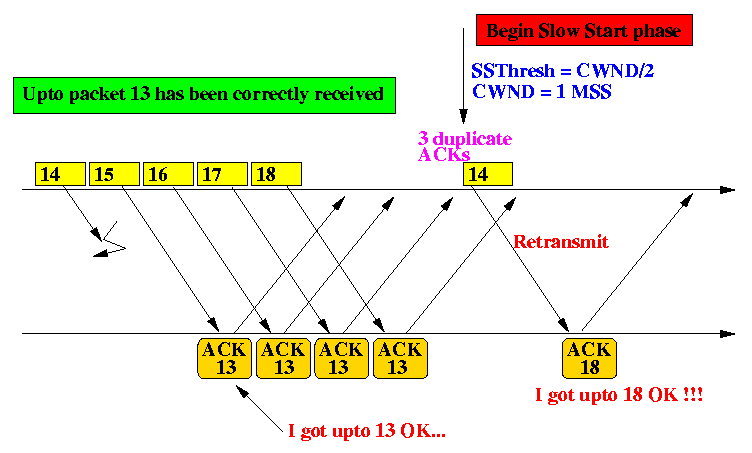
Faza de evitare a congestiei incepe doar atunci cand faza de slow start se termina fara a se detecta pierderea unui pachet , si se opreste atunci cand detecteaza ca s-a pierdut un pachet in transmisie, sau apare fenomenul de timeout.

Conceptul de timeout , in acest caz , se refera la asteptarea indelungata a unui feedback de la receptor ca s-a trimis cu succes pachetul din retea. Daca timpul asteptat de emitator depaseste un anumit timp prestabilit de asteptare , apare fenomenul de timeout.



1. **Fast Retransmit**

Acest mecanism se refera la detectia unei pierderi de pachete prin primirea de confirmari duplicate. Atunci cand se primesc 3 confirmari care sunt identice , TCP retransmite pachetul pierdut , fara sa mai astepte timeout. Apoi , incepe din nou partea de slow start .



1. **Solutia** **noastra**

Pentru rezolvarea implementarii protocolului TCP, s-au folosit 2 scripturi pentru fereastra de transmitere si cea de primire (transmitator.py si receptor.py), cat si implementarea unei clase pentru pachete(package.py).

**Detalii implementare:**

In cadrul scriptului de transmitere, se va astepta alege fisierul dorit pentru transfer, iar acesta va fi impachetat si codificat in binar prin intermediul functiilor implementate in clasa „package”, unde pentru fiecare secventa se va forma header-ul. Header-ul unui pachet este compus din 4 campuri separate prin „||”, care vor contine codificari pentru ca un receptor sa il poata desface corespunzator. Aceste campuri sunt confirmarea de primire, portul sursa, numarul de secventa si datele din pachet.

Se asteapta conectarea receptorului (simularea unui server care asteapta conectarea unui client) si dupa primirea adresei IP si a portului clientului, se va realiza transmiterea de fisier, incepand in faza de slow start (numarul de pachete trimise „vol” incepe cu 1 si creste de doua ori pe transmisie), adaugand intr-o lista pachetele trimise cu succes. In cazul in care nu exista nicio eroare si se depaseste valoarea de threshold prestabilita, va continua in faza de congestion avoidance(numarul de pachete trimise va creste cu 1). In cazul pierderii unui pachet, confirmat prin primirea a 3 pachete cu acelasi numar de secventa, valoarea de threshold va lua valoarea vol / 2 (dar nu mai mica de 16), fereastra de transmisie („vol”) se reseteaza la 1, iar pachetele trimise fara confirmari se vor introduce intr-o coada de retransmitere. Trimiterea va reincepe cu pachetele din coada de retransmitere, reincepand de la etapa de slow start. Procesul se repeta pana la trimiterea ultimului pachet, care reprezinta confirmarea transmiterii fisierului.

In cadrul scriptului de receptie, se vor primi pachetele, se vor prelucra si se vor trimite confirmarile acestora catre transmitator. Pe baza unei probabilitati se va simula o eroare de receptie(trimiterea unei confirmari gresite), in cazul a 3 astfel de erori la acelasi pachet, procesul de primire se opreste pentru pachetele curente receptionate si neprelucrate. In cazul unei primiri corecte, continutul, in functie de numarul de secventa al pachetului, se va adauga intr-o lista, pe baza careia la finalul receptiei se va alcatui fisierul primit.

1. **Bibliografie**

[Intro to Congestion Control (squidarth.com)](https://squidarth.com/rc/programming/networking/2018/07/18/intro-congestion.html)

[Scalabilitatea în reţele de comunicaţii de date (cmu.edu)](https://www.cs.cmu.edu/~mihaib/articole/csfq/csfq-html.html)

[TCP congestion control - Wikipedia](https://en.wikipedia.org/wiki/TCP_congestion_control)

[Network congestion - Wikipedia](https://en.wikipedia.org/wiki/Network_congestion#Congestion_control)

[https://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf](https://inst.eecs.berkeley.edu/~ee122/fa05/projects/Project2/SACKRENEVEGAS.pdf?fbclid=IwAR3GnCUn6YDP3lGCCI8mfD2lwB-4RtvfYuzG5DM9VghFhbFKiefozTAZkak)

[http://www.mathcs.emory.edu/~cheung/Courses/455/Syllabus/A1-congestion/tcp2.html?fbclid=IwAR3IvoaPPqT3pdVRzeaRs\_dDCfcBoxzlt1NhNPNd9Ss0wtvOisuvdo3x934](http://www.mathcs.emory.edu/~cheung/Courses/455/Syllabus/A1-congestion/tcp2.html?fbclid=IwAR33BHPwVGtS1UA7Hqr9FVmYOpVJGqfWgyo1AQcdmKtfSAOQNfFnktU4IIU)

[https://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/](https://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/?fbclid=IwAR15ZiieoX51UXp3Sw00T3I6wTGLaplbOdDStIqLS8_ycNNkUN37xzV_tv4)

[https://en.wikipedia.org/wiki/Additive\_increase/multiplicative\_decrease](https://en.wikipedia.org/wiki/Additive_increase/multiplicative_decrease?fbclid=IwAR3LncSS-DH_5RmOiUpUj0icl5OB9kvonN4SGtIFkkrPOgOnllCEtTLjBGQ)

[nicolaebotezatu/RC-P (github.com)](https://github.com/nicolaebotezatu/RC-P)