FACULTATEA DE ELECTRONICA, TELECOMUNICATII SI TEHNOLOGIA INFORMATIEI

Protocoalele de transport UDP si TCP. Algoritmi de control al congestiei

Coordonator,

Conf. Dr. Ing. Ștefan Stăncescu

Lecu Radu Șerban

Tică Andra Maria

Vidrașcu Mihai

442A

-2011-

Cuprins

10.3. Fast retransmit
10.4. Fast recovery
11. TCP tranzactional (T/TCP)
12. Bibliografie

C. Algoritmi de control al congestiei — Tică Andra Maria

1. Definirea problemei
2. Solutii posibile. Clasificarea Yang&Reddy a algoritmilor
3. Algoritmi de control ai congestiei
   3.1. Algoritmul RED
      3.1.1. Calcularea lungimii medii a cozii
      3.1.2. Calcularea probabilitatii de marcare a pachetelor
      3.1.3. Implementarea algoritmului RED
   3.2. Algoritmul GAIMD
      3.2.1. Modelarea ratei de transmisiune
      3.2.2. TCP-friendly GAIMD
   3.3. Algoritmii HTSCP si STCP
      3.3.1. HTSCP
      3.3.2. STCP
   3.4. Algoritmul binomial BI-TCP
      3.4.1. Binary search increase
      3.4.2. Additive increase
      3.4.3. Slow start
      3.4.4. Implementare
   3.5. Algoritmul SIMD
4. Bibliografie

Protocoalele de transport UDP si TCP. Algoritmi de control al congestiei

A. Protocolul UDP

Vidrașcu Mihai

1. Introducere

Internetul este probabil cea mai mare şi mai complexă realizare a omului, este un sistem global de reţele de calculatoare interconectate, care pune la dispoziţia miliardelor de utilizatori experienta şi cunoştiinţele acumulate de alungul timpului de om. Este o reţea de reţele care interconectează milioane de calculatoare prin tehnologii de comunicare prin fir, radio sau optice.

Însă pentru a funcţiona în armonie, toate calculatoarele trebuie "să vorbească aceeaşi limba". De aceea au fost create şi standardizate o mulţime de protocoale. Protocolul de comunicare este un sistem de formate şi reguli pentru schimbul de mesaje între sisteme de telecomunicaţii. Protocolul pentru internet este un protocol care asigură transmiterea datelor de la un sistem de calcul la altul, prin internet.

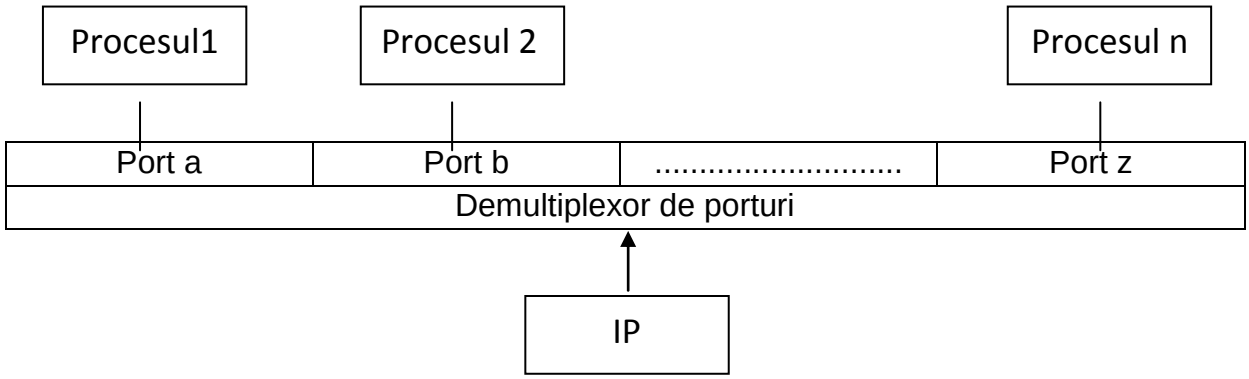
Protocoalele au fost standardizate şi pot fi clasificate şi după modelul stratificat OSI (Open Systems Interconnection). Există 7 nivele: fizic, legatura de date, reţea, transport, sesiune, prezentare şi aplicaţie. Nivelul de transport oferă transparenta transferului de date între utilizatori. Asigura fiabilitatea unei legături prin controlul vitezei, segmentare / desegmentare şi controlul erorilor. [14]

În nivelul de transport există 2 protocoale importante, unul orientat pe conexiune şi altul fără conexiune (sunt complementare unul altuia). Cel fără conexiune este UDP se ocupă în principal numai de transmisia pachetelor între aplicatii. Cel orientat pe conexiune este TCP şi este mult mai complex: realizeaza conexiunea, adauga fiabilitate prin retransmisie, control pentru debit şi congestie. [11]

1.1 Ce este UDP ?

UDP provine de la User Datagram Protocol şi a fost proiectat în 1980 de David P. Reed. UDP oferă numai un serviciu minimal de transport (livrare ne-garantată de datagrame) şi permite aplicaţiilor acces direct la serviciul de datagrame al stratului de IP. UDP este folosit de aplicaţii care nu necesita servicii de nivelul TCP, sau care vor să folosească servicii de comunicare care nu sunt disponibile în TCP (ca multicast).

Singurele servicii pe care le oferă sunt verificarea datelor prin checksum şi multiplexarea pe porturi. Deci o aplicaţie care foloseste UDP trebuie să trateze direct problemele legate de comunicaţia E2E (end-to-end) pe care un protocol orientat pe conexiune le-ar fi soluţionat, ca retransmisia pentru asigurarea fiabilităţii, segmentarea pe pachete şi reasamblarea, controlul debitului, evitărea congestiei etc). [14]



Exemplificarea funcţiei de multiplexare pe porturi [16]

1.2 Comparatie UDP cu TCP

Se ştie ca TCP (Transmission Control Protocol) este orientat pe conexiune, adică se foloseste metoda de tip handshake pentru iniţializarea comunicarii între sisteme. Principalele caracteristici ale TCP sunt următoarele:

Siguranţă: TCP gestionează confirmările de primire, retransmisiile şi momentele de time-out. Se fac mai multe incercări de livrare a mesajului. Dacă se pierde ceva pe drum, serverul va cere din nou partea pierdută. Nu există cale de mijloc: fie nu există pachete lipsa, fie se întrerupe conexiunea.

Ordine: dacă se trimit două mesaje succesiv, ele vor fi receptionate în ordinea în care au fost trimse. În cazul în care pachetele ajung în alta ordine, TCP stocheaza datele dezordonate pana ajung toate pachetele şi apoi le reordonează şi le livrează aplicaţiei.

Streaming: datele sunt citite ca stream de octeti; nu există indicatori care să arate limitele unui segment.

TCP trimite 3 pachete pentru a initializa o conexiune la un socket. Abia apoi poate incepe să trimită date. TCP se ocupă şi de fiabilitate şi controlul congestiei.

UDP este un protocol mai simplu, fără conexiune. Protocoalele fără conexiune nu iniţializează o conexiune dedicată între capete. Comunicarea se face prin transmiterea informaţiei intr-o singura directie, fără a verifica starea sau disponibilitatea receptorului. Totuşi, un puternic avantaj al UDP-ului este viteza, şi este exploatată la maximum în cazul aplicaţiilor VoIP (Voice over IP). O comunicare de tip handshake ar ingreuna transmiterea clară a vocii.

Caracteristicile UDP-ului sunt următoarele:

Nesigur: când un mesaj este trimis, nu se ştie dacă va ajunge la destinaţie, se poate pierde pe drum. Nu se aplica conceptele de confirmare, retransmitere sau timeout.

Fără ordine: dacă două mesaje sunt trimise succesiv către acelaşi receptor, nu se poate prezice ordinea în care vor ajunge.

Operarea bazată pe datagrame: pachetele sunt trimise individual şi sunt verificate pentru integritate doar dacă sosesc la destinaţie. Pachetele au frontiere bine definite.

Nu există controlul congestiei: UDP nu evită congestiile de unul singur, şi este posibil ca aplicatiile de viteza mare să duca la blocaj dacă nu se implementează metode de control al congestiei la nivelul aplicaţiei. [14]

1.3 Structura pachetelor

UDP transmite segmente formate dintr-un header de 8 Bytes, urmat de datele efective de transmis.

32 Biti

Port Sursa	Port destinaţie
Lungime UDP	UDP checksum

Cele două porturi identifică cele două sisteme de calcul care comunica, sursa şi destinaţia. Cand un pachet UDP soseste, incarcatura lui este preluată de procesul ataşat portului destinaţie. Pe scurt, porturile asigură livrarea segmentelor aplicaţiei corecte.

Portul sursa este necesar în cazul în care receptorul trebuie să raspundă emiţătorului. Copiind portul sursa din pachetul abia sosit în câmpul de destinaţie al pachetului de răapuns, procesul care trimite răapunsul poate specifica ce proces de pe emiţător îl primeste.

Câmpul „Lungime UDP” include headerul de 8 Bytes şi datele. Lungimea minima este de 8 Bytes (pentru header). Lungimea maxima este de 65.515 bytes (dimensiune mai mica decat maximul reprezentabil pe 16 biti; limita este data de dimensiunea maxima a pachetelor IP).

Suma de verificare este optionala şi se foloseste pentru cresterea fiabilităţii. Face verificarea header-ului, a datelor şi un pseudoheader IP conceptual. Cand se face calculul, câmpul checksum este setat la 0 şi câmpul de date este bordat cu un byte aditional de 0 dacă lungimea lui este un număr impar. Algoritmul de verificare consta în sumarea tuturor cuvintelor de 16 biti în complement faţă de 1 şi scoaterea complementului lui 1 din rezultat. Deci, când receptorul face calculul pe întregul segment, inclusiv pe câmpul de checksum, rezultatul ar trebui să dea 0. Dacă checksum-ul nu se calculează, atunci se va stoca cu valoarea 0.

În cazul IPV4, pseudoheader-ul arata ca în figura urmatoare:

Adresa sursei (IPV4)		
Adresa destinaţiei (IPV4)		
0 0 0 0 0 0 0 0	Protocol = 17	Lungime UDP

Contine adresele IPV4 pe 32 de biti ale emiţătorului şi receptorului, numărul de protocil pentru UDP(17), şi lungimea segmentului (inclusiv headerul). La IPV6 este similar, insa câmpul de checksum nu mai este optional. Pseudoheader-ul va arata astfel:

Adresa sursei (IPV6)	
Adresa destinaţiei (IPV6)	
Lungimea pachetului din stratul superior	
0000...000 (24 de biti 0)	Urmatorul header

Includerea pseudoheader-ului în calculul checksum-ului ajuta la detectarea pachetelor livrate gresit, da includerea lui violeaza ierarhia protocolului, pentru ca adresele IP din el apartin stratului IP, nu celui UDP. TCP foloseste acelaşi pseudoheader pentru checksum-ul lui.

Trebuie tinut cont ca UDP nu are nici o metoda de control a congestiei şi a debitului, şi nici nu retransmite în cazul primirii unui segment eronat. Totul ramane pe seama proceselor user-ului. Însă, oferă o interfaţă către protocolul IP şi poate şi demultiplexa mai multe procese folosind porturi si, optional, detectie de erori la receptie. Este util în situatii de comunicare client-server. Deseori, clientul trimite o cerere scurta către server şi asteapta un răapuns rapid. Dacă fie răapunsul fie cererea s-au pierdut, clientul asteapta o perioada de time-out, şi apoi incearca din nou. Pe langa faptul ca programarea e mai usoara, sunt necesare şi putine mesaje (unul în fiecare directie) decat în cazul unui protocol care cere o setare initiala, ca TCP.

O aplicaţie care foloseste UDP în acest fel este DNS (Domain Name Server). Pe scurt, un program care trebuie să caute adresa IP a unui host oarecare poate trimite un pachet UDP care contine numele host-ului către un server DNSS. Serverul raspunde cu un pachet UDP care contine adresa IP a hostului. Nu este nevoie de nici o setare initiala şi eliberare ulterioara a conexiunii. [11]

UDP mai este folosit în aplicaţii care pun viteza inaintea calitatii, în aplicaţii de tip FINGER (FINGER este un protocol simplu pentru schimbul de status-uri şi informaţii despre utilizator), si, în general, acolo unde cererile sunt rapide şi necesita un singur pachet de răapuns (exemplu: SNMP – Simple Network Management Protocol, RIP – Routing Information Protocol, DHCP – Dynamic Host Configuration Protocol). [6]

Traficul video şi audio este în general transmis folosind UDP. Protocoalele de streaming audio-video au fost proiectate să suporte eventualele pierderi de pachete, deci va aparea doar o scadere slaba a calitatii, în locul unor intarzieri deranjante care ar aparea în cazul retransmisiei pachetelor pierdute. [7]

2. RPC – Remote Procedure Call

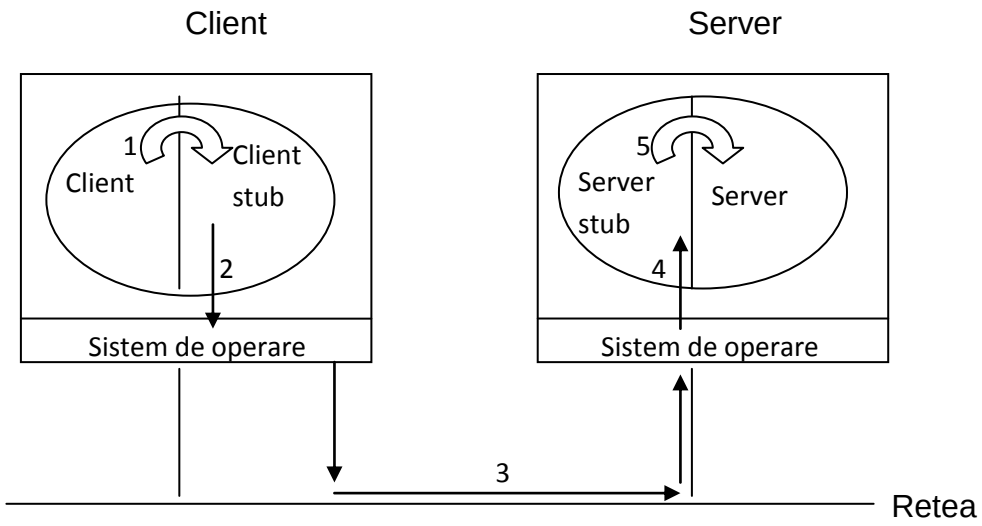
Intr-un fel, trimiterea unui mesaj către un host şi primirea unui răapuns seamana cu apelarea unei funcţii intr-un limbaj de programare. În ambele cazuri se porneste cu unul sau mai multi parametrii şi se returneaza un rezultat. Asemanarea aceasta se transpune în mediul de reţea sub forma apelurilor de procedura. Aceasta formulare usureaza mult programarea aplicatiilor. Exemplu: fie procedura iaAdresaIP(numeHost) , care funcţioneaza prin trimiterea unui pachet UDP către

serverul DNS şi asteapta răapuns, cronometrand şi incercand din nou, dacă acesta nu soseste în timp util. Astfel, toate detaliile reţelei sunt ascunse pentru programator.

Primii care au folosit acest concept au fost Nelson şi Birrell, în 1984. Ei au venit cu ideea de a permite programelor să initieze apeluri la proceduri aflate pe alte host-uri. Cand un proces de pe masina A apeleaza o procedura de pe masina B, procesul apeland se suspenda şi executia procedurii are loc pe masina B. Informatia este transportata de la apelant la apelat sub forma de parametrii, şi se intoarce ca rezultat. Pentru programator, acest schimb de informaţie nu este vizibil. Aceasta tehnica este cunoscuta sub numele de RPC – Remote Call Procedure, şi a devenit baza pentru multe din aplicatiile de lucru în reţea. Apelantul procedurii este denumit client; apelatul se numeste server.

Scopul RPC este acela de a face un apel de procedura să para ca unul local. IN cea mai simpla forma, pentru a apela o procedura aflata în alta parte, programul client trebuie să detina o biblioteca, numita *client stub,* care reprezinta procedura serverului în spatiul de adresa al clientului. Similar, serverul are şi el partea lui: *server stub*. Aceste proceduri ascund faptul ca apelul de la client către server nu este local.

În figura urmatoare sunt prezentate etapele de creare a RPC. Pasul1: clientul apeleaza stub-ul lui. Aici se apeleaza o procedura locala, cu parametrii introdusi în stiva în modul obisnuit. În etapa a doua, stub-ul clientului comprima parametrii intr-un mesaj şi face apel la sistemul de operare pentru a trimite pachetul. Aceasta impachetare se numeste „marshaling”.



În etapa a treia, sistemul de operare trimite mesajul de pe masina clientului către server. Pasul patru: sistemul de operare inmaneaza pachetul către stub-ul serverului, iar la etapa cinci, stub-ul serverului cheama procedura, după ce, în prealabil, a decomprimat parametrii. Raspunsul foloseste aceleasi etape, în sens invers.

De observat: procedura client, scrisa de programator, face un apel simplu de procedura, către stub-ul client, care are acelaşi nume ca procedura de pe server. Din moemnt ce procedura clientului şi stub-ul lui se afla în acelaşi spatiu de adrese, parametrii sunt dati în modul obisnuit. În acelaşi mod, procedura de pe server este apelata de o procedura din acelaşi spatiu de adresa, cu parametrii asteptati. Astfel, în locul operatiilor de intrare-iesire facute pe socket-uri, comunicarea prin reţea se face prin mimarea unui apel de procedura simplu.

În ciuda simplitatii, există probleme, una din cele mai mari fiind legata de folosirea pointer-ilor ca parametrii. În mod normal, un parametru pointer dat unei proceduri nu este o problema. Procedura apelata poate folosi pointerul în aceeaşi modalitate ca apelantul, pentru ca ambele există în acelaşi spatiu virtual de adresa. Însă, în cazul RPC, folosirea pointer-ilor ca parametrii este imposibila, pentru ca serverul şi clientul se afla în spatii total diferite.

Totuşi, există situatii în care ei pot fi utilizati. Fie primul parametru un pointer către o variabila întreaga ”a”. Stub-ul client poate impacheta pe *a* şi trimite către server. Stub-ul server-ului creaza un pointer către *a* şi îl da procedurii de pe server. Cand procedura reda controlul stub-ului, acesta trimite *a* inapoi la client, unde noul *a* este copiat peste cel vechi, în cazul în care a fost modificat de server. Pe scurt, secventa de apel prin referita a fost inlocuita cu secventa de apelare prin copiere şi inlocuire. Din pacate, aceasta metoda nu finctioneaza intotdeauna. De exemplu, nu se poate aplica la în pointer care indica spre structuri complexe de date. De aceea se impun anumite restrictii.

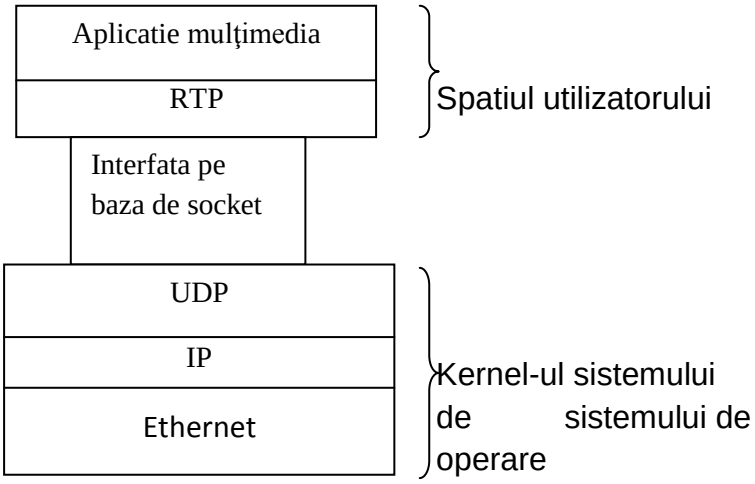
O alta problema este aceea ca în unele limbaje, ca C, este posibil şi nu constituie o eroare, calculul produsului unor vectori, fără specificarea prealabila a dimensiunii lor. Fiecare era terminat de o valoare speciala cunoscuta procedurilor. În cazurile acestea, stub-ul client nu poate comprima parametrii pentru ca nu ştie exact cat de mari sunt.

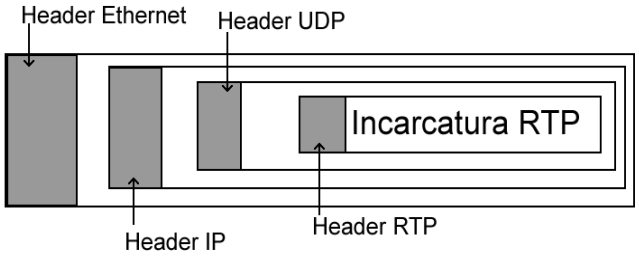
Inca un obstacol: nu se poate deduce intotdeauna tipul de parametrii, nici după o specificare formala a lor în cod. Exemplu: *printf* care poate acea orice număr de parametrii, şi care pot fi în orice ordine şi de aproape orice tip.

A patra problema se refera la utilizarea variabilelor globale. În mod normal, procedurile ar puta comunica folosind variabile globale, dar dacă procedura apelata se afla la distanta, codul nu va reusi să le acceseze. [11]

3. Protocoale de transport de timp real

RPC între client şi server este una din aplicatiile în care este foarte folosit UDP-ul. Alta utilizare: aplicaţii mulţimedia de timp real. Radio-ul prin internet, telefonica prin internet, programele *music-on-demand,* video-conferintele şi altele au devenit extreme de populare şi mai toate folosesc în principiu acelaşi protocol de transport în timp real. Acesta este RTP (Real Time Protocol) . Este descries în RFC3550 şi este foarte raspandit printer aplicatiile mulţimedia. Transporta informaţia audio şi video în pachete, şi procesarea are loc în mare parte la receptor. Locul lor în stiva de protocoale se observa în figura urmatoare:





RTP funcţioneaza în spatiul utilizatorului deasupra UDP-ului (in sistemul de operare). Aplicatia mulţimedia lucreaza cu mai multe stream-uri audio, video, text şi poate şi altele. Acestea sunt introduce apoi în bilioteca RTP, care se afla în spatial utilizatorului, impreuna cu aplicatia. Biblioteca multiplexeaza stream-urile şi le codeaza în pachete RTP, pe care pe trimite la un socket. Pe partea sistemului de operare, se genereaza pachete UDP care le include pe cele RTP şi sunt trimise stratului IP pentru transmisie prin Ethernet. La receptor are loc procesul invers. Aplicatia mulţimedia primeste informaţia de la biblioteca RTP, şi va reda informaţia media. Stiva protocoalelor este în figura de mai sus, în partea stanga. În partea dreapta este ilustrrata imbricarea pachetelor. O consecinta a acestei proiectari este dificultatea de a clasa RTP intr-un anumit strat. Ruleaza în spatial utilizatorului şi este legat de aplicaţie, deci pare a fi un protocol de aplicaţie. Este totuşi un protocol generic, independent de aplicaţie, care oferă doar facilitate de transport, deci se incadreaza în categoria protocoalelor de transport. O descriere mai buna: este un protocol de transport implementat în stratul de aplicaţie.

Functia de baza a RTP este de a multipleza mai multe stream-uri de timp real intr-un singur stream de pachete UDP. Acesta din urma poate fi trimis către o singura destinaţie (unicast) sau mai multe (multicasting). Pentru ca RTP foloseste UDP simplu, pachetele lui nu sunt tratate special de rutere, decat dacă niste caracteristici specifice de QoS (Quality Of Service) ale IP sunt activate. Nu există garantii special ale livrarii, deci pachetele pot fi pierdute, corupte, intarziate etc. [11]

Totuşi, RTP contine niste caracteristici specific care ajuta receptorii să proceseze informaţia mulţimedia. Fiecarui pachet trimis printr-un stream RTP i se aloca un număr cu 1 mai mare decat predecesorului sau. Aceasta număratoare permite masinii destinaţie să determine lipsa unuia sau mai multor pachete. Dacă unul lipseste, aplicatia decide ce se face. Poate omite un cadru video dacă acesta este tipul de informaţie transportat, sau poate interpola o valoare lipsa dacă se transporta informaţive audio. Retransmisia nu este o soluţie practica, pentru ca, cel mai probabil, ar sosi prea tarziu pentru a mai fi utilizat. Consecinta acestui fapt: RTP nu are confirmare şi nici un mechanism de cerere de retransmisie.

Un pachet RTP poate contine mai multe esantioane, care pot fi codate în orice fel (depend de aplicaţie). Pentru a putea conlucra (emiţătorul cu receptorul), RTP defineste cateva profile (de exemplu: stream audio unic), şi pentru fiecare din ele, se accepta mai multe formate de codare. Exemplu: un stream audio unic, poate fi codat PCM cu 8 biti (Pulse Code Modulation), esantionat la 8 KHz, codare predictiva, codare GSM, MP3 etc. RTP are un camp în header în care sursa poate specifica tipul de codare.

Alta facilitate: marcarea timpului (timestamping). În esenta, sursa asociaza un timestamp primului esantion din fiecare pachet. Timestamp-urile sunt relative la inceputul stream-ului, deci sunt mai importante diferentele între ele (valorile absolute sunt nesemnificative). Acest mecanism permite receptorului să plaseze esantioanele intr-o memorie tampon (buffering) pentru a reda fiecare esantion un anumit interval de timp (atat cat trebuie pentru a putea fi perceput corect), indiferent dacă pachetul care trebuie nu a sosit inca.

Timestamping-ul nu numai ca reduce efectele variatilor intarzierilor în reţea, dar permite şi sincronizarea mai multor stream-uri între ele. Exemplu: un program de televiziune digitala, are un stream video şi două stream-uri audio (pentru sunet stereo sau unul pentru coloana sonora originala şi unul pentru dublare). Fiecare stream vine de la un dispozitiv fizic diferit, dar dacă sunt etichetate de un singur contor, pot fi redate sincronizat, chiar dacă sunt transmise dezorganizat.

În figura urmatoare este reprezentat headerul RTP. Contine 3 cuvinte de 32 de biti (si eventual niste extensii). Primul cuvant contine câmpul pentru versiune (s-a ajuns la versiunea 2). Bitul P indica dacă pachetul a fost bordat pana la multiplu de 4 octeti. Ultimul octet de bordare arata cati octeti au fost adaugati. Bitul X indica existenta unei extensii. Formatul şi semnificatia extensiei nu sunt definite. Se specifica doar lungimea, în primul cuvant al extensiei.

Versiune	P	X	CC	M	Tipul de incarcatura	Numarul secventei
Timestamp						
Identificatorul sursei de sincronizare						
....						
Identificatorii surselor contribuitoare						

Câmpul CC mentioneaza cate surse contribuitoare există (între 0 şi 15). Bitul M este un marker specific aplicaţiei. Poate fi folosit pentru a marca inceputul unui cadru video, inceputul unui cuvant intr-un canal audio, sau orice altceva caracteristic aplicaţiei.

Câmpul *Tip de incaratura* specifica ce algoritm de codare este folosit. Din moment ce fiecare pachet contine acest camp, codarea se poate schimba în timpul transmisiei.

Numarul de secventa este un contor care se incrementeaza la fiecare pachet trimis. Este folosit pentru detectarea pierderii de pachete.

Timestamp-ul este creat de de sursa stream-ului pentru a reduce variatia de intarziere (jitter) la receptie, prin decuplarea dîntre redare şi timpul de sosire al pachetului.

Identificatorul sursei de sincronizare spune carui stream apartine acel pachet. Este folosit pentru a multiplexa şi demultiplexa mai multe stream-uri de dare intr-un / dintr-un singur stream de pachete UDP. Identificatorii surselor contribuitoare, dacă există, sunt folositi candin studio-ul de inregistrare se folosesc mixere. În acest caz, mixer-ul este sursa de sincronizare, şi stream-urile mixate sunt listate aici. [11]

3.1 RTCP

RTCP este un protocol care se ocupă numai de răapunsuri, sincronizari şi interfaţă cu utilizatorul, dar nu transporta deloc date, şi este strans legat de RTP. Prima funcţie oferă feedback către surse în legatura cu intarzierile, latimea de banda, congestia şi alte proprietati ale reţelei. Informatia furnizata de acest serviciu poate fi folosita de procesul de codare pentru a creste rata de date (si de a creste calitatea) atunci când reţeaua funcţioneaza foarte bine, sau pentru a o reduce când există probleme în reţea. Primind răapunsuri continue, algoritmii de codare pot fi adaptati mereu pentru a oferi cea mai buna calitate posibila, în limitele performantelor curente ale reţelei. Exemplu: dacă latimea de banda creste (sau descreste) în timpul transmisiei, se poate folosi codare MP3, PCM pe 8 biti, sau codare Delta. Câmpul „tip de incarcatura" este folosit pentru a indica masinii destinaţie ce algoritm de codare s-a folosit pentru pachetul curent (de aceea se poate schimba codarea în timpul transmisiei).

O problema a mecanismului de feedback este aceea ca rapoartele se trimit către toti participantii la comunicare. Pentru o aplicaţie multicast, cu un număr mare de calculatoare, latimea de banda folosita de RTCP creste puternic. Pentru a preveni ocupărea bandei cu rapoarte, rata lor este micsorata, undeva sub 10% din banda media. Fiecare participant trebuie să ştie dimensiunea benzii, pe care o poate calcula cu ajutorul emiţătorului şi a numărului de participanti, pe care îl poate determina ascultand porturile RTCP.

RTCP se ocupă şi de sincronizarea între stream-uri. Diferite stream-uri pot folosi diferite surse de tact („ceasuri" diferite), cu granularitati şi derive diferite. RTCP poate să le sincronizeze, în ciuda acestor impedimente. [11]

3.1.1 Pachetele RTCP

Sunt specificate mai multe tipuri de pachete pentru a transporta diverse infromatii de control:

- SR (*Sender Report*): contine statistici de transmisie şi receptie de pa participantii care emit activ.
- RR (*Receiver Report*): contine statistici de la participantii care nu emit activ
- SDES: descrie obiectele sursa, inclusiv CNAME
- BYE: indica sfarsitul participarii unei masini
- APP: pentru funcţii specifice aplicaţiei.

Fiecare pachet RTCP incepe cu o sectiune fixa, similara pachetelor de date ale RTP. Urmeaza elemente structurate, care pot avea lungime variabila, un funcţie de tipul pachetului, insa trebuie să se limiteze pe o frontiera de 32 de biti. Aceasta cerinta de aliniere şi câmpul „Lungime" din partea fixa a ficarui pachet sunt necesare pentru a se putea „stivui" pachetele. Mai multe pachete RTCP pot fi concatenate, fără a fi nevoie de separatoare, pentru a forma un pachet compus care poate fi trimis intr-un unic pachet al protocolului inferior (ca UDP). Nu se număra pachetele individuale RTCP din pachetul compus pentru ca protocoalele din nivelele inferioare tin cont de lungimea totala şi ele determina sfarsitul pachetului compus.

Fiecare pachet individual din pachetul compus poate fi procesat independent, fără a se impune o anumita ordine. Totuşi, pentru a se putea folosi eficient funcţiile protocolului, se impun anumite constrangeri:

- Statisticile de receptie (in SR sau RR) ar trebui trimise cat mai des (in limita constrangerilor latimii de banda) pentru a maximiza rezolutia statisticilor, deci, periodic, unul din pachetele compuse trebuie să contina şi unpachet de raportare
- Participantii noi care primesc date trebuie să primeasca şi CNAME-ul pentru o sursa cat mai repede posibil pentru a identifică sursa şi pentru a asocia informaţia cu anumite scopuri (ca sincronizarea), deci fiecare pachet compus RTCP trebuie să includa şi SDES CNAME.

În concluzie, toate pachetele RTCP trebuie trimise intr-un pachet compus care contine cel putin 2 pachete individuale, cu formatul urmator:

- Prefix de criptare: dacă pachetul compus trebuie securizat, trebuie prefixat cu un număr pe 32 de biti, aleator, pentru fiecare pachet compus. Dacă este necesara bordarea pentru criptare, trebuie adaugata la ultimul pachet din ansamblu.
- SR sau RR: primul pachet din ansamblu trebuie să fie un pachet de raport pentru a facilita validarea headerului, chiar dacă nu există date de transferat (caz în care trebuie trimis un RR gol) şi chiar dacă şi celalalt pachet din ansamblu este de tip BYE.

- RR aditionale: dacă numărul de surse pentru care se raporteaza statistice depaseste 31 (adică nu mai incape intr-un sungur pachet RR sau SR), atunci primul pachet de RR/SR trebuie urmat de un pachet aditional (sau mai multe, în funcţie de numărul de surse).
- SDES: un pachet SDES care contine un obiect CNAME trebuie inclus în fiecare ansamblu RTCP. Optional pot fi incluse şi alte surse dacă sunt cerute de o anumita aplicaţie, în limita constrangerilor legate de latimea de banda.
- BYE sau APP: alte tipuri de pachete RTCP pot aparea, în orice ordine, dar BYE trebuie să fie ultimul pachet trimis.

Un participant RTP trebuie să trimită doar un singur pachet compus RTCP pe intervalul de raportare, pentru ca estimarile privind banda pentru fiecare participant să fie corecte. Exceptie: când pachetul compus este impartit pentru criptare partiala. Dacă sunt prea multe surse şi nu incap toate pachetele RR intr-un singur ansamblu fără depasirea MTU (maximum transmission unit), atunci doar subsetul care incape intr-un MTU ar trebui inclus în fiecare interval. Subseturile trebuie selectate folosind metoda round-robin, astfel incat toate sursele să fie raportate. [4]

v=2	p	Nr RR	tip de pachet	Lungime mesaj
SSRC al raportului emitator				
Timestamp NTP(2 cuvinte de 32 biti)				
Timestamp RTP				
Contor cumulativ de pachete al emitatorului				
Contor cumulativ de octeti al emitatorului				
Blocul 1 al raportului de receptie				
Blocul 2 al raportului de receptie				
.....				

Pachet RTCP (cu header-ul inclus) [5].

3.1.2 Intervalul de transmisie

RTP este proiectat pentru a permite scalabilitate unei aplicatii, de la sesiuni de dimensiuni mici la sesiuni de mii de participanti. Exemplu: intr-o conferinta, traficul se limiteaza automat, pentru ca nu pot vorbi mai mult de 2-3 oameni deodata (nu ar intelege nimeni nimic). Deci, rata de date, la multicasting, este relativ constanta independent de numărul de participanti. Însă, traficul de control nu se limiteaza de la sine. Dacă rapoartele de receptie de la fiecare participant ar fi trimise cu o rata constanta, traficul de control ar creste liniar cu numărul de participanti. Deci rata trebuie scazuta şi trebuie calculat dinamic un interval între pachetele transmise.

Presupunem ca există o limita a traficului de date, numit „latimea de banda a sesiunii”, care se divide între participanti. Aceasta se poate alege în funcţie de un

cost sau de niste informaţii apriori despre latimea disponibila sesiunii. Deseori, latimea este egala cu suma latimilor nominale ale fiecarui emiţător activ.

Latimea sesiunii o da de obicei o aplicaţie de management a sesiunii, atunci când invoca aplicatia media, insa aplicatiile media pot seta o valoare implicita, pe baza latimii pentru singur utilizator, pentru codarea selectata pentru sesiune. Aplicatia poate impune şi limite la banda regulilo de multicast sau a altor criterii, insa toti participanti trebuie să aiba aceeaşi caloare pentru latimea sesiunii, astfel incat să se poata calcula un singur interval RTCP pentru toata lumea.

Calculul latimii de banda pentru traficul de date şi de control include şi protocoalele inferioare (ca IP şi UDP) (ar trebui să le ştie sistemul de gestiune al resurselor). Este de asteptat ca aplicatia să ştie care din aceste protocoale este folosit. Header-eole de legatura nu sunt incluse în calcul, din moment ce pachetele vor fi incapsulate cu diferite header-e pe masura ce calatoresc.

Traficul de control trebuie limitat la o fractiune cunoscuta din latimea de banda a sesiunii (cat mai mica, dar suficienta pentru a nu impiedica sarcina de a transfera informaţii a protocolului de transport). De asemenea, trebuie să o cunoasca toti partiipantii pentru ca fiecare să îşi poata calcula independent partea lui de banda. Se recomanda ca fractiunea de latime de banda pentru RTCP să fie fixata la 5%. De asemenea, se recomanda ca o patrime din latimea pentru RTCP să fie dedicată participantilor care trimit date, astfel incat, în sesiunile cu număr mare de receptori, dar cu putini emiţători, cei care vor să între în sesiune să poataprimi cat mai repede CNAME-ul pentru emiţători. Desi valorile acestor constante nu sunt critice, este esential ca tori participantii la sesiune să folosească aceleasi valori, astfel incat să se calculeze acelaşi interval. Deci constantele acestea trebuie fixate pentru un anumit profil. Un profl poate să specifice ca latimea de banda asociata traficului de control să fie un parametru separat al sesiuni, şi nu un procentaj strict din banda sesiunii. Folosirea unui parametru separat permite aplicaţiilor cu rata adaptabila să sa seteze o latime care să corespunsa cu una tipica pentru datele trimise, şi care e mai mica decat latimea maxima specificata de parametrul sesiunii.

Un profil mai poate specifica dacă latimea pentru traficul de control este divizata în doi parametri separati pentru participantii activi şi pentru cei pasivi. Fie acesti parametrii S şi R. Urmarind recomandara ca ¼ din banda să fie alocata emiţătorilor, valorile recomandate pentru acesti parametrii sun de 1.25% , respectiv 3.75%.

Folosirea ambilor parametrii permite oprirea rapoartelor RTCP, prin setarea latimii de banda pentru cei care nu emit date la 0. Însă oprirea rapoartelor de receptie RTCP nu este recomandata, pentru ca sun necesare pentru feedback-ul calitatii şi controlul congestiei. Totuşi, poate fi avantajoasa pentru sistema care opereaza pe legături unidirectionale sau sesiuni care n-au nevoie de feedback-ul calitatii receptiei.[4]

4. DCCP – Datagram Congestion Control Protocol

4.1 Introducere

DCCP este un protocol al nivelului de transport orientat pe mesaje. Implementeaza setarea unei conexiuni sigure, inchiderea ei, ECN (Ecplicit Congestion Notification), control de congestie, şi negociere de caracteristici. DCCP oferă o care de accesare a mecanismelor de control al congestiei, fără a fi necesara implementarea lor în nivelul de aplicaţie. Permite fluxuri similar TCP, dar nu asigură şi livrarea în ordinea transmisiei. Livrarea secventiala a mai multor fluxuri (ca în SCTP – Stream Control Transmission Protocol) nu este disponibila în DCCP.

DCCP este utilizat în aplicaţii în care se impun constrangeri temporal asupra livrarii pachetelor. Din aceasta categorie fac parte jocurile online multiplayer, telefonia prin internet, streaming-ul de informaţii media (video, audio). Caracteristica esentiala a acestor aplicaţii este aceea ca mesajele vechi devin rapid expirate, îşi pierd utilitatea. Prioritate mai mare o au mesajele noi, de aceea nu este utila incercarea de retrimitere a pachetelor (ar consuma timp şi resurse de reţea inutil).

De asemenea, DCCP poate fi folosit ca mecanism general de control al congestiei pentru aplicaţii care au la baza protocolul UDP. Se poate adauga şi un mecanism pentru siguranţă sei, eventual, unul pentru livrarea pachetelor în ordinea transmisiei. În acest context, DCCP permite utilizarea diferitelor mecanisme de control al congestiei, în general a celor TCP-friendly.

O conexiune DCCP contine atat trafic de confirmare, cat şi trafic de date. Confirmarile anunta emiţătorul ca pachetele lui au sosut la destinaţie sau dacă au fost marcate de ECN. Confirmarile sunt transmise cu gradul de siguranţă pe care îl cere mecanismul de control al congestiei. Este posibila atingerea unui grad de 100% al sigurantei.

4.2 Congestie

Am tot mentionat cuvantul „congestie”, insa nu a fost definit clar.

Congestia reţelei este fenomenul de deteriorare a calitatii serviciilor (QoS – quality of service) produs de supraincarcarea nodurilor reţelei, deci termenul este asociat mai ales reţelelor de dimensiuni mare, în care se vehiculeaza cantitati importante de date.

Congestia are mai multe cauze: fie router-ele nu sunt suficient de rapide (procesoarele lor sunt prea lente, şi nu reusesc să goleasca cozile de asteptare în timp util, chiar dacă reţeaua este suficient de libera), fie se transmit mai multe stream-uri care trebuie apoi trimise pe aceeaşi linie de iesire (se aduna iar cozi), fie buffer-ele nu sunt suficient de mari şi se pierd pachete. În cazul unui trafic foarte mare, situatia se poate agrava atat de tare incat nu mai este livrat nici un pachet. [7]

4.3 Caracteristici DCCP

DCCP are următoarele caracteristici:

- Un flux nesigur de datagrame, cu confirmare
- Negociere sigura de optiuni, inclusiv negocierea unui mecanism potrivit pentru controlul congestiei
- Protocol de tip handshake sigur pentru iniţializarea şi inchiderea conexiunii
- Descoperirea unitatii maxime transmisibile pe calea aleasa (MTU = Maximum Transmission Unit)
- Mecanisme care permit serverelor să evite pastrarea de stari pentru incercări de realizare deconexiuni, neconfirmate, sau pentru conexiuni deja inchise.
- Control de congestie, inclusiv ECN şi ECN NONCE (....)
- Mecanisme de confirmare care comunica pierderea pachetelor şi informaţii ECN.
- Mecanisme optionale care comunica aplicaţiei emitatoare, cu grad mare de siguranţă, care pachete au ajuns la receptor şi dacă ele au fost marcate de ECN, corupte sau eliminate în buffer-ul receptorului.
- Alegerea mecanismelor modulare de control al congestiei. În momentul de faţă, sunt specificate două metode: TCP-like Congestion Control (control de congestie asemanator TCP) şi TCP-Friendly Congestion Control. [10]

Intregul scop al DCCP este acela de a oferi o metoda standard de a implementa controlul congestiei simplu şi negociarea lui pentru aplicatiile de timp real. Un motiv pentru care DCCP este des folosit este acela ca permite utilizarea ECN, impreuna cu controlul congestiei endd-to-end, pentru aplicaţii care altfel ar folosi UDP (care nu este orientat pe conexiune şi nu are nici un mecanism de control de congestie, deci aplicatia care îl foloseste ar trebui să îşi implementeze propriul sistem de control). A fost proiectat astfel incat să genereze cat mai putine date de overhead, legat atat de dimensiunea antetelor pachetelor, cat şi de stare şi overhead CPU necesar la capatul de receptie.

Pe langa acestea, DCCP implementează şi iniţializarea unei conexiuni sigure, incheierea ei şi negocierea de caracteristici.

Alta motivatie a folosirii DCCP e aceea ca a fost proiectat astfel incat să permita aplicaţiilor să aleaga o alternativa la mecanismul curent de control al congestiei (TCP-Style Congestion Control) care injumatateste fereastra de congestie, ca răapuns la aparitia unei indicatii de congestie. DCCP permite aplicaţiilor să aleaga între mai multe forme de control. Una din ele este controlul similar TCP (TCP-like Congestion Control), şi injumatateste fereastra de congestieca răapuns la eliminarea sau marcarea unui pachet (ca în TCP). O alta alternativa este TFRC (TCP Friendly Rate Control). Este o forma de control bazată pe ecuatii, care minimizeaza schimbarile bruste ale ratei de trimitere.

4.4 Conexiunea DCCP

Orice conexiune DCCP ruleaza între două capete. Le voi numi DCCP A şi DCCP B. Datele circula în ambele directii, folosind 4 tipuri de pachete:

1. pachete de la A la B
2. confirmări de la B la A
3. pachete de la B la A
4. confirmări de la A la B

Fiecare flux de pachete de mai sus reprezinta un set. Voi defini niste termeni pe care ii folosesc în continuare:

- sub-fluxuri = un subflux este un pachet fie de date, fie de confirmare, trimis intr-o singura directie. Fiecare din cele 4 seturi de mai sus reprezinta un subflux (subfluxurile se pot intercala intr-un anumit grad, deoarece confirmările pot fi trimise imediat după pachetele cu date).
- secvente = o seventa consta în totalitatea pachetelor trimise intr-o singura directie, indiferent dacă sunt date sau confirmări. Seturile 1 cu 4, şi 2 cu 3 de mai sus sunt secvente diferite. Fiecare pachet dintr-o secventa are un alt număr de secventa.
- Emitator/receptor pe jumatate de conexiune (half-connection) = în contextul unei conexiuni injumatatite, emiţătorul e cel care trimite date, iar receptorul le primeste, trimitănt inapoi confirmări. Exemplu: în semi-conexiunea A-B, DCCP A este emiţătorul, iar DCCP B este receptorul.

Fiecare semiconexiune este gestionata de un mecanism de control al congestiei. Capetele negociaza aceste mecanisme la iniţializarea conexiunii (cele două mecanisme nu trebuie să fie neaparat identice). Mecanismele conformante de control al congestiei corespund identificatorilor de byte, sau CCID-uri. CCID pentru o semiconexiune descrie cum emiţătorul unei semiconexiuni limiteaza rata pachetelor de date, cum mentine parametrii necesari (ca ferestrele de congestie), cum receptorul trimite feedback-ul de congestie prin confirmări şi cum gestionează rata confirmărilor.

Fiecare conexiune DCCP este initiata activ de unul din participanti, care se leaga la un socket DCCP aflat în stare pasiva de ascultare. Capatul activ e va numi client, iar cel pasiv: server. Majoritatea specificatiilor pentru DCCP sunt identice şi pentru server şi pentru client, insa doar serverul poate cere inchiderea unei conexiuni. Deci clientul nu poate forta serverul să mentina status-ul unei conexiuni după ce aceasta a fost inchisa.

DCCP nu suporta dewschidere simultana, ca TCP. Asta inseamna ca un host nu trebuie să raspundă unei cereri DCCP decat dacă portul destinaţie specificat în cerere corespunde unui socket local deschis pentru ascultare. DCCP nu poate lucra nici cu conexiuni semideschise (adică inchide ambele semiconexiuni ca pe o unitate întreaga).

DCCP foloseste mecanisme generice pentru negocierea proprietatilor conexiunii, ca CCID-urile active pe semiconexiuni. Un nume de proprietate, ca „CCID” este asociat, în general, la două caracteristici, una pentru fiecare semiconexiune. De exemplu, există 2 CCID-uri pentru o conexiune. Capatul care comanda o anumita caracteristica se numeste: locatia caracteristicii. Valorile caracteristicilor sunt negociate de optiunile: „Change”, „Confirm” şi „Prefer”. „Change” (schimba) este trimis către locatia unei caracteristici pentru a cere schimbarea valorii penbtru acea caracteristica. Locatia poate raspunde cu „Prefer” (preferinta), care cere schimbarea valorii de la capatul care a trimis cererea de schimbare, sau îşi poate schimba el insusi valoarea caracteristicii, raspunzand apoi cu „Confirm” (confirmare). Negocierea de caracteristici este una fiabila din cauza retransmisiilor.

4.5. Pachetele DCCP

DCCP foloseste noua tipuri de pachete:

1. DCCP-Cerere
2. DCCP-Raspuns
3. DCCP-Date
4. DCCP-Confirmare
5. DCCP-Confirmare de date
6. DCCP-Cerere de inchidere
7. DCCP-Inchidere
8. DCCP-Reset
9. DCCP-Transfer

Descrierea informaţionala conexiunii:

1. Clientul trimite serverului un pachet DCCPCerere în care specifica porturile clientului şi ale serverului, serviciul cerut şi orice alte caracteristici care se negociaza, inclusiv CCID-ul pe care clientul vrea să îl folosească serverul. Clientul poate trimite în spatele acestui pachet şi niste date, ca o cerere la nivelul aplicaţiei, pe care serverul poate să o ignore.
2. Serverul trimite clientului un pachet DCCP-Raspuns, indicand ca va comunica cu clientl. Raspunsul indica optiunile şi caracteristicile cu care serverul este de acord, incepe (sau continua) alte negocierei de caracteristici (dacă este necesar) si, optional, include şi o procedura de iniţializare de cookie-uri care impacheteaza toate aceste informaţii şi care trebuie trimise inapoi la client pentru a completa conexiunea.
3. Clientul trimite serverului un pachet DCCP-Confirmare care confirma pachetul DCCP-Raspuns. Acesta confirma numărul initial de secventa al serverului şi trimite inapoi iniţializarea de cookie, dacă a există vreuna un pachetul de răapuns. Poate include şi negociere de caracteristici.
4. Pot urma (sau nu) mai multe pachete de confirmare pentru a finaliza negocierea de caracteristici. Clientul poate trimite şi o cerere către aplicaţie în

spatele pachetului de confirmare, producand astfel un pachet DCCP-Confirmare de date.

5. În aceasta etapa, serverul şi clientul fac schimb de pachete de date, pachete de confirmare, si, optional, pachete de confirmare de date, care contin date şi confirmări. Dacă clientul nu are date de trimis, stunci serverulk va trimite pachete DCCP-Date şi DCCP-confirmare de date, în timp ce clientul va trimite numai pachete de confirmare.
6. Serverul trimite un pachet DCCP-cerere de inchidere (cerand inchiderea conexiunii). (numai serverul poate cere inchiderea conexiunii !).
7. Clientul trimite un pachet DCCP-Inchidere pentru a confirma inchiderea.
8. Serverul trimite un pachet DCCP-Reset şi stergte starea conexiunii.
9. Clientul primeste pachetul de RESET şi pastreaza starea conexiunii un interval rezonabil de timp pentru a permite pachetelor ramase să sosească inainte de a inchide conexiunea.

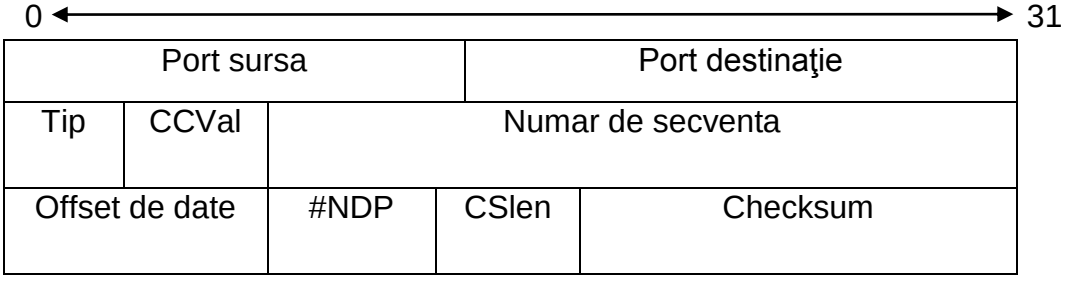
O alternativa la secventa de inchidere de mai sus este cea în care inchiderea este initiata de client:

6. Clientul trimite un pachet DCCP-Incidere pentru a cere inchiderea conexiunii.
7. Serverul trimite un pachet DCCP-Reset cu câmpul „Scop” (campurile sunt tratate în paragrafele urmatoare) setat pe „Inchidere” şi sterge starea conexiunii.
8. Clientul primeste pachetul de RESET şi pastreaza starea conexiunii un interval rezonabil de timp pentru a permite pachetelor ramase să sosească inainte de a inchide conexiunea.

Aceasta metoda de comunicare cu handshake pentru initierea şi inchiderea conexiunii permite serverului să refuze pastrarea oricarei stari pana când are loc handshake-ul cu clientul, ceea ce asigură ca clientul trebuie să tine starea de asteptare la inchiderea conexiunii.

4.6 Header-ul

Toate pachetele DCCP incep cu un antet generic:



Porturile sursa şi destinaţie: representate pe 16 biti fiecare, seamana cu campurile omoloage din TCP şi UDP. Portul sursa reprezinta portul relevant la capatul care care trimite pachete, iar destinaţia este portul care asculta.

TIP: 4 biti, specifica tipul de mesaj DCCP. Sunt definite următoarele valori:

0. Pachet de cerere
1. Pachet de răapuns
2. Pachet de date
3. Pachet de confirmare(ACK)
4. Pachet de confirmare de date (Data Ack)
5. Pachet de cerere de inchidere
6. Pachet de inchidere
7. Pachet de reset
8. Pachet de transfer

CCVal: 4 biti, rezervat pentru trimiterea CCID-ului. CCID-ul emiţător de la A la B, care este activ la DCCP A, poate trimite informaţi receptorului de la DCCP B, codand informaţia în CCVal.

Numarul de secventa: 24 biti. Acest camp este initializat de pachetul de cerere sau de răapuns şi incrementat cu 1 la fiecare pachet trimis. Eceptorul foloseste aceasta informaţie pentru a determina dacă s-au pierdut pachete pe drum. Chiar şi pachetele care nu contin date actualizeaza numărul de secventa. Aceste numere oferă şi o protectie impotriva pachetelor vechi sau a celor trimise de programe malware. DCCP de rata mare poate necesita protectie importuca numerelor de secventa impachetate. Exemplu: un flux de 10Gb/s cu pachete de 1500 octeti va trimite 2^24 pachete în ~ 20 de secunde. În termenii calatoriei prin reţea, este un timp destul de mare, dar tot există riscuri. Numerele de secventa initiale ale celor două subfluxuri sunt setate de primele pachete de cerere, respectiv e răapuns trimise, şi ar trebui alese ca pentru TCP. Un număr initial de secventa trebuie să includa o componenta aleatoare (sau pseudo-aleatoare) care să ingreuneze munca atacatorilor de a interveni în umerele de secventa. Numarul initial de secventa ales pentru un identificator de conexiune dat (adresa şi portul sursa + adresa şi portul destinaţie) ar trebui să creasca cu timpul, pentru a evită livrarea neadecvata a pachetelor vechi.

Offset-ul de date: 8 biti. Este offset-ul de la inceputul header-ului pana la inceputul incarcaturii utile a pachetului, masurata în cuvinte de 32 de biti.

#NDP (number of non-data packets): 4 biti, este numărul de pachete care nu contin date. DCCP seteaza acest camp cu numărul de pachete fără date trimise oana atunci în secventa să (modulo 16). Un pachet non-date este un pachet care nu contin informaţie de la utilizator (pachetele de confirmare, de inchidere, de cerere de inchidere şi de reset sunt mereu fără date, în timp de pachetele de cerere, răapuns şi de transfer pot avea data sau nu). Acest camp ajuta DCCP-ul receptor să decida dacă un pachet pierdut continea date de la utilizator. De exemplum pachetul 10 avea #NDP setat la 5. Pachetul 11 s-a pierdut, iad pachetul 12 are #NDP=5. Atunci receptorul îşi da seama ca pachetul 11 continea date, din moment ce nu s-a schimbat #NDP. Dacă acesta ar fi fost 6 pentru pachetul 12, pachetul nu ar fi continut deloc date.

CSLEN (Checksum length): 4 biti. Câmpul acesta specifica ce parti din pachet sunt acoperite de checksum (acesta acopera cel putin header-ul, optiunile şi un pseudoheader luat de la header-ul nivelului reţea). Dacă lungimea checksum-ului este 0, asta e tot ce acopera. Dacă este 15, acopera şi incarcatura utila a pachetului. Alte valori decat între 0 şi 15 se considera experimentale.

Checksum: 16 biti. DCCP foloseste algoritmul de verificare de la TCP-IP. Câmpul de checksum este egal cu complementul lui 1 al complementului tuturor cuvintelor de 16 biti din header, optiuni şi din pseudoheader-ul luat de la headerul nivelului reţea, si, în funcţie de lungimea specificata în CSLEN, o parte din incarcatura, poate chiar toata.

Pseudoheaderul este calculat tot ca pentru TCP. Pentru IPV4, are lungimea de 96 de biti şi consta din adresele sursa şi destinaţie, numărul de protocol IP pentru DCCP (bordat la stanga cu 8 biti de 0) şi lungimea DCCP (pe 16 biti), cu tot cu header cu optiuni + lungimea datelor.

Pachetele cu checksum invalid trebuie ignorate, şi optiunile lor nu trebuie procesate. [7]

5. TCP-Like Congestion Control

DCCP foloseste identificatori de control al congestiei, sau CCID-uri (Congestion Control Identifiers) pentru a specifica ce mecanism de control al congestie se foloseste pe o semiconexiune.

CCID-ul similar TCP (il voi prescurta TCP-L) trimite date folosind o varianta apropiata de de mecanismul de control folosit de TCP, incluzand mai multe confirmări selective: SACK (Selective Acknowledgements). Aceasta varianta de control al congestiei se numeste CCID2, şi este potrivita pentru emiţătorii care se pot adapta cresterilor bruste din fereastra de congestie, tipica metodei AIMD (Additive increase, Multiplicative Decrease) a lui TCP, şi este utila pentru cei care vor să profite de latimea de banda intr-un mediu în care conditiile se schimba rapid.

Există două tipuri de pachete: pachete de date, pe care le trimit emiţătorii, şi pachete de confirmare (ack), trimise de receptori.

CCID2 este potrivit pentru fluxurile DCCP care au nevoie de cat mai multa latime de banda, pe termen lung. Fluxul trebuie să tolereze variatii mari ale ratei de transmisie, caracteristica AIMD, incluzand şi injumatatirea ferestrei de congestie ca răapuns pentru un eveniment de aglomerare. Poate fi folosit de aplicaţii care trebuie să transfere cat mai multe date în cel mai scurt timp posibil.

În contrast cu CCID2, CCID3 (sau TFRC = TCp Friendly Rate Control) este potrivit pentru fluxuri în care se prefera minimizarea schimbarilor bruste ale ratei de transmisie. De excemplu, CCID2 este m,ai potrivit decat CCID3 pentru streaming mulţimedia care foloseste buffere mari la receptie inainte de redare, izoland intr-un fel aplicatia de schimbarile rapide rata de transmisie. Astfel de aplicaţii ar trebuie să aleeaga CCID2 şi în locul TCP-ului, adaugand optional şi i forma de asigurarea a fiabilităţii.

Alt avantaj al lui CCID 2 este ca mecanismele lui de control al congestiei similare TCP sunt clar intelese, iar dinamica traficului seamana cu cea de la TCP. [1]

5.1 Relatia cu TCP

Diferentele între CCID2 şi controlul direct al congestiei prin mecanismul TCP sunt:

1. CCID 2 aplica controlul la confirmări, un mecanism care nu este standardizat inca pentru TCP.
2. DCCP este un protocol cu datagrame, deci mai multi parametrii ale caror unitati de masura sunt octetii în TCP (ca fereastra de control)se masoara în pachete la DCCP.
3. Fiind un protocol nefiabil, DCCP nu retransmite niciodata un pachet, deci mecanismele de control al congestieicare disting pachetele trimise initial de cele retransmise au fost reproiectate pentru contextul DCCP.

5.2 Exemplu pentru semiconexiune

Exemplul urmator arata evolutia unei semiconexiuni folosind controlul de congestie al CCID2:

1. Emitatorul trimite pachete DCCP de date, numărul lor fiind controlat de o fereastra de congestie, ca în TCP. Fiecare pachet DCCP-Data are un număr de secventa. Semitaorul trimite şi o caracteristica AckRatio (raport de confirmări) în care specifica numărul de pachete de date care trebuie acoperite de un pachet de confirmare de la receptor. Implicit, AckRatio este 2.câmpul CCVal din headerul DCCP este setat la 0. Presupunand ca semiconexiunea are capabilitate ECN, fiecare pachet DCP-Data se trimite cu capacitate ECN.
2. Receptorul trimite un pachet DCCP-Ack, confirmand pachetele de date în funcţie de setarea AckRatio. Fiecare pachet de confirmare are un număr de secventa şi contine vectorul de confirmare (AckVector). Numarul de secventa confirmat intr-un pachet DCCP-Ack este acela al pachetului de date cu cel mai mare număr (nu este ca la TCP, cu confirmări cumulative). Receptorul returneaza suma anunturilor ECN primite prin optiunile vectorului de confirmare permitand emiţătorului să verifice probabilistic faptul ca receptorul primeste corect. Pachetele DCCP-Ack de la receptor au şi ele capabilitate ECN, din moment ce emiţătorul va controla rata confirmărilor intr-o maniera TCP-Friendly, folosind caracteristica AckRatio.
3. Emitatorul continua să emita pachete DCCP-Data, controlate de fereastra de congestie. Dupa primirea pachetelor de confirmare, emiţătorul examineaza vectorii lor de conformare pentru a afla dacă s-au pierdut sau distrus pachete, şi regleaza fereastra de congestie. Fiind un transfer nesigur, emiţătorul nu va retrimite pachetele eliminate.

4. Pentru ca pachetele de confirmare folosesc numere de secventa, emiţătorul are niste informaţii despre pachetele pierdute sau marcate, şi raspunde la pierderi sau marcari prin reglarea raportului de confirmare (AckRatio) trimis receptorului.
5. Emitatorul confirma confirmările receptorului cel putin odata intr-o fereastra de congestie. Dacă ambele semiconexiuni sunt active, confirmarea emiţătorului despre confirmările receptorului este inclusa în confirmarea emiţătorului despre pachetele de date ale receptorului. În cazul în care calea inversa a semiconexiunii este pasiva, emiţătorul trimite cel putin un pachet DCCP-DataAck (confirmare de date) pe fereastra de congestie.

Emitatorul estimeaza timpul de dus-intors, fie prin urmarirea timpilor confirmărilor (asa cum face TCP), fie prin optiunea explicita de timestamping, şi calculează valoarea de expirare (TimeOut) similar cum se calculeazătimpul de retransmisie din TCP. TimeOut-ul determina când un pachet de date nou poate fi transmis, atunci când emiţătorul este limitat de fereastra de congestie şi nu există nici un feedback de la receptor. [1]

6. TFRC – TCP-Friendly Rate Control

TFRC este un mecanism de control al congestiei pentru fluxuri unicast. A fost proiectat pentru aplicaţii care folosesc o dimensiune fixa a pachetului şi poate rula alaturi de TCP cu pachet fix fără a consuma o latime de banda prea mare. Are insa o variatie mai mica a timpului de traversare a reţelei, comparativ cu TCP, ceea ce îl face mai adecvat pentru aplicaţii ca media streaming sau telefonie, unde o rata de transmisie constanta este importanta.

Trade-off-ul pentru rata constanta de transmisie consta în faptul ca raspunde mai greu la schimbarile de banda disponibila. Deci TFRC ar trebui folosit nimai când aplicatia are nevoie explicita de transmisie „lina”, evitănt injumatatirea ratei de transfer (ca la TCP), ca răapuns la pierderea unui pachet. Pentru aplicaţii care trebuie să transfere cat mai multe date în cel mai scurt timp posibil, se recomanda TCP, insa dacă nu este necesara fiabilitatea, se poate folosi şi schema de control al congestiei AIMD (Aditive Increase, Multiplicative decrease).

TFRC a fost proiectat pentru a da performante maxime aplicaţiilor care folosesc dimensiune fixa a segmentului, şi variaza rata de trimitere a pachetelor ca răapuns la congestie. [1]

6.1 Mecanismele protocolului

Pentru mecanismul de control, TFRC implementează direct o ecuatie de transfer pentru rata de transmitere permisa, ca funtie de rata evenimentelor de pierdere şi timpul de dus-intors. TFRC foloseste direct ecuatia de transfer a TCP, unde rata este o funcţie de evenimentele de pierdere, timpulo de dus–intors şi dimensiunea segmentelor. Mecanismul de control funcţioneaza astfel:

1. Receptorul masoara rata evenimentelor de pierdere şi trimite aceasta informaţie emiţătorului.
2. Emitatorul foloseste aceste mesaje pentru a determina timpul dus-intors pentru un pachet (RTT = round-trip time).
3. Rata evenimentelor de pierdere şi RTT sunt introduse în ecuatia de transfer şi rata de transmisie rezultata este limitata la cel mult dublul ratei de receptie, pentru a se putea transmite cu rata X (cea calculata).
4. emiţătorul ajusteaza apoi propria rata de transmitere pentru a se potrivi ratei de transmisie X. [1]

6.2 Ecuatia de tranfer

Ecuatia de transfer este urmatoarea (o forma usor simplificata):

$$X[B/s] = \frac{s}{R \cdot \sqrt{\frac{2 \cdot b \cdot p}{3}} + (t\_RTO \cdot (3 \cdot \sqrt{\frac{3 \cdot b \cdot p}{8}} \cdot p \cdot (1 + 32 \cdot p^2)))}$$

Unde:

- X[B/s] este media ratei de transmisie
- S este dimensiunea segmentului în octeti
- R = timpul dus-intors (RTT) în secunde
- p = rata evenimentelor de pierdere a pachetelor, cuplinsa între 0 şi 1, şi este fractiunea de pachete pierdute din totalul pachetelor emise
- t_RTO = timpul de expirare pentru retransmisie, exprimat în secunde
- b = numărul maxim de pachete care sunt confirmate cu un singur pachet de confirmare TCP. [1]

7 Bibliografie

- [1] RFC5348
- [2] RFC3448
- [3] RFC4828
- [4] RFC3550
- [5] http://www.cs.odu.edu/~cs778/jeffay/Lecture6.pdf
- [6] http://www.networksorcery.com
- [7] http://opalsoft.net/qos/
- [8] RFC4341
- [9] RFC 4336
- [10] RFC4340
- [11] *Andrew S. Tanenbaum*: Computer Networks
- [12] http://www.scribd.com/doc/22272663/All-About-UDP
- [13] http://www.netfor2.com/udp.htm
- [14] www.wikipedia.com
- [15]http://www.ardenstone.com/projects/seniorsem/reports/UDP_Protocol.html

- [16]www.unibuc.ro/prof/niculae_c_m/telecom/user_datagram_protocol_udp.htm

B. Protocolul TCP - Transmission Control Protocol

Lecu Radu Șerban

1) Introducere

Protocolul TCP (Transmission Control Protocol) este unul dinte protocoalele nivelului Internet al modelului TCP/IP

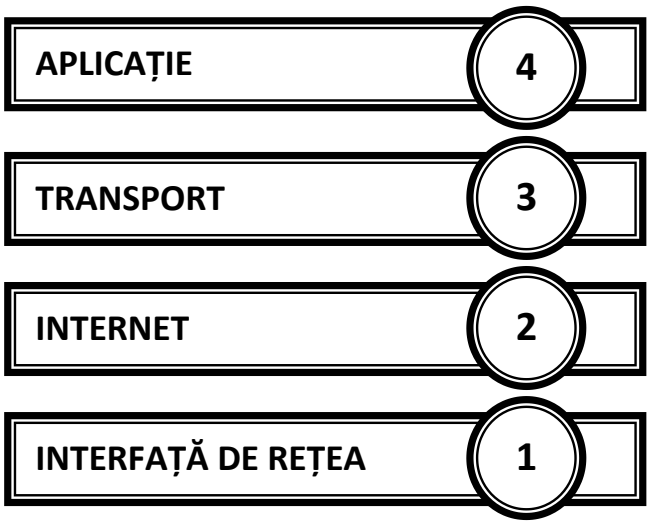
TCP a fost special conceput pentru a oferii un transport de biți de tip capăt-la-căpat, fără erori sau pierderi de date. El a fost definit în standardul RFC 793 (Request For Comment) scris în 1981.

O rețea de internet diferă de una globală deoarece diferite păți pot avea topologii, lațimi de bandă, timpi de întârziere diferiți, dimeniuni ale pachetelor sau alți parametrii diferiți. Astfel TCP a fost astfel conceput să se adapteze la proprietățile diferite ale componentelor folosite în cadrul rețelei.

Fiecare mașină care suportă TCP este astfel concepută încât este capabilă să transmită informații și date prin transmisia unui șir de octeți. Un sir de octeți poate fi grupat în pachete de date de lungime de maxim 64KB incluzând și informațiile de transimise (headerul). În cazul în care informația ,cum ar fi cea conținută de un fișier, depășește această valoare, ea trebuie divizată în mai multe pachete de către emitor în așa manieră încât să poată fi recuperată și reconstruită la recepție. Pachetele sunt transmise independent, iar ele pot ajunge la destinație pe căi diferite și în altă ordine. [5]

Protocolul TCP reprezintă unul din compnentele modelul TCP/IP (Protocol de control al transmisiei/ Protoclo Internet) care a fost creat de US DoD (Departamentul de aparare al Statelor Unite) din necesitatea unei rețele care ar putea supraviețui în orice condiții. Scopul rețelei TCP/IP era ca orice conexiune s-ar rupe în interiorul rețelei rețeaua în ansamblu să rămână intacta.

Astfel a trebuit conceputa o arhitectură complexă , flexibilă și care avea în vedere aplicații divergente, printre care se transmit informații (cum ar fi vorbirea, fisiere) în timp real.



Pornind de la structura modelului OSI, cerințele de mai sus au condus la alegerea a 4 niveluri pentru modelul TCP/IP

Nivelul Transport este similar celui în cadrul stivei OSI, ocupandu-se de probleme legate de siguranță, controlul fluxului și corecție de erori. El a fost astfel conceput încât sa permită conversații între entitățile pereche sursă și destinație. Astfel au fost concepute 2 protocoale capăt-la-capăt: TCP și UDP.

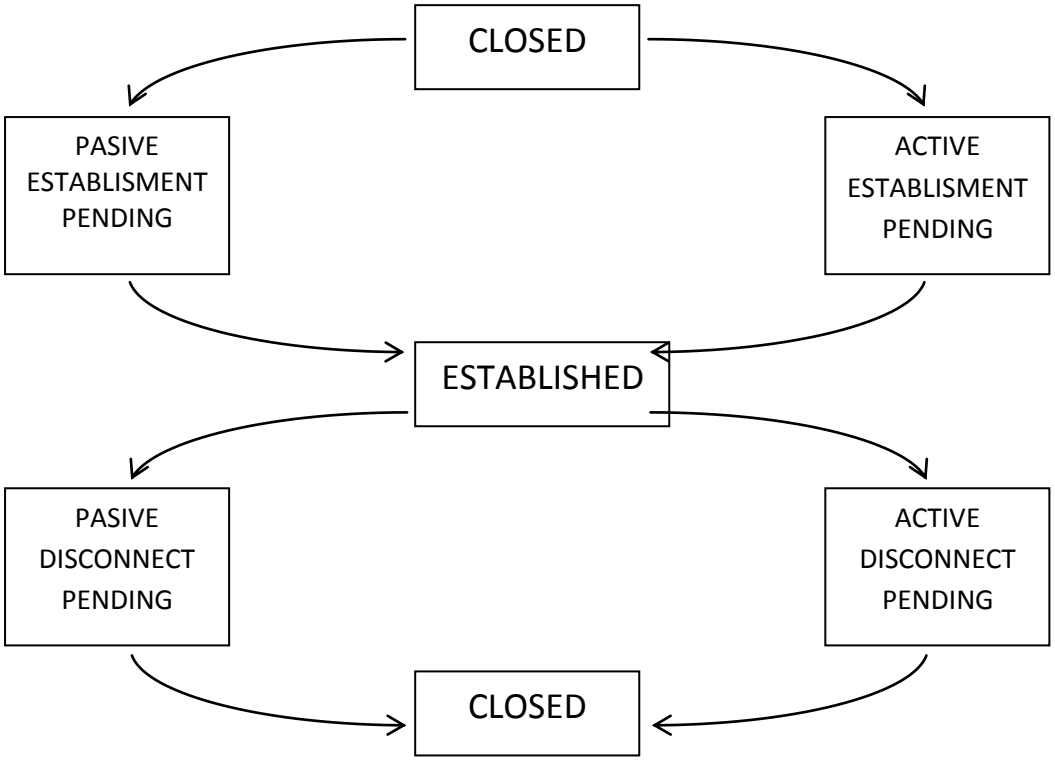
Protocolul TCP este un protocol orientat pe conexiune care permite ca un flux de octeți transmiși să ajunga la destinație ,fără erori, la orice altă mașină din cadrul aceleiași rețea. [4]

Modelul TCP este protocolul folosit de nivelul Transport al stivei TCP/IP.

2) Modelul Serviciului TCP

Serviciul TCP trebuie asigurat atat de emitator cat si de receptor deoarece se bazează pe principiul capăt-la-capăt și folosește socketi. Fiecarui socket îi corespunde un număr cu rol de adresă numit adresa IP a destinatarului si un alt numar alcatuit din 16 biti numit port. Pentru ca serviciul TCP să fie obținut trebuie realizată explicit o conexiune între mașina care transmite mesajul și cea care îl recepționează. Simplificat o conexiune pate fi reprezentat astfel:

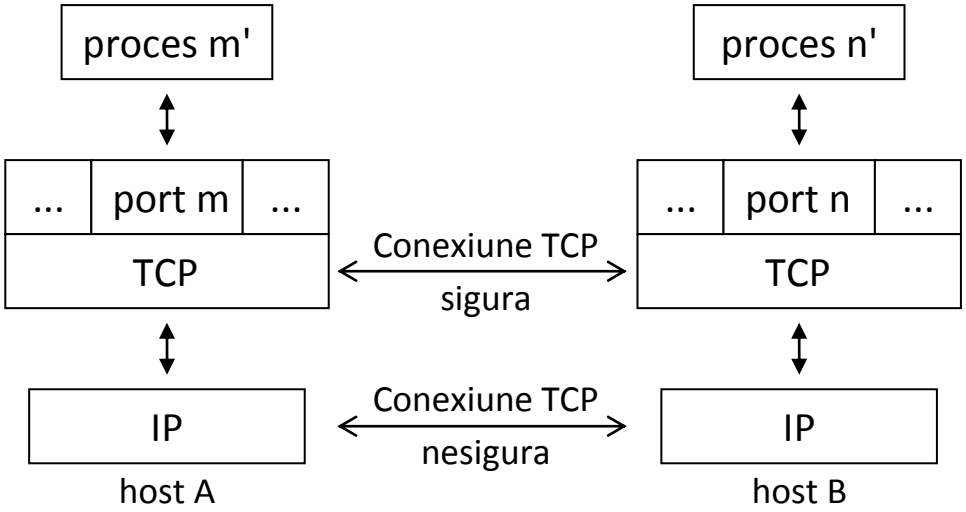
Astfel TCP poate fi comparat cu o conexiune telefonica din punct de vedere al modului în care utilizatorul vede această conexiune. Diferența de baza este dată de faptul că mesajele sunt transmise prin intermediul pachetelor fără a bloca o linie telefoincă în cazul netransmiterii de informație.[1]



Astfel un anumit port se poate afla în diferite stări, cum ar fi IDLE, ESTABLISHED sau stari intermediare în cadrul cărora se realizează stabilirea conexiunii sau deconectarea. Conexiunea presupune 3 faze: realizarea conexiunii (IDLE->ESTABLISHED), transmiterea informației (starea ramane ESTABLISED) și intreruperea conexiunii (ESTABLISHED->IDLE).

Un socket poate fi folosit pentru mai multe conexiuni simultane. Astfel 2 sau mai multe conexiuni se pot termina la același socket. Acestea sunt identificate prin identificatori de socket aflati la ambele capete.

Conexiunea TCP poate fi reprezentată în felul urmator:[6]



Din cele 65536 porturi (2^{16}) , primele 1024 porturi sunt numite well-known-ports și sunt rezervate pentru servicii standard. Spre exemplu se doreste realizarea unui transfer de fișiere prin intermediul protocolului FTP (File Transfer Protocol). Aceasta se realizează prin conexiunea la portul 21 pentru a-și realiza conexiunea cu propriul demon. Demonul reprezintă o apicație care rulează pe un anumit calculator fără control direct realizat de utilizator. Câteva exemple de porturi mai importante sunt cele din tabelul urmator

Port	Protocol	Utilizare
21	FTP	File transfer
23	Telnet	Remote login
25	SMTP	E-mail
69	TFTP	Trivial file transfer protocol
79	Finger	Lookup information about a user
80	HTTP	Worl Wide Web
110	POP-3	Temote e-mail access
119	NNTP	USENET news

Ar fi posibil ca de exemplu demonului FTP să îi fie asociat portului 21 în momentul butării. Similar pentru celelalte porturi. Totuișsi acest lucru ar umple memoria cu demoni ce vor fi idle majoritatea timpului.

Modul de realizare este următorul: Un singur demon, numit inetd (Internet demon) este atașat la rândul său la porturi multiple și așteaptă prima conexiune. Când se realizează acest lucru, inetd realizează un nou proces și execută demonul corespunzător, lăsându-l pe aceesta să se ocupe de cerere. Cu toate acestea este posibilă si realizarea unor demoni utlizați permanent cum ar fi portul 80 (HTTP) și inetd în rest.

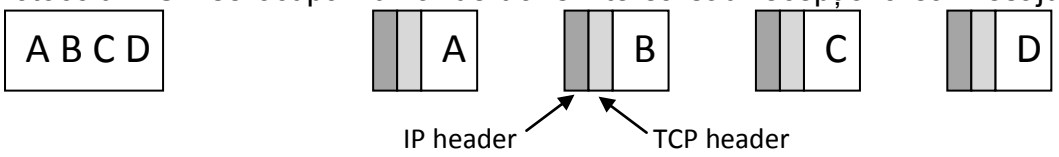
Toate conexiunile TCP sunt de tip full-duplex capăt-la-capăt. Full-duplex înseamnă că transmisia se poate face bidirecțional și simultan între cele 2 calculatoare. Capăt-la-capăt arată că nu pot exista mesaje de tip multicast sau broadcast.

O conexiune TCP reprezintă o înșiruire de biți (octeți), care au o structură bine definită și care este împărțită în diferite componente. Pe langă mesaj (șirul de octeți ce reprezintă informația) la fiecare nivel se mai adaugă și alte informații numite

headere. Acestea conțin informații cum ar fi ip sursa, ip destinație, port destinație, informații de detecție a erorilor, informații de ordonare a pachetelor pentru reconstrucția întregului mesaj din fragmentele sale, etc. și sunt asociate fiecărui nivel în parte. În cazul lipsei unor astfel de informații pachetele pot ajunge la destinație dar sunt inutilizabile, sau pot chiar să nu fie recepționate.

Să presupunem că avem un mesaj înpărțit în 4 fragmente (A,B,C,D). Fiecărui mesaj îi va corespunde un header ip si unul TCP ca în figurile urmatoare:

Protocolul TCP se ocupă numai de transmiterea sau recepționarea mesajului



a) b)

nu și de interpretarea lui. La emisie mesajului i se adaugă informațiile suplimentare din headere. La recepție mesajul este separat de headere fiind transmis nivelelor superioare.

Important este de menționat că pentru realizarea conexiunii se folosesc anumite pachete cu ajutorul cărora emițătorul și receptorul comunică pentru realizarea conexiunii. Pachetele care sunt folosite pentru realizarea conexiunii au proiritate față de altele în care se transmite informație. Acestor pachete li se asociază un flag numit URGENT flag. Cand un pachet urgent este recepționat acesta are prioritate față de celelalte fiind primul interpretat chiar dacă alte pachete anterioare au venit inaintea acestuia.[1]

3) Protocolul TCP

Receptorul si transmițătorul comunică prin intermediul unor segmente. Un segment TCP este alcătuit dintr-un header de 20 biți la care se adaugă mesajul. Dimnesiunea header-ului plus cea a mesajul pot avea împreună până la 65515 octeți.

Protocolul de bază al TCP-ului este sliding window protocol, care se bazează pe principiul ferestrei glisante. În momentul în care emitătorul transmite un segment, acestra porneste și un timer. Când segmentul ajunge la destinație, receptorul transmite un segment (cu sau fără date) care conține un numar de confirmare egal cu următoarea secvență pe care o așteaptă. Daca timerul emițătorului este depășit înainte de recepția mesajului de confirmare, mesajul este retransmis. [6]

Spre deosebire de UDP care transmite pachete între 2 hosturi fără a oferii siguranța că acestea ajung la destinație, TCP este protocol orientat pe conexiune. Astfel pentru fiecare pachet transmis de la sursă la destinație, cu ajutorul arhitecturii TCP, trebuiesc parcurse mai multe etape:

- realizarea conexiunii dintre cele 2 hosturi
- schimbarea de informații între acestea
- întreruperea conexiunii

In cazul UDP, un pachet este transmis fără a avea siguranța că informația a ajuns de la sursă la destinație. Același pachet transmis folosind protocolul TCP, va oferii siguranța că pachetul a ajuns la destinație. Mai mult, în cazul în care locatia dorită nu poate fi găsită, emițătorul va fi informat de acest lucru.

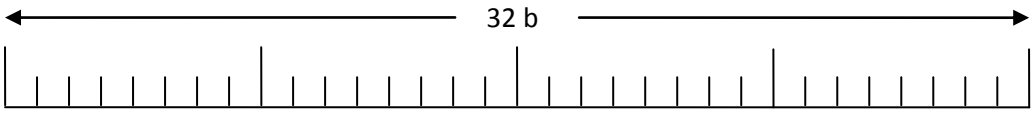
Există 2 cazuri în care nu se poate transmite un pachet.

a) Nu se poate realiza conexiunea între cele 2 hosturi. În momentul în care se doreste realizarea conexiunii locația nu poate fi gasită sau nu acceptă realizarea conexiunii.

b) Conexiunea a fost deja realizată anterior între cele 2 hosturi dar pachetul nu a putut fi transmis. Acest lucru se poate întampla fie din cauză că unul din hosturi s-a deconectat fără a anunța intreruperea conexiunii (ex: întreruperea curentului), fie fără ca cererea de închidere să ajungă la celălalt capăt al conexiunii (legătura dintre calculatoare a fost întreruptă si nu există altă rută alternativă).

4) Header-ul TCP

Fiecare segment începe cu un format fix de 20 B urmat de cuvinte duble de 32 biti optionale si de date. Câmpul de date poate să conțină până la 65495 octeti (65535 - 20 octeti pt header-ului ip -20 coteti ai header-ului TCP).



Source port								Destination port
Sequence number								
Acknowledgement number								
TCP header length		U R G	A C K	P R H	R S T	S Y N	F I N	Window size
Checksum								Urgent pointer
Optinos (0 or more 32-bit words)								
Data (optional)								

Portul sursă si portul destinație arată porturile de emisie și de recepție. Portul plus adresa ip formează o adresa unica de 48 biti.

Sequence number si Acknowledgement number realzeaza functiile de verificare daca a ajuns pachetul la destinatie sau nu. Ambele au 32 de biti.

TCP header length arată lungimea headerului în cuvinte de 32 biți. Informația este necesară deoarece headerul include și campul de opțiuni care este variabil.

Urmează 6 biți nefolosiți si alti 6 care reprezintp flaguri care au rol diferit

- URG - Urgent pointer - descris mai sus
- ACK - Acknowledge flag - arată că Acknowledgent number este valid adică pachetul are rol de confirmare a recepției mesajului
- PSC - Push Function - arată că mesajul trebuie transmis aplicației cât mai repede posibil. Astfel se oferă prioritate pachetului față de alte pachete.

- RST - Reset conection - arată apariția unei probleme ce impune resetarea conexiunii

- SYN - Syncronyze -este folosit pentru realizarea conexiunii.

- FIN - No more data from sender - este folosit pentru închiderea conexiunii.

Window size are rolul de a arăta lungimea câmpului de date în octeți.

Checksum reprezintă o sumă de verificare cu ajutorul căruia se determină dacă datele transmise sunt eronate sau nu. Aceiasi operație este realizată în ambele capete iar dacă ele coincid atunci este foarte probabil ca informația obținută să fie corectă.

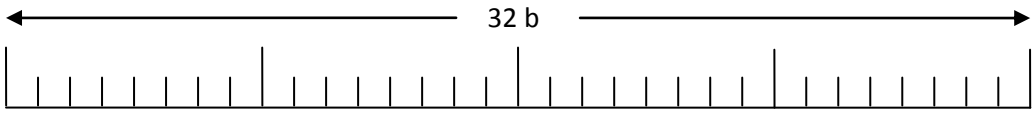
Urgent pointer este folosit când se dorește ca anumite date sa fie procesate în cel mai scurt timp posibil. El indică unde se termină ultimul byte de date urgente.[5]

Options indică anumite informații opționale ce pot fi transmise în scopul obținerii anumitor funcționalități. Una din cele mai importante functionalități este lungimea maximă a segmentului (MSS Maximum Segment Size). Aceasta are importanță în cazul în care avem o linie de transmisie cu multe erori. Dacă segmentul are lungimea mai scurtă, probabilitatea ca să avem cel puțin un bit eronat este mai mică iar în cazul în care se retransmite pachetul se retransmite o cantitate mai mică de informație. Astfel în anumite cazuri este de preferat să împărtim un mesaj în fragmente mai multe dar mai mici decît să avem pachete de dimesiune mare. În aelasi timp numar mare de pachete duce la cantitatemare mare de informație datorată headere-lor dar inutila pentru utilizator și care crește durata de transmsie a întregului mesaj. Valoarea predefinită a dimensiunii mesajului este de 536 B pentru date.

Câmpul de date reprezintă informația ce va fi transmisă către sau recepționată de la nivelul de aplicație pe portul cerut.

Pseudoheader-ul este folosit pentru calculul câmpului Checksum el nefiind transmis în cadrul pachetului. Se observă că acesta conține și informații cara nu aparțin nivelului TCP deoarece IP-ul este o informație a nivelului IP, violând astfel ierarhia TCP/IP.

Urmatoarea figură arată structura pseudoheader-ului[6]:

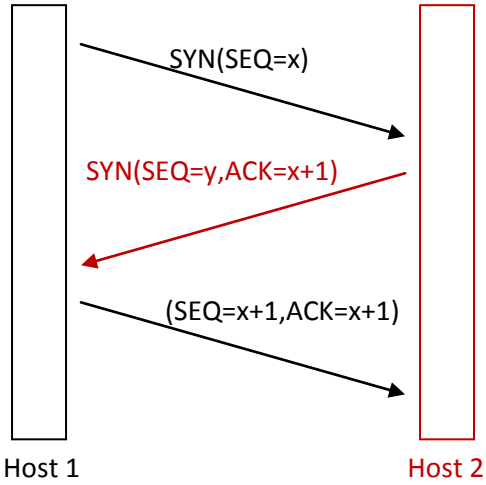


Source address		
Destination address		
00000000	Protocol =6	TCP segment length

5) Stabilirea conexiunii TCP

Realizarea conexiunii presupune o secventă de 3 pachete transmise între cele 2 hosturi cunoscută sub numele de three-way handshake. Pentru stabilirea conexiunii este necesar ca un host, numit Rețea, să aștepte pasiv realizarea conexiunii. În același timp celălalt host, numit Client, va transmite un pachet prin care cere realizarea conexiunii la adresa IP si la portul dorit. Pachetul transmis are setate flagul SYN =1 si ACK=0. Reteaua răspunde prin transmiterea unui pachet având flagurile SYN=1 si ACK=1. În final Clentul retransmite un pachet având setați SYN=0

si ACK=1, moment în care s-a realizat conxiunea în cazul în care nu au aparut alte erori de-a lungul intregului proces. [1]



Exista posibilitatea ca simltan ambele hosturi să încerce realizarea conexiunii între aceiasi 2 socketi. În final este realizată numai o singura conexiune.[1]

6) Întreruperea conexiunii TCP

Conexiunea full-duplex dintre hosturi poate fi văzută si ca 2 conexiuni semi-duplex

Pentru a se realiza întreruperea conexiunii, oricare din cele 2 hosturi pot cere acest lucru cu ajutorul flagului FIN. Deoarece când unul dintre hosturi cere întreruperea conexiunii celălalt poate transmite date, întreruperea conexiunii se face simultan în ambele parți. Porcedeul este realizat similar realizarii conexiunii fiind realizat printr-o secventa de 3 pachete.

Există însă situația ca mesajul de încheiere a conexiunii să nu fie recepționat. Dacă în intervalul de timp maxim în care se poate transmite si receptiona un mesaj nu s-a primit un răspuns se încheie conexiunea numai intr-o singura parte, ramanand ca celalalt host să observe că nimeni nu mai asculta mesajele sale. [1]

7) Managementul conexiunii

Conexiunea TCP poate fi reprezentată printr-un automat cu număr finit de stări. Automatul are 11 stări prezentate în tabelul următor.

Ca orice automat, în funcție de starea curentă nu se poate trece decât în anumite stări. Succesiunea acestori stări este reprezentata în graficul de mai jos. Tot în grafic seunt prezentate si primitivele (comenzile cu care utilizatorul face trecerea dintr-o stare în alta).

Stare	Descrierere
CLOSED	Nici o conexiune nu este activă
LISTEN	Servărul asteaptă un apel
SYN RECEIVED	A fost primită o cerere de conexiune
SYN SENT	Aplicația a început deschiderea conexiunii
ESTABLISHED	Starea normală pentru transferul de date
FIN WAIT 1	Aplicația a zis că a terminat de transmis date
FIN WAIT 2	Cealaltă parte a acceptat închiderea conexiunii
TIME WAIT	Așteptare ca toate pachetele să fie transmise
CLOSING	Ambele părți încearcă închiderea conexiunii simultan
CLOSE WAIT	Cealaltă parte a inițiat închiderea conexiunii
LAST ACK	Așteaptă ca toate pachetele să fie transmise

Fiecare conexiune începe cu starea CLOSED. Această stare este părăsită numai când se face o deschidere pasivă (hostul are rol de retea, adică se așteaptă cererea de conexiune de la alt host) sau când se face una activă (hostul are rol de client, el apelează un alt host cu rol de rețea, căruia îi cere realizarea conexiunii).

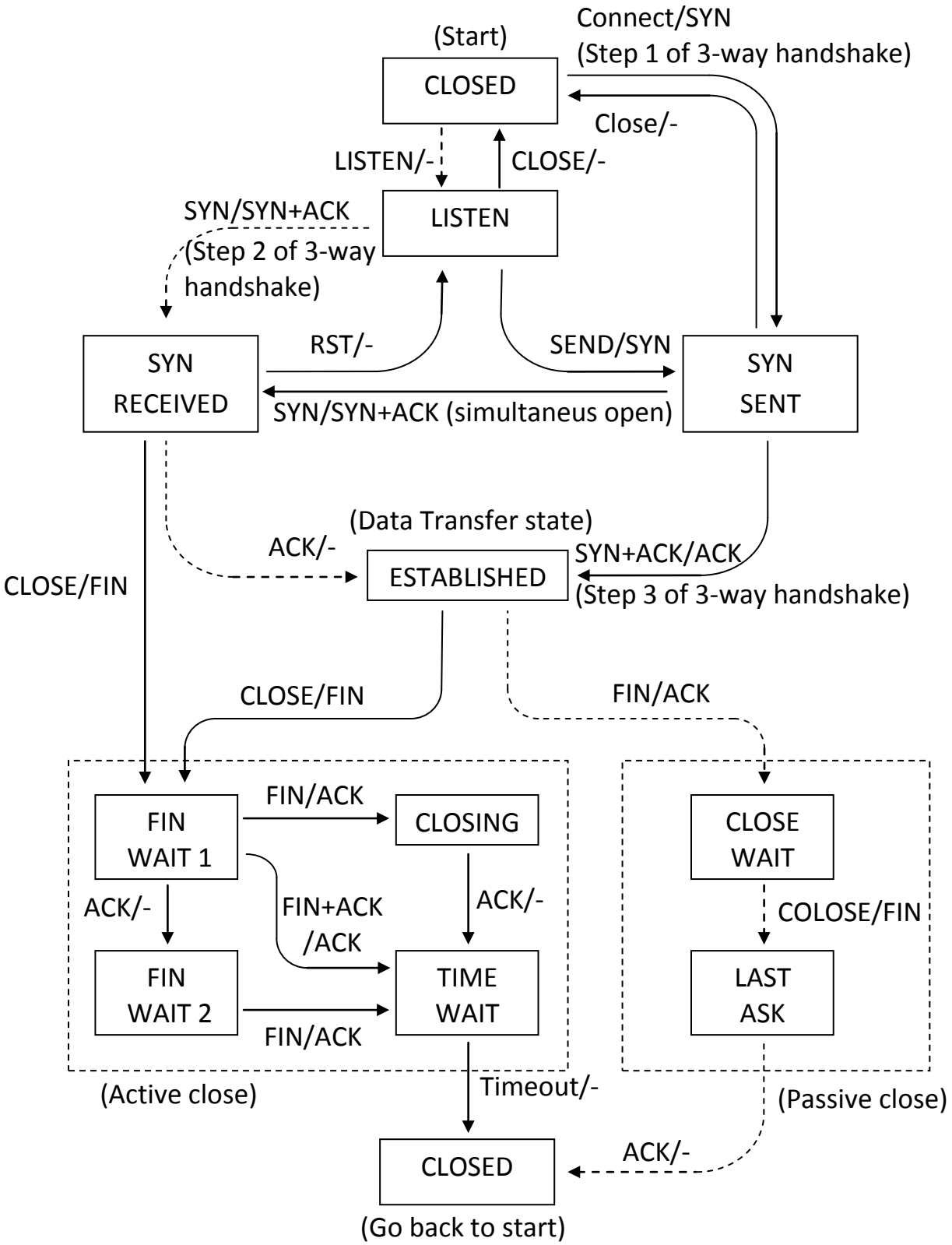
Rețeaua trece în starea LISTEN, stare în care așteaptă cererea de conexiune. Clientul trece în starea SYN SENT și transmite un segment de tip SYN. Dacă existăun host de tip rețea la adresa cerută, care ascultă pe portul corespunzător, acesta recepționează mesajul și transmite un mesaj de tip SYN+ACK. În cazul în care clientul nu primește un astfel de mesaj dupa un anumit interval de timp, el trece înapoi în starea CLOSED. Altfe el transmite un semnal de tip ACK, moment în care se realizează conexiunea.

Atât clientul, cât și rețeaua dupa ce primsc pachetul ACK, trec în starea ESTABLISHED. În această stare se realizează comunicarea bidirecțională între cele 2 hosturi.

Metoda prezentată mai sus poartă numele de 3-way Handshake.

În orice moment, oricare din cele 2 hosturi poate realiza întreruperea conexiunii. Aceasta se realizează prin metoda prezentată mai sus.

Există 2 moduri de întrerupere a conexiunii: activă, în cazul celui care a inițializat întreruperea conexiunii, sau pasivă în cazul celui care este anunțat de întreruperea conexiunii.[1]


(Start)
CLOSED
Connect/SYN
(Step 1 of 3-way handshake)
Close/-
LISTEN/-
CLOSE/-
LISTEN
SYN/SYN+ACK
(Step 2 of 3-way handshake)
SYN RECEIVED
RST/-
SEND/SYN
SYN SENT
SYN/SYN+ACK (simultaneus open)
(Data Transfer state)
ACK/-
ESTABLISHED
SYN+ACK/ACK
(Step 3 of 3-way handshake)
CLOSE/FIN
CLOSE/FIN
FIN/ACK
FIN WAIT 1
FIN/ACK
CLOSING
CLOSE WAIT
ACK/-
FIN+ACK /ACK
ACK/-
COLOSE/FIN
FIN WAIT 2
TIME WAIT
LAST ASK
FIN/ACK
Timeout/-
(Active close)
(Passive close)
ACK/-
CLOSED
(Go back to start)

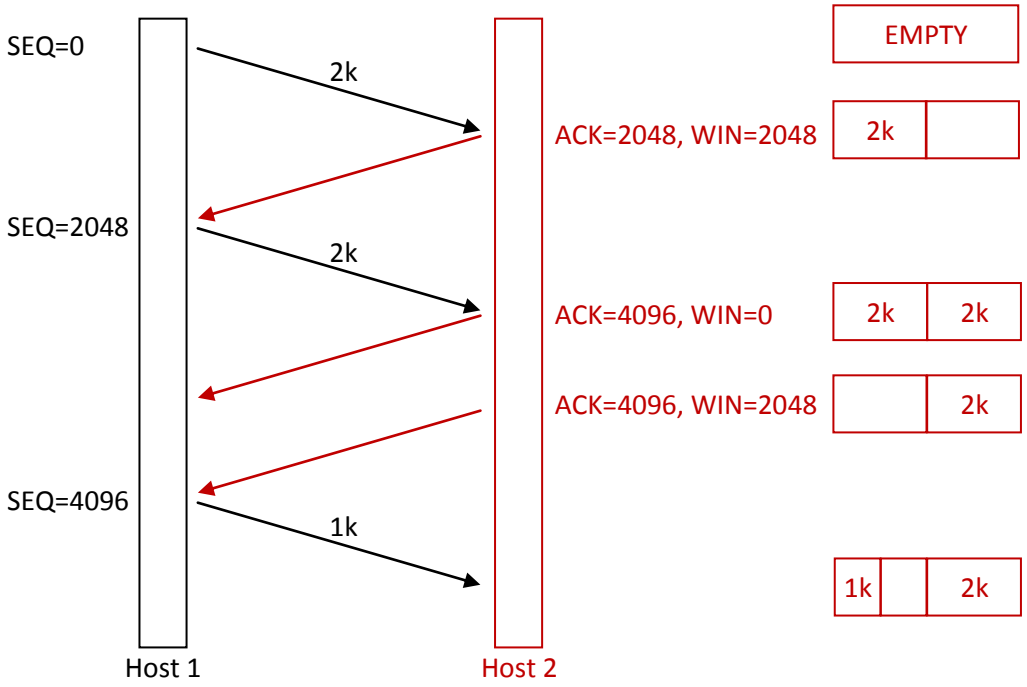
8) Politici de transmisie TCP

In capitolele anterioare s-a discutat despre modul în care se realizează conexiunea dintre 2 entități diferite, cum se întrerupe conexiunea dintre acestea, structura segmentelor și a headerelor acestora. Toate acestea au ca scop final oferirea posibilității de a transmite informații între hosturi fără pierdere de informație.

Transmisia de informații între cele 2 hosturi se face numai când se presupune că ambele capete ale conexiunii sunt în starea ESTABLISHED. Astfel se porneste de la ideea că a fost realizată anterior conexiunea.

Fiecare host are un buffer în care odată pachetele recepționate ele sunt stocate într-un buffer. Acest buffer are o dimensiune limitată. Astfel dacă are loc transmiterea unui pachet care nu poate fi stocat atunci acesta va fi pierdut fiind necesară retransmisia sa.

Pentru a minimiza cantitatea de informație pierdută s-au introdus protocoale de management ale ferestrei. Modul de funcționare este prezentat în figura următoare:



După cum se observă în imagine, receptorul, în momentul în care trimite mesaje de ACK, trimite și informații suplimentare pentru a informa emițătorul despre spațiului liber din buffer, prin utilizarea câmpului WIN. Valoarea câmpului WIN arată spațiul liber disponibil al bufferului. Astfel, dacă bufferul este plin se va oprii transmisia. În momentul în care se eliberează o zonă din buffer, receptorul trimite un mesaj cu dimensiunea zonei eliberate din interiorul bufferului adică cantitatea maximă de informație pe care emițătorul o poate transmite. [1]

9) Managementul timerelor TCP

Pentru transmisia datelor, se pun următoarele probleme:

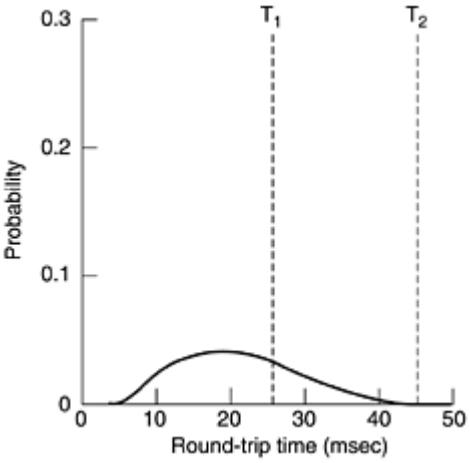
- fiecare segment trebuie să ajungă la destinație;
- un segment ajuns la destinație nu trebuie să aibe biți eronați.

A doua condiție este verificată cu ajutorul checksum din headerul TCP. Prima condiție presupune existența unui timer numit timerul de retransmisie.

TCP-ul folosește multiple timere. Cel mai important este cel de retransmisie. Atunci când un segment este transmis, este pornit un timer. Dacă se recepționează un ACK pentru acel segment înainte ca timerul său să expire atunci se consideră că mesajul a ajuns la destinație. În caz contrar se retransmite segmentul iar timerul se repornește. Se pune problema alegerii dimensiunii timerului.

În cazul în care timerul este mic, există o foarte mare probabilitate ca pachetul să fie retransmis înainte de recepționarea ACK-ului. Astfel se poate provoca o congestie datorită introducerii de pachete inutile în rețea. Dacă timerul este foarte mare atunci intervalul de timp după care are loc retransmisia este foarte mare, astfel că se va aștepta un interval mare de timp recepțiomarea segmentului daca acesta a nu a fost recepționat de prima dată.

Timerul trebuie astfel ales încât să se obțină optimul pentru cele 2 cazuri.[1]



10) Algoritmi utilizați de TCP pentru eficientizarea transmisiei de pachete

Pentru a rezolva diferite probleme ale TCP s-au adoptat 4 algoritmi folosiți de TCP a căror documentație se găseste în RFC 2001 publicat în 1997. Aceștia sunt:

- Slow start
- Congestion avoidance
- Fast retransmit
- Fast recovery algorithms

10.1) Slow start

Versiunile inițiale de TCP ar începe o conexiune cu clientul injectând numeroase segmente în retea de până la lungimea maximă a ferestrei. Acest lucru nu reprezintă o problemă în cazul în care ambele hosturi se află în acelasi LAN. Problemele apar atunci când există rutere între hosturi. Unele rutere intermediare trebuie să păstreze pachetele într-o memorie. Drept urmare vom avea o încărcare exagerată a memoriei și deci la umplerea ei.

Algoritmul pentru evitarea acestor situații este Slow start. Acesta operează prin a observa că rata maximă la care noile pachete ar trebui injectate în rețea este egală cu rata cu care răspunsurile se întorc de la celălalt capăt.

Pentru realizarea algoritmului se adaugă o nouă fereastră emitorului numită fereastră de congestie (cwnd - congestion window), când o nouă conexiune este realizată cu un host din altă rețea, fereastra de congestie este inițializată la un segment (dimensiunea segmentului cerută de la celălalt capăt sau o valoare default, în general egală cu 536 sau 512). De fiecare dată când un nou ACK este recepționat, cwnd este crescută cu încă un segment. Emițătorul poate să transmită până la minimul dintre lungimea ferestrei de congestie si cea de advertisement a receptorului. Fereastra de congestie este controlată de emițător, iar cea de advertisement este controlată de receptor.

Principiul de funționare este următorul: emițătorul începe prin a transmite un segment. Inițial cwin=1. Când receptorul primește mesajul el va trimite un răspuns de tip ACK. Când emițătorul îl recepționează cwin devine 2. De data aceasta se pot

transmite până la 2 pachete. Drept răspuns vor fi 2 ACK-uri deci cwin=4=2^2. În continuare cwin devine 2^3, 2^4, 2^5 și asa mai departe astfel că vom avea o creștere exponențială. Este posibil ca, de exemplu când cwin devine 16, emițătorul să nu mai dorească să mai transmită 16 pachete, ci numai 5. Ca rezultat cwin va deveni 21 nu 32, caz în care nu vom mai avea o creștere exponențială. Dacă continuă creșterea numărului de pachete, la un moment dat capacitatea rețelei poate fi atinsă iar un router intermediar va începe să renunțe la pachete, acestea fiind pierdute. Pierdere lor determină micșorarea ferestrei.

Deși inițial Slow start a fost aplicat numai pentru entități aflate în rețele diferite în prezent este folosită si pentru rețelele locale.[7][9][12]

10.2) Congestion avoidance (Evitarea congestiei)

Congestia poate să apară atunci când, pachetele care intră într-un ruter și care urmează să iasă toate în același LAN, depășesc debitul maxim suportat de LAN. Congestion avoidance este un procedeu prin care se pot evita astfel de situații pentru a minimiza numărul de pachete pierdute.

Presupunerea algoritmului este că pachetele pierdute prin eronare sunt foarte mici (maxim 1%), deci pierderea unui pachet semnealează existența unei congestii între sursă și destinație. Există 2 indicatori ai pierderii unui pachet: expirarea timerului sau recepționarea a 2 ACK-uri.

Congestion avoidance si Slow start sunt algoritmi diferiți cu obiective diferite. Totuși când congestia apare TCP trebuie să încetinească rata de transmisie a pachetelor în rețea, și apoi să invoce Slow start pentru a crește din nou rata de transmisie. De aceea ele sunt în practică implementate împreună.

Cei 2 algoritmi necesită 2 variabile pentru fiecare conexiune:cwind (congestion window) și sstresh(Slow start treshold).

Algoritmul combinat funcționează după cum urmează:

I) Inițializarea conexiunii date. Se setează cwmd la 1 segment și sstresh la 65535.

II) Rutina de transmisie TCP nu transmite mai mult decât minimul dintre cwnd și dimensiunea ferestrei de advertisement a receptorului.

III) Când apare congestia incidcată de expirarea timerului sau de recepționarea de ACK dublu, o jumătate din dimensiunea ferestrei curente (minimul dintre cwnd și dimensiunea bufferului receptorului dar cel puțin egală cu 2) este salvată în sstresh. Suplimentar dacă congestia este determinată datorită expirării timpului cwnd este setat la 1 segment (Slow start).

IV) Când este recepționat un nou ACK de la celălalt capăt, cwmd este crescut cu 1 dar acesta crește în funcție de ce realizează TCP: Slow start sau Congestion avoidance.

Dacă cwnd este mai mic sau egal cu sstresh, TCP se aplică Slow start, altfel este realizat Congestion avoidance. Astfel TCP se află în Slow start până în momentul în care cwnd atinge sstresh după care se realizează Congestion avoidance.

În cazul slow start cwnd creste cu 1 pentru fiecare ACK primit. În cazul Congestion avoidance cwnd este credscut cu segsize^2/cwnd pentru fiecare ACK primit, unde segsize și cwnd au ca unitate de măsură bytes. Astfel vom avea o creștere liniară în cazul Congestion avoidance comparativ cu cea exponențială petru cazul slow start. [7][9][12]

10.3) Fast retransmit (Retransmitere rapidă)

Fast retransmit reprezintă o modalta prin care TCP reduce timpul prin care emițătorul asteaptă pînă când retranmite segmentul pierdut.

Emițătorul TCP ,în varianta sa inițială, foloseste un timer pentru a detecta pierderea de pachete. Dacă un ACK nu este recepționat pentru un anumit pachet până la expirarea timerului, emițătorul va presupune că pachetul a fost pierdut deci va trebui retransmis. De multe ori acest timp duce la întârzieri nedorite care de multe ori pot fi eliminate.

ACK dublu stă la baza algoritmului.

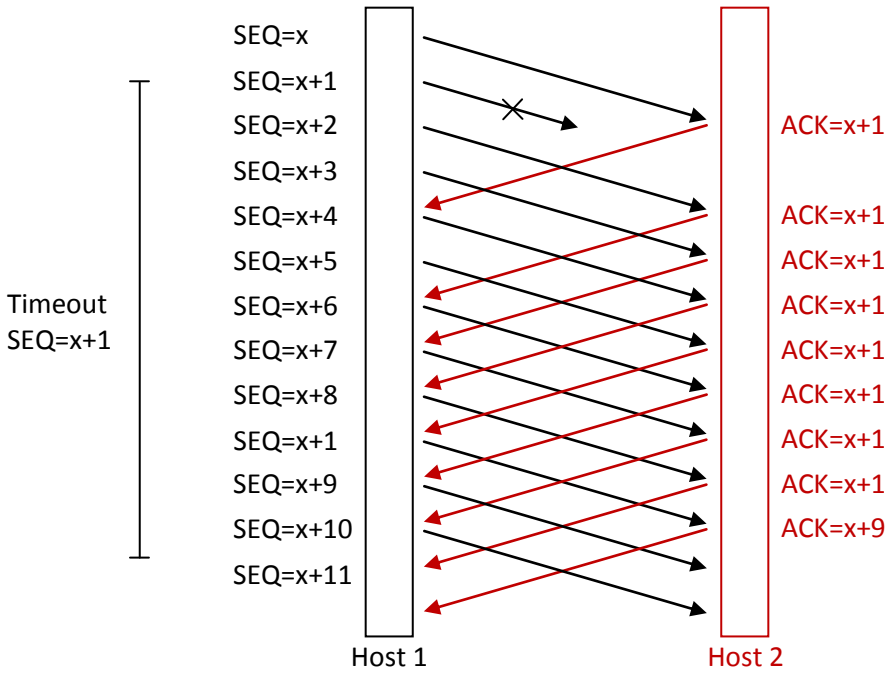
Un ACK dublu poate fi obținut din 2 motive: pierdere de pachet, sau inversare de pachete.

După recepționarea unui pachet cu SQN=x (Sequence number), receptorul transmite un mesaj cu ACK=x+1 (Acknowledgement number) care înseamnă că acesta a primit pachetul x si asteapta pachetul x+1. În ruma acestui procedeu emițătorul este informat despre pachetele care au ajuns la destinație.

Să presupunem că dorim să transmitem un șir de pachete, iar pachetul SQN= x+1 nu ajunge la destinație, el fiind pierdut. SQN=x ajunge decisemițătorul recepționează ACK=x+1. Când SQN= x+2 ajunge la destinație emițătorul recepționează ACK=x+1 în loc de x+3 deoarece se așteaptă încă pachetul x+1. Similar la SQN=x+2, SQN=x+3, SQN=x+4, etc. emițătorul primește ACK=x+1.

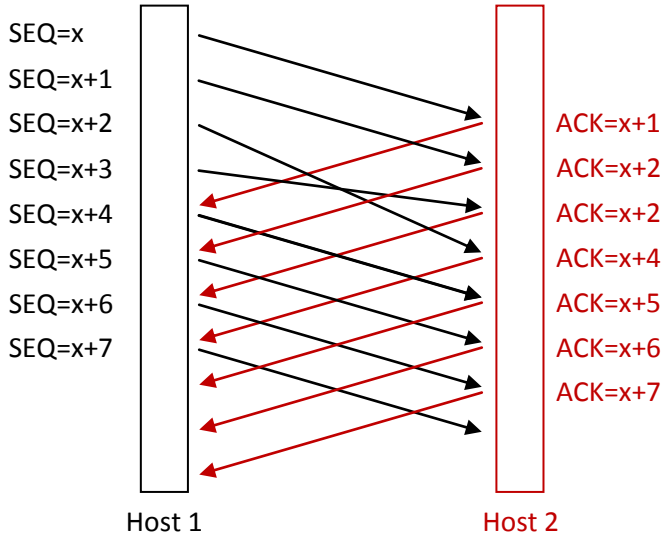
Daca se primesc o succesiune de 4 ACK-uri egale atunci pachetul respectiv se retransmite deoarece este considerat pierdut.

Se observă că deși timerul nu a expirat încă pentru pachetul SEQ=x+1 el este retransmis mai repede. Există posibilitatea ca pachetul să fie retransmis cu mult înaintea timerului și astfel se poate ajunge la întârzieri mult mai mici.



Pentru cazul cu inversare de pachete(pachetele ajung în alta ordine) algoritmul funcționează conform figurii de mai jos.

Se observă că nu există probleme aduse algoritmului în cazul în care 2 pachete sunt inversate. Dacă însă vom avea un pachet întârziat foarte mult,există celu mult riscul ca să fieretransmis, iar la destinație va ajunge de 2 ori. Singura



problemă care se pune este aceea de detecție a pachetului pentru eliminarea sa. [9][12][13]

10.4) Fast recovery algorithms

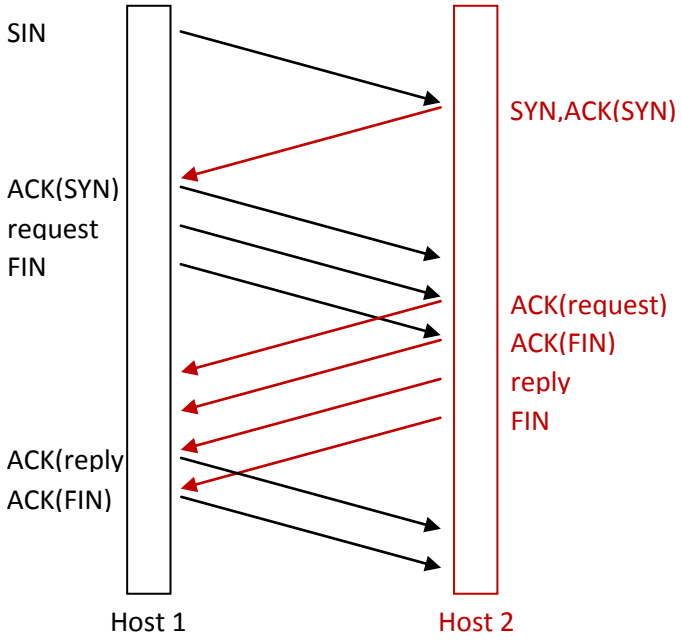
În cazult Fast Recovery Algoritm, după ce fast retransmis cea ce pare a fi un segment pierdut, este aplicat Congestion Avoidante în loc de Slow start. Aceasta este o înbunătățire care permite debite mari cu un nivel de congestie moderat, în special pentru ferestre mari.

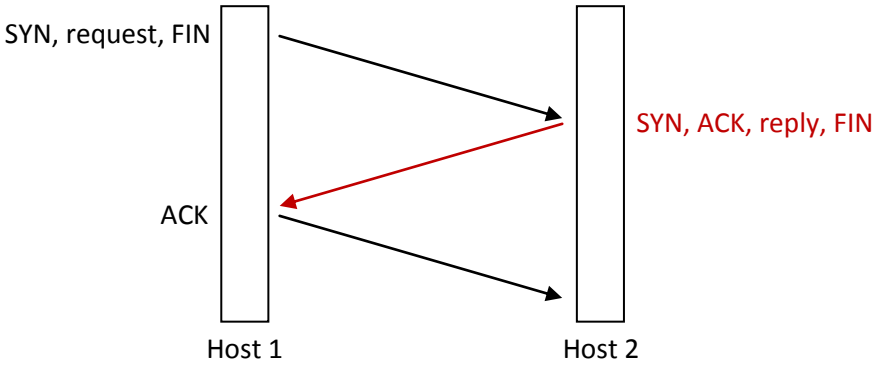
Motivul pentru care nu se aplică Slow start este că recepționarea ACK-urilor duble informează TCP-ul mai mult decât prin faptul că pachetul a fost pierdut. Deoarece receptorul nu poate genera decât ACK-uri duble când un alt segment este recepționat, acel segment a părăsit rețeaua si se găseste în interiorul bufferului receptorului. Există în continuare schimb de date intre cele 2 capete, iar TCP-ul nu vreea să reducă abrupt fluxul de date prin trecerea în Slow start.[7][12]

11) TCP Tranzactional (T/TCP)

Protocolul TCP fiind un protocol orientat pe conexiune presupune realizarea unei conexiuni indiferent de numărul de pachete transmise între cele 2 hosturi. După cum se observă în imaginea următoare pentru transmiterea unui singur pachet de cerere a unei informații (request) și a unia de răspuns (reply) este necesară transmiterea unui număr mare de pachete între cele 2 hosturi care duc la cresterea timpului de transmisie a datelor utile.

În cazul folosirii protocolului TCP Tranzacțional mai multe pacete sunt cumulate într-unul singur astfel fiind necesară transmiterea numai a 3 pachete între hosturi comapartiv cu 11 din varianta inițială.[1]





12) Bibliografie

- [1]Andrew S. Tanenbaum ,Computer Networks
- [2]http://en.wikipedia.org/wiki/Transmission_Control_Protocol
- [3] http://en.wikipedia.org/wiki/File:Tcp_state_diagram_fixed_new.svg
- [4] http://ro.wikipedia.org/wiki/TCP/IP
- [5]http://condor.depaul.edu/jkristof/technotes/tcp.html
- [6]http://www.unibuc.ro/prof/niculae_c_m/telecom/transm_ctrl_prot_tcp.htm
- [7]http://www.cisco.com/web/about/ac123/ac147/ac174/ac195/about_cisco_ipj_archive_article09186a00800c83f8.html
- [8]http://www3.gdin.edu.cn/jpkc/dzxnw/jsjkj/chapter3/35.htm
- [9] http://www.faqs.org/rfcs/rfc2001.html
- [10]http://www.cs.umd.edu/~shankar/417-F01/Slides/chapter3b/sld015.htm
- [11]http://nptel.iitm.ac.in/courses/IIT-MADRAS/Computer_Networks/pdf/Lecture37_TCPConnectionMgmt.pdf
- [12]http://en.wikipedia.org/wiki/Slow-start
- [13] http://www.fcoe.ru/english/data-transfer-protocols-overview/25-fcip

C. Algoritmi de control al congestiei

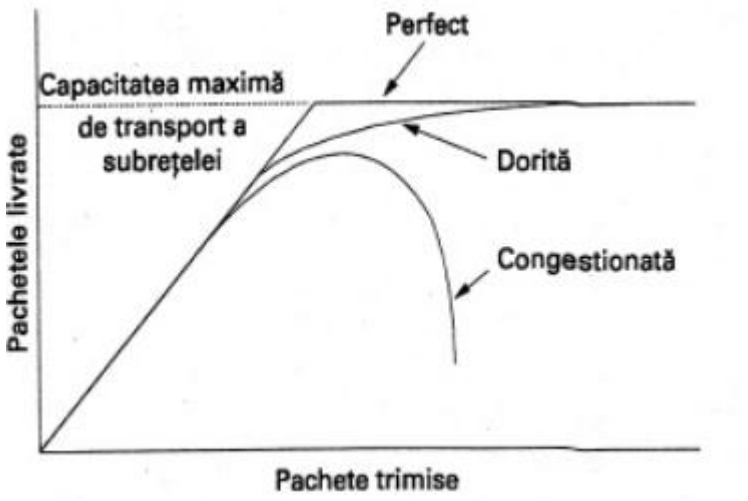
Tică Andra Maria

1. Definirea problemei [5]

Congestia apare in reţelele de calculatoare atunci când incărcarea depăşeşte posibilităţile acesteia de a transfera datele. Incărcarea unei reţele la un moment dat nu depinde numai de capacitatea ei de transmitere ci şi de erorile care apar în mediul de transmisie, de viteza de prelucrare in noduri şi de mecanismele de confirmare folosite de receptor.

Controlul acestui fenomen intră in responsabilitatea nivelelor de transport și de reţea. Apare ca rezultat al traficului intens la nivelul transport, propagându-se la nivelul reţea, performanţele totale ale sistemului degradându-se prin intârzierea și pierderea pachetelor de date.

Această situaţie este prezentată in graficul următor:



[5]

Acesta prezintă cele trei situaţii posibile: transportul optim, trasportul dorit și cazul in care intr-o reţea sunt prezente foarte multe pachete și performanţele acesteia se degradează apărând fenomenul de congestie. Putem spune că atunci când numărul de pachete emise in reţea nu depășește capacitatea de trasport, aceastea sunt livrate integral, excepţie făcând cele care prezintă erori de transmisie. In acest caz numărul pachetelor livrate este proporţional cu numărul celor emise. Atunci când traficul crește prea mult, ruterele incep să nu mai facă faţă și să piardă pachete. Această situaţie se poate deteriora complet la un trafic intens, ducând sistemul in imposibilitatea de a mai livra pachete.

Congestia apare ca rezultat al mai multor factori.

In situaţia in care la sosirea unui număr mare de pachete provenind de pe mai multe linii de intrare intr-o singură linie de ieșire, atunci se va forma o coada. Pentru a

le păstra pe toate, sistemul necesită memorie, iar creșterea acestei capacităţi de memorare duce la inrăutăţirea congestiei și nu la ameliorarea ei.

Un alt factor care cauzează congestia este reprezentat de viteza procesoarelor. Dacă unitatea centrală a ruterelor este lentă in execuţia funcţiilor sale, cozile pot crește. Și liniile cu lăţime de bandă scăzuta pot provoca congestia. Schimbarea liniilor cu unele mai performante și păstrarea aceluiași procesor sau invers ajută puţin, deoarece problema ţine de incompatibilitatea intre părţile sistemului și aceasta va persista până la aducerea la echilibru a tuturor componentelor.

Controlul congestiei trebuie să asigure capabilitatea reţelei de a transporta intreg traficul implicat. Este o problemă globala care implică comportamentul tuturor calculatoarelor gazdă și al ruterelor.

2. Soluţii posibile. Clasificarea Yang&Reddy a algoritmilor.

Prezenţa congestiei inseamnă că incărcarea momentană a sistemului este mai mare decât capacitatea resurselor care pot fi gestionate de sistem la acel moment de timp. Sunt posibile două soluţii dar eficacitatea acesora nu oferă o rezolvare definitivă a acestei probleme: sporirea resurselor sau reducerea incărcării.

Sporirea resurselor constă in creșterea temporară a lăţimii de bandă intre anumite puncte ale reţelei. Dar uneori nu este posibilă creșterea capacităţii de transfer sau aceasta și-a atins deja limitele. In acest moment singura cale de a rezolva congestia este reducerea incărcării prin următoarele metode: interzicerea unor servicii către anumiţi utilizatori, degradarea serviciilor pentru o parte sau pentru toţi utilizatorii sau planificarea cererilor utilizatorilor intr-o manieră mai previzibilă.

Aceasta abordare conduce la impărţirea soluţiilor in două grupe: in buclă deschisă și in buclă inchisă.

Soluţiile in buclă deschisă incearcă să rezolve problema printr-o proiectare atentă, asigurând evitarea apariţiei acesteia. După ce sistemul se pornește și se verifică funcţionarea acestuia, nu se mai fac niciun fel de corecţii. Se iau decizii fără a se ţine cont de starea curentă a reţelei: instrumentele de realizare a controlului in bucla deschisă decid când să se accepte trafic nou, când să se distrugă pachete și care dintre acestea, realizând planificarea deciziilor in diferite puncte din reţea.

Soluţiile in buclă inchisă se bazează pe conceptul de reacţie inversă, această abordare având trei părti:

- Monitorizarea sistemului pentru a detecta când și unde se produce congestia.
- Trimiterea acestor informaţii către locurile unde se pot executa acţiuni.
- Corectarea problemei prin ajustarea funcţionării sistemului.

Se cunosc numeroși algortimi pentru controlul congestiei, iar Yang și Reddy au realizat o clasificare a acestora, studiind in principal cele patru clase de algoritmi centraţi pe host-uri:

- ✓ Algoritmi centraţi pe host-uri
  - Algoritmi in buclă deschisă
    - ➢ Algoritmi care acţionează asupra sursei
    - ➢ Algoritmi care acţionează asupra destinaţiei
  - Algoritmi in buclă inchisă
    - ➢ Algoritmi cu feedback implicit – sursa deduce existenţa congestiei din observaţii locale, cum ar fi timpul necesar pentru intoarcerea confirmărilor.
    - ➢ Agoritmi cu feedback explicit – pachetele sunt trimise inapoi de la punctul unde s-a produs congestia către sursa, pentru a o avertiza.
- ✓ Algoritmi centraţi pe rutere
  - Ruterele inlătura congestia prin renunţarea la pachete.
  - Ruterele semnalează congestia și nu participă activ la inlăturarea ei.

3. Algoritmi de control a congestiei

3.1. Algoritmul RED – Random Early Detection [1] – face parte din clasa algoritmilor de evitare a congestiei centraţi pe rutere, unde ruterele au un rol activ in inlăturarea congestiei prin marcarea pachetelor. Această abordare incearcă să prevină formarea de cozi la intrarea in reţea prin renunţarea la pachete inainte de producerea propriu-zisă a fenomenului de congestie. Congestia incipientă este detectata de gateway-uri prin calculul dimensiunii medii a cozii. Acestea pot anunţa congestionarea unor conexiuni fie prin renuntarea la pachetele care ajung in dreptul gateway-ului respectiv sau prin trimiterea unui bit cu rol de flag in header-ele pachetelor. Când dimensiunea medie a cozii depășește un anumit prag presetat, gateway-urile renunţă sau marchează fiecare pachet care ajunge cu o anumită probabilitate in funcţie de valoarea mediei calculate. In acest mod, algoritmul RED pastrează dimensiunea medie a cozii scăzute dar permite acumularea ocazională in coadă a unui număr mare de pachete.

In reţelele de mare viteză se dorește conceperea de gateway-uri care să suporte cozi de dimensiuni cât mai mari pentru a evita congestionarea acestora. Este din ce in ce mai important să se găsească mecanisme pentru o livrare optimă a

pachetelor din punctul de vedere al intârzierilor, dar in același timp să se păstreze o dimensiune medie a cozilor cât mai scăzută.

Cel mai efectiv mod de detectare al congestiei are loc chiar la gateway-urile reţelei, deoarece numai aici se poate vizualiza comportamentul cozilor in timp. Metoda de monitorizare a dimensiunii medii a cozii la gateway-ul reţelei și de a semnala conexiuni care se află in faza incipientă de congestie este bazată pe presupunerea că va fi util sa avem cozi la gateway-uri unde traficul mai multor conexiuni este multiplexat după algoritmul FIFO. FIFO este util aici la reducerea intârzierii pentru o anumită conexiune atunci când ea e foarte incărcată, prin impărţirea intârzierii totale intre toate conexiunile.

Algoritmul RED este conceput pentru reţelele unde marcarea unui pachet este suficientă pentru a semnala prezenţa congestiei. Acest mecanism de control monitorizează dimensiunea medie pentru fiecare coadă de ieșire și alege intr-un mod aleatoriu ce conxiune să fie notificată de apariţia congestiei. Congestiile tranzitorii sunt eliminate prin creșterea temporară a dimensiunii cozii. Congestiile de lungă durată sunt sesizate prin creșterea dimensiunii medii a cozii, având ca rezultat transmiterea aleatorie de răspunsuri către unele conexiuni pentru a-și micșora ferestrele. Probabilitatea ca o conexiune să fie notificată de apariţia congestiei este proporţională cu proporţia de participare a conexiunii respective la incărcare.

Algoritmul RED calculează dimensiunea medie a cozii folosing un filtru trece-jos de tip EWMA – Exponential/Weighted Moving Average. Dimensiunea medie este comparată cu două praguri, unul de minim și celalalt de maxim. Când media este mai mica decat pragul minim niciun pachet nu e marcat. Când media depășește pragul maxim, toate pachetele care ajung sunt marcate. Dacă media se află intre cele două praguri, fiecare pachet care ajunge este marcat cu probabilitatea pa , unde pa=f(media avg). Astfel, de fiecare dată când un pachet este marcat, probabilitatea ca un pachet să fie marcat ca venind de la o anumită conexiune este proporţională cu proporţia de participare a acelei conexiuni la incărcare.

In cele ce urmează este dat algoritmul general RED:

Pentru fiecare pachet care ajunge

Se calculează dimensiunea medie a cozii avg

Dacă min ≤ avg ≤ max

Se calculează probabilitatea pa

Se marchează pachetul cu probabilitatea pa

Altfel

Dacă max ≤ avg se marchează pachetul.

Dupa cum se vede algortimul RED constă in calcularea dimensiunii medii a cozii și a probabilitaţii de marcare a pachetelor. Prin determinarea mediei se determină gradul de incărcare a reţelei. Probabilitatea de marcare a pachetelor arată frecvenţa cu care pachetele sunt marcate știind că se cunoaște nivelul curent de congestie.

Algoritmul detaliat este următorul:

Iniţializare: avg=0, count=-1

Pentru fiecare pachet care ajunge

Se calculează dimensiunea medie a cozii avg

*Dacă coada nu e goală avg=(1-w)avg+w*q*

Altfel m=f(time-q_time) și avg=(1-w)^{m}avg

Dacă min ≤ avg ≤ max

Cout=cout+1

Se calculează probabilitatea pa : pb=maxp(avg-min)/(max-min)

*pa=pb/(1-count*pb)*

Se marchează pachetul probabilitatea pa și count=0

Altfel

Dacă max ≤ avg se marchează pachetul și count=0

Altfel count=-1

Când coada devine vidă q_time=time.

Variabile salvate: avg, q_time=timpul in care coada e goala, count=numărul de pachete ajunse de la ultimul pachet marcat.

Parametrii ficși: w= „greutatea” cozii, min, max ,maxp=max(pb)

Alţi parametrii: pa, q=dimesiunea cozii curente, time=timpul curent, f(t)=o funcţie liniară de t.

Calculul dimensiunii medii ţine cont de perioada in care coada e goală, estimând numărul m de mici pachete care ar fi putut fi transmise in acest timp. După perioada in care coada a fost goală se calculează dimensiunea medie ca și cum m pachete ar fi ajuns in coada in acea perioadă. Dupa cum avg variază de la min la max probabilitatea de marcare a pachetelor variază liniar de la 0 la maxp: *pb=maxp(avg-min)/(max-min).*

Probabilitatea finală de marcare a pachetelor pa crește incet in timp ce count crește incepând de la marcarea ultimului pachet: *pa=pb/(1-count*pb).* Se asigură astfel diminuarea timpului de așteptare intre marcarea pachetelor.

Există și opţiunea de a masura coada in bytes și nu in pachete. In acest caz dimensiunea medie reflectă intârzierea medie la gateway. Când se folosește această opţiune, algoritmul trebuie modificat pentru a asigura că probabilitatea ca un pachet să fie marcat să fie proporţională cu dimensiunea pachetului in bytes:

pb=maxp(avg-min)/(max-min)

*pb=pb*dim_pachet/dim_max_pachet*

*pa=pb/(1-count*pb)*

„Greutatea" cozii (w) este determinată de dimensiunea și durata incărcărilor bruște ale cozii care sunt admise la gateway. Pragurile min și max sunt determinate de dimensiunea medie dorită a cozii, care depinde la rândul ei de caracteristicile reţelei.

3.1.1 Calcularea lungimii medii a cozii

Se folosește un filtru trece-jos de tip EWMA – Exponential/Weighted Moving Average:

*avg=(1-w)avg+w*q*

„Greutatea" w determină constanta de timp a filtrului trece-jos.

- ✓ **Limita superioara a lui w**

  Dacă w este prea mare, la gateway nu vor putea fi eliminate congestiile cauzate de incărcări tranzitorii.

  Presupunem că iniţial coada este goală, cu o medie 0, lungimea acesteia crescând de la 0 la L pachete. După ce și ultimul L pachet a ajuns in coadă, o să avem:

$$avgL = \sum_{i=1}^{L} iw(1-w)^{L-i} = w(1-w)^{L} \sum_{i=1}^{L} i(\frac{1}{1-w})^{i} = L+1+\frac{(1-w)^{L+1}-1}{w}$$

  Mai sus s-a folosit următoarea identitate:

$$\sum_{i=1}^{L} ix^{i} = \frac{x+(Lx-L-1)x^{L+1}}{(1-x)^{2}}$$

Fiind dat paragul minim min și știind că admitem incărcări de L pachete la gateway, atunci w trebuie să satisfacă urmatoarea inecuaţie:

$$L+1+\frac{(1-w)^{L+1}-1}{w}<\min$$

- ✓ **Limita inferioară a lui w**

  Dacă w are setată o valoare prea mică atunci modificările din coadă se vor sesiza greu, deoarece valorile lui avg vor inregistra fluctuaţii foarte reduse. Astfel stadiile iniţiale de congestie vor fi mai greu de depistat.

  Presupunem că dimensiunea cozii se schimbă de la 0 la un pachet, că odata cu sosirea și plecarea pachetelor dimensiunea ei de un pachet nu se modifică și că dimensiunea medie iniţiala a fost 0. In acest caz este nevoie să ajungă -1/ln(1-w) pachete până când avg ajunge la valoarea 0.63=1-1/e.

- ✓ **Setarea pragurilor de min și max**

  Valorile optime pentru min și max depind de lungimea medie dorită a cozii. Valoarea optimă pentru max depinde mai ales de intârzierea maximă admisă de gateway. Se folosește in general o valoare a lui max astfel incât să fie dublul lui min.

3.1.2 Calcularea probabilitaţii de marcare a pachetelor

Probabilitatea iniţiala de marcare a pachetelor pb este calculată ca funcţie liniară a dimensiunii medii a cozii:

pb=maxp(avg-min)/(max-min)

- ✓ **Metoda 1. Variabile aleatoare geometrice.**

  Fie pb - probabilitatea de marcare a fiecărui pachet și X – numărul de pachete care ajung intre două marcări succesive. X este o variabilă aleatoare geometrică cu parametrul pb și E[X]=1/pb.

  Deoarece fiecare pachet este marcat cu probabilitatea pb avem:

  $$\mathrm{P}(X=n)=(1-pb)^{n-1}pb$$

  Cu o dimensiune medie a cozii constantă, scopul este să marcăm pachete la intervale regulate. Nu este de dorti să avem pachete marcate nici prea des, dar nici să inregistrăm intervale lungi de timp intre marcări succesive. Se realizează astfel o sincronizare globală, anumite conexiuni modificându-și ferestrele in același timp.

- ✓ **Metoda 2. Variabile aleatoare uniforme.**

  In acest caz X este o variabilă aleatoare distribuită uniform in mulţimea {1,2,....1/pb}. Acest lucru se intâmplă dacă probabilitatea de marcare a fiecărui pachet este pb/(1-count*pb), unde count=numărul de pachete nemarcate intre două marcări succesive.

  In acest caz:

$$\mathrm{P}(X=n)=\frac{pb}{1-(n-1)\,pb}\prod_{i=0}^{n-2}\left(1-\frac{pb}{1-i\,pb}\right)=pb\ pentru\ 1\le n\le 1/\ pb$$

$$\mathrm{P}(X=n)=0\ pentru\ n>1/\ pb$$

  Pentru această metoda E[X]=1/(2pb)+1/2.

3.1.3 Implementarea algoritmului RED

Initializare: avg=0 si count=-1

Pentru fiecare pachet care ajunge

Se calculeaza noua dimensiune medie avg

Daca coada nu este goala atunci avg=avg+w(q-avg)

Altfel $avg=(1-w)^{(time-q\_time)s}\,avg$

Daca min≤avg≤max atunci

*count=count+1 si pb=C1*avg-C2*

Daca count>0 si count≥round(R/pb) atunci se marcheaza pachetul care va ajunge si count=0

Daca count=0 atunci R=random[0,1]

Altfel

Daca max≤avg atunci se marcheaza pachetul care va ajunge si count=-1

Altfel count=-1

Cand coada devine vida q_time=time

R – un numar aleator, s – timp de transmisiune tipic.

Concluzie

RED este un mecanism eficient pentru evitarea congestiei la gateway-uri in cooperare cu protocoalele de transport. Este un algoritm relativ simplu care poate fi implementat in reţelele de mare viteză.

Exista incă aspecte de rezolvat in legatură cu acest algoritm: determinarea dimensiunii medii optime a cozii pentru a maximiza numărul de pachete transmise corect și a minimiza intârzierile pentru diverse configuraţii de reţele, implementarea RED cu alte protocoale de transport inafară de TCP.

3.2 Algortimul GAIMD – General Additive Increase Multiplicative Decrease [2] este o strategie de evitare a congestiei prin ajustarea dimensiunii ferestrei. Astfel, in starea de evitare a congestiei dimensiunea ferestrei va fi crescută cu un parametru α, iar la apariţia propriu-zisă a congestiei dimensiunea acesteia va fi micșorata multiplicativ cu β.

Această tehnică a fost pentru prima data considerată de Chiu&Jain. Ei au demonstrat că dacă α și β respectă următoarele relaţii, atunci controlul congestiei GAIMD are stabilitate și se realizează corect: $\alpha>0$ si $0<\beta<1$. Studiul lor a considerat numai cazul când toate fluxurile de transmisiuni din reţea folosesc aceleași valori pentru α și β și nu au studiat efectele acestor parametrii asupra performanţei. Yang&Lam au continuat studiul, explorând in detaliu relaţia dintre performanșa și diferite valori ale celor doi parametrii. In particular, ei au *modelat rata de transmisiune* in funcţie de α, β, a ratei de pierderi, a mediei de timp tur-retur, media timpului de pauză și a numărului de pachete pe care fiecare ACK le confirmă. In continuare au adaptat acest rezultat, găsind o relaţie simplă intre α și β pentru ca fluxul de date să fie de tip TCP-friendly.

3.2.1. Modelarea ratei de transmisiune

O sesiune GAIMD incepe in starea *slowstart* când dimensiunea ferestrei de congestie este dublată pentru fiecare fereastra de pachete confirmate ca primite. La prima indicare a congestiei, fereastra de congestie este tăiata in două și sesiunea intră in starea de evitare a congestiei. In aceasta stare, fereastra de congestie este mărită cu α/W pentru fiecare ACK primit, unde W este dimensiunea ferestrei de congestie curentă. Vom spune astfel că dimensiunea ferestrei va fi crescută cu α la fiecare tur-retur. GAIMD schimbă dimensiunea ferestrei scăzând din aceasta βW când congestia a fost detectată printr-o confirmare de tip triplu-dublu ACK. Dacă detectarea s-a făcut prin detectarea unui timeout, dimensiunea ferestrei va fi setată la valoarea 1.

$W=W+\alpha/W$, $\alpha>0$ – evitarea congestiei

W=W - βW, 0<β<1 – congestie detectată printr-o confirmare de tip triplu-dublu ACK

W=1 - detectarea unui timeout

La modelarea ratei de transmisiune s-au făcut următoarele presupuneri:

- ✓ Sursa mereu are date de transmis.
- ✓ Rata de transmisiune este un proces aleatoriu.
- ✓ Se ignoră impactul etapei de slowstart, punându-se accentul pe mecanismul de avitare a congestiei.
- ✓ Mecanismul GAIMD de evitare a congestiei lucrează cu timpi de tur-retur.
- ✓ Pierderile intr-un tur-retur sunt independente: când un pachet este pierdut inseamnă că toate pachetele care ii urmează in acel tur-retur sunt de asemenea pierdute. Se notează cu p probabilitatea ca un pachet sa fie pierdut.

Formula ratei de transmisiune este:

$$T_{\alpha,\beta}(p, RTT, T_0, b) = \frac{1}{RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}\,p} + T_0 \min(1, 3\sqrt{\frac{(1-\beta^2)b}{2\alpha}\,p})\,p(1+32p^2)}$$

Observăm că numitorul formulei de mai sus este alcătuit din doi termeni pe care ii vom nota astfel:

$$TD_{\alpha,\beta}(p, RTT, b) = RTT\sqrt{\frac{2b(1-\beta)}{\alpha(1+\beta)}\,p}$$

$$TO_{\alpha,\beta}(p, T_0, b) = T_0 \min(1, 3\sqrt{\frac{(1-\beta^2)b}{2\alpha}\,p})\,p(1+32p^2)$$

Numitorul formulei conţine numai TD dacă congestia a fost identificată prin triplu-dublu ACK. Al doilea termen TO este adăugat când congestia poate fi indicată atât prin triplu-dublu ACK cât și prin timeout.

Termenul $Q = \min(1, 3\sqrt{\frac{(1-\beta^2)b}{2\alpha}\,p})$ aproximează probabilitatea de timeout.

3.2.2 TCP-friendly GAIMD

Din formula anterioară se observă că este posibil să controlăm perechile (α, β) astfel incât să obţinem rata de transmisiune dorită. Putem selecta parametrii α, β astfel incât:

$$T_{\alpha,\beta}(p, RTT, T_0, b) = T_{1,\frac{1}{2}}(p, RTT, T_0, b)$$

Aceste perechi (α, β) care satisfac egalitatea de mai sus formează curba TCP-friendly.

- **TD TCP-friendly**

In continuare păstrăm numai primul termen al numitorului, renunţând la variabilele p, RTT și b din ambii membrii ai ecuaţiei:

$$TD_{\alpha,\beta}(p,RTT,b) = TD_{1,\frac{1}{2}}(p,RTT,b)$$

$$\frac{1-\beta}{\alpha*(1+\beta)} = \frac{1-0.5}{1*(1+0.5)}$$

In urma calculelor obţinem:

$$\alpha = \frac{3(1-\beta)}{1+\beta}$$

- **TO TCP-friendly**

Acum păstrăm cel de-al doilea termen al numitorului și renunţăm la variabilele p,To si b din ambii termeni ai ecuaţiei:

$$TO_{\alpha,\beta}(p,T_0,b) = TO_{1,\frac{1}{2}}(p,T_0,b)$$

$$\sqrt{\frac{(1-\beta^2)}{\alpha}} = \sqrt{\frac{(1-0.5^2)}{1}}$$

In urma calculelor obţinem:

$$\alpha = \frac{4(1-\beta^2)}{3}$$

3.3 Algoritmii HSTCP – High Speed TCP si STCP – Scalable TCP [3]

3.3.1 HSTCP – High Speed TCP este un algoritm de tip additive increase multiplicative decrease (AIMD), cu deosebirea că cei doi factori α și β depind de dimensiunea ferestrei.

Considerăm w_i ca fiind dimensiunea ferestrei chiar inaintea unei pierderi și RTTi, timpul de tur-retur, pentru două ferestre i=1,2. Fie t, intervalul dintre două pierderi consecutive.

$$w_1 = w_1(1-\beta(w_1)) + \alpha(w_1)\frac{t}{RTT_1}$$

$$w_2 = w_2(1-\beta(w_2)) + \alpha(w_2)\frac{t}{RTT_2}$$

Substituind $\alpha(w) = \frac{2w^2\beta(w)p(w)}{2-\beta(w)}$ si $w = \frac{0.15}{p(w)^{0.82}}$ obţinem:

$$\frac{w_1}{w_2} = \left(\frac{2-\beta(w_2)}{2-\beta(w_1)}\frac{RTT_2}{RTT_1}\right)^{4.56}$$

3.3.2 STCP – Scalable TCP este un algoritm de tip multiplicative increase multiplicative decrease (MIMD) cu factorii α și β:

$$w_1 = w_1(1-\beta) + (1+\alpha)^{\frac{t}{RTT_1}}$$

$$w_2 = w_2(1-\beta) + (1+\alpha)^{\frac{t}{RTT_2}}$$

3.6 Algortimul binomial BITCP – Binary Increase TCP [3] este un algoritm de control al congestiei care adaptează controlul ferestrei in funcţie de dimensiunea acesteia. Acesta constă in două părţi: căutare prin creștere binară (binary search increase) și creșterea aditivă a dimensiunii ferestrei (additive increase).

3.6.1 Binary search increase

In această etapă privim controlul congestiei ca pe o problemă de căutare in care sistemul poate răspunde prin da sau nu atunci când vrem să stim dacă rata de transmisiune curentă depășește sau nu capacitatea reţelei. Fereastra minimă curentă poate fi estimată ca dimensiunea ferestrei atunci când nu avem pierderi.

Dacă se știe dimensiunea maximă a ferestrei, putem să aplicăm o tehnică de căutare binară pentru a seta dimensiunea ferestrei curente la dimensiunea medie dintre minimul estimat și maximul cunoscut. Dacă această dimensiune dă pierderi, noua fereastră poate fi tratată ca un nou maxim, iar dimensiunea ferestrei după pierderea pachetelor devine noul minim. Media acestor valori devine noul optim.

Acest proces de recalculare a minimului și maximului are loc până când diferenţa dintre dimensiunea maximă și cea minimă scade sub un nivel prestabilit numit incrementare minimă Smin.

3.6.2 Additive increase

Pentru a asigura o convergenţă rapidă a algoritmului și corectitudinea, se combină căutarea binară cu o strategie de creștere aditivă. Când distanţa dintre dimensiunea medie și minimul curent este foarte mare, creșterea dimensiunii ferestrei direct la cea medie este un factor de risc in reţea. Astfel, când distanţa dintre dimensiunea ferestrei curente și cea medie este mai mare decât o valoare presetată, numită increment maxim Smax, in loc să creștem dimensiunea ferestrei direct la acea valoare medie in următorul RTT, o să o creștem cu Smax până când distanţa va deveni mai mica decat Smax, moment in care dimensiunea va fi setata direct la valoarea medie.

3.6.3 Slow start

Atunci când dimensiunea ferestrei curente devine mai mare decat valoarea maximă, nu cunoaștem noul maxim. In acest moment se va aplica tehnica de căutare binară care va seta un maxim egal cu o valoare presetată și dimensiunea ferestrei curente la o valoare minimă. Vom folosi o strategie de tip „slow start” pentru a căuta o valoare a maximului mai mica decât Smax. Dacă considerăm cwnd dimensiunea curenta a ferestrei și incrementul maxim Smax, la fiecare RTT dimensinea ferestrei va crește astfel: cwnd+1, cwnd+2....cwnd+Smax. In acest mod se caută banda disponibilă până când se asigură faptul că o creștere cu Smax este nedăunătoare. După această etapa se va crește dimensiunea la fiecare RTT cu Smax.

3.6.4 Implementare

Se folosesc următorii parametrii presetaţi:

- low_window – acest algoritm este activat in momentul in care dimensiune ferestrei curente depaseste aceasta valoare
- Smax – increment maxim
- Smin – increment minim
- β – factorul de scadere multiplicativa a dimensiunii ferestrei
- default_max_win – maximul presetat

Se folosesc următoarele variabile:

- max_win – dimensiunea maxima a ferestrei care la inceput a valoarea presetata
- min_win – dimensiunea minima a ferestrei
- prev_win – dimensiunea maxima chiar inainte de setarea nouui maxim
- target_win – media dintre minim si maxim

- cwnd – dimeniunea ferestrei de congestie
- is_BITCP_ss – variabila booleana care indica daca suntem sau nu in etapa slow start. Se initializeaza cu false.
- ss_cwnd – o varibila care stocheaza cu cat se creste cwnd curent in etapa slow start
- ss_target – stocheaza fiecare valoare a cwnd dupa fiecare RTT in etapa slow start

Algoritmul BI-TCP:

Când se intră in fast recovery:

```
if(low_window<=cwnd){
        prev_max=max_win;
        max_win=cwnd;
        cwnd=cwnd*(1-β);
        min_win=cwnd;
        if(prev_max>max_win)  max_win=(max_win+min_win)/2;
        target_win=(max_win+min_win)/2;
} else { cwnd=cwnd*0.5; }
```

Cand nu se află in fast recovery și ajunge un ACK pentru un nou pachet:

```
if(low_window>cwnd) { cwnd=cwnd+1/cwnd; return;}
if(is_BITCP_ss is false ){
        if(target_win-cwnd<Smax)   cwnd+=(target_win-cwnd)/cwnd;
        else cwnd+=Smax/cwnd;
        if(max_win>cwnd){
                min_win=cwnd;
                target_win= (max_win+min_win)/2;
                } esle { is_BITCP_ss=true;
                ss_cwnd=1;
                ss_target=cwnd+1;
```

```
                max_win=default_max_win;
                }
} else {
        cwnd+=ss_cwnd/cwnd;
        if(cwnd>=ss_target){
                        ss_cwnd=2*ss_cwnd;
                        ss_target+=ss_cwnd;
}
if(ss_cwnd>=Smax)   is_BITCP_ss=false;
}
```

3.7 Algortimul SIMD – Square Increase Multiplicative Decrease [4] este un mecanism de control al congestiei cu memorie. Folosește micșorarea multiplicativă precum AIMD dar creșterea dimensiunii ferestrei se face cu rădăcina timpului scurs de la detectarea ultimei pierderi. SIMD este TCP-friendly și TCP-compatibil. Are un comportament convergent mult mai bun decat TCP-friendly GAIMD si algoritmul binomial descrise in subcapitolele anterioare.

SIMD folosește pe lângă dimensiunea ferestrei curente și informaţii memorate referitoare la conexiunile reţelei. Această informaţie este reprezentată de dimensiunea ferestrei la momenul detectării ultimei pierderi. Fereastra este micșorată multiplicativ dar creșterea se realizează cu rădăcina timpului scurs de la detectarea ultimei pierderi.

Algortimul SIMD

Creștere: $W = W + \alpha\sqrt{W - W_0}$, α>0 unde W_0 este dimensiunea ferestrei după ultima micșorare a acesteia.

Micșorare: W=W- βW, 0<β<1

Fie w(t) aproximarea continuă a dimensiunii ferestrei la momentul t scurs din momentul inceperii creșterii ferestrei și w=w(0).

Folosind interpolarea liniară și aproximarea continuă in formula de creștere, obţinem:

$\frac{dw(t)}{dt} = \alpha\sqrt{w(t) - w_0}$, de unde rezultă: $\frac{1}{\sqrt{w(t) - w_0}} dw(t) = \alpha dt$.

După integrare se obţine: $2\sqrt{w(t)-w_0} = \alpha t + C$. Pentru t=0 avem w(t)=w(0) de unde rezultă C=0.

Se obţine: $w(t) = w_0 + \frac{\alpha^2}{4} t^2$. Se poate astfel spune că SIMD devine „agresiv" cu timpul adică poate să pună la dispoziţie lăţime de bandă in plus când dispune de aceasta.

SIMD TCP-friendly – definirea lui α atunci când se cunosc β și variabila de stare wmax.

Definim $\alpha = \frac{3\sqrt{\beta}}{(1-2\beta/3)\sqrt{2w\max}}$ și o inlocuim in formula lui w(t): $w(t) = w_0 + \frac{9\beta}{8(1-2\beta/3)^2 w\max} t^2$.

Din ultima formulă putem spune că SIMD poate fi privit ca un AIMD al cărui parametru α variază.

Concluzie

SIMD este un algoritm cu memorie care converge mai rapid decât algoritmii fără memorie AIMD și binomial. De exemplu, dacă există două fluxuri de tip SIMD competitive atunci cel cu dimensiunea ferestrei mai mică este mai „agresiv". Această proprietate duce la obţinerea unui comportament convergent mai bun.

3.9 Bibliografie

[1] „Random Early Detection Gateways for Congestion Avoidance", Sally Floyd, Van Jacobson

[2] „General AIMD Congestion Control", Yang Richard Yang, Simon S. Lam

[3] „Binary Increase Congestion Control for Fast, Long Distance Networks", Lisong Xu, Khaled Harfoush, Injong Rhee

[4] „TCP - friendly SIMD congestion Control and Its Convergence Behavior", Shudong Jin, Liang Guo, Ibrahim Matta, Azer Bestavros

[5] ''Computer networks'', Andrew S. Tanenbaum, David J. Wetherall, Fifth Edition, 2011