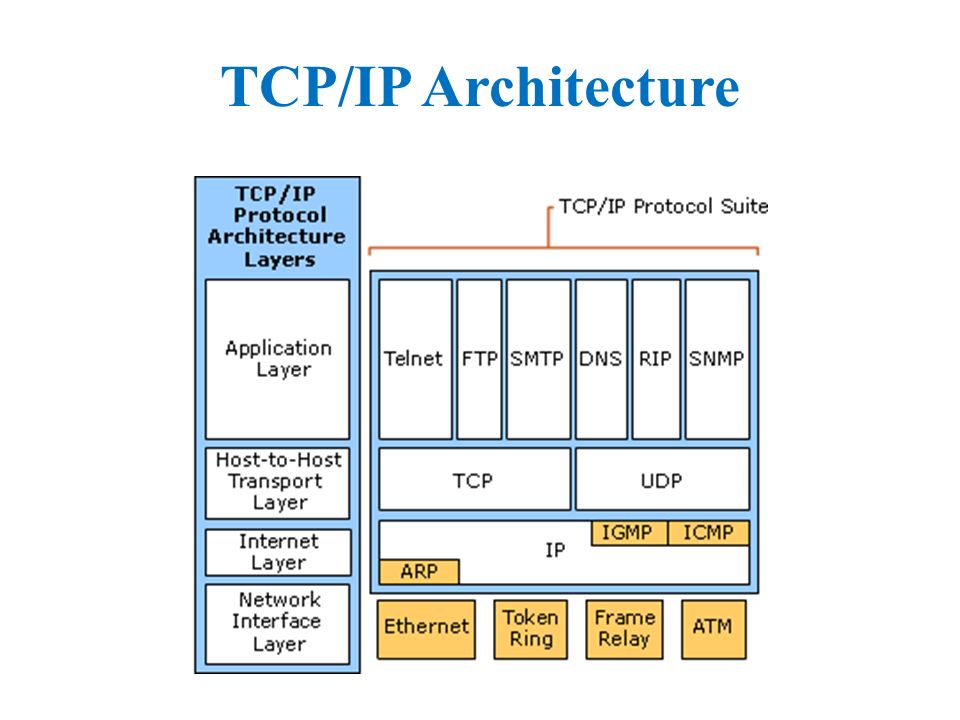
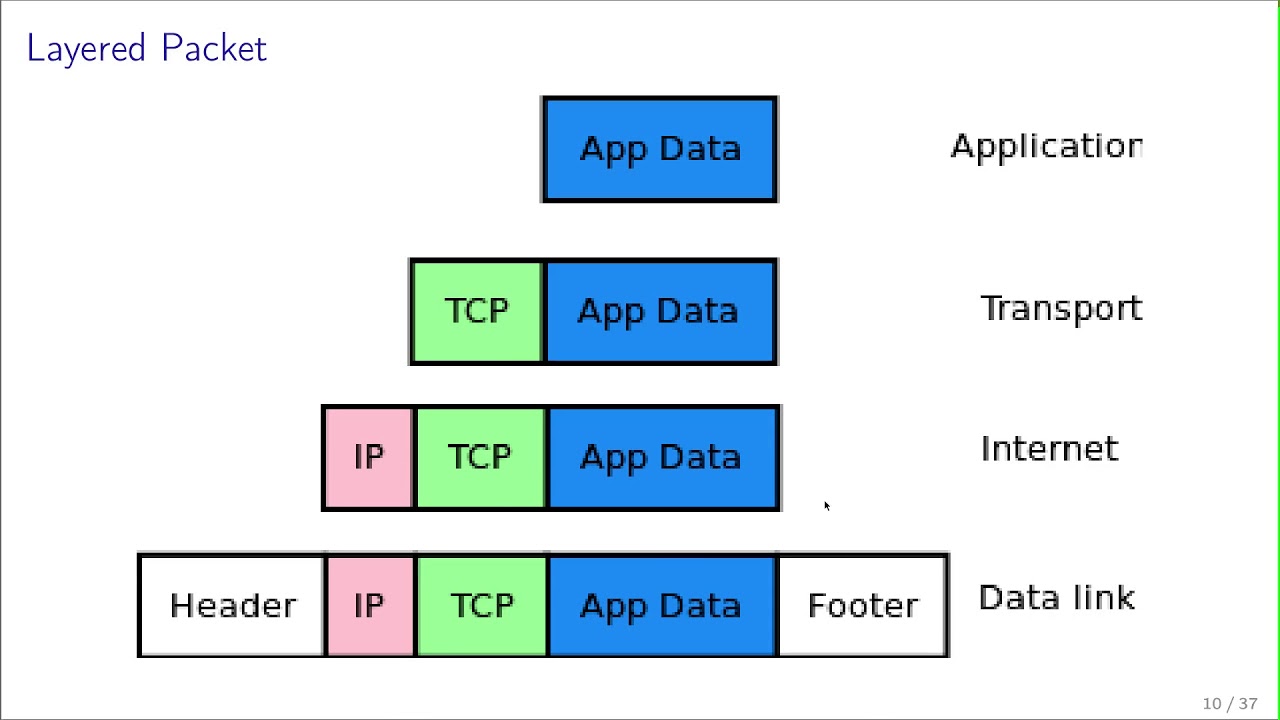
**Transferul fișierelor – Controlul congesiei**

1. **Introducere**

În decursul timpului, Internetul a cunoscut o dezvoltare foarte puternică, ajungând să fie o necesitate primară pentru o mare parte a planetei. Odată cu această evoluție, au fost întâmpinate și diverse probleme soldate cu prăbușirea unor mari părți de rețea, una dintre cauze fiind supraîncărcarea rețelei, rezultat al algoritmilor folosiți în protocoalele de control a transmisiei.   
 Ne vom preocupa de analiza protocoalelor de transmisie (TCP și UDP), de principiul congestiei și vom oferi o soluție a implementării unui mecanism de control a congestiei, prin folosirea protocolului UDP și algoritmul Tahoe de control al congestiei.

1. **TCP. UDP. Controlul congestiilor**

****

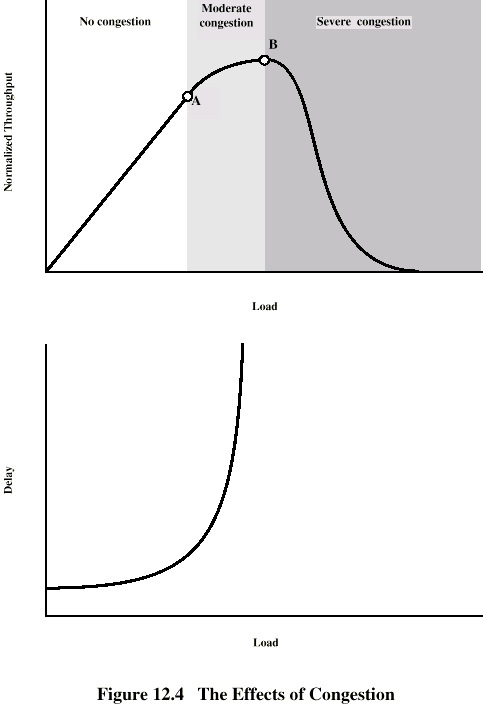
****

TCP este unul dintre protocoalele de bază ale suitei de protocoale Internet și are ca scop asigurarea controlului transferului pachetelor, prin livrarea ordonată a fluxului de octeți de la o sursă la o destinație, în rețea. Pe lângă sarcinile sale de gestionare a traficului, TCP controlează și mărimea segmentului de date, debitul de informație, rata la care se face schimbul de date, precum și evitarea congestionării traficului de rețea.

Astfel, cu siguranță cel mai folosit protocol de control a traficului de date, TCP joacă un rol integral în determinarea performanțelor generale a rețelei.

Modul de operare a protocolului TCP implică existența a trei faze : realizarea conexiunii, transferul de date, închiderea conexiunii. Dintre acestea, vom analiza faza transferului de date.

Transferul de date reprezintă trimiterea datelor de la sursă la destinație, sub formă de pachete. Pachetul este unitatea de informație de bază când ne referim la transferul datelor în rețea și este alcătuit dintr-un header conținând sursa și destinația pachetului, alte secvențe pentru control și datele efective ce trebuie transmise. Premisa de la care plecăm este faptul că destinația este responsabilă pentru controlul rata fluxului de date. Trebuie să ne asigurăm de faptul că rata de transmisie se potrivește cu rata de recepție, rată care trebuie să fie potrivită pentru a asigura o performanță bună, și pentru a nu supraîncărca rețeaua sau destinația. Pentru aceasta, trebuie să ținem cont de controlul congestiilor.

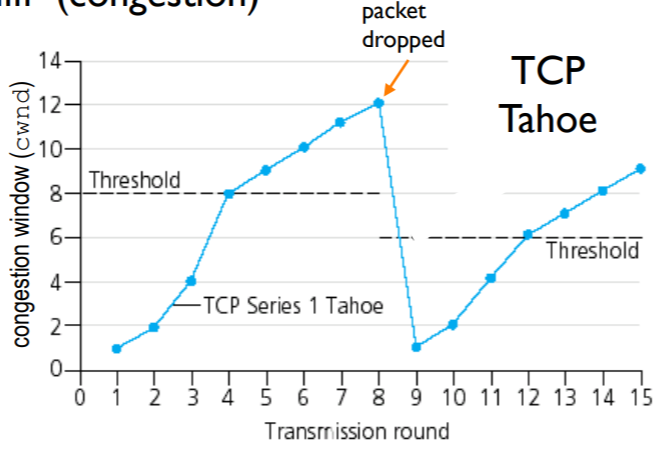
Congestia este un fenomen care apare atunci când pe o rețea se transmit date la capacitatea maximă a rețelei. Acest fenom va produce probleme, cum ar fi pierderea pachetelor sau chiar blocarea rețelei. Există diverși algoritmi implementați în cadrul TCP pentru evitarea congestiei, aceștia bazându-se pe retransmiterea pachetului, ceea ce trebuie foarte atent analizat, deoarece retransmiterea agresivă poate crește congestia, chiar dacă rata de transfer a fost redusă la un nivel la care, în mod normal, nu ar trebui să apară congestie.

Pentru a cunoaște starea conexiunii, emițătorul folosește un timer, astfel încât, pentru fiecare pachet trimis, emițătorul se așteaptă să primească un acknowledgement în perioada de timp impusă, altfel, este generată o eroare, care poate sugera faptul că există congestie. În astfel de situații, TCP trebuie să corecteze această problemă, în primul rând, prin retransmiterea pachetului și să dubleze timpul în care se așteaptă semnal de recepție pentru acest pachet. By default, acest timp este de 1.5 secunde. Se permite încercarea retransmiterii pachetului, până la un maxim de 64 secunde. Totodată, se folosește acest timp și pentru a modifica timpul de așteptare pentru celelalte pachete.

1. **UDP (User Data Protocol)**

* are pachete mai mici header = 8 bytes (versus 20 bytes TCP)
* nu e nevoie de conexiune inainte de transmise (se elimina pasii cu realizarea si inchidrea conexiunii)
* control sporit asupra transmiterii (datele se transmit imediat)
* implementeaza detectarea erorilor (optional), dar sterge pachetele cu erori sau seteaza warning flag
* trimite pachetul, daca se pierde din cauza retransmiterii, nu retransmite
* pachetele pot ajunge in ordine diferita => vom folosi un camp pentru ordine
* fara control al congestiei
* message-oriented -> aplicatii face impartirea in pachete (TCP = stream-oriented isi imparte singur fisierul in pachete)
* voce, data streaming, (uneori blocat de firewall), DNS lookup, aplicatii care tolereaza pierderi pachete VS text, file transfer, aplicatii pentru care acuratetea e importanta

1. **Mecanismul Tahoe de control a congestiei**

* 
* Are la bază mecanismul Slow Start și Fast Retransmit
* Slow Start : Se pornește cu dimensiunea ferestrei de congestie având valoarea 1 și se dublează pe măsură ce sunt primite ACK-uri, până la un threshold stabilit; la acest threshold se miscoreaza rata de transmisie, dimensiunea ferestrei urmand sa se mareasca doar cu cate o unitate, până intervine pierderea unui pachet, moment în care se generează un timeout.
* Fast Retransmit : Când se primește un ACK duplicat (în cazul nostru 3) , se sugerează faptul că pachetul respectiv nu a fost primit iar expeditorul va retrimite acel pachet; totodată se setează un anumit sstresh la jumatate din dimensiunea ferestrei de congestie.
* Se pornește cu Slow Start și se setează un threshold ; Pornind de la dimensiunea ferestrei cu valoarea 1, se dublează până este atins acest threshold, moment în care se continuă transmisia prin mărirea ferestrei doar cu o unitate.
* Când un pachet este pierdut, se reduce dimensiunea ferestrei la 1;
* Dacă pachetul este pierdut din cauza unui timeout (nu se primește ACK), respectivul pachet este retransmis și se reia cu acelasi Slow Start de la 1.
* Dacă pierderea pachetului este semnalată de 3 ACK-uri duplicate, se execută Fast Retransmit, setându-se threshold-ul la jumatate din valoarea ferestrei de congestie din momentul pierderii, și se reia Slow Start de la 1, ținând cont de noul threshold.
* Ideea primirii de ACK-uri duplicate ( de implementat )
* După ce se recepționează un pachet (nr.seq = 1), se trimite ack adaugand 1 la nr seq; astfel daca se transmite pachetul cu nr seq = 1 si se primeste ack = 2 se sugereaza faptul ca pachetul 1 a fost primit si se asteapta pachetul 2. Consideram ca se pierd 1 pachet => se trimite pachetul cu nr. Seq = 3, receptorul primeste aceasta valoare, dar trimite ack pentru pachetul asteptat si nereceptionat = 2.. Se trimite pachetul cu nr seq = 4 , este vazut de receptor, dar se asteapta acelasi pachet 2. In functie de numarul de duplicari setat, se ia decizia retransmiterii. Dupa retransmiterea pachetului pierdut si receptionarea acestuia, se trimite ack corespunzator ultimului pachet primit ( nr seq ultimul pachet + 1).

1. **Proiectarea Aplicatiei**

* Python in PyCharm IDE
* UDP
* algoritm asemanator TCP
* folosim Tahoe

Transmitator:

* citeste fisier la nivel de octet
* sparge fisier in segmente de dimensiune fixa
* adaugare simbol inceput/sfarsit de fisier
* impachetare suplimentara
* ID pachet
* ID fisier
* suma de control
* initializare conexiue UDP
* transmisie prin UDP
* (CONTROLUL CONGESTIEI - TAHOE)

Receptor

* initializare conexiune UDP
* astept sa primesc (intrerupere?) date
* (CONTROLUL CONGESTIEI - TAHOE)
* extragere pachet din buffer
* despachetare
* reconstruiesc fisier la nivel de octet (in functie de ID file, ID packet)

TCP Tahoe

**Idei scurte:**

**-** nivelul transport permite ca mai multe aplicatii sa foloseasca simultan aceeasi retea **-** adresa si portul sunt impachetate la nivel de UDP, nu scriem noi cod in aplicatie pentru asta  
- dimensiunea bufferului este date de intrare la initializare