## *8. Компютърни мрежи и протоколи – OSI модел. Канално ниво. Маршрутизация. IP, TCP, HTTP.*

Ще разгледаме моделът **OSI** (open system interconnection), създаден от международната организация **ISO** (international standard organization) за връзка между отворени системи. Всъщност OSI-моделът е абстрактен модел на мрежова архитектура, който описва предназначението на слоевете, но не се обвързва с конкретен набор от протоколи.
Всеки слой от модела на една машина взаимодейства с едноименния слой на друга машина. Правилата по които се осъществява това взаимодействие се определят от **протокол**, който отговаря на съответния слой. На практика при комуникацията между съответните слоеве на двете машини не се предават данни. Всеки слой предава данни и контролна информация на непосредствено по-долния слой, докато се достигне най-долния слой. Под него е физическата среда за предаване, където се осъществява реалната комуникация между машините. В приемника получените данни се разпространяват в обратна посока - от най-долния слой нагоре, като всеки слой премахва контролната информация, която се отнася до него. Всеки слой предоставя **интерфейс** на непосредствено по-горния слой, който определя какви функции и услуги му се предоставят. В OSI-модела има седем слоя – физически, канален, мрежов, транспортен, сесиен, представителен, приложен – изброени са в последователност от най-долния към най-горния слой.
**Физическият слой** има за задача да реализира предаването на битове през физическата среда. Основна функция на физическия слой е да управлява кодирането и декодирането на сигналите, представящи двоичните цифри 0 и 1. Този слой не се интересува от предназначението на битовете. Физическият слой трябва да осигурява възможност на по-горния канален слой да активизира, поддържа и прекратява физическите съединения.
Основна функция на **каналният слой** е откриването и евентуалното коригиране на грешки при предаването на данните. Данните на канално ниво се обменят на порции, наречени **кадри** (обикновено с дължина от няколко стотин до няколко хиляди байта). При надеждна комуникация приемникът трябва да уведомява изпращача за всеки успешно получен кадър като му

изпраща обратно потвърждаващ кадър. Функциите на каналния слой обикновено се реализират смесено - апаратно и програмно.
**Мрежовият слой** отговаря за функционирането на комуникационната подмрежа. Приложните програми, които се изпълняват в двете крайни системи взаимодействат помежду си посредством **пакети** от данни. Основна задача на мрежовия слой е маршрутизирането на тези пакети. Пакетите са с фиксирана големина в рамките на една мрежа. За системите, реализиращи възлите на комуникационната подмрежа този слой е последен. Функциите на мрежовия слой, както и на по-горните слоеве се реализират програмно.
**Транспортният слой** осигурява транспортирането на произволно дълги съобщения от източника до получателя. Той е най-ниският слой, който реализира връзка от тип "край-край" между комуникиращите системи. В транспортния слой на изпращача съобщенията се разбиват на пакети и се подават на мрежовия слой, а в транспортния слой на получателя подадените от мрежовия слой пакети се реасемблират. Транспортният слой освобождава по-горния сесиен слой от грижата за надеждното и ефективно транспортиране на данните между крайните системи.
**Сесийният слой** е отговорен за диалога между две комуникиращи програми. Съобщения се обменят след като двата крайни абоната установят **сесия**. Сесийният слой осигурява различни режими на диалог – двупосочен едновременен диалог, двупосочен алтернативен диалог, еднопосочен диалог. Освен това той предоставя възможност за прекъсване на диалога и последващо възстановяване от мястото на прекъсването. При липсата на сесиен слой всяко съобщение се предава независимо от другите съобщения.
**Представителният слой** е най-ниският слой, който разглежда значението на предаваната информация.
Първата функция на този слой е да определи общ синтаксис за предаване на съобщенията. Втората функция на слоя е да унифицира вътрешната структура на представените данни в съобщенията. По този начин за по-горния приложен слой няма значение дали двете крайни системи използват различни представяния на данните.
**Приложният слой** е най-горният слой, към който се свързват потребителските процеси в двата крайни абоната. Някои потребителски процеси са интерактивни - взаимодействат си в голям период от време с кратки съобщения от тип заявка-отговор. Други потребителски процеси взаимодействат с малко на брой големи по обем порции от данни. За двата вида процеси се предвиждат различни протоколи на приложния слой.

Каналното ниво има три основни функции - да осигури подходящ интерфейс на по-горното мрежово ниво, да открива грешки по време на предаването и да управлява информационният обмен.

Данните за каналното ниво представляват последователност от **кадри** (frame). Каналното ниво взима пакетите, които му се подават от мрежовото ниво и ги затваря в кадри. Всеки кадър се състои от заглавна част (header), поле за данни, което съдържа пакета и опашка (trailer). Дължината на кадъра обикновено е ограничена отгоре. Физическото ниво възприема информацията от каналното ниво като поток от битове, без да се интересува от нейната структура. Получателят идентифицира в потока от битове кадрите и въз основа на служебната информация в тях ги контролира за грешки. За целта опашката на кадъра съдържа контролна сума (обикновено 2 байта), която се изчислява върху останалата част от кадъра преди той да бъде предаден. Когато кадърът пристигне в получателя, контролната сума се преизчислява и ако тя е различна от предадената контролна сума, то получателят отхвърля кадъра и евентуално изпраща съобщение за грешка към източника. Ако контролните суми съвпаднат, то получателят приема кадъра и изпраща потвъждаващ кадър до източника. След това се премахва служебната информация на кадъра и информационният поток се предава на мрежовото ниво вече под формата на пакети.

Разглеждаме ситуация при която машината *A* изпраща кадри към машината *B* по канал с шум. Кадрите могат да се изкривят по време на предаването или изцяло да се изгубят.
Когато в *B* постъпи нов кадър от *A*, *B* изчислява контролната сума на кадъра. Ако тази контролна съвпадне с изпратената контролна сума, *B* изпраща данните на неговото мрежовото ниво, формира потвърждаващ кадър и го изпраща към *A*. Ако контролната сума не съвпадне, то кадърът е сгрешен и се отхвърля от *B* и *B* не изпраща потвърждение или изпраща негативно потвърждение обратно към *A*.
Възможно е *A* да изпрати кадър към *B*, но този кадър да се изгуби. Тогава *B* не може да реагира, тъй като не е регистрирал грешка. За да се избегне тази ситуация, *A* стартира брояч на време (**time-out**) с изпращането на всеки кадър. Времето, което отчита брояча трябва да е по-голямо от времето за предаване на кадъра, обработката му в приемника и получаване на потвърждение.
Ако кадърът не се потвърди в рамките на това време, то *A* предава кадърът отново. Тук се обхваща случая в който *A* получава от *B* негативно потвърждение (това е равносилно с изтичане на времето за потвърждение).
Възможно е *A* да изпрати кадър към *B*, този кадър да се получи в *B*, но потвърждението да се изгуби. В този случай *A* не знае дали изпратеният кадър въобще е пристигнал до *B*.
При всички положения *A* изпраща наново кадъра и ако не се вземат мерки, *B* ще получи същия кадър и ще го изпрати към мрежовото ниво, което ще доведе до недопустимо дублиране на данните. За целта с всеки кадър се свързва пореден номер.

В случая е достатъчно номерът да е един бит (0 или 1). Във всеки един момент *B* очаква кадър с определен номер. Ако *B* получи кадър с друг номер, този кадър е дубликат и се отхвърля. Ако *B* получи кадър с очаквания номер, кадърът се приема и очакваният номер на кадър се инвертира (ако е бил 0 става 1, ако е бил 1 става 0). От своя страна *A* номерира алтернативно кадрите, които изпраща към *B*. Естествено, ако даден кадър бъде изпратен отново неговият номер не се променя.

Едно подобрение на разгледания подход е следния – тъй като кадрите текат и в двете посоки, потвържденията може да се закачат за кадрите с данни (за целта се използва поле в заглавието на кадъра). Проблемът е, че данните от мрежовото ниво, към които трябва да се прикрепи потвърждението може да се забавят прекалено дълго и броячът на време да изтече, което ще доведе до повтаряне на кадъра. Обикновено решението е следното - изчаква се фиксиран брой милисекунди и ако дотогава не пристигне пакет от мрежовото ниво се изпраща самостоятелен потвърждаващ кадър.
Ще разгледаме някои **протоколи с прозорци**. Те са по-ефективни от горния протокол спри и чакай, тъй като позволяват изпращане на повече от един кадъра преди да се чака за потвърждение.
При тези протоколи всеки кадър се номерира с число от 0 до някакъв максимум, обикновено от вида
$2^n$ - 1, така че номерът да се вмества точно в $n$ бита.
Във всеки един момент предавателят поддържа множество от поредни номера на кадри, които попадат в **прозореца на предавателя**. От друга страна, получателят поддържа **прозорец на получателя**.
Поредните номера в рамките на прозореца на предавателя съответстват на кадри, които вече са били изпратени и чакат потвърждение. Когато от мрежовото ниво на предавателя пристигне нов пакет, той разширява прозореца откъм горната му граница. Когато в предавателя пристигне потвърждение, долната граница на прозореца се придвижва напред. Предимството е, че с едно потвърждение могат да се потвърдят повече от един последователно номерирани кадри. Тъй като кадрите в прозореца на предавателя могат да се изкривят или изгубят, те трябва да се съхраняват за евентуалното им повторно изпращане. Така предавателят трябва да разполага с достатъчен брой буфери.
Ще отбележим една особеност – прозорецът на предавателя никога не трябва да е изцяло запълнен (т.е. да съдържа всички номера), тъй като потвърждаването на изцяло запълнен прозорец не може да се интерпретира еднозначно – то може да означава както, че всички кадри са били приети, така и че всички кадри са били отхвърлени.
Номерата на кадрите в прозореца на получателя съответстват на кадри, които могат да бъдат получени. Когато в получателя пристигне кадър, чийто номер съвпада с долната граница на

неговия прозорец, данните от този кадър се предават към мрежовия слой на получателя и прозорецът се завърта напред, т.е. придвижват се и горната и долната му граница. Ако номерът на пристигналия кадър попада в прозореца, но не съвпада с долната му граница, този кадър не се отхвърля, а се буферира.
За разлика от прозореца на предавателя, прозорецът на получателя има фиксиран размер. За да поддържа прозорец на предавателя трябват буфери и битова карта, която показва кои буфери са запълнени.
При наличие на сгрешен или изгубен кадър, предавателят ще продължи да предава кадри, преди да разбере че има проблем. Въпросът е какво да прави получателят с успешно получените кадри след сгрешен или изгубен кадър.
Едната стратегия (**go back n**) е тези кадри да се отхвърлят. Тя съответства на прозорец на получателя с размер 1. С други думи, получателят приема единствено следващия поред кадър, който трябва да се предаде към мрежовия слой. В даден момент броячът на време на предавателя ще изтече и той ще изпрати наново всички кадри, започвайки от сгрешения (изгубения).
Другата стратегия (**selective repeat**) е получателят да буферира успешно получените кадри след сгрешен или изгубен кадър. Когато броячът на време в предавателя изтече, той изпраща наново само най-стария сгрешен (изгубен) кадър. Ако повторното изпращане е успешно, получателят може последователно да изпрати към своя мрежов слой кадрите, които е буферирал. Обикновено при тази стратегия получателят изпраща служебен кадър, който известява на предавателя за сгрешен или изгубен кадър - това води до по-бързо повторно предаване на съответния кадър. Стратегията съответства на размер на прозореца на получателя по-голям от 1. Всеки успешно получен кадър, чийто номер попада в прозореца на получателя се буферира и се изпраща към мрежовия слой чак след като са изпратени предшестващите го в прозореца кадри.

Мрежите с общодостъпно предаване се характеризират с общ комуникационен канал, който се споделя от всички машини, включени в мрежата. Всеки изпратен кадър минава през общия канал и достига до всички машини в мрежата. Адресно поле в кадъра посочва за кой е предназначен този кадър. Когато една машина получи кадър, тя проверява дали той е предназначен за нея. Ако това е така, кадърът се приема и обработва, в противен случай се отхвърля. Общодостъпни многоточкови канали се използват най-вече при локалните мрежи.
Най-разпространената локална мрежа е **Ethernet**. Един персонален компютър се свързва в Ethernet мрежа с помощта на NIC (Network Interface Card) - това е каналната станция, която осъществява обмена по Ethernet канала.
Преди да изпрати кадър, каналната станция проверява състоянието на канала. Ако той е свободен, тя веднага започва предаване. Ако каналът не е свободен (т.е. предава друга станция),

то станцията изчаква неговото освобождаване. След като започне предаването, каналната станция продължава да подслушва канала. Ако се открие изкривяване на предавания сигнал, това означава, че по същото време е започнала да предава друга станция и е настъпила **колизия**. В този случай двете станции спират предаването и всяка от тях изчаква случаен интервал от време преди да предава отново.
Кадрите в Ethernet имат максимална дължина 1500 байта. Когато една предаваща станция разбере за конфликт, тя веднага спира предаването като орязва настоящия кадър. За да може да се прави разлика между валидни и орязани кадри, дължината на кадъра трябва да е поне толкова голяма, че да може предаването да не е завършило, преди станцията да разбере за конфликта. Затова кадрите имат минимална дължина 64 байта.
Адресите са по шест байта. Адрес на получател, състоящ се само от 1 е предназначен за всички станции.
В началото в Ethernet се използва коаксиален кабел и скоростта на предаването е достигала 10 Mb/s. По-нататък се въвежда използването на **хъбове** (hub) и скоростта на предаване скача десетократно – до 100 Mb/s. Каналните станции се свързват към хъба чрез две медни усукани двойки. По една от усуканите двойки се предава към хъба, а по другата се приема от него. Ако хъбът получи кадър по някоя линия, той изпраща този кадър по всички останали линии. Важно е да се отбележи, че хъбът работи на физическо ниво и не знае адресите на каналните станции.
Алтернатива на хъбовете са по-интелигентни устройства, които работят вече на канално ниво – **мостовете** и **превключвателите**, които не предават кадрите по всички възможни линии. Те имат информация (във формата на таблици), чрез която въз основа на адреса на кадъра определят по коя изходна линия да се изпрати този кадър.

Основната функция на мрежовото ниво е да маршрутизира пакетите от източника към получателя. В повечето мрежи пакетите ще изминат това разстояние за няколко хопа.
**Маршрутен алгоритъм** е част от софтуера на мрежовото ниво, която определя по коя от изходните линии да се изпрати пристигнал пакет. За целта всеки маршрутизатор притежава **маршрутна таблица**.

Маршрутизиращите алгоритми са два вида - **неадаптивни** и **адаптивни**. При неадаптивните алгоритми маршрутизацията не се извършва на базата на текущата топология на мрежата. Маршрутите между всеки два възела в мрежата се изчисляват предварително и маршрутните таблици се попълват ръчно от мрежовите администратори. При промяна на топологията на мрежата (например при отпадане на възел или на връзка), администраторите ръчно трябва да променят маршрутните таблици, така че всеки два възела да останат свързани.

Това прави неадаптивните алгоритми приложими само в малки мрежи, при които рядко настъпват промени.
Неадаптивните алгоритми се наричат още **статични**.
Една маршрутна таблица в един възел съдържа по един ред за всяко възможно местоназначение. При статична маршрутизация, един ред съдържа толкова полета, колкото са непосредствените съседи на конкретния възел. За всеки ред, с всеки съсед се свързва едно тегло между 0 и 1, така че сумата от теглата на всеки ред да е точно 1. Когато във възела пристигне пакет, първо се определя кое е местоназначението, след това се генерира едно случайно число и в зависимост от теглата в реда на съответното местоназначение се определя към кой съсед да се изпрати пакета. Така колкото е по-голямо теглото на един съсед, толкова повече пакети ще се изпращат към него.

При адаптивните алгоритми маршрутните таблици се променят динамично за да отразяват промени в топологията и натовареността на трафика. Важна характеристика на един адаптивен алгоритъм е неговата **скорост на сходимост** - тя се определя от времето, което е необходимо да се преизчислят маршрутните таблици на всички маршрутизатори в мрежата при промяна в топологията или трафика.

При **централизираните** адаптивни алгоритми в мрежата се създава един маршрутен управляващ център. Той изчислява маршрутните таблици на всички възли и им ги изпраща. За да се адаптират маршрутните таблици към текущата топология и текущия трафик, всички възли трябва да изпращат информация към маршрутния център. На базата на получените сведения, маршрутният център изчислява теглата на ребрата и след това пресмята оптималният маршрут между всеки два възела. Добре е да се поддържат алтернативни пътища между възлите.
Информацията от по-близките до маршрутния център възли ще пристигне по-бързо отколкото от по-далечните. Поради тази причина, периодът на обновяване на маршрутните таблици трябва да е поне два пъти по-голям от времето за преминаване на пакет от маршрутния център до най-отдалечения от него възел.
Преизчислена маршрутна таблица, получена в един възел не трябва да се използва веднага, тъй като маршрутните таблици пристигат по различно време в различните възли.
Ако по някаква причина маршрутният център отпадне, мрежата остава без управление. За целта може да се дублира маршрутният център, но тогава служебният трафик би се увеличил твърде много.
При маршрутизацията с **вектор на разстоянието** всеки маршрутизатор изгражда и поддържа маршрутна таблица, в която всеки ред съдържа адрес на дадено местоназначение, адрес на следващата стъпка към това местоназначение по най-добрия известен до момента път и дължината на този път.

Маршрутизаторите разменят на фиксиран брой милисекунди съдържанието на маршрутните си таблици само с директно свързаните към тях съседни маршрутизатори.
Предполага се, че всеки маршрутизатор знае метриката на връзките до своите съседи. Да предположим, че за метрика е избрано време-закъснението на пакетите.
Нека даден маршрутизатор $J$ получи маршрутната таблица на съседа си $X$, като $X_i$ е обявеното от $X$ закъснение до маршрутизаторът $i$. Ако закъснението от $J$ до $X$ е $m$, то от $J$ до всеки маршрутизатор $i$ има път през $X$ със закъснение $X_i + m$.
Възможни са четири случая:

- ако в маршрутната таблица на $J$ няма ред за направлението $i$, то $J$ добавя такъв ред и записва в него следваща стъпка $X$ и закъснение $X_i + m$;
- ако в маршрутната таблица на $J$ има ред за направлението $i$ и в него е записана следваща стъпка $X$, то стойността на закъснението се актуализира с $X_i + m$ независимо дали тя е по-голяма или по-малка от предходната стойност;
- ако в маршрутната таблица на $J$ има ред за направлението $i$, в него е записана следваща стъпка различна от $X$ и закъснение по-голямо от $X_i + m$, то редът се актуализира като за следваща стъпка се записва $X$, а за закъснение $X_i + m$;
- ако в маршрутната таблица на $J$ има ред за направлението $i$, в него е записана следваща стъпка различна от $X$ и закъснение по-малко или равно на $X_i + m$, то редът не се променя.

Сериозен недостатък на маршрутизиращите алгоритми с вектор на разстоянието е ниската им скорост на сходимост. Добрите новини се разпространяват бързо в мрежата, но лошите новини обикновено изискват твърде голям брой периодични съобщения за да достигнат до всички маршрутизатори. Този проблем се нарича **броене до безкрайност**. За него съществуват някои частични, но не изчерпателни решения (като например разделяне на хоризонта).

При маршрутизирането със **следене състоянието на връзката**, всеки маршрутизатор трябва да извършва следните пет основни действия:

1. Откриване на съседните маршрутизатори и техните мрежови адреси.
2. Измерване на цените на връзките до съседните маршрутизатори.
3. Конструиране на пакети с информация за състоянието на връзките.
4. Изпращане на тези пакети до всички останали маршрутизатори.
5. Изчисляване на най-късия път до всеки маршрутизатор в мрежата.

След включването на един маршрутизатор неговата първа задача е да научи кои са съседите му. Това се постига чрез изпращане на “ехо” пакет по всяка от изходящите линии на маршрутизатора. От своя страна, всеки от съседите отговаря като съобщава името си. Това име трябва да бъде уникално в мрежата.

Всеки маршрутизатор трябва да може да определи време-закъснението до своите съседи. Най-простият начин е маршрутизаторът да изпрати "ехо" пакет към всеки свой съсед на който трябва директно да се отговори. Времето от изпращането на "ехо" пакета до получаване на отговора се дели на две и по този начин се получава времето-закъснение до съответния съсед.
Друг въпрос е дали при измерването да се взима предвид натовареността на възлите. Разликата се постига в зависимост от това кога маршрутизаторът стартира измерването - когато пакетът постъпва в съответната изходяща опашка или когато пакетът се придвижи в началото на опашката.

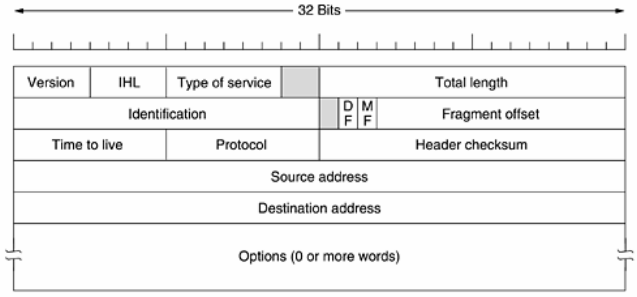
След като събере необходимата информация за състоянието на връзките си, следващата задача на маршрутизатора е да конструира пакет, който съдържа тази информация.
Пакетът трябва да съдържа уникалното име на подателя, пореден номер, срок на годност и списък със съседите на подателя, като за всеки съсед е указана цената на връзката до него.
Определянето на момента, в който трябва да бъдат подготвени и изпратени пакетите е важна задача.
Един възможен начин е това да става през определени равни интервали от време. Друга по-добра възможност е пакетите да се подготвят и изпращат само при промяна в топологията на мрежата - след отпадане или поява на нов съсед или промяна в цената на някоя връзка.

Най-съществената част на алгоритъма е надеждното доставяне на пакетите с информацията за състоянието на връзката до всички маршрутизатори.
За разпространението на пакетите се използва методът на наводняването (flooding). При него всеки пакет се изпраща по всички линии, освен линията по която е пристигнал.
Обработката на всеки пристигнал пакет започва с проверка дали пакетът има по-голям пореден номер в сравнение с най-големия пореден номер, който е пристигнал до този момент от този източник. Ако номерът е по-голям, информацията от пакета се записва в таблицата с информация за състояние на връзките и пакетът се предава по останалите линии. Ако номерът е по-малък или равен, пакетът се отхвърля.
Този алгоритъм има някои проблеми. Ако поредният номер не е достатъчно голям, той може да се превърти. Затова се използват 32-битови поредни номера. В полето за срок на годност маршрутизаторът-подател указва продължителността на

интервала от време в секунди, през който пренасяната от него информация трябва да се счита за валидна. Всеки маршрутизатор, който получи даден пакет намалява с единица стойността на това поле преди да го предаде към своите съседи. Освен това, след като маршрутизаторът запише данните от пакета в своята таблица, той продължава да намалява срока на годност на тези данни на всяка следваща секунда. Ако срока на годност стане 0, данните се изтриват.

След като един маршрутизатор получи пълна информация за състоянието на връзките на всички останали маршрутизатори, той може да приложи алгоритъма на Дейкстра.
Изчислените маршрути се записват в маршрутните таблици.
Големите по размер мрежи изискват използване на маршрутизатори с голям обем памет.

От гледна точка на мрежовото ниво Internet е съвкупност от **автономни системи**. Всяка автономна система се състои от една или повече мрежи. В рамките на една автономна система има фиксирани правила за предаване и фиксиран размер на пакета. Internet всъщност изгражда правила за връзка между отделните автономни системи. За целта се използва протоколът **IP**. Между автономните системи данните се придвижват под формата на **дейтаграми**. Задачата на IP е да извърши успешно предаване на дейтаграмите от източника до получателя без значение дали те са в една и съща мрежа или в различни мрежи. Всяка дейтаграма се изпраща самостоятелно, като по пътя тя може да се фрагментира на по-малки единици. Когато фрагментите достигнат до получателя той ги реасемблира за да получи оригиналната дейтаграма.
Ще разгледаме формата на IP-дейтаграмата във версия 4 (4-байтови адреси). IP-дейтаграмата се състои от заглавна част и част за данни. Заглавната част е 20B+опции с променлива дължина и има следния формат:




Полето Version указва версията на протокола, към който принадлежи дейтаграмата.
Полето IHL указва дължината на заглавната част в 32-битови думи. То е необходимо, тъй като полето Options има променлива дължина.
Полето Type of service показва какво обслужване очаква дейтаграмата. Различните видове данни, например

видеоизображение, глас, файлове предполагат различно обслужване. Практически сегашните маршрутизатори не обръщат внимание на това поле.
Полето Total length съдържа общата дължина на дейтаграмата (заглавна част + данни).
Полето Identification съдържа номер на дейтаграмата. Всички фрагменти на една и съща дейтаграма имат еднакъв номер и по този начин получателя разбира кой фрагмент към коя дейтаграма принадлежи.
Флагът DF (don't fragment) указва на маршрутизаторите да не фрагментират дейтаграмата. Всички автономни системи трябва да могат да приемат фрагменти от поне 576B. Ако размерът на фрагментите е по-голям и флагът DF е 1, то дейтаграмата може да пропусне някоя автономна система с по-малка дължина на пакета, дори тя да се намира на оптималния маршрут.
Флагът MF (more fragments) за всички фрагменти на дейтаграмата, освен последния е 1, а за последния е 0, т.е. той показва дали получен фрагмент е последен в дейтаграмата или не.
Полето Fragment offset указва къде се намира фрагмента в оригиналната дейтаграма.
Полето Time to live е брояч, който ограничава продължителността на живота на дейтаграмата. Това поле се намаля с единица на всеки hop, а освен това се намаля с единица и за всяка секунда престой в маршрутизатор. Когато полето стане 0, дейтаграмата се премахва и в обратна посока се изпраща предупредителен пакет.
Полето Protocol указва протокола на транспортно ниво, към който трябва да се предаде дейтаграмата. Той може да бъде TCP, UDP или някой друг.
Полето Header checksum е контролна сума само на заглавната част. Тя трябва да се преизчислява на всеки hop, тъй като поне едно поле се променя - Time to live.
Полетата Source Address и Destination address съдържат съответно адрес на източника и адрес на получателя.
Възможни са различни опции в полето Options. Най-често се използват опции, които налагат ограничения върху пътя, който трябва да измине дейтаграмата и опции, в които се записва пътя по който е минала дейтаграмата – те се използват с цел по-лесно да се проследяват грешки при маршрутизирането.
Всеки хост и маршрутизатор в мрежата има IP-адрес.
Всички IP-адреси са 32-битови. Всеки IP-адрес се дели на две части – номер на мрежа и номер на хост. Номерът на мрежата е непрекъсната порция от битове в лявата част на адреса, а номерът на хоста е останалата непрекъсната порция от битове в дясната част на адреса.
В зависимост от структурата си IP-адресите се делят на пет класа. Класовете се различават по първите няколко бита на адреса, които се наричат **сигнални битове**.
За клас A сигналният бит е 0, номерът на мрежата е 7 бита, така че в него са възможни приблизително 120 мрежи, номерът на

хоста е 24 бита, така че всяка мрежа има приблизително 16000000 хоста.
За клас B сигналните битове са 10, номерът на мрежата е 14 бита, така че в него са възможни приблизително 16000 мрежи, номерът на хоста е 16 бита, така че всяка мрежа е с приблизително 65000 хоста.
За клас C сигналните битове са 110, номерът на мрежата е 21 бита, така че са възможни приблизително 2000000 мрежи, номерът на хоста е 8 бита, така че всяка мрежа е с приблизително 250 хоста.
Клас D, със сигнални битове 1110 е предназначен за работа с групови адреси, а клас E със сигнални битове 1111 е резервиран за бъдеща употреба.

Големият недостатък на IP-адресацията е, че половината адреси са от клас A и се разпределят само между малко повече от 120 автономни системи, въпреки че всяка от тях може да съдържа милиони хостове.
Всяка мрежа трябва да има уникален номер и всички хостове в дадена мрежа трябва да имат един и същ номер на мрежата. Това води до проблеми при нарастване на броя на мрежите. Решението на проблема е да се разреши разделянето на една мрежа на **подмрежи**, но за външния свят тя да изглежда като една мрежа. За целта полето за мрежов номер се разширява надясно, като се отнемат битове от номера на хост.
За имплементация на подмрежите маршрутизаторите се нуждаят от **мрежова маска**, която определя границата между номера на мрежата + номера на подмрежата и номера на хоста.

За адресация в Internet се използват 32-битови IP-адреси. Хостовете, свързани към локална мрежа Ethernet, притежават уникални 48-битови физически адреси от тази мрежа.
За установяване на съответствието между IP адреса и Ethernet адреса на хостовете в локалната мрежа се използва протокол за право преобразуване на адресите **ARP**. Когато даден хост трябва да изпрати дейтаграма към машина от локалната мрежа, чийто IP адрес е известен, но не е известен Ethernet адреса, мрежовият слой разпространява в локалната мрежа ARP пакет-заявка. Този пакет-заявка е от тип broadcast, т.е. предава се до всички машини. В полето “Ethernet адрес на подателя” е записан съответният адрес на хоста, който изпраща ARP заявката.
В полето “Данни” е записано ARP съобщение от вида “who is X.X.X.X tell Y.Y.Y.Y”, където X.X.X.X и Y.Y.Y.Y са IP адреси съответно на получателя и на подателя. Всички машини от локалната мрежа игнорират заявката с изключение на хоста, чийто адрес съвпада с X.X.X.X. Този хост изпраща ARP пакет-отговор само на подателя, тъй като вече знае неговия Ethernet адрес от получената заявка. В полето “Данни” на пакета-отговор е записано ARP съобщение от вида “X.X.X.X is hh:hh:hh:hh:hh:hh”,

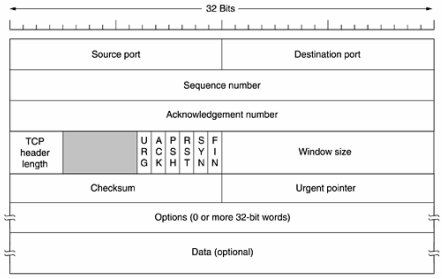
където hh:hh:hh:hh:hh:hh е Ethernet адреса на хоста, изпращащ пакета-отговор. Обикновено хоста, който изпраща ARP заявката запомня (кешира) получените 48-битови Ethernet адреси, за да могат да се използват при следващо предаване. При определяне на Ethernet адреса на получателя на дадена дейтаграма първо се проверява дали този адрес вече е кеширан и ако не е, се изпраща ARP заявка. Освен това всеки хост при първоначалното си стартиране изпраща broadcast съобщение, което съдържа неговият IP адрес и Ethernet адрес, което се получава от всички останали хостове в локалната мрежа и те записват тази информация в своите кешове.
Протоколът **RARP** е за намиране на IP адреси по Ethernet адреси. За функционирането му е необходимо в мрежата да е включен хост, който функционира като RARP сървър. Този сървър съхранява съответствието между Ethernet и IP адреси на станциите в мрежата. Обикновено RARP се използва от машини без твърди дискове, които чрез него научават своя IP адрес.

Големият проблем на IP протокола е недостига на адреси. По принцип съществуват над 2 милиарда адреси, но организацията им по класове е неефективна и заради нея се губят милиони адреси. Друг проблем е големината на маршрутните таблици. Съхраняването на таблици с милиони редове да кажем е възможно, въпреки че повечето маршрутизатори съхраняват таблиците си в оперативната памет, но друг сериозен проблем е сложността на алгоритмите за маршрутизация. Освен това, маршрутизаторите периодично трябва да изпращат маршрутните си таблици – колкото са по-големи, толкова по-голяма е вероятността части от тях да се загубят при предаването.
Решение на описаните проблеми е използване на безкласова маршрутизация **CIDR**. Основната идея е, че незаетите IP адреси се разпределят по блокове с различна големина (степен на 2), без да се взимат под внимание класовете.
Премахването на класовете усложнява маршрутизацията. Всеки ред от маршрутната таблица се разширява с 32-битова маска.
По този начин маршрутната таблица е масив от тройки (IP адрес, маска, изходяща линия). Когато пристигне пакет, се извлича IP адресът на получателя. После маршрутната таблица се сканира ред по ред, като за всеки ред съответната маска се прилага към IP адреса и се сравнява получения номер с номера в реда. Възможни са няколко съвпадения – в такъв случай се избира реда с най-дълга маска.

Най-важните протоколи, обслужващи транспортния слой, са **TCP** (transmission control protocol) и **UDP** (user datagram protocol).

Предназначението на TCP е да осигурява надеждно предаване на данните между предавателя и приемника чрез установяване на връзка. Обменът на информация, който осъществява TCP се

извършва посредством **сегменти**. При предаване TCP получава данни от по-горния слой, разделя ги на части, опакова ги в сегменти и ги изпраща на IP протокола. Той от своя страна опакова сегментите в дейтаграми и извършва маршрутизирането на всяка дейтаграма. При приемане IP протоколът разопакова пристигналите дейтаграми, след което предава получените сегменти на TCP протокола, който сглобява и подрежда данните от сегментите в съобщения към по-горните слоеве така, както те са били изпратени.
Всеки край на TCP връзката се идентифицира с IP адреса на съответния хост и с 16-битово число, наречено **номер на порт**, което определя съответната приложна програма, използваща тази връзка. Комбинацията от адреса на хоста и номера на порта се нарича **socket**. Всеки TCP сегмент съдържа номерата на портовете на източника и на получателя и това позволява на TCP протокола да определи за коя приложна програма е предназначен съответния сегмент. Комбинацията от socket-а на източника и socket-а на получателя е уникална и тя идентифицира TCP връзката. Първите 1024 номера на портове са така наречените **well-known** портове, които са резервирани за най-често използваните стандартни приложни програми.
TCP сегментът се състои от заглавна част и част за данни и има следния формат:




Заглавната част включва задължителни полета с фиксиран размер 20 байта, към които може да бъде добавено поле Options. След опциите (ако има такива) следва полето на обменяните данни - Data, което също не е задължително.
Полетата Source port и Destination port са двубайтови и представляват номер на порта на източника и на получателя съответно, които заедно с IP адресите на източника и на получателя образуват номера на socket-и, идентифициращи уникално връзката.
Полето Sequence number е поредния номер на първия байт (в рамките на последователността от байтове, предавани от източника), който е записан в полето Data на сегмента.
Полето Acknowledgement number е номерът на първия байт данни, който се очаква да се получи със следващия сегмент, изпратен от другия край на TCP връзката.
Полето TCP header length е 4-битово и определя дължината на заглавната част на TCP сегмента в 32-битови думи. То е задължително, тъй като полето за опции е с променлива дължина.

Заглавната част на TCP сегмента съдържа и 6 еднобитови флага. Те имат следното предназначение:

- URG – указва, че е валиден е указателят за спешни данни;
- ACK – валиден е номерът на потвърждение, записан в полето Acknowledgement number на заглавната част;
- PSH – при активирането на този флаг, програмните модули управляващи транспортния слой на източника и на приемника трябва да изпратят незабавно наличните данни колкото е възможно по-бързо към техния получател;
- RST – сегмент, в който е установен този флаг, служи за прекратяване на TCP връзката;
- SYN – сегмент с установен флаг SYN се използва при установяване на TCP връзка и за изпращане на началния номер, от който ще бъдат номерирани байтовете на изходящия информационен поток;
- FIN – сегмент, в който е установен този флаг, означава, че изпращачът прекратява предаването на данни.

Полето Window size определя темпа на информационния обмен от гледна точка на получателя на информационния поток. Стойността на прозореца указва на отсрещната страна колко байта могат да бъдат изпратени и съответно приети без препълване на входия буфер след последния потвърден номер на байт. При получаване на данни, размерът на прозореца намалява. Ако той стане равен на 0, изпращачът трябва да престане да предава данни. След като данните се обработят, получателят увеличава размера на своя прозорец, което означава, че е готов да получава нови данни.
Полето Urgent pointer се използва да укаже позицията на първия байт на спешните данни спрямо началото на полето данни.
Полето Checksum се изчислява върху целия TCP сегмент. При неговото изчисляване участват и някои полета от заглавната част на IP дейтаграмата, в която е опакован сегмента.
Полето Options на заглавната част на TCP сегмента е предназначено да предостави допълнителни възможности за управление на обмена. Най-важната възможност е указване на максимална дължина на сегмента. Всеки хост указва своята максимална дължина на сегмента и за осъществяване на обмена се приема по-малката от двете. Ако максималната дължина на сегмента не се договори се приема по подразбиране, че нейната стойност е 556 байта, което е допустимо за всички интернет хостове.
При първоначално отваряне на връзката между два хоста е необходимо всеки от тях да изпрати на другия началния номер на байтовата последователност, която ще изпраща, и съответно да получи насрещното потвърждение за получаването на този номер. Процедурата за установяване на връзка се нарича **трикратно договаряне** и в нормалния случай е следната:

1. Хостът (клиентът), който отваря връзката, изпраща SYN сегмент. В същия сегмент клиентът указва номера на порта

на сървъра, с който трябва да се установи връзка, и началният номер x на потока байтове, който клиентът ще предаде към сървъра.
2. Сървърът отговаря със собствен SYN сегмент, включващ началния номер y на неговия поток от байтове. В сегмента се съдържа потвърждение за SYN сегмента с номер на потвърждението, равен на x+1, тъй като за самия SYN сегмент е необходим един пореден номер.
3. Клиентът трябва да потвърди получаването на SYN сегмента от сървъра, като изпрати сегмент с потвърждаващ номер y+1.

Затварянето на връзката също се извършва чрез трикратно договаряне. Тъй като TCP връзката е пълен дуплекс, тя се затваря, когато всеки от двата хоста прекрати своя изходящ информационен поток. Такъв тип затваряне на връзката се нарича още симетрично. Вариантът, при който даден хост прекратява информационния обмен и в двете посоки се нарича асиметрично прекратяване на връзката.
За симетричното затваряне на една връзка е необходим обмен на 4 сегмента – по два за всяка посока. Даден хост може да инициира затваряне на своята част на връзката, когато изпрати сегмент с установен флаг FIN, след като приключи с предаването на данни. Хостът, получил този сегмент, може да продължи да изпраща данни при положение, че не е затворил връзката. Всеки хост, който получи FIN сегмент, изпраща обратно потвърждение с номер, равен на получения пореден номер + 1, тъй като FIN сегментът изисква един пореден номер.
За асиметрично затваряне на връзката се изпраща сегмент с вдигнат флаг RST.

UDP е прост транспортен протокол за предаване на дейтаграми в мрежите с комутация на пакети. За разлика от TCP, той не осъществява надежден транспорт. Дейтаграмите се изпращат от източника без да се контролира дали са достигнали до получателя. Затова форматът на заглавната част на UDP дейтаграмата е много по-прост: тя съдържа само номерата на портовете на източника и на получателя, дължината на дейтаграмата и контролна сума. UDP има смисъл да се използва при мрежи с висока надеждност.
Най-мощното приложение на Internet е **WWW**.
Основното, на което се базира web-технологията е хипертекста. Това е текст, който съдържа в себе си информация как да бъде изобразен на екрана. Изобразяването става чрез специална програма, наречена хипертекстов browser.
Хипертекстът се оформя като съвкупност от страници. Всяка страница си има уникално URL – уникален адрес, който еднозначно указва местоположението на страницата в целия Internet. Другото нещо е хиперлинкът – под част от текста, който се изобразява отдолу стои URL на друга страница. С други думи

хипертекстовите страници съдържат препратки към други хипертекстови страници.
Browser е клиентът, който изтегля и изобразява страниците.
Web server е сървърът, който съхранява страниците. За комуникация между browser-ите и сървърите е създаден протокола **HTTP**.
Едно URL съдържа протокол, име на домейн и пътя на страницата върху диска на сървъра. При осъществяване на връзка между browser-а и сървъра по URL-то първо по името на сървъра, а после по пътя върху диска на сървъра страницата физически се изтегля от сървъра, предава се на browser-а и той я изобразява. Една страница се прехвърля в рамките на една HTTP-сесия. Ако човек кликне върху линк на изтеглената страница, browser-ът установява нова сесия, подава се новото URL, изтегля се страницата от сървъра и отнове се изобразява от browser-а.
Протоколът HTTP се базира на TCP. HTTP клиентът отваря сесия на произволен порт с номер по-голям от 1024. HTTP сървърът слуша за заявки на порт 80.
При HTTP протокола имаме подготвителна фаза – прави се заявка за HTTP сесия към сървъра, след това се прави HELLO към сървъра, потвърждава се от сървъра и се изпращат методи на HTTP. Методите са GET, HEAD, POST, PUT, TRACE, CONNECT.
Основният метод е GET. Като аргумент му се подава пълното име на страницата върху диска на сървъра. Страниците могат да се кешират върху proxy-сървъри, затова в метода GET има if-условие. Всяка хипертекстова страница има заглавие, което съдържа описание на възрастта на страницата, датата на последната модификация и др. Методът HEAD взима само заглавието на страницата. Той позволява на browser-а бързо да провери дали я има физически страницата и кога е модифицирана последно (спестява се тегленето на тялото на страницата).
Методът PUT служи за прехвърляне на страница върху сървъра. Той е свързан с обмен на два етапа – първо се дава адресът на страницата, след това се прехърля самата страница. Методът POST е аналогичен на метода PUT с тази разлика, че той добавя новите данни към съществуващ адрес.
Методът TRACE връща от сървъра получените данни по заявката. Той се използва за тестване – да видим дали сървърът е получил това, което сме изпратили.
При първите версии на HTTP протокола за изпълнението на всеки метод се прави отделна HTTP сесия – тя отваря TCP съединение, праща нещо, след това затваря последователно съединението и сесията. С други думи имаме 1:1 – една сесия, едно съединение. След това започва развитие – стремежът е да не се затваря съединението, т.е. да има няколко сесии върху едно съединение. Ако за всяко изпълнение на GET се затваря съединението, а предстои четене на серия от страници това е много неефективно. При версията 1.0 на HTTP на едно съединение отговаря една сесия.

При версията 1.1 на HTTP на едно TCP съединение отговарят няколко сесии (няколко команди). Тези команди касаят различни хипертекстови страници, но достъпът до тях се прави с едно TCP съединение. За целта се създава съобщителен канал. По него в пълен дуплекс текат заявки, а в обратна посока – отговори. Така не се работи по метода спри и чакай за всяка заявка.
Друга особеност е, че зад един IP адрес може да има няколко имена на сървъри (така наречените виртуални сървъри). За клиента те са различни сървъри, но реално зад тях стои един и същ IP адрес. По-късно те могат да мигрират към други компютри, но вътре в тях URL-то ще се запази и по този начин не е необходима промяна на хиперлинковете към страници върху виртуалните сървъри.