***1. Компютърни мрежи. Апаратура и топологии.***

Компютърните мрежи са комплекс от цифрова, компютърна апаратура. Компютрите в мрежата се свързват по специални правила и с определен софтуер.

Компютрите в мрежата се наричат **хостове** (host).

Най-общата класификация на мрежите, основана на логическото преминаване на данните през комуникационната среда е следната:

* **мрежи със селекция** – представляват физически много бърз канал, по който минават всички данни, които се обменят между компютрите; всеки компютър в мрежата извършва селекция върху общия поток от данни;
* **мрежи с маршрутизация** – компютрите са свързани към специални възли, чрез които се осъществява разпознаване на входния поток, разделяне на входния поток и доставка на данните до съответния получател.

**Топологията** на една мрежа определя геометричното свързване на физическите канали и възлите. Тя се определя на основата на географското разположение на компютрите, на обема на предаваните данни, на пропусквателната способност на линиите за връзка и др.

По отношение на физическия размер на мрежите класификацията е следната:

**LAN** (local area network) - локална мрежа - характеризира се с физическа ограниченост в рамките на една или няколко сгради с диаметър няколко километра. Традиционните локални мрежи предават със скорост 10 Mb/s до 100 Mb/s. Скоростите при по-новите LAN достигат 10 Gb/s.

Локалните мрежи са мрежи със селекция - хостовете сами селектират това, което е предназначено за тях.

При локалните мрежи са възможни различни топологии.

При **шинната топология** всички компютри са свързани към линеен кабел. В даден момент само една машина може да предава данни, а всички останали машини трябва да чакат. За да се разрешават конфликти при едновременно желание за предаване на две или повече машини се използват арбитриращи механизми.

Те могат да са централизирани – при тях има специално устройство, което извършва арбитража и децентрализирани (разпределени) – при тях всяка машина сама определя кога да изпраща. Пример за мрежа с шинна топология и разпределен контрол е **Ethernet**.

При **кръговата топология** хостовете са разположени в кръг на определено разстояние един от друг. Обикновено всеки предаден пакет извършва една пълна обиколка на кръга. И тук трябва да има подходящ арбитраж. При мрежата **IBM Token Ring** например, машините се редуват да предават чрез използване на специален маркер, който обикаля кръга.

**MAN** (metropolitan area network) - градска мрежа - високоскоростна мрежа, която трябва да обслужва едно населено място. Отделни участъци от мрежата, например в рамките на една сграда, са на принципа на селекцията. Връзката между тези участъци се осъществява по принципа на маршрутизацията чрез оптически кабели, разположени в комуникационни шахти.

При градските мрежи е важен въпросът за разширяемост в рамките на определената територия. Най-добър пример за MAN е кабелната телевизия. В началото кабелните мрежи са служели за предаване само на телевизионни програми, но впоследствие кабелните оператори започват да предоставят на своите потребители достъп до Интернет, като използват незаетите от телевизионните сигнали честоти.

**WAN** (wide area network), **WLN** (wide large network) - регионални мрежи - обхващат широки географски области, най-често страни или континенти. Изцяло работят на маршрутизация. Хостовете в WAN са свързани помежду си чрез **комуникационна подмрежа**. Тя най-често е собственост на телефонна компания или на доставчика на Интернет услугите (**ISP**). Задачата на комуникационната подмрежа е да предава съобщения от хост до хост. В повечето WAN тя се състои от предавателни линии и превключващи (комутиращи) елементи.

По предавателните линии се пренасят пакетите с данни между машините. Те могат да са медни жици, оптически влакна или да са основани на радио връзки. Комутиращите елементи свързват две или повече предавателни линии. Те най-често се наричат **маршрутизатори** (**router**). Когато един пакет пристигне на входна линия на даден маршрутизатор, той трябва да избере на коя изходна линия да предаде този пакет. За целта се използват маршрутизиращи алгоритми.

Някои WAN използват сателитни връзки – при тях всеки маршрутизатор има антена, чрез която приема или предава съобщения от сателита.

За да могат потребителите на различни мрежи да комуникират помежду си се използват специални устройства, наречени шлюзове (**gateway**). Те реализират свързване на различни мрежи, които могат да използват различни технологии. Съвкупност от такива свързани мрежи се нарича **интернет**.

Най-популярен пример е световната мрежа Интернет (**Internet**).

При всички мрежи съществени са линиите между възлите – те се изграждат от телефонни компании. В рамките на града най-често се използват възможностите на съществуващата телефонна мрежа. Нейните възли са телефонните централи. Всеки отделен абонат е свързан чрез медна усукана двойка кабели към районна телефонна централа. Всяка районна централа се свързва към една или повече регионални централи. Телефонните централи се свързват помежду си чрез цифрови високоскоростни магистрали, най-често се използват оптически влакна. Свързването между два отделни абоната се базира на комутация на каналите.

Телефонната мрежа осигурява надеждна работа в ниския честотен диапазон от 300 Hz до 3300 Hz за предаване на глас. При аналоговите телефонни мрежи се налага преобразуване на цифровите сигнали към аналогови (**модулация**) и обратно – преобразуване на аналоговите сигнали към цифрови (**демодулация**). Тези преобразувания се извършват от устройства, наречени **модеми**. При споменатите честоти между два абоната може да се предават данни със скорост 33.6 Kb/s. Скоростта може да достигне 56 Kb/s, ако единият абонат е свързан директно с цифрова линия към телефонна централа (както са повечето ISP).

Ограниченията в честотната лента при предаване на говор се дължат най-вече на наличието на филтри в края на линиите.

Без тези филтри по медната усукана двойка проводници могат да се предават данни с много висока скорост. Цифровата технология **ADSL** (asymmetric digital subscriber line) позволява предаване на данни от централата към абоната със скорост 8 Мb/s и от абоната към централата със скорост 1 Mb/s.

***2. Структура и еталонен модел на мрежите. Нива.***

Основен принцип в съвременните мрежови архитектури е принципът за разслояване на функциите по управление на връзките, като всеки слой ползва услугите, предоставени от по-долните слоеве без да знае как са реализирани тези услуги.

Слоят *n* на една машина взаимодейства със слоят *n* (на същото ниво) на друга машина. Правилата по които се осъществява това взаимодействие се определят от протокола на *n*-то ниво.

Най-общо под **протокол** се разбира съгласувани правила между комуникиращите страни за това как да протича комуникацията.Всеки слой *n* предоставя **интерфейс** на слой *n+1*. Интерфейсът определя функциите и услугите, които слоят *n* предоставя на слой *n+1*. При определянето на интерфейсите между отделните слоеве трябва ясно да се знае какви функции изпълнява всеки слой.Разслояването позволява да се промени изцяло имплементацията на даден слой *n*, без да се променя имплементацията на другите слоеве – достатъчно е да се запази множеството от услугите, които слой *n* осигурява на горния слой *n+1*.

Една **мрежова архитектура** се определя от множеството на слоевете, услугите които те предоставят и протоколите, по които се осъществява взаимодействие между слоевете на едно и също ниво. Списъкът от протоколи, използвани от една система, по един протокол за всеки слой се нарича **протоколен стек**.

Ще разгледаме моделът **OSI** (open system interconnection), създаден от международната организация **ISO** (international standard organization) за връзка между отворени системи. OSI-моделът ще наричаме още **еталонен модел**.

В еталонния модел има седем слоя – физически, канален, мрежов, транспортен, сесиен, представителен, приложен – изброени са в последователност от най-долния към най-горния слой.

**Физическият слой** (physical layer) 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. Той не се интересува от предназначението на битовете. Физическият слой трябва да осигурява възможност на по-горния слой да активизира, поддържа и прекратява физическите съединения.

Основна функция на **каналният слой** (data-link layer) 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.Форматът на кадрите се определя от избрания протокол на канално ниво. Функциите на каналния слой обикновено се реализират смесено - апаратно и програмно.

**Мрежовият слой** (network layer) 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** (transport layer) осигурява транспортирането на 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** (session layer) 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** (presentation layer) 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. Например, за унификация на символни данни е съставена двубайтовата кодова таблица UNICODE.

**Приложният слой** (application layer) 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 (request-reply). 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 - например протокол **FTP** (file transfer protocol) за обмен на цели файлове, протокол **HTTP** (hyper text transfer protocol) за обмен на уеб-страници и др.В ARPANET - първата компютърна мрежа, която прераства в Internet се използва моделът **TCP**/**IP**. За разлика от OSI-модела, този модел се обвързва с конкретни протоколи и не е приложим за описание на мрежи, които не използват тези протоколи.При модела TCP/IP се запазват приложният и транспортният слой, липсват сесийният и представителният слой, мрежовият слой е известен като интернет-слой, а каналният и физическият слой са обединени в един слой за достъп до мрежата, който почти не се коментира.

***3. Видове мрежи – комутация и съобщения.***

Различават се три режима на предаване на съобщения от източника до приемника - **комутация на канали**, **комутация на съобщения** и **комутация на пакети**.

При мрежите с комутация на канали между двата крайни абоната се създава временен физически канал, а след това по този канал се предава едно съобщение (серия от кадри). Когато съобщението се предаде напълно, каналът се освобождава. Когато пристигне следващото съобщение отново се установява физическа връзка между крайните машини, съобщението се предава и т.н. Превключването на каналите става много бързо в специални електронни възли. Същественият недостатък е, че физическият канал се ангажира непрекъснато, докато се обменя съобщение, т.е. целият път по канала се блокира. Пример за мрежа с комутация на канали е телефонната мрежа.

При мрежите с комутация на съобщения всяко съобщение, което трябва да се предаде се изпраща в комуникационната подмрежа. Тя избира неговия маршрут до назначението му. С други думи, изпращачът подава съобщението на прилежащия му междинен възел, след което съобщението се придвижва на **хопове** (скокове между непосредствено прилежащи междинни възли) към получателя. Мрежи с такава организация се наричат **мрежи със запомняне и препредаване** (store-and-forward network).

Съобщенията са с неограничена дължина, което изисква възловите компютри да притежават големи по обем буфери дори върху дискове. На канално ниво съобщенията се предават като серия от последователни кадри. Това може да доведе до блокиране на линия между два маршрутизатора при предаване на едно съобщение, което прави мрежите с комутация на съобщения неприложими при интерактивен трафик.

При мрежите с комутация на пакети съобщенията се разбиват на части, наречени **пакети** (по 1000-10000 бита). Пакетите са с фиксиран размер в рамките на една мрежа. Всеки пакет се предава индивидуално в комуникационната подмрежа и възловите компютри се грижат за съхраняването и препредаването на пакети, а не на цели съобщения. Тъй като пакетите са значително по-къси от съобщението те могат да се буферират в оперативната памет на възловите компютри и да се обменят по-бързо.

Мрежите с комутация на канали и с комутация на пакети значително се различават. При мрежите с комутация на канали, преди да започне комуникацията между двете крайни системи трябва да се създаде физически канал. При мрежите с комутация на пакети това не се изисква - предаването на един пакет може да се извърши веднага след формирането му в източника.

Като следствие от създаването на физически канал при мрежите с комутация на канали, всички пакети на едно съобщение минават по един и същи път и пристигат в получателя в реда, в който са били изпратени. При мрежите с комутация на пакети различните пакети на едно съобщение могат да преминат по различни физически пътища до получателя и да пристигнат в разбъркан ред. Мрежите с комутация на пакети са по-надеждни от мрежите с комутация на канали в смисъл, че при отпадане на един възел, всички физически канали, минаващи през този възел стават неизползваеми, а при комутацията на пакети отпадналите възли могат да се избегнат по обиколен маршрут.

Когато се резервира един физически канал, той може да се използва само за трафик между двете крайни системи. Това води до неефективно използване на ресурсите, което се избягва при мрежите с комутация на пакети.При комутацията на пакети се използва техниката на запомняне и препредаване. При комутацията на канали тази техника няма смисъл - тя само би забавила предаването. Теоретично при мрежите с комутация на канали кадрите преминават транзит през междинните възли.

Мрежите с комутация на пакети съществуват в две основни разновидности - **мрежи с виртуални канали** и **дейтаграмни мрежи**. При мрежите с виртуални канали пакетите между два хоста се предават по точно определен маршрут, който се установява при създаване на виртуалния канал. Пакетите пристигат в реда, в който са изпратени. Дейтаграмните мрежи осигуряват предаване само на независими един от друг пакети – дейтаграми. **Дейтаграмата** представлява пакет, който се предава от източника към приемника, съгласно указан в него адрес, без маршрутът да е определен предварително. Комуникационната подмрежа не гарантира на приемника същата последователност на получаване на дейтаграмите, каквато е използвана при изпращането - след като получи всички дейтаграми приемникът оформя съобщението. Предимството на дейтаграмните мрежи е, че отделните дейтаграми могат да се изпращат по различни канали в различно време, което води до уплътняване на физическите канали. Недостатъкът е, че обемът на дейтаграмата е голям - тя трябва да носи адрес на приемника и друга служебна информация - което увеличава дължината на предаваното съобщение.

***4. Физическо ниво в мрежите. Теоретически основи и среди за предаване.***

Целта на физическото ниво е да транспортира поредица от битове между две машини. За предаване на информацията се използват физически канали, които се определят като среда за предаване.

Физическото ниво трябва да предоставя обслужване на по-горното канално ниво. Пример е серийният интерфейс RS 232, по който един персонален компютър може да се свърже с модем.

Цифровата информация се предава като цифрови сигнали, т.е. логическа последователност от нули и единици, но тя често трябва да се преобразува в аналогов сигнал.

Всяка частично гладка периодична функция може да се развие в безкраен ред на Фурие - безкрайна сума от синуси и косинуси. По този начин всеки цифров сигнал може да се апроксимира със сума от аналогови хармонични сигнали.

В зависимост от качествата на предавателната среда, честотите над определена стойност затихват много бързо, което изкривява съответните Фуриерови компоненти. Обхватът на предаваните честоти, които преминават без да затихват се нарича **широчина на честотната лента**. Колкото е по-широка честотната лента, толкова по-точно може да се възпроизведе цифровият сигнал.

При тясна честотна лента (например при телефонните линии) цифровите сигнали не могат да се предават точно, поради което се използва **модулация**. Въвежда се носещ синусоидален сигнал и информацията се предава чрез смяна на неговата честота (честотна модулация), амплитуда (амплитудна модулация) или фаза (фазова модулация). На практика при предаване се комбинират няколко техники за модулация.

Времето за предаване на един бит зависи както от широчината на честотната лента, така и от метода на кодиране. Броят на превключванията на стойностите на един цифров сигнал се измерва в **бодове**.

Най-старата и все още най-разпространена среда за предаване е **усуканата двойка**. Тя се състои от два изолирани медни проводника с дебелина около 1 милиметър, които се усукват един около друг. Най-разпространеното приложение на усуканата двойка е за свързване на телефоните към телефонните централи. Усуканите двойки могат да предават на разстояние няколко километра без усилване, но за по-големи разстояния трябва да се използват повторители (**repeater**).Обикновено усуканите двойки, които свързват два обекта се оформят в един кабел, който се затваря със защитна обвивка. Двойките проводници се усукват именно за да се намали взаимното им влияние в общия кабел. При неголеми разстояния (няколко километра) може да се постигне скорост на предаване няколко Mb/s.

Няколко разновидности на усуканите двойки са важни за компютърните мрежи. Основно са два типа - неекранирани усукани двойки (**UTP**) и екранирани усукани двойки (**STP**). Екранираните усукани двойки са обвити със специален защитен екран от алуминиево фолио, който предпазва проводниците от външни влияния. Усуканите двойки от категория **3 UTP** обикновено се групират по четири в един кабел, а тези от категория **5 UTP** са по-нагъсто усукани, което повишава качеството на сигнала и ги прави по-подходящи за високоскоростно предаване - до 100 Mb/s. 3 UTP позволява предаване със скорост до 10 Mb/s.

Друга разпространена среда за предаване е **коаксиалният кабел** (coaxial cable, coax). Той има по-надеждна защита от усуканите двойки и позволява предаване на по-големи разстояния с по-високи скорости. Състои се от меден проводник, обвит с диелектричен материал. От своя страна диелектрикът е обвит с медна оплетка, която изпълнява ролята на екран, предпазващ кабела от външни електромагнитни смущения. Върху медната оплетка се нанася изолиращ слой.С коаксиален кабел може да се организира местна високоскоростна връзка, при която данните се предават директно в първичния си вид като правоъгълни импулси (**baseband**). Коаксиалните кабели позволяват да се осъществи и така нареченото широколентово предаване (**broadband**), при което наличната честотна лента се разделя на определен брой подканали - техниката се нарича **мултиплексиране** с разделяне на честотата. Коаксиалните кабели намират приложение в кабелните телевизии и при градските мрежи MAN.

Развитието на оптиката позволи създаването на друга среда на предаване - **оптическите влакна** (fiber optic). Една оптическа система включва три компонента: източник на светлина, предавателна среда и детектор. Обикновено с един импулс светлина се представя логическата 1, а отсъствието на такъв импулс означава логическа 0. Използват се два източника на светлина - светодиод или полупроводников лазер. Предавателната среда представлява много тънко влакно, изработено от изключително чисто стъкло. Детекторът (фотодиод) генерира електрически импулси когато светлината попадне върху него. Защитното покритие предпазва кабела от механични повреди. Теоретично оптичните кабели могат да предават със скорост десетки терабита в секунда.Съвременната горна граница е 10 Gb/s поради невъзможността за по-бързо преобразуването на електрическите сигнали в светлинни.

**Безжичната комуникация** е възможна, благодарение на разпространението на електромагнитни вълни в пространството. Посредством антени с подходящ размер, електромагнитните вълни могат да се предават и приемат. Радиовълните, микровълните, инфрачервените лъчи и видимата светлина могат да се използват за предаване на информация като се модулира тяхната амплитуда, честота или фаза.

***5. Канално ниво – основни характеристики: кадри, грешки, прост протокол. Протоколи с прозорци.***

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).

Каналите са три вида - **симплексни**, **полудуплексни** и **дуплексни**. Дуплексните канали позволяват едновременно предаване в двете посоки. Полудуплексните канали позволяват предаване и в двете посоки, но в даден момент може да се предава само в една посока. Симплексните канали позволяват предаване само в една посока.

Най-общата услуга, която каналното ниво предоставя е надеждното прехвърляне на данни между мрежовото ниво на източника и мрежовото ниво на получателя (всъщност самото предаване се извършва от физическото ниво, но това остава невидимо за мрежовото ниво). Ако един кадър се загуби поради шум в линията, каналното ниво не прави опит да възстанови този кадър. Това обслужване е подходящо при канали с много малка честота на грешките

По принцип потвърждаването на получената информация е функция на транспортното ниво, но там то се отнася до последователности от пакети. Потвърждаването на каналното ниво има смисъл при ненадеждна комуникационна среда, каквато е безжичната, тъй като повторно ще се предават само непотвърдените кадри.

При потвърденото и установено обслужване има три фази. През първата фаза се установява връзка и се заделят необходимите ресурси (локални буфери, броячи и т.н.). През втората фаза се изпращат кадрите, а през третата фаза се освобождават ангажираните ресурси. Друг проблем, който е свързан с управлението на обмена на канално ниво е източникът да изпраща кадри по-бързо, отколкото те могат да бъдат приети от получателя. За целта се въвеждат механизми за управление на потока от кадри, който осигурява обратна информация на източника за темпа на предаване.

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.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 се премахва служебната информация на кадъра и информационният поток се предава на мрежовото ниво вече под формата на пакети.

Разделянето на потока от битове на кадри не е тривиална задача.Понастоящем основно се използват три метода.

При първия метод се **броят отделните символи**. В заглавието на кадъра се указва броя на символите в целия кадър. Когато каналното ниво на получателя прочете броя на символите в заглавната част на кадъра, то знае колко символа предстоят до края на кадъра. Основният проблем на този метод е, че броят на символите може да бъде сгрешен по време на предаването, при което получателят ще загуби синхронизация и няма да може да определи началото на следващия кадър.

При втория метод в началото и края на кадъра се вмъкват специални служебни символи - **STX** (start of text) за начало на кадър и **ЕТX** (end of text) за край на кадър, които маркират границите на кадъра. Техниката е известна като **вмъкване на символи** (byte stuffing, character stuffing).

С развитието на мрежите стана възможно кадрите да съдържат произволно цяло число битове. За такива кадри се използва третия метод, при който началото и края на всеки кадър се маркира с битовата последователност 01111110, наречена **флагов байт**.

***6. Протоколи на канално ниво – HDLC и PPP.***

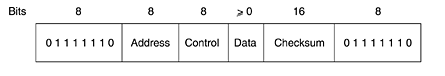
Първият протокол на канално ниво, който се използва в IBM е

**SDLC** (synchronous data link control). По-късно организацията по стандартизация ISO разработва на базата на SDLC протоколът

**HDLC** (high-level data link control).

И двата протокола са битово-ориентирани и използват вмъкване на битове за правилно идентифициране на кадрите.

Форматът на кадъра в HDLC е следния:



В началото и в края на кадъра са флаговете за маркиране на границите на кадъра.

Полето *Address* се използва при многоточкови канали (multipoint) и чрез него се идентифицира получателя на кадъра.

Полето *Control* се използва за номериране на кадрите, за потвърждения и за други цели.

Полето *Data* съдържа данните на кадъра. По принцип има неограничена дължина.

Полето *Checksum* е контролната сума на кадъра (използват се циклични кодове).

Минималната дължина на кадъра, без да се включват флаговете за начало и край е 32 бита.

Кадрите са три вида - **information**, **supervisory** и **unnumbered**.

Полето *Control* за information-кадрите има следния формат:



В протокола се използва прозорци с 3-битови номера.

Полето *Seq* е поредния номер на кадъра в прозореца на предавателя. Полето *Next* е прикачено потвърждение за насрещния поток - то съдържа номерът на следващия кадър, който се очаква в получателя.

Битът *P/F* се използва при изпращане на кадри към терминали.

Ако той е 1, предавателят указва на терминала да предава.

Всички кадри, които терминалът изпраща освен последния имат стойност 1 за този бит. За последния изпратен кадър *P/F* е 0.

Понякога битът *P/F* се използва за да се укаже на получателя да изпрати моментално потвърждение, а не да го прикачва към насрещния трафик.

Полето *Control* за supervisory-кадрите има следния формат:



Полето *Type* определя типа на кадъра:

* тип 0 (RECEIVE READY) - кадър за потвърждение, в полето *Next* се указва номерът на следващия очакван кадър;
* тип 1 (REJECT) - кадър за негативно потвърждение, в полето *Next* се указва номерът на първия неполучен кадър, предавателят трябва да изпрати наново всички кадри, започвайки от *Next* (това отговаря на стратегията go back n);
* тип 2 (RECEIVE NOT READY) - кадър за потвърждение, подобен на RECEIVE READY, но указващ на предавателя да спре да изпраща кадри;
* тип 3 (SELECTIVE REJECT) - кадър за негативно потвърждение, в полето *Next* се указва номер на неполучен кадър, предавателят трябва да изпрати наново само кадърът с номер *Next* (това отговаря на стратегията selective repeat).

Полето *Control* за unnumbered-кадрите има следния формат:

쀀퀠S-1-친

Тези кадри са служебни и касаят поддържането на съединението, наричат се още **команди**. Някои от командите са:

* DISC (DISConnect) - команда за разпадане на съединение;
* SNRM (Set Normal Response Mode) - команда за установяване на режим, в който едната машина управлява, а другата изпълнява;
* SABM (Set Asynchronous Balanced Mode) - команда за установяване на режим, в който двете машини имат еднакво влияние върху съединението;
* SNRME, SABME (Extended) - аналогични команди на SNRM и SABME, но номерацията на кадрите при тях е 7-битова (полетата *Seq* и *Next* се разширяват с по 4 бита);
* FRMR (FRaMe Reject) - команда, която указва за кадър с погрешна семантика - например, кадър с дължина по-малка от 32 бита или кадър за потвърждение на неполучен кадър.

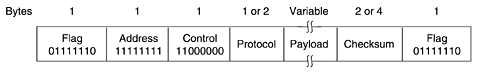
Командите също могат да се изкривят или изгубят, затова те също трябва да се потвърждават. За целта се използва специален unnumbered кадър UA (unnumbered acknowledgement). След изпращането на всяка команда се изчаква съответно потвърждение преди да се изпрати друга команда.

**Протокол PPP**

Протоколът PPP (Point-to-Point Protocol) е протокол за двуточкова връзка. Този протокол се използва за свързване на домашни компютри до доставчици на Интернет услуги по телефонна линия.

Протоколът PPP е байтово-ориентиран и за идентифициране на кадрите се използва техниката вмъкване на байтове.

Форматът на кадъра е наследен от HDLC:



При PPP няма индивидуални адреси на станциите, затова полето *Address* съдържа 11111111, което означава адресите на всички станции.

Полето *Control* съдържа 11000000, което означава

unnumbered-кадър. С други думи, PPP не осигурява надеждно предаване чрез номера на кадрите и потвърждения.

Полето *Protocol* съдържа идентификатор на протокол, който указва как да се интерпретира полето *Payload*, в което се помества съответния пакет.

Максималната дължина на *Payload* е 1500 байта.

Дължините на полетата *Protocol* и *Checksum* се договарят при установяването на съединение.

След установяване на съединение, двете страни се договарят за мрежовите протоколи, които ще се използват. След това започват да се предават кадрите с данни, като полето *Protocol* съдържа идентификатор на един от уговорените мрежови протоколи, а *Payload* съдържа съответната дейтаграма.

***7. Канално ниво в ETHERNET. Превключватели и мостове.***

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.

При мрежите с общодостъпно предаване основен проблем е да се определи кой да започне да използва канала, когато има състезание за него.

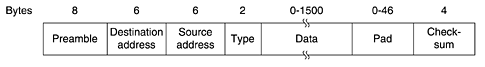
Протоколите, които се използват за да се разреши този проблем се отнасят към подниво на каналния слой, наречено **подниво за достъп до средата** (medium access control).

Градските и регионалните мрежи обикновено използват връзки "точка-точка" (point-to-point), общодостъпни многоточкови (multipoint) канали се използват най-вече при локалните мрежи.

Най-разпространената локална мрежа е **Ethernet**. Тя е описана в стандарта 802.3, издаден от ΙΕΕΕ (Institute of Electrical and Electronic Engineers) през 70-те години. 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.

Форматът на кадрите в Ethernet е следния:



Полето *Preamble* е синхронизираща последователност от байтове.

Полето *Destination address* съдържа адресът на получателя на кадъра, а полето *Source address* - адресът на изпращача на кадъра. Най-старшият бит на адреса на получателя е 0 за нормален адрес и 1 за групов адрес. При групов адрес, кадърът е предназначен за група станции (multicast). Адрес на получател, състоящ се само от 1 означава, че кадърът е предназначен за всички станции (broadcast).

Полето *Type* указва как получателя трябва да обработи кадъра.

Данните се съдържат в полето *Data* и максималната им дължина е 1500 байта. Освен максимална дължина на кадъра има и минимална дължина на кадъра. 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. В стандарта 802.3 минималната дължина на кадъра е 64 байта. Ако данните са

по-малко от 46 байта, то се използва полето *Pad* за запълване на кадъра до 64 байта.

Полето *Checksum* е контролна сума, която се използва за откриване на грешки при предаването.

При физическото предаване кадрите се кодират в **манчестърски код** (manchester encoding). Периодът за предаване на един бит се разделя на две равни части. Бит 1 се кодира високо напрежение в първия период и ниско напрежение във втория период. Бит 0 се кодира с ниско напрежение в първия период и високо напрежение във втория период. Преходът в средата на периода служи за синхронизация. По този начин няма нужда от паралелен синхронизиращ сигнал.

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). При окабеляване 100Base-T4 каналните станции се свързват към хъба чрез четири усукани двойки 3 UTP, а при 100Base-TX чрез две усукани двойки (5 UTP). По една от усуканите двойки се предава към хъба, а по другата се приема от него (при 100Base-T4 останалите две усукани двойки се превключват по посока на предаването). Станциите се свързват към хъба в **прав кабел**, т.е. предаващата двойка на всяка станция съответства на предаващата двойка на хъба и съответно приемащата двойка на всяка станция съответства на приемащата двойка на хъба.

При свързване на два хъба чрез усукана двойка, обаче, се използва **cross кабел**, т.е. предаващата двойка на единия хъб се свързва с приемащата двойка на другия хъб и обратно.

Предаването достига скорост 100 Mb/s.

Ако хъбът получи кадър по някоя линия, той изпраща този кадър по всички останали линии. Важно е да се отбележи, че хъбът не знае адресите на каналните станции.

Хъбът е пример за устройство, чрез което се препредават кадри от един кабел към друг. Той работи на физическо ниво. Друго подобно устройство на физическо ниво е **повторителят** (repeater).

Той приема сигнал на единия си порт, усилва го и предава сигналът на другия си порт. По този начин може да се увеличи максималната дължина на кабела в една локална мрежа.

**Мостът** (bridge) е устройство, което работи на канално ниво и служи за свързване на няколко локални мрежи. За разлика от повторителите и хъбовете, мостът анализира получените кадри.

Той прочита адреса на получателя и по него определя към коя изходна линия да изпрати кадъра (за целта се поддържа специална таблица). Ще отбележим, че мостът предава кадъра само към определената от него изходна линия, а не по всички изходни линии. Подобно устройство е **превключвателят** (switch) - той също прочита адресите на постъпилите в него кадри. Преключвателите най-често се използват когато на всяка линия има по една канална станция. Всяка линия е самостоятелна, така че кадри не могат да бъдат изгубени поради колизии. За сметка на това в превключвателя трябва да има достатъчно буферно пространство за да може да се препращат кадрите. По-добра алтернатива е използването на **cut-through превключвател**, който препраща кадъра към съответната изходна линия (стига тя да е свободна) веднага след като е прочетен адресът на получателя.

17. Маршрутни протоколи OSPF.

Протоколът OSPF (open shortest path first) е разработен за IP мрежи и използва алгоритъма за маршрутизиране със следене състоянието на връзката (link state).

OSPF маршрутизаторите поддържат топологични бази данни с информация за състоянието на връзките в мрежата. Тези бази данни периодично се обновяват посредством обмен на съобщения за състоянието на връзките и съдържат входните данни за алгоритъма на Дейкстра, който се изпълнява от всеки маршрутизатор. В резултат от неговото изпълнение, всеки OSPF маршрутизатор намира най-късите от своя гледна точка пътища до всички известни местоназначения в мрежата.

Важна особеност на OSPF е йерархичното разделяне на автономните системи на области (area). Връзките между различните области се осъществяват задължително през опорна мрежа (backbone), която е особена област с номер 0. Според принадлежността към OSPF областите различаваме 4 вида маршрутизатори:

- вътрешни маршрутизатори (internal routers), всичките интерфейси на които са свързани към мрежи от една OSPF област;

- областни гранични маршрутизатори (area border routers), интерфейсите на които са свързани към мрежи от две или повече OSPF области, една от които задължително е опорната мрежа. Тези маршрутизатори поддържат топологични бази данни с информация за състоянието на връзките в областите, към които са свързани, и изпълняват алгоритъма на Дейкстра поотделно за всяка от тях;

- опорни маршрутизатори (backbone routers), на които поне един от интерфейсите е свързан с опорната мрежа на област 0. Опорните маршрутизатори могат да бъдат областни гранични маршрутизатори, но това не е задължително;

- гранични за автономната система маршрутизатори (AS boundary routers), които са свързани към поне две различни автономни системи и изпълняват освен OSPF друг външен маршрутизиращ протокол (например BGP).

Опорните маршутизатори могат същевременно да бъдат вътрешни, ако всичките им интерфейси са свързани към опорната мрежа.

В рамките на една област всички маршрутизатори имат една и съща топологична база данни. Нейното предназначение е да се изчислят най-късите пътища между дадения маршрутизатор и всички останали маршутизатори в областта.

Йерархичните области в OSPF мрежите увеличават скоростта на сходимост на протокола и намаляват натоварването на необходимите за работата му мрежови и изчислителни ресурси.

OSPF поддържа три вида различни връзки:

- връзка от тип “точка-точка” между два маршрутизатора;

- многоточкови мрежи с broadcast (например, повечето LAN);

- многоточкови мрежи без broadcast (например, повечето WAN с комутация на пакети).

OSPF работи чрез обмяна на информация между прилежащи маршрутизатори, което не е същото като съседни маршрутизатори. Това се прави, тъй като не е ефективно всеки два маршрутизатора да обменят данни. За целта се избира един титулярен маршрутизатор (designated router), който става прилежащ към всички останали маршрутизатори и всеки от тях синхронизира топологичната си база данни с неговата. Ако настъпи промяна в състоянието на връзките на даден маршрутизатор, той изпраща съобщение на титулярния маршрутизатор, а от своя страна той съобщава промяната на всички останали маршрутизатори в мрежата. При отпадане на титулярния маршрутизатор има заместник, който е в състояние веднага да поеме неговите функции.

Форматът на OSPF пакета е следния:

Полето Version number съдържа номера на версията на OSPF, която се използва.

Полето Type определя типа на съобщението. Има няколко типа съобщения:

- “ехо” пакет (hello packet) – той се използва при откриване на съседни маршрутизатори, за проверяване на работоспособността на връзките и за избор на титулярен маршрутизатор;

- описание на база данни (database description) – разменят се при първоначално установяване на връзка между два маршрутизатора и служат за обмен на информация за състоянието на връзките в областта от техните топологични бази данни;

- пакет за запитване (link-state request) – чрез него даден маршрутизатор може да поиска информация от друг маршрутизатор за част от записите в неговата топологична база данни за състоянието на връзките;

- пакет за обновяване (link-state update) – тези пакети се изпращат периодично от всеки маршрутизатор и съдържат данни за неговото състояние и за цените, използвани в топологичната база данни; разпространяват се по метода на наводняването към всички останали маршрутизатори в границите на дадена област;

- пакет за потвърждение (link-state acknowledgement) – механизмът за надеждно доставяне на пакетите за обновяване изисква при получаването им да се изпращат потвърждения.

Полето Packet length съдържа общата дължина на OSPF пакета, включват се заглавната част и данните.

Полето Router ID съдържа уникален за автономната система идентификатор на маршрутизатора, изпратил пакета.

Полето Area ID съдържа номера на областта, към която е свързан интерфейса на маршрутизатора-подател на пакета.

Полето Check sum е контролна сума, която се изчислява върху цялото съдържание на OSPF пакета. Служи за проверка дали пакета е бил предаден правилно.

Полето Authentication type указва вида на използвания механизъм за идентифициране на подателя на пакета. Проблемът е, че в мрежата може да проникнат фалшиви пакети, които дублират информацията за състояние на връзките на някой маршрутизатор. За това трябва да се установява автентичността на пакетите.

Полето Authentication съдържа автентицираща информация.

18. Портален маршрутен протокол BGP.

В рамките на една автономна система се използват вътрешни протоколи за маршрутизиране, например RIP или OSPF. Тяхната цел е максимално бързо и ефективно да предават пакети от източника до получателя.

За маршрутизиране между различни автономни системи се използват външни маршрутизиращи протоколи. Такъв протокол е BGP (border gateway protocol). В него се залага на политиката на маршрутизация – например, най-прекият път между две автономни системи не винаги е разумен. В политиката се включват икономически, административни и други фактори. Примери за политически ограничения:

- да не се позволява транзитен трафик през дадена автономна система;

- трафикът за Пентагона да не минава през Ирак;

- трафикът, започващ или свършващ в IBM да не минава през Microsoft.

Политическите ограничения се конфигурират ръчно и не са част от BGP протокола.

От гледна точка на един BGP маршрутизатор, светът се състои от автономни системи, свързани с линии. Две автономни системи са свързани, ако има линия между гранични маршрутизатори във всяка от тях. От гледна точка на транзитния трафик BGP мрежите се делят на три вида:

- stub мрежи – те имат само една връзка в BGP графа и не могат да се използват за транзитен трафик;

- multiconnected мрежи – те могат да се използват за транзитен трафик, но самите те отказват да го правят;

- transit networks – например опорните мрежи, които пренасят транзитен трафик, обикновено срещу заплащане.

BGP протокола се базира на алгоритъма с вектор на разстоянието, но се различава доста от него. Вместо да се поддържа само цената към всяко местоназначение, всеки BGP маршрутизатор поддържа целия път към местоназначението. Освен това, вместо периодично да подава на съседите си цените до местоназначенията, всеки BGP маршрутизатор подава на съседите си целите пътища, които те използват. Това има смисъл, тъй като графът с възли автономните системи не е безмерно голям.

При BGP лесно се решава проблемът “броене до безкрайност”. Например, да предположим, че G пропадне или, че линията FG става недостъпна. Тогава F получава маршрути от останалите си три съседа. Тези маршрути са BCD, IFGCD и EFGCD. F бързо премахва последните два пътя, тъй като те са безсмислени – преминават през него и за това избира FBCD като нов маршрут. Другите алгоритми с вектор на разстоянието често правят погрешен избор, тъй като те не могат да съобщят кой от съседите има независим маршрут до местоназначението и кой няма.

16. Маршрутни протоколи RIP. Рутери.

RIP (routing information protocol) e широко използван маршрутизиращ протокол с вектор на разстоянието. Той е подходящ предимно за малки мрежи, в които относително рядко настъпват промени в топологията.

Всеки ред в маршрутната таблица на RIP маршрутизаторите съдържа информация за направлението, следваща стъпка към това направление и метрика. Метриката обозначава разстоянието в стъпки до местоназначението, т.е. метриката използвана от RIP протокола е брой хопове.

Както повечето маршрутизиращи протоколи, RIP също използва таймери. Обикновено на всеки 30 секудни се изпраща копие на маршрутната таблица към съседните маршрутизатори. Този интервал се задава от таймера за обновяване (route update timer) и е общ за всички маршрути. Таймерът за невалиден маршрут (route timeout timer) определя интервала от време, след който даден маршрут се счита за невалиден, ако маршрутизаторът не е получил съобщения за него. Когато даден път бъде отбелязан като невалиден, се изпращат съобщения с тази информация към съседните маршрутизатори и се преустановява използването му. Тези съобщения се изпращат до изтичането на таймера за изтриване на маршрут (garbage-collection timer), след което пътя се изтрива от маршрутната таблица.

Първата версия на RIP не поддържа подмаски, т.е. от гледна точка на IP не поддържа подмрежи, затова в края на 80-те години се разработва втора версия на RIP. Формата на пакетите на версия RIP-2 е следния:

Първите три полета Command, Version и Routing domain представляват заглавната част на пакета, а останалите шест полета съдържат данни за маршрути и комбинация от тях може да се повтаря до 25 пъти в един RIP-2 пакет. За пренасяне на информацията от по-големи маршутни таблици се използват няколко RIP-2 пакета.

Полето Command указва дали пакетът съдържа заявка или отговор. Командата “Заявка” изисква отговарящата система да изпрати цялата или част от маршрутната си таблица. Командата “Отговор” представлява отговор на получена команда “Заявка” или периодично съобщение за обновяване на маршрутите, в което се включва цялата маршрутна таблица.

Полето Version указва версията на протокола, за RIP-2 тази стойност е 2.

Полето Routing domain не се използва.

Полето Address family е идентификатор на адресна фамилия. Въвежда се идентификация на група от маршрутизатори. Всички маршутизатори в една група имат една и съща адресна фамилия.

Полето Route tag указва дали информацията за даден маршрут произхожда от RIP или от друг вътрешен или външен маршутизиращ протокол.

Полето IP address съдържа IP адреса на мрежа или хост, която представлява местоназначението на описвания маршрут.

Полето Net mask е мрежовата маска, отнасяща се за горния IP адрес.

Полето Next hop IP address съдържа IP адрес на най-близкия машрутизатор, към който ще се изпрати пакета.

Полето Metric указва броя хопове до съответното местоназначение и може да има стойност от 1 (директно свързана мрежа) до 16 (недостъпен маршрут – безкрайност).

В първия от 25-те записа на RIP-2 пакета полето Address family може да има стойност 0xFFFF, която указва че следва идентификационна информация за подателя на пакета. В този случай полето Route tag определя използвания алгоритъм, а следващите 16 байта съдържат парола. Използването на този механизъм намалява максималния брой маршрути в един RIP-2 пакет на 24.

При промяна в топологията на мрежата се налага всички маршрутизатори да преизчислят своите вектори на разстоянията и да достигнат до непротиворечиво описание на новата топология. За увеличаване на скоростта на сходимост на RIP се използват различни методи, например разделяне на хоризонта. Тези методи намаляват вероятността за поява на цикли в маршрутите, но не могат да гарантират отсъствието им.

Максималният брой хопове в RIP е 15. Всяко местоназначение, което е на разстояние над 15 хопа се приема за недостижимо. Това прави невъзможно прилагането на RIP в мрежи с диаметър над 15 хопа, но ограничава ситуацията “броене до безкрайност”, при която могат да се получат цикли в маршрутите.

Във версията RIP-2 са избегнати някои от недостатъците на RIP-1, но тя продължава да бъде приложима само в малки мрежи поради ниския максимален брой хопове и сравнително ниската и скорост на сходимост. Въпреки наследените от RIP-1 недостатъци и наличието на протоколи, в които те са избегнати, протоколът RIP-2 продължава да се използва, тъй като е лесен за реализация и конфигуриране и се нуждае от сравнително малко машинни и мрежови ресурси.

Internet е съвкупност от физически различни мрежи, които са обединени посредством общ протоколен стек, така че логически да се формира една обща мрежа. Най-лесният начин да се изгради такава мрежа е чрез свързване на две или повече мрежи чрез маршрутизатор (router). Маршрутизаторът представлява специализирано устройство, което дава възможност за свързване на различни типове физически мрежи. Основни функции на маршрутизатора са определяне за всеки получен пакет на най-добрия маршут до хоста-получател на пакета и препредаване на този пакет към следващия маршрутизатор по този маршрут. Последният маршрутизатор от пътя препредава пакета директно към хоста-получател. Информацията за най-добрите маршрути се съхраняват в маршрутни таблици. За определяне на най-добрия маршрут маршрутизаторите обменят помежду си информация, като за оценка използват различна метрика. Обикновено тази метрика включва следните величини: дължина на пътя, надеждност, закъснение на пакета при предаването от източника до получателя, пропусквателна способност на комуникационните линии, натоварване на маршрутизаторите, цена на комуникационните линии.

10. Преобразуване на IP адреси и физически адреси в локални мрежи – ARP и RARP.

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. При опаковането в Ethernet кадри на IP дейтаграми, за всяка от които е известен IP адреса на хоста-получател, в полето “адрес на получателя” на Ethernet кадъра трябва да се запише Ethernet адреса на съответния хост. За установяване на съответствието между IP адреса и Ethernet адреса на хостовете в локалната мрежа се използва протокол за право преобразуване на адресите ARP (address resolution protocol).

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 полетата “Ethernet адрес на подателя” и “IP адрес на подателя” са записани съответните адреси 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 (в шестнадесетичен вид) 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. Хостът може да използва и адреси, записани в конфигурационен файл.

Освен това всеки хост при първоначалното си стартиране уведомява чрез broadcast съобщение от вида “I am X.X.X.X and my Ethernet adress is hh:hh:hh:hh:hh:hh”, където X.X.X.X и hh:hh:hh:hh:hh:hh са съответно IP адреса и Ethernet адреса на хоста, всички останали хостове в локалната мрежа, които ще запишат тази информация в своите кешове.

Чрез ARP могат да се определят физическите адреси само на хостове, които са включени в локалната мрежа и имат IP адреси от IP мрежата (подмрежата) на изпращача. Дейтаграмите, чийто получател е хост от друга IP мрежа (подмрежа), се изпращат към маршрутизатора, включен в локалната мрежа. Неговият Ethernet адрес се получава чрез ARP заявка, ако не е кеширан. Този маршрутизатор избира маршрут и препраща дейтаграмата към нейния получател.

Протоколът RARP (reverse address resolution protocol) е за намиране на IP адреси по Ethernet адреси. Обикновено IP адресът на хоста е записан в конфигурационен файл, който се намира на твърдия диск на машината. При първоначално зареждане на операционната система файлът се прочита от твърдия диск и хостът научава своя IP адрес. В случай, че в локалната мрежа е включена машина, която не притежава собствен твърд диск, за определяне на нейният IP адрес се използва RARP протоколът. За целта в мрежата трябва да е включен хост, който функционира като RARP сървър. Този сървър съхранява съответствието между Ethernet и IP адреси на станциите в мрежата. Действието на RARP се основава на наличието на уникален физически Ethernet адрес на всяка система в локалната мрежа. При инициализиране на машината без твърди дискове RARP протоколът прочита този адрес от интерфейсната карта и предава до всички станции в мрежата пакет-заявка. RARP сървърът отговаря на тази заявка, като в пакета-отговор се съдържа IP адресът, съответстващ на изпратения Ethernet адрес.

Протокола DHCP- dynamic host configuration protocol се изпълнява за разпределяне на IP адреси при компютри в които няма статичен такъв. Единия начин за едно устройство да се определи IP адрес е той ръчно да се въведе, заедно със subnet mask и адрес на рутера. Ако не е хубаво този компютър да е със статично IP , то тогава се изпълнява DHCP протокола. За да работи това трябва да има DHCP сървър в конфигурация с него. Има пространство от динамични адреси, които той може да им раздава в определен срок. На този сървър трябва да се каже диапазона, който се раздава за статични IP адреси. Трябва да има DHCP агент, който знае къде е DHCP сървъра. Може да има повече от един DHCP сървър в конфигурацията.  
В началото клиента нищо не знае. Първата му работа е да открие дали има DHCP сървър и да получи предложения за разни адреси. Процедурата по получаване на един адрес е четирифазна. Новия клиент на сървъра означен със (?) прави в началото UDP datagram, в която негов source address е 0.0.0.0 защото е абсолютно бос. Портът, по който се пита за DHCP е порт 68. Той се нарича bootstrap client. Destination address е broadcast адрес със порт 67 – това е порта на DHCP сървъра. Типът се нарича DHCP discover. Питащия слага номер на транзакцията, в случая клиента и агента са в рамките на един сегмент, тоест тази UDP datagram е затворорена в 1 broadcast кадър. Този broadcast попада в агента, който е маршрутизатор и този агент го пуска в другия сегмент. Този агент, освен че е маршрутизатор знае и адреса на DHCPсървъра. За discovery този сървър получава кадрите и си записва номера на транзакцията и пуска своята оферта. Затова има предварително разузнаване и офериране. В тази оферта се съдържа IP адрес, който отговаря на този номер на транзакция. Устройство 3 трябва да получи отговора и да разпознае че това е неговия адрез. Може клиента да получи няколко оферти. Тези адреси ще бъдат коректни, но могат да са в различни диапазони. За два сървъра адресите не се пресичат. На новия клиент му се дава избор от адреси, които са специалиално предназначени за тази цел. Има и друга възможност DHCP да помни какви адреси от кого са взимани и да предпочете да дава същия адрес, особено ако включването и изключването стават често. Клиента чака да събере оферти. Има и time out. Той взима решение към коя оферта да се насочи. Насочва се към офертата на сървъра, който е най близо. След като е оферирал той прави отново request, в който слага адреса, който иска. Пак го подава като broadcast и вкарва в него разни чисълца. Това се прави, защото може да е изтекла офертата. Не винаги DHCP пази предложените адреси. Сървъра му предоставя този адрес като му дава DHCP acknowledge със получения адрес и клиента си конфигурира настройките с него. Тези 4 фази се правят, защото иначе могат да се създадат различни конфликтни ситуации.  
Важно за интернет провайдърите е да разполагат с динамично пространство за да не си конфигурират сами клиентите настройките. DHCP сървъра си води сметка какъв адрес е дал, няма да се получи IP conflict по този начин.

**8.Мрежов протокол IPv4-адресация,подмрежи и маски.CIDR**

*IP, автономни системи*

**IP** е основен и един от най-важните протоколи в съвременните компютъни мрежи. Той е протокол от трето, мрежово (network) ниво, създаден някога за обмен между различни мрежи. Основната му цел е да свързва **хостове** (станции, компютри) от **различни мрежи** (да припомним: ниво 2 - обмен **в локална мрежа**; ниво 3 - обмен между устройства, **независимо в коя мрежа се намират**; ниво 4 - обмен между **процеси, работещи на съответните хостове**). С течение на времето доста от идеите, залегнали в него, се изменят.

В сегашната обстановка, комуникационната подмрежа, в която работи IP, не е хомогенна - дели се на **автономни системи** (AS) - това са, грубо казано, множества от отделни (локални) IP мрежи, обикновено администрирани от едно и също ръководство. Internet представлява конгломерат от такива AS. Вътре в AS, маршрутизатори (рутъри) направляват трафика между отделните потребители, наричани **хостове**. Между мрежите работят отново рутъри, наричани**гранични**. От административна гледна точка, една автономна система е множество от хостове и рутъри.

Във всяка AS се работи по определени правила. Някои от най-важните са:

* С какви размери на пакетите се работи?
* Как вътрешните рутъри се разбират един с друг?

Напомняне: AS е набор от локални мрежи, между които трафикът се **маршрутизира** (направлява). Локалната мрежа (мрежов сегмент, collision domain, etc.) емрежа с **пряка селекция** - абонатите се "виждат" един друг през канала, или се прилага съвсем проста маршрутизация.

В IP е установено правилото, че всяка AS ще поддържа размер на пакета поне [576 байта](http://fixunix.com/tcp-ip/364013-576-bytes.html). Отново напомняне: **пакет** и **дейтаграма** често се ползват с подобни и еднакви значения, тъй че и с двете може да се опише ПБД от ниво 3.) 2 AS могат да ползват различни размери на дейтаграмите (>= 576 B), което ще наложи по-големите пакети да се разбият на части (да се фрагментират).

*Формат на IP пакета*

IP дейтаграмата се дели на 32-битови думи. Всеки пакет започва с IP хедър, форматиран по следния начин:  
Описание на важните полета:

* **Version** - версията на протокола. За сега използвания носи стойност 4.
* **IHL** - дължина на IP хедъра, изразена в брой 32-битови думи (между 5 и 15). 15 е недостатъчно, но така е останало от едното време - за 60те 20 B хедър е изглеждало огромно, при обичайна дължина на пакетите от други видове мрежи около 128 B. В ARPA обаче се е мислело далновидно - за преминаване през различни IP мрежи - което е помогнало много за създаването на днешния интернет.
* **Type of Service** - указва какви данни се пренасят в пакета, като смисълът е какво обслужване трябва да получи - приоритети и т.н. . В момента няма особено значение, не се разглежда от рутърите на големи производители като Cisco, затова и тук няма да се разглежда.
* 2 неизползвани бита.
* **Total Length** - Общата дължина в брой октети. Тъй като в Ethernet е заложено максималният обем данни в един кадър да бъде 1500 B (MTU = 1500), обикновено не се ползват пакети с по-голяма дължина. Ако допустимото от някоя AS е по-малко, то пакетът се реже на части.
* **ID** - 16-битов идентификатор, обикновено цикличен (65535++ -> 0). Сравнително голямо число, целта му е дълго време да се запази уникалността му. Стойността се задава от подателя ("емитента") на пакета, и никое междинно устройство няма право да я пипа.
* Един неизползван бит.
* **DF** - Don't fragment. Забранява на междинните устройства да разделят пакета на части. Ако такъв пакет стигне мрежа с по-малък максимален размер, то той се отхвърля ("дропва").
* **MF** - More Fragments. При фрагментиране на дейтаграмата, всички нови дейтаграми, образувани от нейните фрагменти, носят нейния ID номер (един и същ). Тогава във всички от тях, без последната, се поставя MF флаг, указващ на получателя, че следват още данни (разпознават се по същия ID).
* **Fragment offset** - Използва се при фрагментиране. Указва отместването на данните от фрагмента (в байтове) спрямо началото на първоначалните данни. Размерът на фрагмента е кратен на 8, с изключение на последния фрагмент.
  + Когато една дейтаграма е цяла и е позволено да се разбива на части, то DF = 0 и Offset = 0. След разбиването:
  + В първия пакет: MF = 1, Offset = 0.
  + Във втория: MF = 1, Offset = дължината на предишния фрагмент.
  + В n-тия: MF = 1, Offset = сума от дължините на предходните.
  + …
  + В последния: MF = 0
  + Тъй като за рутърите фрагментите се явяват самостоятелни дейтаграми, то те могат да ги насочат по различни маршрути. Еднаквият ID и стойността на Offset помагат на получателя да си сглоби първоначалния пакет.
* **TTL** - Time to live. Ограничава престоя на дейтаграмата в мрежата и предпазва от безкрайни цикли или неефективна маршрутизация. Първоначалната идея е то да бъде стойност в секунди, като мрежовите устройства я измерват при получаване и намаляват с 1 всяка секунда. При 0 пакетът става невалиден и се унищожава. Оказва се обаче, че е много тежка задача за рутърите да мерят времена. Затова TTL обозначава просто максималния брой устройства, през които дайтаграмата може да премине. Всяко междинно устройство намалява стойността с единица. Така, когато някой рутър получи дейтаграма с TTL = 1, той го намалява на 0 и изхвърля пакета.
* **Protocol**: указва кой протокол от четвърто (транспортно) ниво е капсулиран в пакета. Теоретично третото ниво не би трябвало да се интересува от протокола от по-горно ниво, но тук се прави компромис. [Възможните стойностти](http://www.iana.org/assignments/protocol-numbers/) са TCP, UDP, ICMP, други.
* **Header checksum** - контролна сума на хедъра.
* **Source, Destination Address** - 2 32-битови стойности, представляващи IP (логически) адреси на хостовете. Разделят се на две части: **мрежов адрес**(**номер на мрежата**) и **номер на хоста** (повече за това - надолу). Мрежовата част се използва от рутърите за намиране на подходящ маршрут.

*IP адрес, класове адреси*

IP адресът е ключов елемент от IP мрежата. Неговата цел е да идентифицира уникално своя хост - забележете, че говорим за идентификация на високо, дори световно, ниво. Всеки хост в мрежата има 1 или повече адреси. Адресът се обработва от мрежовото оборудване като 32-битово двоично число. Това представяне не е удобно за хората, затова е въведен **точковият запис** (точкова нотация) - запис от вида **A.B.C.D**, където A, B, C, D са числа между 0 и 255.

Както казахме, IP адресът се разделя на две части - мрежова и хостова. От самото начало, допреди известно време, начинът за това е бил следният: Определят се няколко класа мрежи. Всеки клас използва определен брой битове от адреса за мрежова, а останалите - за хостова част. Класовете:

* Клас **A**: адресът започва с **0**. Първите **8** бита указват мрежовата част, другите **24** - хостовата. 27 = 128 мрежи с по 224 = 16 777 216 хоста във всяка от тях (първият бит е строго определен). **Диапазон: 0.0.0.0 до 127.255.255.255**
* Клас **B**: адресът започва с **10**. Първите **16** бита указват мрежата, вторите **16** - хоста. 214 = 16384 мрежи с по 216 = 65536 хоста. **Диапазон: 128.0.0.0 до 191.255.255.255**
* Клас **C**: адресът започва с **110**. **24** бита указват мрежата, **8** - хоста. 221 = 2 097 152 мрежи с по 28 = 256 хоста. **Диапазон: 192.0.0.0 до 223.255.255.255**
* Клас **D**: адресът започва с **1110**. Запазен е за **multicast**-адресация. Тези адреси **не могат да се задават** като уникални адреси на мрежови устройства.**Диапазон: 224.0.0.0 до 239.255.255.255**
* Клас **E**: адресът започва с **1111** (т.е., каквото е останало). Запазен за изследване и експериментали цели. Тези адреси **не могат да бъдат задавани** на хостове в IP мрежите. **Диапазон: 240.0.0.0 до 255.255.255.255**

При създаването си адресното пространство е изглеждало огромно: 232 = 4 294 967 296 адреса (имайте предвид, че протоколът е създаден за ARPAnet). Случва се така обаче, че недомислените IP класове разхищават [огромна част от пространството](http://edge.networkworld.com/subnets/cisco/chapters/1587054620/graphics/01fig36.jpg). За илюстрация, да разгледаме класа А. 126 мрежи съдържат половината от цялото адресно пространство, а [по-голямата част от тези мрежи](http://en.wikipedia.org/wiki/List_of_assigned_/8_IPv4_address_blocks) са собственост на американски организациии и компании, закупили ги още преди години, които не могат да ги оползотворят. B-класът също е разпродаден (*всъщност в момента не се делят адресите по класове A, B, C, но доцентът не е наблегнал на този факт*).

*Маршрутизиране, регулация, subnetting, supernetting*

Маршрутизацията в IP мрежите се извършва само по мрежовия номер (мрежова част, мрежов адрес) - маршрутизаторите не се интересуват за кой от хостовете става дума. Подадено от единия краен абонат съобщение се прехвърля от мрежа в мрежа посредством поредица от рутъри, като последният (надяваме се!) е свързан в мрежата на получателя (другия краен абонат), на когото го доставя директно в мрежовия сегмент.  
Рутърът има поне 2 интерфейса, като на всеки интерфейст стои различен мрежов адрес.

**Опциите** на IP протокола се отнасят до маршрутизирането на дейтаграмата. Те могат да бъдат от 0 до 10 32-битови думи. В днешно време те се използват рядко. Някои от тях са:

* **Security** - в ранните дни, разграничавало частта от ARPAnet, принадлежаща на военните, от гражданската.
* **Strict source routing** - Твърдо задава маршрута, по който трябва да се движи дейтаграмата. След тази опция следват полета, задаващи адреси на рутъри. Поради ограниченията обаче, това означава до 9 адреса, което много често не е достатъчно.
* **Loose source routing** - Зададени са първите няколко адреса, през които трябва да се премине (до 9).
* **Record route** - Записва се изминатият маршрут. Полето се модифицира от маршрутизаторите. Отново, до 9 адреса.
* **Timestamp route** - Записва се адресът на рутъра и времето, когато пакетът е минал оттам. До 4 записа (крайно недостатъчно).

Рутърът може да бъде настроен да предава broadcast-пакетите по някой интерфейс, или да ги ограничава в сегмента. При препредаване, пакетът ще се декапсулира, адресът ще се прочете, след което пакетът ще се капсулира наново, за разлика от препредаването при устройствата от ниво 2.

След образуването на ARPAnet, тя почва да расте и след време се превръща в множество свързани американски мрежи, а след още повече време - в Интернет. С увеличаването на размера, става все по-трудно да се поставят уникални адреси, и то по такъв начин, че да могат да се администрират. Затова се учредява организация (IANA - Internet Assigned Numbers Authority), която се заема със задачата в Интернет да не се дублират адреси. Желаещите да получат адреси - Интернет-доставчици или големи корпорации - правят заявка пред IANA, която им продава мрежи (*класови мрежи - в момента се продават адресни диапазони, ама това не сме го обсъждали*). Вътре в тези мрежи, клиентът може да си разпределя адресите на хостовете, както намери за добре.

За да може една организация да си администрира успешно адресите, или за да отговарят те на нейната вътрешна йерархия, понякога се налага тя да ги структурира в отделни участъци, между които трафикът се маршрутизира. Тези участъци се наричат **подмрежи** (subnets). За целта, от хостовата част се "заемат" допълнителни битове, които идентифицират подмрежата (на практика, те отиват към мрежовата част). Subnetting се прави главно в огромните мрежи от клас A.  
Възниква въпрос: колко бита от даден адрес определят мрежата, подмрежата и хоста? Тук за пръв път се въвежда терминът **подмрежова маска** (subnet mask). Маската е 32-битово число, което в двоичен запис започва с блок от единици и завършва с блок от нули. Броят единици е равен на броя битове, отговарящи за мрежа и подмрежа, докато броят нули е равен на броя на хостовите битове. Записът е такъв, тъй като хостовете определят мрежата и подмрежата, в която се наира даден адрес, извършвайки логическо **AND** върху него и съответната му маска. След това адресът се сравнява със собствения мрежов адрес, извлечен по същия начин.

Маските се записват в точкова нотация, подобно на IP адресите. Например, за една клас B мрежа, тази маска би означавала **4** подмрежи:  
**255.255.192.010** = **11111111.11111111.11000000.000000002**  
Всеки клас мрежи си има стандартна подмрежова маска (която не е била нужна, преди да се въведат подмрежите):

* A - **255.0.0.0**
* B - **255.255.0.0**
* C - **255.255.255.0**

По-кратък запис на маската е чрез **префикс** - обратно на името си, той се записва след адреса и указва колко бита от 32-та са мрежови. Например, адресът**152.121.183.34** с маска **255.255.248.0** би се записал като **152.121.183.34 /21**.  
Възможно е подмрежите допълнително да се "надробяват" на други подмрежи. В IP маршрутизацията се извършва по IP адрес, а при subnetting дейтаграмата се придвижда до определена част от мрежата.

Съществува и обратната концепция: **supernetting**. При него се увеличава броят хостове чрез обединяване на две или повече мрежи (броят е число, кратно на 2). Това не се позволява при класова адресация (по ред причини, свързани най-вече с маршрутизацията). Прилага се при **CIDR** (Classless Inter-Domain Routing) - схема на адресация, при която мрежи представляват не класови мрежи, а адресни блокове с определна маска (например, **202.135.64.0 /27** - ако не е ясно, разпишете си го в двоичен вид и вижте откъде почват само нули). В безкласовите мрежи работят безкласови протоколи за маршрутизация (маршрутните протоколи изграждат "таблица на маршрутите", според която рутърът насочва пакетите). Необходимо условие за supernetting е субектът да притежава всичките обхванати от мрежовия отрязък класови мрежи. За тази цел са заделени няколко мрежови диапазона:

* 194.0.0.0 - 195.255.255.255
* 198.0.0.0 - 199.255.255.255
* 200.0.0.0 - 201.255.255.255
* 202.0.0.0 - 203.255.255.255

*Това е стара информация - повече за*[*CIDR*](http://en.wikipedia.org/wiki/Classless_Inter-Domain_Routing)  
В момента **граничните** рутъри (тези между отделните AS) работят с класово адресиране. *Това също не е вярно - но вие знаете какво да пишете…*

**9.Специални IPv4 адреси.NAT.ICMP**

С неимоверното разширяване на ARPAnet и Internet, адресното пространство започва да се изчерпва.  
За решението на този проблем има различни нововъведения, които отлагат неизбежния край. Едно от тях са така наречените **частни мрежи**. Във всеки от трите класа A, B, C, е заделен диапазон от мрежи за частни нужди, както следва:

* A: мрежов адрес **10.0.0.0** - 1 мрежа. **Диапазон: 10.0.0.0 - 10.255.255.255**
* B: мрежови адреси **172.16.0.0 до 172.31.0.0** - 16 мрежи. **Диапазон: 172.16.0.0 - 172.31.255.255**
* C: мрежови адреси **192.168.0.0 до 192.168.255.0** - 256 мрежи. **Диапазон: 192.168.0.0 до 192.168.255.255**

Адресите от частните обхвати са предвидени да се използват в мрежи, за които е сигурно, че няма да имат достъп до интернет, тъй като те със сигурност ще се дублират, ако се появят в световната мрежа. Възможен е достъп на тези машини до глобалната мрежа чрез механизъм, наречен **NAT**, който ще се разглежда в по-късна теми.\*\*

**Мрежовият адрес** представлява номера на мрежата, а хостовата част е запълнена с 0. Пример: **123.160.14.24** се намира в мрежа **123.0.0.0** от клас A. Ако мрежовият номер се допълни с 1, се получава **мрежовият broadcast адрес** на съответната мрежа. В случая - **123.255.255.255**. Мрежовият адрес и broadcast-адресът на мрежата не могат да се задават като хостови адреси в една мрежа. Затова, реално броят адреси е **2n - 2**, където **n** е броят битове в хостовата част.

Някои от адресите са запазени за специални цели. Адресът **0.0.0.0** служи за "локална идентификация" - на практика се използва, когато хостът все още няма свой уникален адрес (например, при DHCP разпределение). Адресът **255.255.255.255** е **broadcast** адрес - някога идеята е била той да бъде broadcast до абсолютно всички хостове, в момента той бива ограничаван до мрежовия сегмент. Използва се, например, когато хостът не знае своя мрежов адрес (DHCP).

В момента се разработва, и, донякъде, прилага, **IPv6** - нов вариант на IP. В него адресите са 4 пъти по-дълги - 128 бита, което прави **огромно** адресно пространство - **340 282 366 920 938 463 463 374 607 431 768 211 456** адреса, или средно **667 x 1021** адреса на **квадратен метър земна повърхност**. [Не вярвате ли ?](http://spectrum.ieee.org/tech-talk/semiconductors/devices/oops_how_many_ip_addresses)

*ICMP*

За да работи ефективно IP мрежата - въпреки, че IP е **unreliable** и **connectionless** - е добре да се получават съобщения за възникналите при предаване на мрежово ниво грешки. За това се грижи **ICMP** - Internet Control Message Protocol. Неговите пакети имат собствена стойност на полето **Protocol**, а в данните си съдържат тип и код на съобщението за грешка. Някои от кодовете:

|  |  |  |
| --- | --- | --- |
| Тип | Код | Значение |
| 0 | 0 | Echo reply |
| 3 | 0 | Destination network unreachable |
| 3 | 1 | Destination host unreachable |
| 3 | 6 | Destination network unknown |
| 3 | 7 | Destination host unknown |
| 4 | 0 | Source quench ("затваряне" на източника) |
| 8 | 0 | Echo request |
| 11 | 0 | TTL expired in progress |
| 12 | 0 | Invalid IP header |

Как работи ICMP? Изпраща се ICMP Echo (това прави програмата/командата **ping**). Получателят може да отговори, а може и да не отговори - в зависимост от настройките за сигурност и наличието на firewall. Ако съобщението не достигне до получателя, то ще отговори (няма да отговори) някой междинен рутър. Във върнатото съобщение за грешка се съдържат първите 8 байта от изпратената дейтаграма.

NAT

IP адреса може да се преобразува от една мрежа в друга – това е така наречения NAT – network address translation. Тази транслация се прави от специялен маршрутизатор на изхода на една локална мрежа. Той освен маршрутизация прави и транслация на адресите. От практическа гледна точка има две предимства – намалява се броя на използваните световни IP адреси. Под един адрес могат да се скрият много абонати. СЪс това скриване се прави и защита от неоторизиран достъп.

**23.Електронна поща**

Електронната поща, обикновено наричана просто [e-mail / email / eMail](http://communication.howstuffworks.com/email.htm) е много стара концепция. Много. Първите системи за обмяна на съобщения между компютри са се появили не по-късно от 1966 година. Употребата на електронна поща в интернет е един от факторите за неговото развитие. E-mail системи са разработвани от организацията [ISO](http://www.iso.org/iso/home.htm), от комитета [CCITT](http://en.wikipedia.org/wiki/ITU-T), от различни американски университети, и на [други места](http://en.wikipedia.org/wiki/Semi_Automatic_Ground_Environment). Поради това съществуват и множество стандарти - например CCITT [X.400](http://searchexchange.techtarget.com/sDefinition/0,,sid43_gci213396,00.html).  
E-mail системата се основава на TCP/IP пакетна мрежа, но действа на принципа на **комутацията на съобщения**. Съобщенията се предават от субект на субект, като при приемане се съхраняват (буферират) и се препращат при първа възможност, докато не достигнат получателя.. Доставянето е **асинхронно** - не е нужно адресатът да е онлайн, за да му бъде доставено съобщението.Най-разпространената в момента форма на електронна поща се основава на протокола [SMTP](http://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol).

*Структура на електронно съобщение*

Едно e-mail съобщение се състои от **плик (envelope)**, **хедър** и **тяло** на съобщението.

**Плик (envelope)**

Предназначението на плика е аналогично на това на плика в традиционната поща (напоследък често наричана snail mail): той съдържа минималното количество информация, необходимо на пощенските сървъри да доставят съобщението до получателя, а също и как да го третират по пътя. Някои от полетата, съдържащи се в плика, са:

* name
* address
* priority
* encryption

**Хедър**

Хедърът съдържа служебна информация за писмото. Хедърът включва:

* **To** - E-mail адрес на първичен получател (recipient). При единствен получател, това е той.
* **Cc** - E-mail адрес на вторичен получател. Идва от *carbon copy* и е взето от дните, когато копия на писма са се правели с индигова хартия.
* **Bcc** - E-mail адрес на третичен получател. Идва от *blind carbon copy*. Теоретично, би трябвало получателите в **Bcc**: да получат копие от писмото, без тези в**To:** и **Cc:** да знаят това. На практика, това **не се изисква** по стандарт. Различните имплементации го обработват по различен начин.
* **From** - Името на лицето / програмата, автор на писмото.
* **Sender** - E-mail адрес на актуалния изпращач (адрес, от който всъщност е тръгнало писмото).
* **Received** - Едно или много такива полета, отразяващо движението на писмото през сървърите. Всеки запис указва една транзакция между MTA-та.
* **Return-Path** - Подобно на **Received**, съдържа информация за източника на съобщението и за обратния път до него[2](javascript:;).
* **Date** - Дата на изпращане на писмото.
* **Reply-To** - E-mail адрес, на който да се изпрати отговорът. Има смисъл, когато писмото не е дошло от автора - например, в пощенските списъци[3](javascript:;).
* **Message-Id** - Уникален за изпращача идентификатор на съобщението.
* **In-Reply-To** - Идентификатор на съобщението, на което настоящето отговаря.
* **Keywords** - Ако присъства, съдържа ключови за писмото думи.
* **Subject** - Тема (предмет) на съобщението.
* **X-<…>** - Нестандартни хедъри, обикновено специфични за имплементацията. Някои от тях се използват много често и са *de facto* стандартни.

**Тяло**

Тялото съдържа полезната информация на съобщението - текст и/или двоични данни.  
Първоначално в тялото се е допускал само текст в ASCII. Поради недалновидност, обаче, е допусната височайша глупост: предвидено е да се използва **ASCII-7**(първите 128 знака от ASCII таблицата, или нулевата страница). Това е било идеално за САЩ, но вгорчава живота на всички, които използват други символи. Години наред в държавите с друга азбука, например кирилица, съществуват проблеми с кодирането на символите, най-често поради съществуването на няколко различни схеми за борба със 7-битовото ASCII[4](javascript:;), и разчитането на писмото често е въпрос на шанс - дали получателят ще приложи същата кодировка, каквато и подателят.

За борба със седемте бита са измислени много начини. Най-масовата схема прекодира ASCII-8 в двоичен код, който изпраща като ASCII-7, а клиентът го обръща отново в ASCII-8. Тя се реализира по няколко начина:

[**Base64**](http://en.wikipedia.org/wiki/Base64)**encoding**

Текстът се разбива на части по 3 октета (24 бита), а те се делят на 4 групи по 6 бита, които могат да имат стойности между 0 и 63. След това групите се обръщат в ASCII-7:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Стойност** | 0 | … | 25 | 26 | … | 51 | 52 | … | 61 | 62 | 63 |
| **ASCII-7 символ** | A | … | Z | a | … | z | 0 | … | 9 | + | / |

Ако текстът не е достатъчно дълъг, първо се допълва със символи **=**.  
Кодирането е икономично (около 25% надбавка).

[**Quoted-printable**](http://kb.iu.edu/data/aepj.html)

Полезно за текст, съдържащ предимно ASCII-7 знаци. Те се запазват, а останалите се заместват с шестнайсетичната си стойност, предшествана от знак **=**, и цялото това се огражда в кавички. Например, главната кирилска буква **А** (ASCII 192 в [Windows-1251](http://en.wikipedia.org/wiki/CP1251)) се кодира като **"=C0"**.

**Двоично кодиране -**[**uuencoding**](http://en.wikipedia.org/wiki/Uuencode)

Наследство от Unix и неговите (стари) mail-системи. Кодира символите двоични числа, например в [BCD](http://en.wikipedia.org/wiki/Binary_coded_decimal) (Binary Coded Decimal).Все по-популярна става директната употреба на Unicode, без нужда от кодирания на символи.

**MIME**

Практиката с (пре)кодирането на информацията се оказва лоша - води до практическа неизползваемост на системата. Затова, след като електронната поща се разпространява извън САЩ, започват да се търсят други начини за представяне на съдържанието. В един хубав момент, в [RFC 1341](http://www.faqs.org/rfcs/rfc1341.html), допълнено от [RFC 1521](http://www.faqs.org/rfcs/rfc1521.html), и допълнено още безброй пъти, се появяват [MIME](http://en.wikipedia.org/wiki/MIME) (Multipurpose Internet Mail Extensions]. Това е разширение на e-mail стандарта, което описва вида на представяната в тялото информация. MIME има поне две много важни характеристики:

* Позволява пращане на съдържание, различно от обикновен текст - разнообразни двоични файлове, например.
* При текстово съдържание, указва начина на кодиране на символите.

Всъщност MIME е такава гъзария, че се възприема и от HTTP и WWW за указване на съдържанието[5](javascript:;) .  
MIME дефинира следните полета в хедъра:

* **MIME-Version**
* **Content-Description** - ASCII текст, описва съдържанието.
* **Content-ID** - подобно на Message-Id
* **Content-Transfer-Encoding** - Основно се ползват ASCII-7, ASCII-8, Base64, Quoted Printable, UU. Ползват се и други.
* **Content-Type** - Тип на съдържанието. Доста пълен списък има [тук](http://www.utoronto.ca/web/HTMLdocs/Book/Book-3ed/appb/mimetype.html). Най-разпространени (дадените стойности са малка част от възможния набор):
  + **text** - plain, richtext (например, .rtf - Rich Text Format; позволява неща като **bold**, underline, *italic*)
  + **image** - gif, jpeg
  + **audio** - basic
  + **application** - octet-stream, postscript
  + **message** - rfc822, partial (тялото съдържа част от съобщението) / external-body (тялото се предава отделно)
  + **multipart**
* И други.

*Компоненти на e-mail системата*

Първите системи, разглеждани в RFC 822, са свързани чрез малки, прости мрежи, в които работят специализирани, скъпи компютри, които непрекъснато са онлайн. Затова при тях съобщенията се доставят от краен потребител до краен потребител, директно от SMTP. Сега ситуацията е по-различна. Комуникацията се осъществява по модела **клиент - сървър**. Според функциите си се открояват няколко вида субекти: [MUA](http://en.wikipedia.org/wiki/Mail_user_agent) (Mail User Agent), [MTA](http://en.wikipedia.org/wiki/Mail_transfer_agent) (Mail Transfer Agent), [MDA](http://en.wikipedia.org/wiki/Mail_delivery_agent) (Mail Delivery Agent), [MSA](http://en.wikipedia.org/wiki/Mail_submission_agent) (Mail Submission Agent), описани в [RFC 5068](http://tools.ietf.org/html/rfc5068). Най-важните са MTA и MUA.

* MUA - Работи при крайния потребител (изпращач / получател). Създава и съобщения и ги изпраща към e-mail структурата. Получава от нея съобщения и ги предоставя на потребителя. Почти синоним на **e-mail клиент**. Най-често ползва протоколи като [POP](http://en.wikipedia.org/wiki/Post_Office_Protocol) версия 3, работещ по подразбиране на порт **25**, и[IMAP](http://en.wikipedia.org/wiki/Internet_Message_Access_Protocol), работещ по подразбиране на порт **143**. Чрез тези двата протокола се реализира взаимодействието **сървър -> клиент**[6](javascript:;).
* MTA - Основна част от SMTP, работи на e-mail сървърите. Грижи се за получаването, съхраняването, обработката и препращането на e-mail съобщения. Като негови подфункции могат да се обособят MSA и MDA. Реализира взаимодейстията **клиент -> сървър** и **сървър -> сървър**.

Обикновено процесът на изпращане и получаване на електронна поща протича така: MUA изпраща написаното от потребителя съобщение до своя пощенски сървър. Там съобщението се поема от MTA и се съхранява. Ако адресатът се обслужва от същия сървър, съобщението се съхранява в неговата **пощенска кутия**(mailbox) - структура, намираща се на сървъра, или директно - на потребителския компютър. Ако не, то съобщението се препраща, директно или през други сървъри (MTA-та), до пощенския сървър на клиента, където се съхранява в кутията му. E-mail системата **не работи** в реално време. По пътя си съобщението се съхранява от MTA-тата, в случай, че то не може да бъде препратено на момента, и се предава, когато това стане възможно. Затова изпратеното съобщение може да бъде получено моментално, а може да се забави няколко минути, а даже и няколко часа.[7](javascript:;)

*Работа на SMTP*

SMTP работи в следната последователност:

1. Създава TCP съединение с по-голям от 1023 source port и destination port 25. Извършва DNS resolving на домейна на получателя, получава от неговия DNS сървър IP адреса на пощенския сървър, който обслужва клиента, и използва това IP за сокета (който не помни: **ipaddress:port**). При успешно установяване, SMTP сървърът отговаря с **220 READY**. ([кодовете на командите](http://the-welters.com/professional/smtp.html) са подобни, ако не същите, на тези при FTP).
2. Клиентът праща **HELO *hostname***, където*hostname* е [FQDN](http://en.wikipedia.org/wiki/FQDN) на клиента. Сървърът отговаря с **250 OK**.
3. Клиентът праща **MAIL FROM: *address***, където *address* е e-mail адрес на изпращача, стойността на **reply-to** полето от хедъра. **250 OK**.
4. Клиентът праща **RCPT TO: *address | списък от адреси на един и същи сървър***. **250 OK**.
5. Клиентът праща **DATA**. **250 OK**.
6. Следва самото съобщение. Кодировката му се описва в хедъра. Краят на съобщението се обозначава с два нови реда, обозначени с **<CR><LF><CR><LF>**[8](javascript:;).
7. Клиентът затваря сесията с **QUIT**. Сървърът отговаря с **221 CLOSING**.

*И малко за POP3*

Post Office Protocol, версия 3. Използва се за достъп до съобщенията в пощенската кутия. Подобен на SMTP откъм команди. Най-голямата разлика е възможността за удостоверение (**authentication**) - чрез парола се контролира достъпът до пощенската кутия.

При POP3 процесът на четене протича така:

1. Клиентът установява връзка към пощенския сървър по порт 110. Сървърът отговаря с **OK**.
2. Клиентът праща потребителското си име и парола. Сървърът ги утвърждава с **OK** (или не ги утвърждава с **ERR**). Данните се предават в явен вид. Има възможност за криптиране / хеширане, но обикновено не се имплементира.
3. Клиентът подава команда. Сървърът я изпълнява и връща **OK**.
4. Клиентът подава **QUIT** за край на ссесията.

**24.Хипертекстов протокол**

Най-мощното приложение на Internet e WWW (world wide web). Като технология и идея е създадено от физика Тим Бърнър Лий през 1990 година с цел да улесни комуникацията на група учени.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. Всяка страница съдържа произволен брой URL референции (не се изисква тези страници да се намират на същия сървър).

Web значи паяжина – идва от страниците, които са пръснати и са свързани като паяжина чрез хиперлинковите връзки.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 (hyper text transfer protocol).

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-a 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 адресът 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, къде се намира, 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. Правенето на съединение означава resolving на URL-адреса за да се вземе IP адрес, с който да се направи TCP съединение. На самото TCP ниво се договаря трикратно съединението. След това се договаря HTTP сесията. С други думи, мотае се много служебен трафик – не е ефективно, ако от един сървър се четат няколко страници за всяка страница да се започва отначало.

От друга страна не може всяко обръщение към една страница да отваря сесия към някой сървър и да я държи за неопределено време – изглежда нормално сесията да приключи с изтеглянето на страницата. Също така, един сървър отговаря за много страници. Ако е претоварен, докато изпълни заявката ще мине много време. През цялото това време сесията стои отворена и browser-ът е блокиран.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 за 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.

Основен недостатък е, че не знае кога са обновени хиперлинковете. Затова е важно използването на виртуални сървъри.В момента се използват два HTTP сървъра – apache основно за UNIX и IIS на Microsoft.

HTTP browser-ите са:

* MOSAIC – първият browser;
* NETSCAPE – дълго време е бил най-добрият;
* INTERNET EXPLORER – най-масовият в момента;
* OPERA – счита се за най-бърз.

Езикът за написване на страници се нарича HTML (hyper text mark-up language). Това е език с набор от правила (тагове), в които се помества служебна част, която не се изобразява. В последствие в browser-а се правят виртуални машини, които интерпретират други езици като javascript например

**22.DNS.Резолвинг**

DNS (domain name system) разчита на раздробяването на имената на **домейни**. DNS се създава за нуждите на Internet, където имената са много и имат подструктура. В Internet обектите образуват йерархично дървовидно пространство. Създават се области от имена, като имената могат да са имена на други подобласти. Домейните представляват области от имена. Домейните са от първо, второ и трето ниво. Няма пречки да има домейни от четвърто ниво, но те почти не се използват.

Основният домейн е така нареченият **root** домейн. Той няма име и е един единствен. Под него се нареждат домейните от първо ниво, т.е. в root домейна стоят имената на всички домейни от първо ниво. Домейните от първо ниво исторически са получили фиксирани имена:

- com – комерсиални организации

- net – мрежови организации

- org – некомерсиални организации

- gov – правителствени организации

В началото всички те са в САЩ, но после в тях влизат още много имена на обекти извън САЩ, те нарастват твърде много и затова се въвежда друга голяма група от домейни на първо ниво, свързани с географското разположение по държави – uk, de, bg и др.

Цялостното име, което включва домейните и обекта се нарича **URL** (uniform resource locator). Пример за URL е

<http://www.fmi.uni-sofia.bg>

В това URL bg е името на домейна от първо ниво, uni-sofia е името на поддомейна на bg от второ ниво, fmi е името на домейна от трето ниво, www е web-сървъра от домейна fmi, http е името на протокола по който клиента се свързва към съответния обект.

Колкото са точките в едно URL, толковата са нивата на домейните без да се брои root.

DNS е йерархична именна система с три компонента – именно пространство (как се изграждат имената), resolver-и и именни сървъри (name servers).

Resolver-ите са абонатите в Internet, които знаят URL и искат да получат съответния IP адрес. Процесът на преобразуване се нарича resolving. Той се извършва от DNS протокола.

Именните сървъри са основните обекти на DNS протокола. Те съхраняват имената на домейните. Те са физически устройства, за разлика от домейните, които са логически компоненти на именната система. Класификацията на именните сървъри е следната:

* първични именни сървъри;
* вторични именни сървъри;
* главни именни сървъри;
* кеш именни сървъри.

Първичните именни сървъри управляват своя зона с информация за един или повече домейни. Зоната е основен атрибут и тя представлява част от домейн-базираното именно пространство. Зоналните файлове се намират локално в първичния сървър и се променят от него (той ги държи локално на диск). В една зона може да има най-много един домейн от първо ниво или част от него и няколко домейна от второ ниво. Зоналните файлове съдържат ресурсни записи – основното е съотвествие между имена и IP адреси.

Вторичните именни сървъри съхраняват копие на зонална информация, която се намир в първични сървъри. Със зонален трансфер първичните сървъри прехвърлят зонална информация във вторични сървъри. Първичните сървъри съдържат оригинала на зоналната информация, ако се прави промяна тя става в тях. Вторичните сървъри съдържат копие на тази информация – ако нещо се случи с първичния сървър да не се изгуби информация, също и за разпределение на натоварването. Един първичен сървър може да има няколко вторични сървъра.

Главните именни сървъри прехвърлят своите зонални файлове към вторични сървъри. Обикновено първичните сървъри са главни, но може вторичен сървър да е главен за друг вторичен сървър – това се прави с цел първичният сървър да не се грижи за обновяването на съдържанието на всички вторични сървъри.

Кеш именните сървъри нямат зонални файлове, разположени в локалното дисково пространство. Те са посредници – през тях минават заявки за resolving и ги препращат към първични и вторични сървъри. Кеш сървърът временно помни отговорите на преминалите през него заявки. Ако времето на отговора не е изтекло и в кеш сървъра дойде същото допитване, той връща запаметения отговор и не препраща заявката. Обикновено кеш сървъри се слагат пред защитни стени (firewall), които отделят вътрешна за предприятие мрежа от свободния Internet.

В DNS имената са глобални, за разлика от NETBIOS. Регистрирането на име не е автоматично, а става чрез специална заявка към съответна агенция. В класическите първични DNS сървъри информацията се вкарва ръчно. Напоследък се въведе и динамичен DNS, но в класическия си вариант DNS-системата е със статичен характер.

За да се използва системата на URL-имената в клиента (resolver) трябва да има агент, който да може да работи с URL. Този агент е началото на resolving процеса. Освен това в клиента трябва да има и малък кеш, в който да се съхранява информация за вече заявени и resolve-нати адреси за този клиент. Също така, клиентът трябва да разполага с адрес на DNS сървър, който отговаря за съответната област. Ако не разполага с такъв адрес, той трябва да може да попита чрез broadcast има ли именен сървър в мрежата.

Когато към агента се подаде URL за resolve-ане той първо проверява дали отговора не стои в кеша. Ако това не е така, той изпраща заявка до DNS сървър. DNS сървърът може да формира три типа заявки – рекурсивна, итеративна или инверсна.

При рекурсивна заявка DNS сървърът има прилежащ към него друг именен сървър. Този именен сървър също може да има кеш, който евентуално да съдържа отговора. Именният сървър може да съдържа отговора в своите зонални файлове. Ако и двата случая не са налице, но за именния сървър има конфигуриран друг именен сървър, той ще изпрати заявката към него и т.н.

Рекурсивни заявки се формират в защитени области, където има защитна стена (firewall). Чрез тях се изнасят заявки за resolve-ане извън защитената област. В един момент някой именен сървър по описаната верига може да направи рекурсивната заявка в итеративна.

При итеративната заявка именният сървър е в свободното Internet пространство. Той започва да раздробява съответното URL и постъпково, съгласно структурата на URL започва resolve-ането. Първо се изпраща заявка към root-сървъра, като се иска адреса на сървъра, който отговаря за домейна от първо ниво, който участва в URL-то. След това се праща заявка към сървъра от първо ниво за адреса на сървъра, който отговаря за домейна от второ ниво, участващ в URL-то и т.н.

**20.Tранспортен протокол TCP**

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. Това означава, че на приложната програма, която използва TCP, е гарантирано доставянето на предадената информация до получателя. От друга страна, UDP не е ориентиран към установяване на връзка и е ненадежден протокол, който не гарантира, че предаденото съобщение ще достигне до местоназначението си.

TCP протоколът представлява множество правила за осъществяване на надежден информационен обмен между приложните слоеве на два хоста. TCP установява логическа връзка “край-край” между две приложни програми, които в общия случай се изпълняват на два различни хоста. Данните, обменяни между приложните програми по протокола TCP, се интерпретират и обслужват като поток от байтове. TCP не вмъква и не интерпретира никакви разделители между логическите записи.  
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.

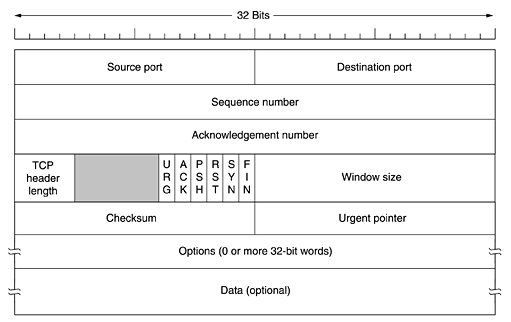
Всеки край на TCP връзката се идентифицира с IP адреса на съответния хост и с 16-битово число, наречено **номер на порт**, което определя съответната приложна програма, използваща тази връзка. Комбинацията от адреса на хоста и номера на порта се нарича **socket**. Всеки TCP сегмент съдържа номерата на 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. Съответствието между номер на порт и приложна програма се осъществява локално във всеки хост. Първите 1024 номера на портове са така наречените **well-known** портове, които са резервирани за най-често използваните стандартни приложни програми.

TCP осигурява на протоколите от по-горните слоеве възможност за двупосочно предаване на данните (пълен дуплекс), при което всеки сегмент може да се използва едновременно за пренос на данни и на информация за управление на обмена, съдържаща се в неговата заглавна част. Това означава, че всеки сегмент може, както да пренася данни от източника до получателя, така и да обслужва насрещния информационен поток чрез заглавната си част – например да извърши потвърждение за получаване на данните на вече приет сегмент. За да се осигури получаването на данните по реда на тяхното изпращане, всеки изпратен байт се номерира, а получаването на всяка последователност от байтове се потвърждава. Насрещните информационни потоци са относително независими в рамките на TCP връзката, което се отнася и до номерацията на байтовете. Данните се препредават, когато не е получено потвърждение за получаването им.

Протоколът TCP използва 32-битови поредни номера за идентифициране на всеки байт от данните, обменяни с приложното ниво. Във всеки TCP сегмент е записан поредния номер на първия байт от неговото поле за данни. Например, ако протоколът TCP предава данни с размер на сегмента 500 байта и в първия сегмент за началния байт данни е записал пореден

номер n, то във втория ще бъде записан пореден номер n+500, в третия пореден номер n+1000 и т.н.

Форматът на заглавната част на TCP сегмента е следния:



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. Максималната допустима дължина на полето Data е 65495 байта. Тя се получава от максималната допустима дължина на IP дейтаграмата 65535 байта, намалена с размера на задължителните полета на IP заглавната част (20 байта) и TCP заглавната част (20 байта).

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 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. Например, при успешно получаване на сегмент с размер на полето данни 500 байта и пореден номер на началния му байт n, към източника на този сегмент се изпраща TCP сегмент, в който потвърждението е с номер n+501.

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. Фактически с това поле се определя началото на полето Data в рамките на TCP сегмента.

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

* URG – валиден е указателят за спешни данни (Urgent pointer). Установяването на този флаг означава, че трябва да се преустанови обработката на получените данни, докато не се обработят байтовете, към които сочи 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, т.е. източникът не изчаква да се съберат данните за образуване на пълен сегмент с избрания размер и съответно получателят не чака запълването на приемния буфер;
* RST – сегмент, в който е установен този флаг, служи за прекратяване на TCP връзката. Използва се в случаите, когато връзката е нарушена (например, поради повреда в хоста) или когато се отхвърля невалиден сегмент или се отказва опит за установяване на връзка;
* 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. Поради двупосочния характер на информационния обмен това не означава, че TCP връзката е прекратена.

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, които не се осигуряват от останалите полета на заглавието. 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 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 този SYN сегмент от сървъра, като изпрати сегмент с потвърждаващ номер y+1.

При затварянето на връзката се има пред вид процедура, при която прекратяването на връзката става по начин, предотвратяващ загубата на информация. 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, ще наричаме 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.

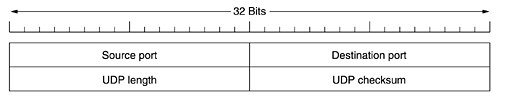
За асиметрично затваряне се изпраща сегмент с вдигнат флаг RST. То е необходимо когато по връзката се получи некоректен сегмент.

Ако предаден сегмент не бъде потвърден за определено време, TCP протоколът предполага, че е налица претоварване на мрежата и сегментът е изгубен, при което следва да се предаде отново. Времето на изчакване на потвърждение от получателя на сегмента се определя от таймер за препредаване. Излишното препредаване на сегменти ще доведе до засилване на претоварването, а по-голямото време на изчакване ще доведе до намаляване на скоростта на TCP връзката под възможностите на мрежата.

**21.UDP.Задръствания и оправление на потоците**

UDP е прост транспортен протокол за предаване на дейтаграми в мрежите с комутация на пакети. Той не осъществява надежден транспорт. Дейтаграмите се изпращат от източника без да се контролира дали са достигнали до получателя.

Форматът на заглавната част на UDP дейтаграмата е следния:



Полетата Source port и Destination port съдържат номерата на портовете, идентифициращи еднозначно изпращащия и получаващия процес.

Полето UDP length задава в брой байтове дължината на цялата UDP дейтаграма – заглавна част + данни. Минималната му стойност е 8, което съответства на UDP дейтаграма, състояща се само от заглавна част.

Полето UDP checksum е контролна сума, която се изчислява върху цялата UDP дейтаграма, заедно с някои полето от IP дейтаграмата, в която тя е опакована.

UDP има смисъл да се използва при мрежи с висока надеждност.

Рутърите (които са частен случай на мрежови хостове) трябва някак да се борят със задръстванията. За тази цел има 5 класически подхода:

*1. Пререгулиране на буферите*

Този подход най-добре се осъществява чрез [виртуални канали](http://fmi.wikidot.com/km3#toc4). В такива случаи се изпраща сервизен пакет за установяване на виртуален канал: в устройствата, през които минава пакетът, се записва номерът на виртуалния канал. След това този канал се ползва монополно чак до неговото разпадане в края на връзката.[2](javascript:;).

В добавка към виртуалния канал идва схемата за регулация на буферите: при установяване на канала, междинният хост си заделя 1-2 буфера за него. Тези буфери обслужват само този канал и нищо друго.Същевременно, по канала вървят единствено пакети, предназначени за него, за които пък каналът е единственият допустим маршрут. Получава се така, че пакетите се движат и пристигат в реда, в който са изпратени, и каналът не се задръства от чужд трафик.

Проблем се появява, когато твърде много буфери бъдат ангажирани за канали и не останат достатъчно, че да обслужват бъдещи канали. Тъй като установените канали могат да стоят свободни през голяма част от съществуването си, това означава пилеене на мрежов ресурс. Ако в това време бъде направен опит за установяване на нов канал, той ще се провали.

*2. Отстраняване на пакети*

При този подход хостът премахва пакети, когато преносната среда (мрежата) се претовари.  
Тъй като в някакъв момент източникът ще трябва да препрати изхвърления пакет, не е добре пакет да бъде изхвърлян към края на пътя си. Ако пък той съдържа прикрепено потвърждение, тогава става още по-лошо. Тези две неща важат най-силно за протоколите, които установяват връзки (connection-oriented) и се стремят да постигнат надеждна (reliable) комуникация (**IP не е такъв**).

За да се избегнат тези неприятни положения, трябва да не се стига дотам, че всички буфери да са разпределени за една и съща опашка. За целта, на всеки вход на хоста стои по един свободен буфер, в който се приемат пакети. При нужда от изхвърляне на пакети, новополучените се разглеждат, и ако не са важни и не съдържат ACK, се премахват. Ако пък са важни, се премахва някой от предишните пакети.

Свободните буфери се разпределят по опашки. Ако всичките отидат на една опашка, не е хубаво: на всички входни линии ще има буфер за приемане, но само от една опашка ще може да се предава. Затова се ограничава дължината на опашката. Ако броят свободни буфери е **k**, а броят изходящи интерфейси - **s**, едно възможно решение е всяка опашка да не става по-дълга от *ks*. След проучване, се оказва, че е достатъчно да няма опашка, по-дълга от *ks*√. Ограниченията се налагат локално, в устройството, поради което методът е прост и ефективен.

*3. Ограничаване броя на пакетите*

При този подход в мрежата се движат **разрешителни пакети** (permits). Мрежово устройство, което има буферирани пакети за изпращане, трябва първо да получи такова разрешително. При получаване разрешителното се унищожава и се изпраща един от чакащите пакети. Нови разрешителни се създават централизирано (от един мрежови център), или разпределено, между устройствата - при доставка до крайния получател, съответният рутър пуска ново разрешително. Идеята на схемата е по всяко едно време броят на пакетите с данни и на разрешителните пакети в мрежата да е константен. Това гарантира, че мрежата няма да се препълни с пакети.

Проблемите при такава логика на работа са няколко. Ако се случи така, че разрешителните започнат да изчезват (авария в устройството, проблем по трасето), намалява и общата пропускателна способност на мрежата. Освен това, разрешителните се движат по неопределен начин: една част от мрежата да е затлачена от пакети, но да няма достатъчно разрешителни, докато в друга част да има много разрешителни и малко пакети. Самите разрешителни също допринасят за натоварването на мрежата. Възстановяването на липсващите рарешителни е сложно, тъй като мрежовото устройство не знае колко разрешителни съществуват в момента. Устройствата пък трябва да могат да обработват (да се съобразяват с) тези разрешителни, което води до тяхното усложняване. Всичко това прави успешното осъществяване на тази идея трудно.

*4. Контрол над потока (flow control)*

Когато абонатът (крайният хост) трябва да изпрати голям трафик, обичайно е да се регулира темпото, с което той изпраща информацията. Пример за това са[механизмите на TCP](http://en.wikipedia.org/wiki/Transmission_Control_Protocol#Flow_control): при претоварване на получателя, той сигнализира на изпращача да забави темпото. Управлението обикновено се осъществява на OSI ниво 4.  
Управлението на трафика между крайните абонати, обаче, регулира трафика между крайните абонати. То влияе слабо върху натовареността в междинните устройства (рутърите, суичовете), тъй като там обработката се извършва на по-ниски нива.

*5. Ограничаване (управление) на входа*

ИЗвършва се автоматично регулиране - *обратна отрицателна връзка*: когато хостът усети усилено натоварване, той уведомява другия хост да забави темпото. Пример за такава връзка е ICMP Source Quench съобщението[3](javascript:;).  
Хостът следи каква част от буферите му са заети. При 60% запълване той изпада в състояние *натоварен*; при 80% - *претоварен* ; при 100% - *задръстен*. При преминаването на границата от 60%, хостът сигнализира на изпращача с *жълт пакет* (yellow packet). При преминаване 80% се изпраща *червен пакет* (red packet). Схемата е аналогична на фуния, в която влизат пакетите и излизат контролни пакети.[4](javascript:;).

Този подход се счита за най-добър, тъй като решава локални проблеми[5](javascript:;), като работи с предвиждане и се стреми да не позволи загуба на трафик. Трудност при него е това, че е нужно хостовете да могат да боравят със специалните пакети. За целта има множество разработки, които целят повишаване на QoS (Quality of Service). Един от тях е [Differentiated services](http://en.wikipedia.org/wiki/Differentiated_services). Работата с цветовата схема[6](javascript:;), е описана в RFC-та като [2697](http://tools.ietf.org/html/rfc2697).

*Ситуации без изход (Deadlocks)*

В тази картинка се наблюдава следното: четирите рутъра, които имат максимална дължина на опашковите буфери **4**, са разпределили всичките си буфери за изпращащите си интерфейси. Така те не могат да разпределят буфер за входящи пакети, които на свой ред седят в буферите на техните рутъри, не ги освобождават, тези рутъри също не могат да приемат… и мрежата замръзва. От тази ситуация няма изход (то си го пише горе).

Съществува просто решение на проблема: някой от рутърите да изхвърли един пакет. Освобождавайки буфер, той ще приеме пакет от друг рутър, който също ще освободи и приеме, и тъй нататък. Проблемът обаче се корени във факта, че рутърите не знаят, че са в мъртва хватка, и, гонейки максимално качество на услугата, няма да изхвърлят пакет.

Ситуации без изход се срещат във всякакви системи[7](javascript:;), и с тях отдавна се води борба чрез предотвратяване (превенция). При мрежовите протоколи, алгоритмите за пренос се правят така, че да не позволяват появата на затворени контури, които при запълването си биха довели до deadlock.

**19.Транспортно ниво.Процедури за съединенията**

IP протоколът, както се знае, е **connectionless** и **unreliable** - няма механизъм нито за установяване на сесии, нито за надеждно предаване на трафика (best effort - пакетът я пристигне, я не). Единствената възможност за известяване на източника, че информацията не е достигнала целта си, е протоколът **ICMP**: когато не може да се намери път за съобщението до хоста, или когато хостът не може да приеме съобщението, **е възможно** да се изпрати ICMP съобщение за грешка. Но, първо, такова може и **да не се изпрати**, и, второ, честа практика е ICMP трафикът да се блокира по съображения за сигурност. В такъв случай, IP хостът-изпращач не знае нищо за съдбата на изпратената дейтаграма.

*Роля на транспортното ниво*

Несигурността на предаване е част от предназначението на IP - неговата цел е максимално бързодействие при минимално натоварване на комуникационната подмрежа. За надеждността на комуникацията се грижи хостът (ако му е необходимо). Тази функционалност е работа на **транспортното ниво**. От негова гледна точка, няма значение какво се е случило на по-ниските нива. То просто праща информацията, и ако протоколът осигурява надеждност, слуша за отговор от отсрещното ниво 4. Ако отговор не последва, се счита, че информацията е загубена, и обичайното действие в такъв случай е нейното препращане. За транспортното ниво информация (дейтаграми) се губи не поради грешки на предавателя, а поради логистическа грешка.[1](javascript:;)  
Протоколите от транспортното ниво работят по два начина на работа: **без установяване на връзка (съединение)** - [connectionless](http://en.wikipedia.org/wiki/Connectionless_protocol), и **със съединение** -[connection-oriented](http://en.wikipedia.org/wiki/Connection_oriented).[2](javascript:;)

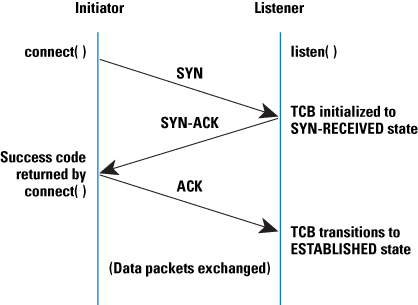
*Connectionless*

Транспортните протоколи, които не установяват връзки (съединения), изпращат данните към получателя директно, без предварителни уговорки за установяване на транспортна сесия и нейните параметри. Тяхното предназначение е, подобно на IP, да осигуряват бърза и лека комуникация. Затова, обикновено те пращат възможно най-малко сервизна информация в хедъра си - например source и destination port. Информацията се праща:

* без предварително известие
* без следене на нейното приемане и без положително / отрицателно потвърждаване
* без фиксиран ред на получаване - отново, както в IP, се разчита на **best effort delivery** и дейтаграмите могат да пристигнат в разбъркан ред.

Крайното приложение, което получава информацията, се грижи за нейната цялост. То толерира загуба на данни и/или извлича липсващата информация от наличната. Типичен пример за употреба на ненадеждни транспортни протоколи e аудио / видеоизлъчването (streaming), където моментните грешки при преноса на информация се проявяват като дефекти в звука / изображението и се *замазват* от приемащия софтуеър. Други примери включват [IP телефонията](http://www.fcc.gov/voip/), различни мрежови услиги (DNS, RIP, etc.) и компютърни игри (Starcraft).  
Протоколите без установяване на съединение се наричат също **stateless**, тъй като комуникацията не преминава пре различни етапи, а е непосредствена. Пример за протокол без съединение е [UDP](http://en.wikipedia.org/wiki/User_Datagram_Protocol).

*Connection-oriented*

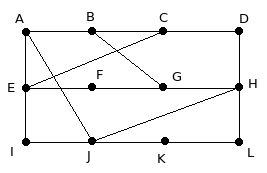
Тези протоколи първо договарят установяват връзка (съединение) с получателя и чак тогава започват да прехвърлят данни. Чрез установяването на връзка (сесия) се осъществява контрол върху предаването - могат да се следят получаването и потвърждаването на информацията, редът на нейното получаване, натоварването на връзката, а също и сигурността на комуникацията[3](javascript:;).  
Договорката фиксира дадени параметри на комуникацията - например, максималната дължина на съобщението, което е важно за приемащата страна. Договорката минава през няколко фази: излъчващият хост подава *оферта*, на която приемащият отговаря със съгласие или заявка за промяна на праметрите (но задължително отговаря, освен ако този вид комуникация не е забранен, например от firewall).  
Пример за договорка е [3-way handshake](http://www.pccitizen.com/threewayhandshake.htm).  
  
Предаващият хост пуска заявка за комуникация, след което чака за отговор до time-out (опитите продължават до няколко тайм-аута) или до получаване на такъв. Ако адресатът приеме предложението, той пуска положителен отговор. При неговото приемане, изпращачът заделя буфер за комуникацията и пуска потвърждение от своя страна. Когато адресатът получи потвърждението, той също заделя буфер. Следва преносът на данни. Ресурси (буфери) не се заделят, преди да е ясно, че желанието за общуване е двустранно.  
При загуба на потвърждението от страна на предавателя, получателят ще се *досети* при получаване на първото съобщение от потока данни, че е извършил договорка с изпращача, и ще започне да приема. Съществуват още много *какво ще стане, ако…* при различни ситуации на приемане или загуба на различни служебни съобщения. За всяка от тях в протокола има предвидено решение - досега случилите се събития определят в какво **състояние** се намира протоколът. Затова често протоколите със съединение се описват като **stateful**.  
Пример за такъв протокол е [TCP](http://fmi.wikidot.com/km20).

**15.Маршрутизация със следене на връзката.Йерархична маршрутизация**

Първите динамични маршрутни протоколи използват метода на дистантния вектор. С течение на времето и нарастването на рутираните мрежи, в частност - интернет, започват да се открояват техните недостатъци. Започва разработката на следващи поколения distance vector протоколи, но същевременно се разработва и изцяло нов клас маршрутни протоколи, следящи **състоянието на връзката** ([link state](http://en.wikipedia.org/wiki/Link-state_routing_protocol)).

Link state протоколите, за разлика от distance vector, строят в паметта си цялостна карта на графа от рутъри. По нея те прилагат алгоритъма на Dijkstra, за да изчислят дърво на най-късите пътища, с което после попълват маршрутната си таблица. За да се постигне това, е нужно всеки рутър да съдържа една и съща актуална карта, в която възможно най-бързо да се отразяват промените в състоянието на връзките, а съответно - и на маршрутите. Това те постигат чрез малки пакети, съдържащи състоянието на конкретна връзка (връзки), които се разпращат моментално при промяна на нейното състояние. Ако за distance vector протоколите е подходящо сравнението с навигация по пътни табели, то за link state може да се каже, че тяхната работа прилича на навигация по пътна карта - на нея са указани всички налични пътища с техните цени (дължини), а наблюдаващият картата си избира този път, който счита за оптимален.

*Link state пакет*

Основен момент в работата на един link state протокол е пакетът, носещ информацията за състоянието на връзката. Чрез такива пакети рутърите си обменят информация за своите непосредствени съседи и връзките си с тях. След употреба пакетите биват разпращани **моментално** до останалите съседи на рутъра, спазвайки правилото split horizon. Сравнително малките размери на пакетите позволяват при това наводнение мрежовата натовареност да остане в поносими граници.  
  
За тази позната конфигурация, пакет със състояние на връзките, изпратен от рутър **J**, би могъл да има следния вид:

|  |  |
| --- | --- |
| ID на изпращащия рутър (J) | |
| Пореден номер (sequence number) | |
| Възраст на пакета (age) | |
| A | 8 |
| I | 10 |
| H | 12 |
| K | 16 |

* **ID** - уникален идентификатор на рутъра в графа (или областта на действие на протокола)
* **Sequence number** - служи за различаване на новите ъпдейти, пуснати от **J**, от старите, които може би още циркулират из мрежата.
* **Age** - указва преди колко време пакетът е пуснат в обръщение, увеличава се при всяко следващо предаване. Ако нещо се случи на **J** и той излезе временно от строя, при връщането си, **Sequence Number** на новите му пакети ще е по-малко от това на старите, пуснати преди отпадането му. По възрастта обаче рутърите ще изберат най-новите пакети.
* Списък от двойки съсед - метрика.

*Стъпки при работата на link state протокол*

Протоколите със следене състоянието на връзката изпълняват 5 основни стъпки.

1. **Уточняване на съседите**. Рутърът открива своите съседи, свързани към неговите активни интерфейси.
2. **Измерване на състоянието (метриката)**. Чрез **Hello** пакети се измерва времето за пътуване на пакета, и по специфичен за протокола метод се определя метриката.
3. **Конструиране на link state пакет**. Директните съседи се попълват в картата на графа и в маршрутната таблица, след което картата се копира в един или няколко пакета, предназначени за съседите на рутъра.
4. **Flooding**. Готовият пакет се разпраща до всички съседи на рутъра, спазвайки правилото на разделения хоризонт.
5. **Събиране на пакети**. Известно време протоколът слуша за изпратени link state пакети, които копира и препредава. След това върху събраната информация се прилага алгоритъмът на Dijkstra. Най-добрите маршрути се записват в маршрутната таблица.

*Части от нещата, изброени по-долу, са специфични за протокола OSPF.*

**Уточняване на съседите**

Чрез обмяна на специални съобщения, рутърът открива други рутъри, към които има директна свързаност в съответния локален сегмент (обикновено, всеки негов интерфейс се намира в различен локален сегмент). След това той пристъпва към оформяне на **съседства (adjacencies)**. Ако в един локален сегмент има повече от два рутъра - тоест, връзката не е point-to-point - се избира **designated router**. Той представя рутърите в сегмента - оформя съседства, от тяхно име, получава предназначените за тях link state пакети и им ги препраща, получава изпращаните от тях пакети и ги подава на външния свят. По този начин се намалява потокът от служебни пакети, които наводняват връзката.

**Измерване**

С изпращане на Hello пакети се измерва времето за получаване на отговор (round-trip time), което служи за определяне на метриката. Различните протоколи ползват различни фактори за изчисляване на метриката, като скорост на връзката (bandwidth), закъснение (delay), натоварване (load).  
Често първата и втората фаза се извършват едновременно.

**Flooding**

След като пакетът е готов, възниква въпросът - какво да се прави с него? Начините за разпращане са няколко. Първият е подобен на distance vector: през определен интервал от време, да се разпращат пакетите. Това, обаче, натоварва комуникационната мрежа. Затова най-често работи вторият механизъм:**triggered updates** - пакети с ъпдейти се сглобяват и разпращат само при промяна състоянието на някоя връзка - например, когато някой канал се претовари, когато на рутъра му угасне токът, или конкуренцията му резне кабелите. Понякога се разпращат и цялостни описания на мрежовите карти - например, когато OSPF рутър се включи към мрежата, той цели максимално бързо да попълни своята карта.

При получаване на link state пакет, който е нов за рутъра, рутърът си записва данните от него, а също и **ID**, **Sequence number** и **Age**, след което записва пакета в буфер и го разпраща към своите съседи. При получаване на нови пакети със същия ID, те се сравняват по **Age** и **Sequence number** - по-старите се изтриват, а по-новите се записват на мястото на досега пазения пакет и отново се наводнява с тях мрежата. При получаване на един и същи пакет втори път, той се унищожава.

За безпроблемна работа, е необходимо полетоо **Sequence number** да е достатъчно голямо. При късо поле, често ще се прехвърля максимумът и новите пакети ще почват от **0** - пакетите няма да остаряват достатъчно и протоколът ще се бърка, съответно - ще конверира по-бавно. Определено е, че с 32-битово поле и изпращане на пакет всяка секунда, максимумът ще се прехвърли след повече от един век непрестанна работа.  
Ако **Sequence number** се повреди при преноса на пакета, може да се случи така, че стойнотта изведнъж да скочи много. Това би довело до изхвърлянето на множество валидни пакети. Тук се намесва полето **Age** - протоколът решава дали да приеме пакета, или да го изхвърли. Чрез сравняване на двете се откриват събития като рестартиране на рутър.

**Събиране и изчисляване**

Рутърът има две възможности: или след време да спре да приема ъпдейти и да пресметне дървото на пътищата, или непрекъснато да събира и периодично да пресмята.  
При мерене на закъснението, може отговорът да се изпрати моментално, или да се нареди в опашка в рутъра. При втория вариант, се отчита размерът на съобщението[1](javascript:;).  
Лесен за изпълнение вариант е метриката на всяка дъга да се счита за единица. Този случай не отразява добре състоянието на мрежата, но се изчислява бързо. Освен това, при него липсва **осцилация на трафика**: В графа по-горе, маршрутите ABCD и AECD имат еднаква цена. Тъй като в дървото има единствен път, само единият маршрут ще се запише в таблицата. Когато трафикът започне да се движи по него, метриката му ще се повиши и той ще стане неоптимален, при което другият ще го смени. Тогава пък неговата метрика ще стане висока, и така нататък. При фиксирано тегло, натоварването не участва в метриката и осцилация не се случва. Повечето протоколи, ползващи комплексна метрика, позволяват **load balancing**: всики маршрути с еднакво тегло се записват в таблицата, като маршрутизаторът ги използва последователно.

Колкото по-голяма е мрежата, толкова по-сложен става графът и изчисленията стават твърде тежки в големи мрежи. Тъй като тук няма максимален мрежов диаметър, ограничения върху растежа на мрежата налага изчислителната възможност на устройствата, чието действие се забавя с уголемяването на графа. Съществува механизъм, който намалява натоварването върху рутърите

При йерархичния подход мрежата, която се рутира, се разделя на непресичащи се области. Рутърите, които са вътрешни за дадена област, изчисляват най-къси пътища само **вътре в областта**. Рутърите, които свързват две области, се занимават с трансфера между областите, през други гранични рутъри. Така всяко устройство работи върху доста по-малка част от цялата мрежа. На картинката, рутъри като **1B** са гранични (**area border router**), а такива като **1A** - вътрешни.

Граничните рутъри участват в broadcast / flooding вътре в зоната, както и вътрешните рутъри. Извън зоната, обаче, те трябва да изпращат ъпдейти само към други гранични рутъри, за да не се товари излишно мрежата. За целта трябва да има надеждни механизми, по които да се избират ABR и да се подменят, когато излязат извън линия. Също така, необходимо е да се обслужват отделно вътрешни и външни графи. Поддръжката на всичкото това разделение увеличава сложността на протокола, но за сметка на това намалява потока от служебни съобщения и улеснява изчисленията във всяко от устройствата.

Разделението на йерархични нива налага рутърите да знаят в коя зона работят, и кои рутъри са от тяхната зона, за да обработват само техните ъпдейти. Възможни решения са списъци на идентификатори на рутъри в съответната област, или добавяне на идентификатор на област в конфигурацията на рутъра и в пакета с ъпдейти.

**14.Разпределена маршрутизация с дистантен вектор**

[Distance vector](http://computer.howstuffworks.com/routing-algorithm4.htm)-протоколите определят най-добрия маршрут, като разглеждат два параметъра: **метрика (metric)** и **следващ съсед (next hop)**[1](javascript:;). Метриката играе ролята на **цена**: ниската метрика означава добър (бърз, ненатоварен) маршрут. Next hop пък указва откъде минава този маршрут. Ако двете се разглеждат като разстояние и посока, се получава нещо като *вектор на отстоянието (distance vector)* - оттам и името на метода. Метриката се мери по различен начин в различните протоколи - **брой хостове** от източника до целта (hop count), **бързина на връзката** (bandwidth), **натовареност** (load), **надеждност**(reliability), или някаква линейна комбинация на изброените.

За принципа на дистантния вектор, доста добра аналогия са [указателните табели](http://www.bulpress.net/documents/newsimages/2008-11-07-cl-CACJQKXBB.jpg): единственото, което се научава от тях, е какво разстояние има до целта и откъде се минава за натам - и това е напълно достатъчно, за се вземе решение накъде да се тръгне.

За да може да се достигне целта обаче, е нужно на всяко разклонение по пътя да има набор от табели, които да дават точна информация за отстоянието и посоката на целта. В термините на маршрутизацията, нужно е всеки рутър по избрания маршрут да съдържа вярна и актуална маршрутна таблица. За да постигнат това, рутърите, работещи по протокол с дистантен вектор, изпълняват алгоритъма алгоритъма на **Bellman-Ford** (в някои случаи, може би този на**Dijkstra**). Те периодично обменят таблиците си със своите съседи, и извършват релаксация на текущите маршрути[2](javascript:;).

*Изграждане на таблиците*

При включване на нов рутър в мрежата, първата му работа (чрез **Echo** пакети или по друг начин) е да открие своите съседи и (ако е нужно) да измери метриката до тях. С тази информация се попълва таблицата му. До възли (рутъри), до които той няма директна връзка, се счита, че съществува връзка с метрика ∞ за нуждите на алгоритъма. През определен интервал от време маршрутната таблица се разпраща до съседите на рутъра, а от тях се получава тяхната и се извършва релаксация. Така поетапно таблицата се попълва и подобрява, докато се достигне до оптималното състояние - тогава мрежата е**конвергирала**.

*Проблеми на Distance vector-протоколите.*

**Бавно конвергиране**

Тъй като ъпдейтите се изпращат през период от време, новата информация се разпространява на периодични "вълни": директно свързаните съседи научават на края на първия интервал за ъпдейти, тези през един възел - на края на втория, и тъй нататък. Получава се така, че по-далечните рутъри научават за промяната след известно време. В мрежи с голям диаметър (най-голямото минимално разстояние между два рутъра) това се превръща в значителен проблем, тъй като в тях е голяма вероятноста да се случват фалове, и в голяма част от времето мрежата няма да е конвергирала.

**Цикли (Routing loops)**

[Routing loops и решенията за тях](http://www.certificationzone.com/newsletter/SL/IE-RIP-WP1-F03_RLP.html)  
В горната конфигурация, може да се случи така, че връзката **A-B** да се разпадне. Тогава **B** ще отбележи в таблицата си, че **A** е недостъпен - метриката на връзката ще бъде безкрайност. В това положение може да се получи **routing loop**: **C** има в таблицата си маршрут през **B** до **A**, чиято цена е по-добра от безкрайност. При ъпдейт, **B** ще получи този маршрут, увеличен с единица, но няма да знае, че той всъщност минава през него, и ще го запише в таблицата си. Тогава, пакет от **D** за **A** ще мине през **C**, оттам през **B**, оттам отново през **C**, отново през **B**, и така нататък, докато TTL-полето на пакета се занули.

**Count to infinity**

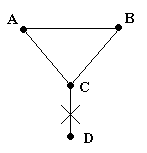
В горната ситуация, метриката от **C** до **A** е 2, а от **B** до **A** - 3. При следващ ъпдейт, **C** ще получи от **B** маршрут с цена 4. Понеже най-добрият маршрут на **C**минава през **B**, то **C** ще извърши релаксация, и тъй като новият маршрут е единствен, **C** ще го запише с метрика 4.[3](javascript:;). На следваща стъпка, това се повтаря в обратна посока - **B** получава от **C** маршрут с цена 5, и по същите съображения го впише с таблицата си. Това продължава потенциално до безкрайност.

*Решения на проблемите*

**Безкрайност**

Безкрайността е стойност на метриката, при която дадена мрежа (рутър) се счита за **недостъпна**. Тази стойност зависи от конкретния distance vector протокол. За [RIP](http://en.wikipedia.org/wiki/Routing_Information_Protocol), например, тя e 16, докато за [EIGRP](http://en.wikipedia.org/wiki/EIGRP) e 224. Безкрайността предотвратява count-to-infinity (и може да се ползва при инициализацията). Същевременно, обаче, безкрайността ограничава мрежата по диаметър (най-голямото минимално разстояние между два рутъра), което означава, че distance vector протоколите могат да се ползват в малки до средни мрежи.

**Split horizon**

Това правило диктува, че ъпдейт за даден маршрут не се изпраща по същия интерфейс (към същия рутър), по който е получен. В нашия пример, ако **C** спазва split horizon, няма да изпрати фаталния ъпдейт до **B** с метрика 3. Правилото се използва, за да предотврати маршрутните цикли, но, само по себе си, не е достатъчно за тази цел. Примерът:  
  
Тук, маршрутът **C-D** отпада. При ъпдейт, **C** известява **A** и **B** за отпадналия маршрут. Може да се получи така, обаче, че **A** да получи ъпдейта от **C**, но **B** да изпрати своя ъпдейт, с вече невярната информация, към **A**, преди да получи ъпдейта си от **C**. Така отново ще се завърти порочен кръг и **A** и **B** ще влязат в count-to-infinity. Затова обикновено split horizon се използва заедно с други механизми.

**Таймъри**

Различните distance vector протоколи имат различни таймъри, които използват, за да регулират ъпдейтите.  
Ако за определен период от време по даден маршрут не бъде получен ъпдейт, то маршрутът се счита за неработещ (невалиден).  
За да успеят останалите рутъри да научат за отпадането на маршрута, е възможно той да се постави в **holddown**. Това е един вид *замразяване*: в ъпдейтите, които рутърът изпраща се отбелязва, че маршрутът е невалиден, но метриката му остава същата в таблицата. Ъпдейти за този маршрут се получават, но поради факта, че е бил най-добър, той се променя **само** ако пристигне ъпдейт със същата или по-добра метрика - ще рече, че или връзката се е възстановила, или се е появила някоя нова и по-добра. В такъв случай holddown се премахва.  
Маршрут, който е бил в holddown достатъчно дълго, се премахва от таблицата.

**13.Статична и централизиране маршрутизация**

Статичната маршрутизация е друг подход към проблема за намиране на оптимален маршрут. При нея маршрутизаторите взимат решение според маршрутна таблица, съдържаща се в паметта им. За всеки рутър **X**, в таблицата му за всеки **Y** от другите рутъри в графа има записи, описващи съседите на **X**, през които може да се достигне **Y**, и коефициенти - "оптималност на ребрата" до тези съседи. Коефициентите се определят от администратора или се изчисляват по някакъв начин, като сборът им е единица. При постъпване на пакет, рутърът генерира произволно число в интервала **[0.00, 1.00]**. Коефициентите разбиват интервала на подинтервали, и в зависимост от това, в кой подинтервал попада числото, пакетът се предава към съответния съсед. Накратко: съседите с по-големи коефициенти имат по-голям шанс да получат пакета.

Възможно е схемата да се допълни с метода на Берън (горещия картоф): коефициентите са свързани с натоварването на опашката. За да се поддържа такава маршрутна таблица, е необходимо да се знаят мрежовата топология, скоростта на линиите (ребрата) и натоварването им в дадения момент. Поради това става трудно таблицата да бъде актуална[1](javascript:;), и схемата отново се оказва *дървена*.

*Централизирана маршрутизация*

Решение на проблема дава [централизираната маршрутизация](http://ntrg.cs.tcd.ie/undergrad/4ba2/network/luke/classes5.html). При нея, един компютър играе централна роля (**маршрутизиращ център**) - обикновено голяма машина с обслужващ персонал, изпълняваща сложни програми. Неговата задача е през определен период от време да събира информация за скоростта и състоянието на връзките (линиите), да изчислява маршрутни таблици за останалите рутъри (например, по Dijkstra) и да им ги праща. По този начин таблиците се поддържат в относително актуално състояние, ще отразяват промените в натоварванията и отпадналите маршрути, а отделните рутъри няма да имат нужда от сложен hardware / firmware за съставянето им. Една известна мрежа, работеща на този принцип, е била френската TRANSPAC.[2](javascript:;)  
Това решение обаче има куп недостатъци:

1. Приложимо е за достатъчно големи мрежи (стотици рутъри).
2. Ако центърът излезе извън строя, цялата идея отива по дяволите - рутърите ще работят със стари таблици. Следователно има нужда да се поддържат поне 2 маршрутизиращи центъра заедно с обслужващ ги персонал, което намалява рентабилността на работа (*$$$*) и ограничава потребителите до големи организации. Така е било и в TRANSPAC - държавата плаща, за да се използват центровете.
3. Данните за връзките пристигат по-бавно от крайните рутъри, отколкото от близките. Това затруднява достоверното измерване на натовареността на ребрата.
4. Линиите около централния възел се затлачват със служебен трафик.
5. Въпреки, че този метод се счита за най-добър по отношение на лесна и ефективна администрация, в една децентрализирана мрежа (например, ARPAnet) финансирането се оказва трудно, поради което още от тогава в нея и нейните наследници не се използва такава маршрутизация.

В търсене на решение за тези проблеми, се появява динамичната **децентрализирана (разпределена) маршрутизация**

**12. Въведение в маршрутизацията. Маршрутни алгоритми. Софтуер за маршрутизация Quagga Routing Suite с отворен код.Инструментариум iproute2.**

Маршрутни алгоритми Основната функция на мрежовото ниво е да маршрутизира пакетите от източника към получателя - през 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.

Ако мрежата е с пакетна комутация (дейтаграми), решението трябва да се взима наново за всяка пристигнал пакет, тъй като оптималният маршрут може да се е променил.

Ако се използва виртуален канал, решенията по маршрутизацията се взимат при създаването му.

Маршрутизиращите протоколи трябва да отговарят на множество изисквания. Да са достатъчно прости и лесни за конфигуриране и да осигуряват надеждна и стабилна работа на мрежата. Да реагират своевременно на отпадане на маршрутизатори или връзки между тях. Да бъдат в състояние да открият алтернативни пътища за доставяне на пакетите, ако такива съществуват. Две други цели на маршрутизиращите протоколи си противоречат (на пръв поглед):

- минимизиране на времето за закъснение (по-малък престой на пакетите в междинните възли);

- максимизиране на общия поток предполага буферите в маршрутизаторите да работят на максимален капацитет.

Освен това максимизирането на общия поток може да влезе в противоречие с изискването мрежовите ресурси да могат да се използват от всички потребители в мрежата.

Маршрутизиращите алгоритми са два вида - неадаптивни и адаптивни. При неадаптивните маршрутизацията не се извършва на базата на текущата топология на мрежата. Маршрутите между всеки два възела в мрежата се изчисляват предварително и се записват ръчно от мрежовите администратори, след което влизат в маршрутните таблици.

При промяна на топологията на мрежата (например при отпадане на възел или на връзка), администраторите ръчно трябва да променят маршрутите. Това прави неадаптивните алгоритми приложими само в малки мрежи, при които рядко настъпват промени. Неадаптивните алгоритми се наричат още статични.

Важна характеристика на адаптивния алгоритъм е неговата **скорост на**

**сходимост**:времето, което е необходимо да се преизчислят маршрутните таблици на всички маршрутизатори в мрежата при промяна в топологията или трафика. (конвергенция)

Оптималните пътища между всеки два възела в мрежата се изчисляват по някои от алгоритмите за намиране на най-къс път в граф (след като е въведена метрика в графа, представящ мрежата). Всички тези алгоритми се базират на **принципа за оптималност**:всяка част от оптимален път е също оптимален път между съответните два върха.

Като следствие от този принцип, оптималните пътища от един връх към всички останали образуват дърво (sink tree).

Алгоритъмът на Дейкстра е алгоритъм за намиране на най-къс път в граф от даден връх до всички останали върхове. Важно е да се отбележи, че при алгоритъма на Дейкстра теглата на ребрата трябва да са положителни. Резултатът от алгоритъма е дърво на оптималните пътища от дадения връх до всички останали.

**Quagga** е open source софтуерен пакет за маршрутизация. Поддържа: RIPv1, RIPv2, RIPng, OSPFv2, OSPFv3, BGP-4 и BGP-4+ (т.е IPv4 и IPv6)

Quagga рутер

Компютър с Quagga си е рутер със **Cisco** **CLI**. Обменя информация за маршрутите с помощта на маршрутни протоколи. Quagga я използва, за да обновява таблицата с маршрутите в ядрото.

Quagga. Архитектура.

Мултипроцесна архитектура. (все още няма **multi-thread**).Всеки демон - .conf файл и терминал. Статичен маршрут – zebra.conf.BGP - bgpd.conf

XORP рутер с интерфейс на **Juniper**.

Също е open source платформа за IPv4 и IPv6 маршрутизация. Поддържа OSPF, RIP, BGP, OLSR, VRRP (Virtual Router Redundancy Protocol), PIM, IGMP (Multicast).

Iproute2 – сбор от средства за контрол на TCP/IP мрежи и трафик в Linux.

**11. Мрежов протокол IPv6 – общи положения, начин на записване, видове адреси, статична и динамична адресация**

Преходът към IPv6 е неизбежен. IPv4 адресите са на изчерпване. IPv6 не е обратно съвместим с IPv4, необходими са промени в мрежови устройства и услуги. Трудности при едновременна работа на IPv4 и IPv6, която е наложителна в дългия преходен период.

Подобрения в IPv6

адресното пространство от 32-битово става 128-битово: **232** (4.3 x 109) с/у **2128** (3.4 x 1038). автоматично (plug-and-play) присвояване на адрес без помощта на DHCP сървър като в IPv4.

Header

В IPv6 е по-опростено от IPv4; с фиксирана дължина 40 байта (RFC 2460):

- 2 \* 16-byte IPv6 адреса;

- 8 байта друга информация.

По-бързо и лесно обработване на пакетите. traffic class (заменя IPv4 ТoS); flow label (ново QoS management); payload length (до 64KB); next header (заменя IPv4 protocol); hop limit (заменя IPv4 TTL).

Поддържа IPsec

IP security (IPsec) съдържа протоколи за аутентикация на изпращача и гарантиране на данните в IP комуникациите:

- Encapsulating Security Payload (ESP);

- Authentication Header (AH);

- Internet Key Exchange (IKE).

IPsec е част от IРv6. Задължителен е IPsec за защита на Mobile IPv6 и OSPFv3.

Mobile IPv6

**MIPv6** поддържа roaming за мобилни възли (RFC 3775). MIPv6 използва Neighbor Discovery (RFC 4861), за да реши проблема с прехвърлянето (handover) на мрежов слой и оптимизация на маршрута (RFC 4449).

Quality of Service (QoS)

IP третира всички пакети еднакво – best effort. TCP (Transmission Control Protocol) гарантира доставянето, но не контролира закъснение, честотна лента и т.п.

QoS – опции за въвеждане на политики и приоритети на трафика.

IPv4 и IPv6 сходни QoS възможности:

QoS

В IPv6 header има две полета за QoS:

- Traffic Class и

- Flow Label.

Traffic Class е разширено по-прецизно диференциране на различните типове трафик.

Новото Flow Label поле - съдържа етикет за идентифициране или приоритетизиране на определен поток от пакети като VoIP или видеоконференции, чувствителни към времето на доставяне.

Jumbograms

IPv6 Hop-by-Hop Option - **jumbograms**, IPv6 пакет с поле за данни(payload) > 65 535 октета. Важи за IPv6 интерфейси, които могат да поемат кадри с такива дължини (>= 1 gbps). 16-бит поле Payload Length (в IPv6 Header) = 0

IPv6 Fragment Extension Header

В IPv6 фрагментирането на пакетите става още при източника. В IРv4 рутерът фрагментира пакета, когато MTU на следващия канал е по-малък. Ако отсреща не се възстанови оригиналния пакет, сесията се разваля.

В IРv6 всеки хост използва Path Maximum Transmission Unit (PMTU) Discovery, за да

научи размера на MTU по пътя, за да не се налага фрагментиране.

IРv6 адрес (пример):

**2001:0db8:9095:02e5:0216:cbff:feb2:7474**

**8** групи с по **4** щестнадесетични числа

8 \* 4 \* 4 = 128

Формат на IРv6 aдрес

Мрежовият префикс (network prefix) – идентифицира дадена мрежа или специален адрес. Присвоява се от ISP (PA) или RIR (PI). Идентификаторът на подмрежата (subnet ID) – връзка вътре в мрежов обект. Присвоява се от администратора на обекта. Един обект ≥ 1

subnet IDs. Определя на кой мрежов сегмент принадлежи даден хост. host ID идентифицира конкретен възел в мрежата – конкретен негов интерфейс.

Мрежовият префикс (RFC 4291) е аналогичен на означението с **“/”**  на SM в IРv4: IPv6 address**/prefix length**

**Например адрес с 32-bit мрежов префикс:** 2001:0db8:9095:02e5:0216:cbff:feb2:7474**/32**

Алокация на IPv6 префикси

IPv6 (подобно на IPv4) се присвояват от RIRs и ISP. Големите провайдери (LIRs) могат да получат префикс с минимална дължина 32 бита:

- най-старшите 32 бита са мрежовия префикс;

- останалите 96 бита са на разположение на администратора за раздаване на subnet ID-та и за host ID.

48-битов мрежов префиксПравителствени, образователни (СУ), търговски и

др. организации обикновено получават от големите ISPs (PA) или от RIRs (PI) 48-битови алокации (/48), оставяйки 80 бита за subnet ID и host ID.

64-битов мрежов префиксПодмрежите в рамките на организация обикновено са 64 битови (/64)64 бита остават за host ID - 64-bit идентификатор на интерфейса в EUI-64 формат.

За да се улесни записването на адреси, съдържащи нули, те се компресират по определени правила.

**"::"**  - една или повече 16-битови групи от нули; за компресиране на водещи или завършващи нули.

"::" може да се появи само веднъж в адреса.

Запис на IРv6 адреси и префикси

**НЕПРАВИЛНО** представяне на 60-bit префикс: 2001:0DB8:0**:CD3**/60 в 16-bit число от адреса се пропускат водещи (незначещи) нули, но не и крайни (значещи) нули

Unspecified ::/128(неопределен)

Loopback ::1/128

Multicast FF00::/8

Link-Local FE80::/10

Няма Broadcast адреси**Broadcast** адреси **не са дефинирани** в IРv6.

Multicast адресирането в IPv6 поема функциите и на broadcast.

IPv6 unicast адресите, подобно на IPv4 CIDR, имат префикси с произволни дължини. Един възел в IPv6 мрежа има повече или по-малко знание за вътрешната структура на адреса (дали е хост или рутер).

Interface ID Идентификаторите на интерфейси в IРv6 трябва да са уникални в рамките на subnet prefix. Всички unicast адреси с изключение на започващите с с двоична стойност 000 Interface IDs са 64 бита в Modified EUI-64 формат.

EUI-64 формат Отнасят се до конкретна LAN или мрежов канал. Всеки IPv6 интерфейс в LAN трябва да има такъв адрес.

IPv6 Multicast. Формат.

Въвеждането на обсег (scope) в IPv6 multicast ограничава разпространението на пакети само до необходимите части от мрежата: интерфейси, мрежови сегменти и префикси.

IPv6 Multicast. Scope. Някои добре известни multicast Group IDs се дефинират с различни обсези (scopes).

Защо няма broadcast?IPv6 няма broadcast адреси, а Solicited Node multicast групи и all routers multicast адреси. Така по-оптимално се използват мрежовите ресурси.

Neighbor Discovery (ND) е процес, чрез който IPv6 възел може да научи адреси на 2 слой на интерфейси, свързани към локалния мрежов сегмент. ND замества ARP в IРv4.Работи в комбинация с ICMP Router Discovery и Redirect.

Функциите на ND се осъществяват чрез:Router Solicitation (**RS**). При активиране на интерфейс хостът изпраща RSs, със заявка рутерите веднага да генерират RAs.

Router Advertisement (**RA**). Рутерите рекламират присъствието си и някои параметри периодически или веднага след RS. RA съдържа префикси на връзката, конфигурации на адреси, брой hop-ве, MTU и др.

ND. Neighbor Solicitation. Възлите изпращат NSs, за да определят адреса на 2 слой на съседа или да се уверят, че съседът е все още достижим. NSs разпознават и дублирани

адреси (Duplicate Address Detection – DAD). Neighbor Advertisement (**NA**). Отговор на NS. Възел може да изпраща самостоятелно NAs, за да съобщи за промяна на адрес.

Redirect Message. Рутерите информират хостовете за по-добър първи хоп до дестинацията.

Autoconfiguration.

В IPv6 има и Stateful (с определено съъстояние), и Stateless (неопределено) автоконфигуриране на адреси - SLAAC. SLAAC не изисква ръчно конфигуриране на хостове, минимално на рутери, сървъри не са необходими. Рутерите рекламират мрежов префикс, a хостът генерира interface ID. Ако в мрежовия сегмент няма рутер, хостът генерира само адрес на 2 слой, с който може да комуникира само в мрежовия си сегмент. Stateful автоконфигуриране в IPv4 е DHCP. За IPv6 версията е DHCPv6, който е много различен от DHCPv4.

Механизмите за преход трябва да осигуряват взаимодействието.

Определят се и от вида на хостовете:

- само IРv4;

- само IPv6 и

- dual stack IPv4/IPv6.

Видове механизми

● Dual stack

● Tunneling

● Translation (NAT)

Dual Stack IPv4/IPv6 (SUnet)

За потребителите на СУ е прозрачно дали за дадена услуга ползват IPv4 или IРv6. Постига се с оборудване, което поддържа и двата протокола:

- втора ръка сървъри за маршрутизатори, работещи под Linux с пакет Zebra Quagga;

- DNS е един и същ за IPv4 и IРv6;

- присвояване на адреси по IРv4 – статично или DHCP, IРv6 – автоматично;

- Web (Apache) – “слуша” по IPv4 и IРv6.