Лекции по компютърни мрежи и комуникации

Писани са от мен, Иван Димитров Георгиев (вече завършил) студент по информатика, електронната ми поща е ivandg@yahoo.com.
Четени са през зимния семестър на учебната 2004/2005 година. Лектор тогава беше доц. Антон Петков.
Използвал съм мои (и на мои колеги) записки от лекции, но основно съм използвал следните учебници: Компютърни мрежи на Андрю Таненбаум и Принципи на компютърните мрежи и Интернет на Боянов, Турлаков, Тодоров, Димитров и др.
Не претендирам за изчерпателност, нито че съм разбрал добре нещата. Мисля, че доц. Петков включва последните години някои допълнителни неща, така че моят съвет е да се посещават редовно неговите лекции.

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 пропускателната 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 Mb/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.

На практика при комуникацията между съответните слоеве на двете машини не се предават данни. Всеки слой *n* предава данни и контролна информация на непосредствено по-долния слой *n-1*, докато се достигне най-долния слой *1*. Под слой *1* е физическата среда за предаване, където се осъществява реалната комуникация между машините. В приемника получените данни се разпространяват в обратна посока - от слой *1* нагоре, като всеки слой премахва контролната информация, която се отнася до него.

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. Имплементацията на слоевете, както и интерфейсът между отделните слоеве не се включват в мрежовата архитектура, тъй като те са видими само в рамките на една машина. Те дори не е задължително да са едни и същи на машините в една мрежа – достатъчно е всеки слой *n* да може да комуникира със съответния слой *n* по определения протокол и да предоставя съответните услуги на по-горния слой. 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. Под отворена система се разбира система, чиито ресурси могат да се използват от другите системи в мрежата. Всъщност OSI-моделът е абстрактен модел на мрежова архитектура, който описва предназначението на слоевете, но не се обвързва с конкретен набор от протоколи. Поради тази причина 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.

Когато започват да се изграждат реални мрежи, използвайки OSI-модела и съществуващите протоколи се вижда, че те не отговарят на изискваните спецификации за обслужване.
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. Съществения 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. Използват се различни среди 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**. Една линия със скорост от b бода може да предава b символа за секунда, т.е. цифровият сигнал може да се превключва b пъти за една секунда. Ако един символ е 0 волта за логическа 0 и 5 волта за логическа 1, то линията ще има скорост b bps. Обикновено, обаче, в един символ се кодират повече битове - например, ако цифровият сигнал има 8 нива на напрежение, то с един символ се предават 3 бита информация и тогава скоростта на линията ще бъде $3b$ bps.

Найкуст е доказал, че при широчина на честотната лента H Hz един сигнал може да се превключва най-много $2H$ пъти за една секунда, т.е. скоростта на предаване не може да е по-голяма от $2H$ бода. По този начин, ако един цифров сигнал има V различни нива, то максималната скорост за предаване е $2H \log_2 V$ bps. Формулата на Найкуст е в сила за идеални канали без шум. Например, идеален 3-KHz канал не може да предава двоични цифрови сигнали със скорост по-голяма от 6000 bps.

По-късно Шенон предлага друга формула, в която се въвежда отношението сигнал/шум. Ако означим силата на сигнала със S, а силата на шума с N, то отношението сигнал/шум е S/N. Обикновено това отношение се изразява в децибели по формулата $10\log_{10} S/N$. Така 10 Db определят отношение S/N от 10, при 20 Db отношението S/N е 100 и т.н. Формулата на Шенон гласи, че максималната скорост за предаване по канал с шум, който има широчина на честотната лента H и отношение сигнал шум S/N е $H\log_2(1+S/N)$. Например, по 3-KHz канал с отношение сигнал/шум от 30 Db може да се предава със скорост не повече от 30000 bps.

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.
По усуканите двойки могат да се предават, както аналогови така и цифрови сигнали. Широчината на честотната лента зависи от дебелината на проводниците и от тяхната дължина. 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. Поставяйки в единия край на влакното източник на светлина, а в другия край детектор получаваме система за еднопосочно предаване на данни, която приема електрически сигнали, преобразува ги в светлинни импулси, предава ги и след това преобразува светлинните импулси обратно в електрически сигнали. Светлинните импулси преминават през влакното чрез пълно вътрешно отражение. Оптическите кабели се състоят от сърцевина, оптична обвивка и защитно покритие. Сърцевината е оптическото влакно, през което преминава светлината. Чрез

оптичната обвивка се осъществява пълното вътрешно отражение. 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). Основните варианти на тази услуга са: непотвърдено неуставено обслужване, потвърдено неустановено обслужване и потвърдено установено обслужване.

При непотвърденото неустановено обслужване източникът изпраща независими кадри към получателя без получателя да ги потвърждава. Няма установяване на връзка между двете машини. 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, което позволява функциите по възстановяване на загубената информация да се поемат от по-горни нива в йерархията. Такова обслужване се реализира в повечето LAN. То също се използва

когато навременното получаване на кадрите е по-важно от тяхната достоверност.
При потвърденото и неустановено обслужване отново не се установява връзка между източника и получателя, но получаването на всеки кадър се потвърждава самостоятелно от получателя. Това дава възможност за повторно изпращане на непотвърдените кадри.
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. При това обслужване се гарантира не само успешното предаване на кадъра, но и последователността в която се предават кадрите.

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. Обикновено механизмите по управление на обмена се изпълняват в транспортния слой над по-големи информационни единици, обхващащи последователност от кадри.

Каналното ниво взима пакетите, които му се подават от мрежовото ниво и ги затваря в кадри.
Всеки кадър се състои от заглавна част (header), поле за данни, което съдържа пакета и опашка (trailer). Дължината на кадъра обикновено е ограничена отгоре.
Физическото ниво възприема информацията от каналното ниво като поток от битове, без да се интересува от нейната структура. Получателят идентифицира в потока от битове кадрите и въз основа на служебната информация в тях ги контролира за грешки. За целта опашката на кадъра съдържа контролна сума (обикновено 2 байта), която се изчислява върху останалата част от кадъра преди той да бъде предаден. Когато кадърът пристигне в получателя, контролната сума се преизчислява и ако тя е различна от предадената контролна сума, то получателят отхвърля кадъра и евентуално изпраща съобщение за грешка към източника. Ако контролните суми съвпаднат, то 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. Един начин е между всеки два кадъра да се въведе времеви интервал. Този подход е твърде несигурен, тъй като времевите интервали могат да се променят по време на предаването. 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. Дори при неправилна контролна сума, получателят не знае началото на следващия кадър, въпреки че разбира, че текущият кадър е сгрешен.

При втория метод в началото и края на кадъра се вмъкват специални служебни символи - **STX** (start of text) за начало на кадър и **ETX** (end of text) за край на кадър, които маркират границите на кадъра. Техниката е известна като **вмъкване на символи** (byte stuffing, character stuffing).
Възможно е, обаче, служебните символи да се срещат като битови последователности в оригиналните данни. За решение на този проблем се въвежда друг служебен символ **DLE** (data link escape), който се вмъква преди всяко срещане на служебен символ (STX, ETX, DLE) в данните. Например, ако потокът, предаван от мрежовия слой на източника е A STX DLE B, той ще се преобразува в A DLE STX DLE DLE B.
Каналното ниво на получателя ще премахне символите DLE (като при два последователни DLE, единият се запазва) преди да предаде данните на мрежовото ниво на получателя.
При по-новите протоколи се използва един и същ символ за маркиране на началото и края на кадъра.
Недостатъкът на този метод е, че той се обвързва с 8-битови символи, кодирани в ASCII.

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**.
За да се предотврати погрешното определяне на граница на кадър, ако тази последователност от битове се срещне в данните на кадъра, след всеки 5 единици в данните източникът добавя по една нула. Техниката се нарича **вмъкване на битове** (bit stuffing). Например, ако потокът от битове, предаван от мрежовия слой на източника е 0110111111111110, то каналният слой ще го преобразува в 011011111011111010. Каналното ниво на получателя премахва нулата след всеки 5 единици в данните, преди да ги подаде на мрежовото ниво.

За постигане на допълнителна сигурност при много протоколи броенето на символи се комбинира с някой от другите два метода.

Протокол спри и чакай с алтернативен бит

Разглеждаме ситуация при която машината *A* изпраща кадри към машината *B* по канал с шум. Кадрите могат да се изкривят по време на предаването или изцяло да се изгубят. Предполагаме, че ако един кадър се изкриви, *B* ще разбере това като изчисли контролната сума. В случай, че кадърът се изкриви по такъв начин, че контролната сума се изчислява правилно - събитие с доста малка вероятност, което няма как да се установи - то към мрежовия слой на *B* ще се изпрати некоректен пакет.

A трябва да има свободен буфер с размер максималната дължина на кадър, в който да формира кадри от пакетите, подадени от неговото мрежово ниво и да изпраща тези кадри към *B*.
B също трябва да има подобен свободен буфер за да може да приема кадри от *A*. Когато в буфера на *B* постъпи нов кадър от *A*, *B* изчислява контролната сума на кадъра. Ако тази контролна съвпадне с изпратената контролна сума, *B* изпраща данните на неговото мрежовото ниво, формира потвърждаващ кадър в друг буфер и го изпраща към *A*. Ако контролната сума не съвпадне, то кадърът е сгрешен и *B* не изпраща потвърждение. Възможно е в този случай *B* да изпрати кадър към *A*, който да уведоми за грешно получения кадър, но както ще видим по-долу това не е необходимо.

Възможно е *A* да изпрати кадър към *B*, но този кадър да се изгуби. Тогава *B* не може да реагира, тъй като не е регистрирал грешка. За да се избегне тази ситуация, *A* стартира брояч на време (**time-out**) с изпращането на всеки кадър. Времето, което отчита брояча трябва да е по-голямо от времето за предаване на кадъра, обработката му в приемника и получаване на потвърждение. Ако кадърът не се потвърди в рамките на това време, то *A* предава кадърът отново. Тук се обхваща случая в който *A* получава от *B* сгрешено потвърждение (това е равносилно с изтичане на времето за потвърждение).

Възможно е *A* да изпрати кадър към *B*, този кадър да се получи в *B*, но потвърждението да се изгуби. В този случай *A* не знае дали изпратеният кадър въобще е пристигнал до *B*.
При всички положения *A* изпраща наново кадъра и ако не се вземат мерки, *B* ще получи същия кадър и ще го изпрати към мрежовото ниво, което ще доведе до недопустимо дублиране на данните. За целта с всеки кадър се свързва пореден номер.
В случая е достатъчно номерът да е един бит (0 или 1). Във всеки един момент *B* очаква кадър с определен номер. Ако *B* получи кадър с друг номер, този кадър е дубликат и се отхвърля. Ако *B*

получи кадър с очаквания номер, кадърът се приема и очакваният номер на кадър се инвертира (ако е бил 0 става 1, ако е бил 1 става 0). От своя страна *A* номерира алтернативно кадрите, които изпраща към *B*. Естествено, ако даден кадър бъде изпратен отново неговият номер не се променя.

Протоколи с прозорци

В разгледания протокол, кадрите с данни се прехвърлят само в една посока - от *A* към *B*. Всъщност кадри текат и в двете посоки, така че каналът може да се използва за предаване на кадри с данни от *B* към *A* и съответно на потвърждения от *A* към *B*.
При този подход кадрите с данни от *A* към *B* (съответно от *B* към *A*) се смесват с потвържденията от *A* към *B* (от *B* към *A*) и за да се различават отделните кадри, в заглавната част на кадъра се добавя поле за тип на кадъра. Възможно е още едно подобрение. Когато в *B* или в *A* пристигне кадър с данни, то *B* (съответно *A*) не изпраща потвърждаващ кадър веднага, а изчаква мрежовият слой да подаде данни и потвърждението се прикачва към кадъра с данни (за целта отново се използва поле в заглавната част на кадъра). Предимството на този подход е, че полето за потвърждение е само няколко бита, докато отделен кадър за потвърждение включва собствена заглавна част и контролна сума. Проблемът е, че данните от мрежовото ниво, към които трябва да се прикрепи потвърждението може да се забавят прекалено дълго и броячът на време в *A* (съответно в *B*) да изтече, което ще доведе до повтаряне на кадъра. Обикновено решението е следното - *A* (съответно *B*) изчаква фиксиран брой милисекунди и ако дотогава не пристигне пакет от мрежовото ниво, *A* (съответно *B*) изпраща самостоятелен потвърждаващ кадър.

Ще разгледаме два протокола, които спадат към класа на **протоколите с прозорци**. Те са по-ефективни от протокола спри и чакай, тъй като позволяват изпращане на повече от един кадъра преди да се чака за потвърждение.

При тези протоколи всеки кадър се номерира с число от 0 до някакъв максимум, обикновено от вида
2^n - 1, така че номерът да се вмества точно в *n* бита.
Във всеки един момент предавателят поддържа множество от поредни номера на кадри, които са готови за изпращане - тези кадри попадат в **прозореца на предавателя**. От друга страна, получателят поддържа **прозорец на получателя**, в който се буферират получените кадри. Не е задължително двата прозореца да имат един и същ размер.

Ще отбележим, че е съществено пакетите подадени от мрежовото ниво на предавателя да се получават в същия ред в мрежовото ниво на получателя, въпреки че протоколите с прозорци

позволяват по-голяма свобода за реда в който се изпращат и приемат кадрите.

Нека размерът на прозореца на предавателя е 2^s, s ≤ n.
Поредните номера в рамките на прозореца на предавателя съответстват на кадри, които вече са били изпратени и чакат потвърждение. Когато от мрежовото ниво на предавателя пристигне нов пакет, той се номерира с (k+1) % 2^n, където k е номерът на последния кадър в прозореца и веднага се изпраща към получателя. Съответно горната граница на прозореца се придвижва напред. Когато в предавателя пристигне потвърждение, долната граница на прозореца се придвижва напред. Тъй като кадрите в прозореца на предавателя могат да се изкривят или изгубят, те трябва да се съхраняват за евентуалното им повторно изпращане. Така предавателят трябва да разполага с 2^s буфера. Освен това, предавателят трябва да може да преустанови подаването на пакети от мрежовото ниво, ако прозорецът бъде запълнен изцяло.

Нека размерът на прозореца на получателя е 2^m, m ≤ n.
Номерата на кадрите в този прозорец съответстват на кадри, които могат да бъдат получени. Когато в получателя пристигне кадър, чийто номер съвпада с долната граница на неговия прозорец, данните от този кадър се предават към мрежовия слой на получателя и прозорецът се завърта напред, т.е. придвижват се и горната и долната му граница. Ако номерът на пристигналия кадър попада в прозореца, но не съвпада с долната му граница, този кадър не се отхвърля, а се буферира.
За разлика от прозореца на предавателя, прозорецът на получателя има фиксиран размер.
Ако прозорецът на получателя има размер 1, то кадрите се приемат в реда в който са изпратени, но при прозорец на получателя с по-голям размер това не е така. За да поддържа прозорец на предавателя трябват 2^m буфера и битова карта с размер 2^m, която показва кои буфери са запълнени.
При m = 0 няма нужда от битова карта - буферът е един и кадърът в него директно се изпраща към мрежовия слой на получателя.

При наличие на сгрешен или изгубен кадър, предавателят ще продължи да предава кадри, преди да разбере че има проблем. Въпросът е какво да прави получателят с успешно получените кадри след сгрешен или изгубен кадър.

Едната стратегия (**go back n**) е тези кадри да се отхвърлят. Тя съответства на прозорец на получателя с размер 1. С други думи, получателят приема единствено следващия поред кадър, който трябва да се предаде към мрежовия слой. В даден момент броячът на време на предавателя ще изтече и той ще изпрати наново всички кадри, започвайки от сгрешения (изгубения).

При тази стратегия, когато в предавателя пристигне потвърждение за кадърът с номер k, кадрите с номера k-1, k-2, ... до долната граница на прозореца на предавателя се потвърждават автоматично и съответно новата долна граница на прозореца на предавателя се придвижва към (k+1) % 2^n.
Ако прозорецът на предавателя има ширина 2^n, той може да съдържа най-много 2^n - 1 кадри за предаване, поради следният проблем - ако прозорецът на предавателя съдържа 2^n кадъра, то потвърждаването на кадър с номер 2^n - 1 може да означава както, че всички кадри са били приети успешно, така и че всички кадри са били отхвърлени (изгубени).

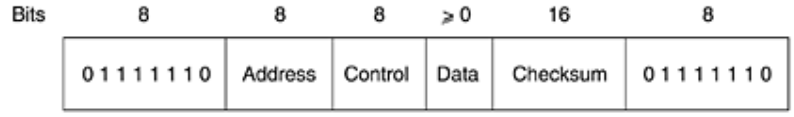
Другата стратегия (**selective repeat**) е получателят да буферира успешно получените кадри след сгрешен или изгубен кадър. Когато броячът на време в предавателя изтече, той изпраща наново само най-стария сгрешен (изгубен) кадър. Ако повторното изпращане е успешно, получателят може последователно да изпрати към своя мрежов слой кадрите, които е буферирал. Обикновено при тази стратегия получателят изпраща служебен кадър, който известява на предавателя за сгрешен или изгубен кадър - това води до по-бързо повторно предаване на съответния кадър. Стратегията съответства на размер на прозореца на получателя по-голям от 1. Всеки успешно получен кадър, чийто номер попада в прозореца на получателя се буферира и се изпраща към мрежовия слой чак след като са изпратени предшестващите го в прозореца кадри.
При тази стратегия възниква следният проблем - когато прозорецът на получателя се придвижва в даден момент той може да се припокрие с прозореца на предавателя (горната граница на прозореца на получателя да настигне долната граница на прозореца на предавателя). Тъй като получателят приема всички кадри в рамките на своя прозорец, то има опасност непотвърден кадър от прозореца на предавателя да се дублира. За целта ширините на прозорците се избират така, че $2^s+2^m \leq 2^n$, което гарантира че не е възможно да се получи подобно припокриване. Например, при 4-битови номера за ширина и на двата прозореца може да се избере 8.

Ширината на прозорците (по два прозореца за двете посоки) и максималната големина на кадъра се уговарят между предавателя и получателя при установяване на съединение в началото на предаването. Обикновено се използват едни и същи параметри и стратегии за двете посоки на предаването.

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.

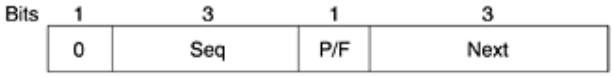
Полето *Control* се използва за номериране на кадрите, за потвърждения и за други цели.

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.

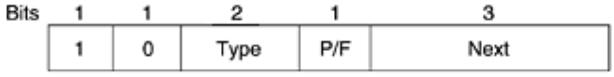
Кадрите са три вида - **information**, **supervisory** и **unnumbered**.
Полето *Control* за information-кадрите има следния формат:



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.

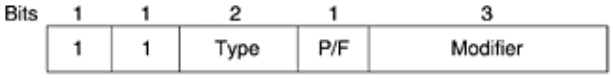
Полето *Control* за supervisory-кадрите има следния формат:



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:



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.

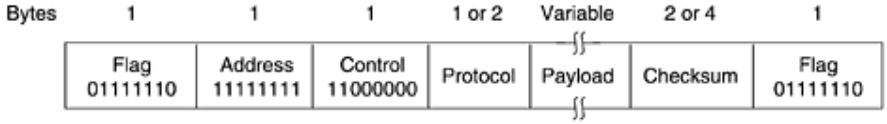
Командите също могат да се изкривят или изгубят, затова те също трябва да се потвърждават. За целта се използва специален unnumbered кадър UA (unnumbered acknowledgement). След изпращането на всяка команда се изчаква съответно потвърждение преди да се изпрати друга команда.

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.

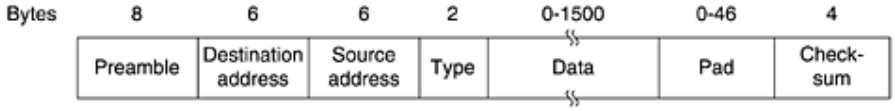
Мрежите с общодостъпно предаване се характеризират с общ комуникационен канал, който се споделя от всички машини, включени в мрежата.
Всеки изпратен кадър минава през общия канал и достига до всички машини в мрежата. Адресно поле в кадъра посочва за кой е предназначен този кадър. Когато една машина получи 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 IEEE (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.

8. Маршрутни алгоритми – оптимален път, статична маршрутизация, обхождане, наводняване, метод на Берън, централизиране.

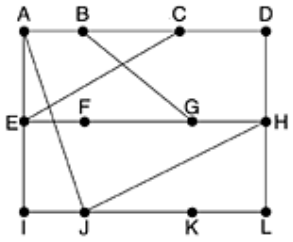
Основната функция на мрежовото ниво е да маршрутизира пакетите от източника към получателя. В повечето мрежи пакетите ще изминат това разстояние за няколко хопа.

Маршрутен алгоритъм е част от софтуера на мрежовото ниво, която определя по коя от изходните линии да се изпрати пристигнал пакет. За целта всеки маршрутизатор притежава **маршрутна таблица**.
Ако мрежата е с дейтаграми, решението трябва да се взима наново за всяка пристигнала дейтаграма, тъй като оптималният маршрут може да се е променил след последното изпращане. Ако мрежата използва виртуални канали, решенията по маршрутизацията се взимат само когато се създава виртуалният канал. Всички пристигнали пакети следват установения маршрут.

Маршрутизиращите протоколи трябва да отговарят на множество изисквания. Те трябва да са достатъчно прости и лесни за конфигуриране и да осигуряват надеждна и стабилна работа на мрежата. Отпадането на маршрутизатори или връзки между тях не бива да пречи на нормалното функциониране на останалите маршрутизатори, които трябва да бъдат в състояние да открият алтернативни пътища за доставяне на пакетите, ако такива съществуват. Минимизирането на времето за закъснение и максимизирането на общия поток в мрежата са други две цели на маршрутизиращите протоколи. Тези две цели са противоречиви - минимизирането на времето за закъснение е свързано с по-малък престой на пакетите в междинните възли, докато максимизирането 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 алгоритми маршрутизацията не се извършва на базата на текущата топология на мрежата. Маршрутите между всеки два възела в мрежата се изчисляват предварително и маршрутните таблици се попълват ръчно от мрежовите администратори. При промяна на топологията на мрежата (например при отпадане на възел или на връзка), администраторите ръчно трябва да променят маршрутните таблици, така че всеки два възела да останат свързани.
Това прави неадаптивните алгоритми приложими само в малки мрежи, при които рядко настъпват промени.
Неадаптивните алгоритми се наричат още **статични**.

Ще разгледаме примерна маршрутна таблица.
Мрежата представяме като граф - върховете са възлите в мрежата, а дъгите са комуникационните линии.
Метриката в графа се определя на базата на дължината на линиите, време-закъснението за минаване на един пакет, надеждност на линията, цена, брой хопове и др. Възможно е да се комбинират една или повече от изброените характеристики.
Нека е дадена следната мрежа:



Маршрутната таблица на възела J би могла да изглежда по следния начин:

A	A	0.63	I	0.21	H	0.16
B	A	0.46	H	0.31	I	0.23
C	A	0.34	I	0.33	H	0.33
D	H	0.50	A	0.25	I	0.25
E	A	0.40	I	0.40	H	0.20
F	A	0.34	H	0.33	I	0.33
G	H	0.46	A	0.31	K	0.23
H	H	0.63	K	0.21	A	0.16
I	I	0.65	A	0.22	H	0.13
J	--	--	--	--	--	--
K	K	0.67	H	0.22	A	0.11
L	K	0.42	H	0.42	A	0.16

Първата колона показва дестинацията на пакета. Нека, например, в J пристигне пакет за A. Тогава J генерира случайно число между 0 и 1. Ако числото е ≤ 0.63, пакетът се изпраща към A, ако числото е > 0.63 и ≤ 0.84 пакетът се изпраща към I,
ако числото е > 0.84, пакетът се изпраща към H.

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

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**, който гласи че 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).

Алгоритъм на Дейкстра

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.

При този алгоритъм на всеки връх се присвоява етикет, който съдържа дължината на най-добрия намерен път до върха за момента, както и непосредственият предшественик на върха по този път.

В началото всички върхове имат етикети ∞ (достатъчно голямо число), само началният връх има етикет 0.
На всяка стъпка на алгоритъма се избира върха с най-малък етикет, който не е бил избиран до момента. Ако пътят до някой от съседите на избрания връх е по-лош отколкото пътя до този съсед, минаващ през избрания връх, то етикетът на съседа се променя. След евентуалната промяна на етикетите на всички съседи на избрания връх се преминава към следващата стъпка на алгоритъма.
Резултатът от алгоритъма е дърво на оптималните пътища от дадения връх до всички останали.

Метод на наводняването

Методът на наводняването (flooding) е статичен алгоритъм за маршрутизация. Когато един пакет пристигне по дадена линия, той се изпраща по всички останали линии. Ясно е, че при този метод се получават дубликати на пакетите (тъй като между два възела обикновено има повече от един път). За да се избегне този проблем се въвежда идентификация на пакетите - всеки пакет съдържа идентификация на възела-източник и пореден номер. Всеки маршрутизатор поддържа по един списък с номерата на пакетите, които са минали през него за всеки възел-източник. Ако пристигнал пакет присъства в списъка, той не се препредава, в противен случай списъкът се актуализира и пакетът се предава. Списъците може да нарастнат много - при една голяма мрежа алгоритъмът не е приложим. За намаляване на дължината на един списък може да се добави допълнителен брояч k, който указва, че всички пакети до номер k са предадени. При това положение, пакетите с номера по-малки от k може да не присъстват в съответния списък.

Метод на случайното обхождане

Методът на случайното обхождане е неадаптивен алгоритъм. Когато един пакет пристигне на някоя линия, той се изпраща по случаен начин по една от останалите линии. Този метод няма сигурна сходимост. Ако всеки възел помни историята си, т.е. кой пакет в коя посока е изпратил, то може да се предотврати повторно изпращане на един и същи пакет в една и съща посока, което ще доведе до сходимост.
Такава информация, обаче, ще има твърде голям обем.

Централизирани адаптивни алгоритми

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

Изолирана маршрутизация

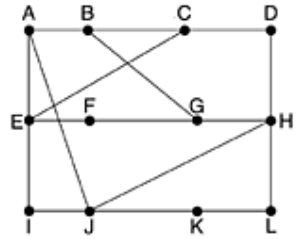
При изолираната маршрутизация всеки възел се разглежда сам за себе си. За всяка линия възелът поддържа по една изходяща опашка. При алгоритъма на Берън всеки пристигнал пакет се добавя към най-късата изходяща опашка в момента на пристигането. Този алгоритъм се адаптира към пиковете на предаване, но маршрутите не са оптимални.

9. Разпределена динамична маршрутизация с вектор на разстоянията.

При маршрутизацията с **вектор на разстоянието** (distance vector routing) всеки маршрутизатор изгражда и поддържа маршрутна таблица, в която всеки ред съдържа адрес на дадено местоназначение, адрес на следващата стъпка към това местоназначение по най-добрия известен до момента път и дължината на този път. Маршрутизаторите разменят само с директно свързаните към тях съседни маршрутизатори съобщения с информация от маршрутните си таблици за всички възли в мрежата.
Предполага се, че всеки маршрутизатор знае метриката на връзките до своите съседи. Ако метриката е брой хопове, разстоянието до всеки съсед е 1. Ако метриката е натоварване на възела, разстоянието до всеки съсед е броя на пакетите в изходящата опашка към този пакет. Ако метриката е време-закъснение, маршрутизаторът периодично изпраща "ехо" пакети до съседните му маршрутизатори и измерва закъснението на техния отговор.
Нека да предположим, че за метрика е избрано време-закъснението на пакетите.
Веднъж на всеки T милисекунди маршрутизаторите изпращат на своите съседи съдържанието на маршрутната си таблица.
Нека даден маршрутизатор J получи маршрутната таблица на съседа си X, като X_i е обявеното от X закъснение до маршрутизаторът i. Ако закъснението от J до X е m, то от J до всеки маршрутизатор i има път през X със закъснение $X_i + m$.
Възможни са четири случая:

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

Нека е дадена примерната мрежа



Векторите, които *J* получава от своите съседи *A*, *I*, *H*, *K* са следните:

To	A	I	H	K
A	0	24	20	21
B	12	36	31	28
C	25	18	19	36
D	40	27	8	24
E	14	7	30	22
F	23	20	19	40
G	18	31	6	31
H	17	20	0	19
I	21	0	14	22
J	9	11	7	10
K	24	22	22	0
L	29	33	9	9

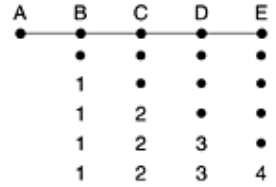
Измерените закъснения между *J* и съседите му *A*, *I*, *H*, *K* са следните:

JA	JI	JH	JK
8	10	12	6

На базата на тази информация, новият вектор на разстоянията, изчислен от *J* е следния:

To		
A	8	A
B	20	A
C	28	I
D	20	H
E	17	I
F	30	I
G	18	H
H	12	H
I	10	I
J	0	–
K	6	K
L	15	K

Сериозен недостатък на маршрутизиращите алгоритми с вектор на разстоянието е ниската им скорост на сходимост. Добрите новини се разпространяват бързо в мрежата, но лошите новини обикновено изискват твърде голям брой периодични съобщения за да достигнат до всички маршрутизатори.
Да разгледаме следния пример с метрика в хопове.

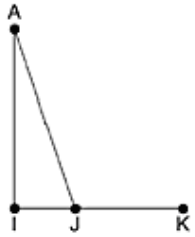


Нека маршрутизаторът *A* в началото не е включен в мрежата. Всички останали маршрутизатори знаят това - в маршрутната им таблица към направлението *A* е записано ∞ (достатъчно голямо число, трябва да е поне с единица повече от диаметъра на мрежата). Това е отразено на първия ред по-горе.
След включването на *A* останалите маршрутизатори научават за това събитие чрез няколко обмена на своите вектори на разстоянията, всеки от които се извършва едновременно между всички съседни маршрутизатори. При първата обмяна *B* научава от *A* за път с дължина 0 до *A* и записва в своята таблица, че *A* е на разстояние 1. В този момент останалите маршрутизатори все още не са научили за включването на *A*. Това е отразено на втория ред по-горе. При следващия обмен *C* научава, че от *B* съществува път до *A* с дължина 1 и записва в своя вектор път до *A* през *B* на разстояние 2 и т.н. По-общо в мрежа с диаметър *k* хопа са необходими най-много *k* размени на съобщения за разпространяване на новината за появил се по-добър път.

Да разгледаме друг пример.

Нека всички маршрутизатори в началото са включени в мрежата. Да предположим, че *A* спира да работи или се прекъсва връзката от *A* до *B*, което от гледна точка на *B* е същото. При първия обмен *B* не получава информация от *A*, но получава информация от *C*, че има път до *A* с дължина 2. *B* не знае, че пътя от *C* до *A* минава през него - от негова гледна точка би могъл да съществува друг независим път от *C* до *A*, затова *B* записва в таблицата си в реда за *A* път с дължина 3 и следваща стъпка *C*. *D* и *E* не променят маршрутните си таблици при първия обмен на векторите на разстоянията. На следващия обмен *C* научава за два възможни пътя до *A*, и двата с дължина 4, единият през *B*, другият през *D* - *C* избира и записва в маршрутната си таблица единия от тях в зависимост от реда на обработването на съобщенията от *B* и *D*. Резултатът от продължаващия обмен е отразен в следващите редове по-горе. Той ще продължи докато стойностите по направленията към *A* и в четирите машрутизатора не достигнат ∞. Този проблем се нарича **броене до безкрайност**.
Едно частично негово решение е т.н. **разделяне на хоризонта** (split horizon). При него се въвежда ново правило - ако в маршрутната таблица на *X* в реда за *Y* е записана следваща стъпка *Z*, то *X* не изпраща към *Z* информация за маршрута към *Y*.

В горния пример на втория обмен на вектори, *C* не изпраща към *B* информация за маршрута към *A*, тъй като маршрутът от *C* към *A* минава през *B*.
Въвеждането на разделяне на хоризонта не решава напълно проблемът броене до безкрайност.
Да разгледаме следния пример.



В началото *A* и *I* имат пътища с дължина 2 стъпки до *K* през *J*, а *J* има път с дължина 1 до *K*. Да предположим, че връзката между *J* и *K* отпадне. Тогава на първия последвал обмен *J* няма да получи информация за нов път до *K* през *A* или *I* по правилото за разделяне на хоризонта и правилно ще заключи, че *K* е недостижим. На следващия обмен *A* и *I* научават, че няма път до *K* през *J*, но *A* научава за път до *K* през *I* с дължина 3 и *I* научава за път до *K* през *A* с дължина 3.
Така въпреки разделянето на хоризонта, *A* и *I* ще броят до безкрайност.

10. Адаптивни маршрутни алгоритми – следене състоянието на връзката. Йерархична маршрутизация.

При маршрутизирането със следене състоянието на връзката (link state routing), всеки маршрутизатор трябва да извършва следните пет основни действия:

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

В резултат на тези пет действия се събира и разпространява до всички маршрутизатори информация за цялата топология на мрежата. Ще разгледаме по-подробно отделните действия.

Откриване на съседните маршрутизатори

След включването на един маршрутизатор неговата първа задача е да научи кои са съседите му. Това се постига чрез изпращане на “ехо” пакет по всяка от изходящите линии на маршрутизатора. От своя страна, всеки от съседите отговаря като съобщава името си. Това име трябва да бъде уникално в мрежата.
Ако два или повече маршрутизатора са свързани в мрежа с общодостъпно предаване (например Ethernet), откриването на съседите е малко по-сложно. Един възможен начин за представяне на връзките между тях е да се въведе допълнителен възел, който да отговаря на общата среда за предаване.

Измерване на цените на връзките

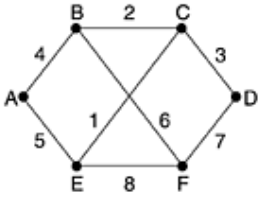
Всеки маршрутизатор трябва да може да определи време-закъснението до своите съседи. Най-простият начин е маршрутизаторът да изпрати "ехо" пакет към всеки свой съсед на който трябва директно да се отговори. Времето от изпращането на "ехо" пакета до получаване на отговора се дели на две и по този начин се получава времето-закъснение до съответния съсед. За по-точно измерване, този процес може да се повтори няколко пъти и да се вземе средната стойност. Този метод предполага, че връзките са симетрични, което не винаги е вярно.
Друг въпрос е дали при измерването да се взима предвид натовареността на възлите. Разликата се постига в зависимост от това кога маршрутизаторът стартира измерването - когато пакетът постъпва в съответната изходяща опашка или когато пакетът се придвижи в началото на опашката.
Включването на натовареността на възлите има предимства и недостатъци. Предимството е, че от две линии, които имат еднаква скорост за по-къса ще се счита по ненатоварената линия. Това ще доведе до по-голяма ефективност.
Недостатъкът може да се илюстрира със следния пример - нека една мрежа е разделена на две части, които са свързани чрез две линии *A* и *B*. Да предположим, че в даден момент по-голямата част от трафика между двете части на мрежата минава по линия *A*. Тогава при следващото изчисляване на маршрутните таблици трафикът ще се насочи към по-добрата линия *B*. Този процес ще се повтаря циклично и ще доведе до нестабилност в работата на мрежата.

Подготвяне на пакети с информация за състоянието на връзките (link state packets)

След като събере необходимата информация за състоянието на връзките си, следващата задача на маршрутизатора е да конструира пакет, който съдържа тази информация.
Пакетът трябва да съдържа уникалното име на подателя, пореден номер, срок на годност (ще разгледаме тези полета по-долу) и

списък със съседите на подателя, като за всеки съсед е указана цената на връзката до него.
Определянето на момента, в който трябва да бъдат подготвени и изпратени пакетите е важна задача.
Един възможен начин е това да става през определени равни интервали от време. Друга по-добра възможност е пакетите да се подготвят и изпращат само при промяна в топологията на мрежата - след отпадане или поява на нов съсед или промяна в цената на някоя връзка.

Нека да разгледаме следната примерна мрежа. Ребрата имат етикети със съответното време-закъснение.



Пакетите със състоянието за връзките за шестте маршрутизатора изглеждат по следния начин:

Link		State		Packets	
A	B	C	D	E	F
Seq.	Seq.	Seq.	Seq.	Seq.	Seq.
Age	Age	Age	Age	Age	Age
B 4	A 4	B 2	C 3	A 5	B 6
E 5	C 2	D 3	F 7	C 1	D 7
	F 6	E 1		F 8	E 8

Разпространяване на пакетите с информация за състоянието на връзката

Най-съществената част на алгоритъма е надеждното доставяне на пакетите с информацията за състоянието на връзката до всички маршрутизатори.
За разпространението на пакетите се използва методът на наводняването (flooding). При него всеки пакет се изпраща по всички линии, освен линията по която е пристигнал.
Обработката на всеки пристигнал пакет започва с проверка дали пакетът има по-голям пореден номер в сравнение с най-големия пореден номер, който е пристигнал до този момент от този източник. Ако номерът е по-голям, информацията от пакета се записва в таблицата с информация за състояние на връзките и пакетът се предава по останалите линии. Ако номерът е по-малък или равен, пакетът се отхвърля.

Този алгоритъм има някои проблеми. Ако поредният номер не е достатъчно голям, той може да се превърти. Затова се използват 32-битови поредни номера. Ако на всяка секунда пристига по един пакет, то за превъртане на номера ще са необходими около 137 години, което е достатъчно много.

В полето за срок на годност маршрутизаторът-подател указва продължителността на интервала от време в секунди, през който пренасяната от него информация трябва да се счита за валидна. Всеки маршрутизатор, който получи даден пакет намалява с единица стойността на това поле преди да го предаде към своите съседи. Освен това, след като маршрутизаторът запише данните от пакета в своята таблица, той продължава да намалява срока на годност на тези данни на всяка следваща секунда. Ако срока на годност стане 0, данните се изтриват. По този начин се премахва опасността остаряла информация за състоянието на връзките да се разпространява и използва прекалено дълго време от маршрутизаторите.

Таблицата с информация за състоянието на връзките, която се използва от маршрутизатор *B* в горния пример изглежда примерно по следния начин:

			Send flags			ACK flags			
Source	Seq.	Age	A	C	F	A	C	F	Data
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

Всеки ред от таблицата съответства на пристигнал, но все още необработен пакет. Полето *Source* е източникът на пакета, *Seq* е поредният му номер, *Age* е срокът на годност. С всеки пакет се свързват флагове за изпращане (*send flags*) и флагове за потвърждение (*ACK flags*) за всяка от изходните линии на маршрутизатора *B*. Флаговете за изпращане указват по кои линии трябва да се изпрати пакета. Флаговете за потвърждение указват по кои линии да се изпрати потвърждение за получаването на пакета.
Ако в *B* пристигне дубликат на някой от пакетите в таблицата, то съответните флагове трябва да се актуализират. Например, ако пристигне дубликат на пакета със състоянието на *C* от *F*, преди този пакет да бъде препратен към *F*, то флаговете за изпращане на пакета ще се променят на 100, а флаговете за потвърждение - на 011.

Изчисляване на новите маршрути

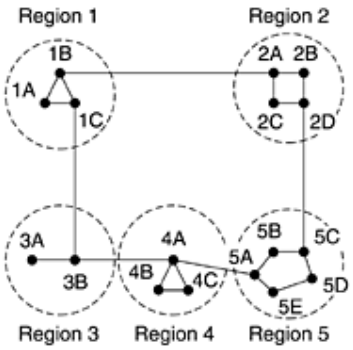
След като един маршрутизатор получи пълна информация за състоянието на връзките на всички останали маршрутизатори, той може да приложи алгоритъма на Дейкстра. Всъщност всяка връзка се представя два пъти, по веднъж за всяка посока. Двете цени могат да се усреднят или да се използват отделно.

Изчислените маршрути се записват в маршрутните таблици. Необходимата памет за съхраняване на информацията за състоянието на връзките за мрежа с n маршрутизатори, всеки от които има по k съседи е пропорционална на nk.
Така големите по размер мрежи изискват използване на маршрутизатори с голям обем памет.

Йерархична маршрутизация

С увеличаването на размерите на мрежата нараства обемът на маршрутните таблици, което изисква повече памет и процесорно време за тяхната обработка. Това налага въвеждането на йерархично маршрутизиране, при което мрежата се разделя на **области**. Маршрутизаторите в една област знаят всичко за вътрешната структура на своята област, но не знаят вътрешната структура на останалите области. За по-големи мрежи може да е необходима йерархия с повече от две нива.

Като пример да разгледаме следната мрежа с йерархична маршрутизация на две нива. Метриката е в хопове.



Пълната таблица за маршрутизатора *1A* съдържа 17 реда и има следния вид:

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2A	1B	2
2B	1B	3
2C	1B	3
2D	1B	4
3A	1C	3
3B	1C	2
4A	1C	3
4B	1C	4
4C	1C	4
5A	1C	4
5B	1C	5
5C	1B	5
5D	1C	6
5E	1C	5

Съкратената таблица за маршрутизатора *1A* съдържа 7 реда и има следния вид:

Dest.	Line	Hops
1A	–	–
1B	1B	1
1C	1C	1
2	1B	2
3	1C	2
4	1C	3
5	1C	4

В нея са запазени маршрутите към направленията в "област 1". Маршрутите към направленията в останалите области са обобщени в един ред, като се използва маршрутизатора от съответната област, който е най-близо до маршрутизатора *1A*. Спестяването на памет от маршрутните таблици има отрицателен ефект - някои от пътищата увеличават своята дължина. Например, най-късият път от *1A* до *5C* минава през "област 2", но при йерархично маршрутизиране ще се използва пътят през "област 3", тъй като това е по-добре за повечето маршрутизатори от "област 5".

Ако n е броят на маршрутизаторите в една мрежа, може да се покаже, че оптималният брой области, всяка с по равен брой маршрутизатори е най-близкото цяло до $\sqrt{n}$.

11. Натоварвания и управление на потоците в мрежата.

Когато броят на изпратените пакети в мрежата надвиши капацитета на мрежата се получава задръстване.
Задръстването може да се причини от няколко фактора.
Ако в даден маршрутизатор започват да постъпват пакети от три или четири линии и всички те трябва да се изпратят по една и

съща линия ще се образува опашка. При недостатъчно памет ще се изгубят пакети.
Бавни процесори също могат да причинят задръстване.
Ако маршрутизаторите не могат да обработват достатъчно бързо пристигащите пакети отново ще се образуват опашки, въпреки че линиите не използват докрай своя капацитет.

Разликата между управление на натоварването и управление на потоците е, че управлението на натоварването е свързано с това цялата мрежа да може да се справи с определен трафик - то е на глобално ниво, засяга всички възли в мрежата, съхраняването и препращането на пакетите в маршрутизаторите и всички останали фактори, които намаляват капацитета на мрежата. Управлението на потоците е свързано с двупосочния трафик между даден източник и приемник. То е свързано с това бърз източник да не изпраща продължително данни, по-бързо отколкото приемника може да ги обработи. Управлението на потока обикновено се осъществява чрез обратна връзка от приемника към източника по която се съобщава на източника за състоянието на приемника. То се осъществява най-вече на транспортно ниво.

Ще опишем някои стратегии за справянето с описаните проблеми.

Броят на буферите в един възел на мрежата е ограничен.
Пристигнал пакет постъпва във входен буфер, след това преминава в изходен буфер, след това се изпраща потвърждение към източника.
Нека възелът има K изходни буфера, m е максималният брой буфери на една изходна линия, s е броят на изходните линии. Ако $K = m$, тогава е възможно всички изходни буфери да се ангажират с една натоварена линия, което ще доведе до задръстване.
Оптимално натоварване на възела се постига при $m = K/\sqrt{s}$.
С една изходна линия може също да се свърже минимален брой буфери, които не могат да се ползват от други линии.

Друга стратегия за избягване на задръстването е маршрутизаторите да отстраняват пакети по желание.
Най-просто е да се отстраняват произволни пакети.
Друга възможност е пакети да се отстраняват в зависимост от вида на предаваната информация. При трансфер на файлове има смисъл да се отстраняват по-новите пакети, тъй като отстраняването на стар пакет може да доведе до повторно предаване на следващите го пакети, ако получателят приема пакети само с поредни номера.
От друга страна, при трансфер на мултимедия, по-новите пакети са по-важни от по-старите, тъй като се цели минимално закъснение.

Трета стратегия е ограничаването на общия брой пакети в мрежата. За целта в мрежата се пускат служебни пакети, които представляват разрешения за изпращане на пакет. За да изпрати пакет един хост, прилежащият му маршрутизатор трябва да улови разрешение. Обратно, когато един хост получи пакет, разрешението се възстановява.
По този начин броят на пакетите в мрежата се контролира от броя на разрешенията. Този алгоритъм е сложен за поддръжка.

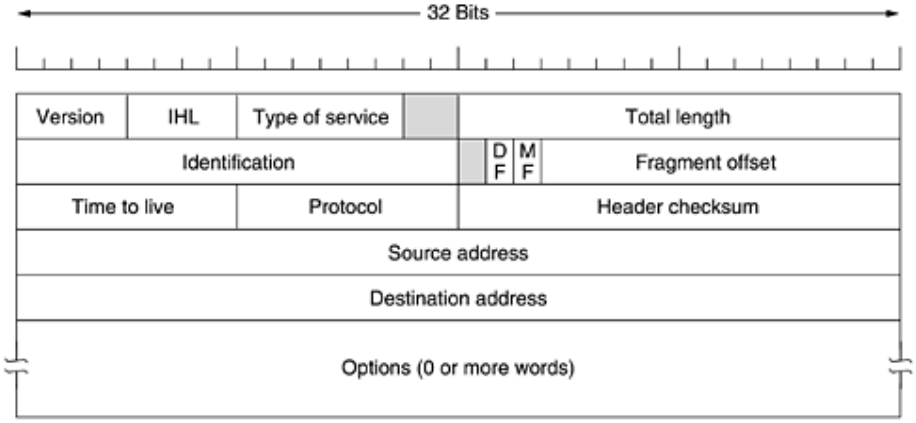
Последната стратегия е регулиране на входа при задръстване. Във всеки маршрутизатор се следи натовареността на изходните линии. Когато се надвиши някакъв праг на натовареност линията се слага в състояние на предупреждение. Когато линията е в такова състояние в обратна посока се изпращат специални задръстващи пакети. Чрез тях изпращача разбира, че трябва да намали скоростта на предаване. В крайна сметка задръстващите пакети достигат до хоста, който генерира пакетите, довели до задръстването и го уведомяват да намали темпото на предаване. Проблем е, че регулирането на входа при задръстване не се прави на транспортно ниво, където е известен пътя между хостовете и затова е трудно задръстващите пакети да намерят пътя си до хоста-източник.

Друг голям проблем при управление на натоварването на мрежите е проблемът **deadlock**. Да разгледаме пример с три възела *A*, *B* и *C*. Буферите и на трите възела са пълни. *A* има пакети за предаване към *B*, *B* има за *C*, а *C* има за *A*. По този начин трите възела са блокирани. Кръгът от блокирани възли може да е много по-голям. Проблемът е, че трудно се установява наличието на deadlock в мрежата. Поради това deadlock трябва да се предотвратява.

12. Мрежи с IP-протокол – адресация, подмрежи и маски.

От гледна точка на мрежовото ниво Internet е съвкупност от **автономни системи** (**AS**). Всяка автономна система се състои от една или повече мрежи. В рамките на една автономна система има фиксирани правила за предаване и фиксиран размер на пакета. Internet всъщност изгражда правила за връзка между отделните автономни системи. За целта се използва протоколът **IP** (internet protocol). Между автономните системи данните се придвижват под формата на **дейтаграми**. Задачата на IP е да извърши успешно предаване на дейтаграмите от източника до получателя без значение дали те са в една и съща мрежа или в различни мрежи.
Комуникацията в Internet най-общо се извършва по следния начин: транспортното ниво взима потоци от байтове и ги разделя на дейтаграми. Дейтаграмите могат на теория да достигнат 64Kb, но на практика те не са по-големи от 1500b. Всяка дейтаграма се

изпраща самостоятелно, като по пътя тя може да се фрагментира на по-малки единици. Когато тези единици достигнат до получателя те се реасемблират от мрежовото ниво за получаване на оригиналната дейтаграма. По-нататък тази дейтаграма се подава на транспортното ниво на получателя, което я вмъква в съответния поток от байтове.
Ще разгледаме формата на IP-дейтаграмата във версия 4 (4-байтови адреси). Ще отбележим, че дейтаграмата се предава в Big-Endian формат, т.е. от старшите към младшите битове.
IP-дейтаграмата се състои от заглавна част и част за данни. Заглавната част е 20B+опции с променлива дължина и има следния формат:

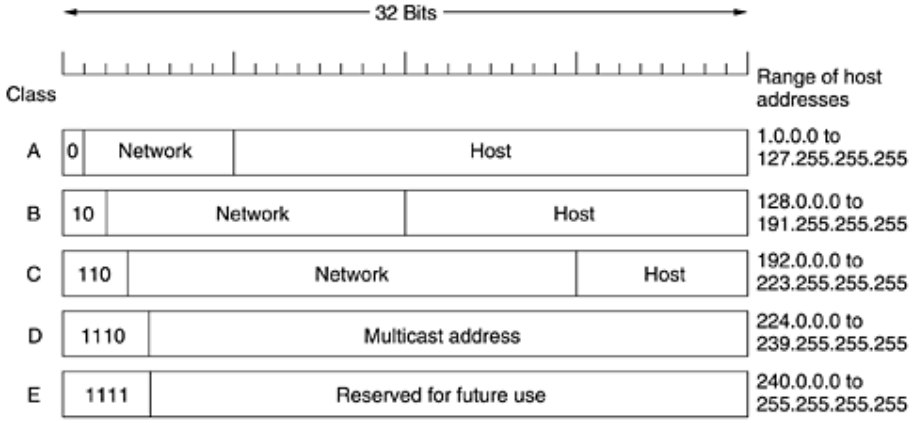


Полето Version указва версията на протокола, към който принадлежи дейтаграмата.
Полето IHL указва дължината на заглавната част в 32-битови думи. То е необходимо, тъй като полето Options има променлива дължина. Минималната стойност на това поле е 5, което отговаря на случая когато полето Options е празно. Максималната стойност е 15, което ограничава заглавната част до 60B, т.е. полето за опции до 40B.
Полето Type of service показва какво обслужване очаква дейтаграмата. Различните видове данни, например видеоизображение, глас, файлове предполагат различно обслужване. Практически сегашните маршрутизатори не обръщат внимание на това поле.
Полето Total length съдържа общата дължина на дейтаграмата (заглавна част + данни). Максималната дължина е 65535 байта.
Полето Identification съдържа номер на дейтаграмата. Всички фрагменти на една и съща дейтаграма имат еднакъв номер и по този начин получателя разбира кой фрагмент към коя дейтаграма принадлежи.
Флагът DF(don't fragment) указва на маршрутизаторите да не фрагментират дейтаграмата. Всички автономни системи трябва да могат да приемат фрагменти от поне 576B. Ако размерът на фрагментите е по-голям и флагът DF е 1, то дейтаграмата може да пропусне някоя автономна система с по-малка дължина на пакета, дори тя да се намира на оптималния маршрут.

Флагът MF(more fragments) за всички фрагменти на дейтаграмата, освен последния е 1, а за последния е 0, т.е. той показва дали получен фрагмент е последен в дейтаграмата или не.
Полето Fragment offset указва къде се намира фрагмента в оригиналната дейтаграма. Всички фрагменти, освен последния трябва да са с дължина кратна на 8B. Тъй като полето Fragment offset е 13 бита максималният брой фрагменти в една дейтаграма е 8192.
Полето Time to live е брояч, който ограничава продължителността на живота на дейтаграмата. То отброява времето в секунди, има дължина 8 бита, така че максималното време за живот е 255 секунди. Това поле се намаля с единица на всеки hop, а освен това се намаля с единица и за всяка секунда престой в маршрутизатор. Когато полето стане 0, дейтаграмата се премахва и в обратна посока се изпраща предупредителен пакет.
Полето Protocol указва протокола на транспортно ниво, към който трябва да се предаде дейтаграмата. Той може да бъде TCP (transmission control protocol), UDP (user datagram protocol) или някой друг.
Полето Header checksum е контролна сума само на заглавната част. Тя трябва да се преизчислява на всеки hop, тъй като поне едно поле се променя - Time to live.
Полетата Source Address и Destination address съдържат съответно адрес на източника и адрес на получателя.
Възможни са различни опции в полето Options.
Първата опция е Security – тя обявява секретност на дейтаграмата.
Втората опция е Strict source routing и тя дава целия път от източника до получателя като последователност от IP-адреси. Дейтаграмата трябва задължително да следва този път.
Третата опция е Loose source routing и тя указва дадена последователност от маршрутизатори да бъде посетена в указания ред от дейтаграмата, но е възможно тя да премине и през други маршрутизатори.
Четвъртата опция е Record route и тя указва маршрутизаторите по пътя на дейтаграмата да добавят своя IP-адрес към полето на опцията. Това позволява да се проследят грешки при маршрутизирането.
Последната опция е Timestamp, която е подобна на опцията Record route, но с тази разлика, че маршрутизаторите освен IP-адреса си записват и времето по което е минала дейтаграмата.

Всеки хост и маршрутизатор в мрежата има IP-адрес.
Всички IP-адреси са 32-битови. Всеки IP-адрес се дели на две части – номер на мрежа и номер на хост. Номерът на мрежата е непрекъсната порция от битове в лявата част на адреса, а номерът на хоста е останалата непрекъсната порция от битове в дясната част на адреса.

В зависимост от структурата си IP-адресите се делят на следните пет класа:

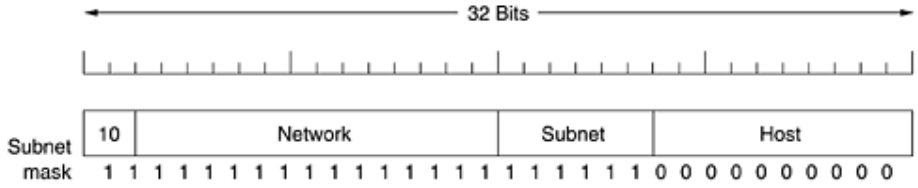


Указаните битове в началото на адреса, които определят неговия клас се наричат **сигнални битове**.
В клас A са възможни 127 мрежи, всяка с приблизително 16000000 хоста.
В клас B са възможни приблизително 16000 мрежи, всяка с приблизително 65000 хоста.
В клас C са възможни приблизително 2000000 мрежи, всяка с по 254 хоста.
Клас D е предназначен за работа с групови адреси, а клас E е резервиран за бъдеща употреба.

За удобство IP-адресите се изписват в точкова десетична нотация, като всеки от четирите байта се изписва като десетично число от 0 до 255. Най-малкия IP-адрес е 0.0.0.0, а най-големия 255.255.255.255.
Числото 0 се интерпретира по специален начин в IP-адресите – то означава тази мрежа или този хост. Адрес, който съдържа само единица се интепретира като broadcast-адрес, т.е. адресират се всички хостове в дадена мрежа.

Големият недостатък на IP-адресацията е, че половината адреси са от клас A и се разпределят само между 127 автономни системи, въпреки че всяка от тях може да съдържа милиони хостове.
Всяка мрежа трябва да има уникален номер и всички хостове в дадена мрежа трябва да имат един и същ номер на мрежата.
Това води до проблеми при нарастване на броя на мрежите.
Решението на проблема е да се разреши разделянето на една мрежа на **подмрежи**, но за външния свят тя да изглежда като една мрежа. За целта полето за мрежов номер се разширява надясно, като се отнемат битове от номера на хост.
Например за един адрес от клас B вместо 14 бита за номер на мрежата и 16 бита за номер на хост се използват 20 бита за номер на мрежа, като десните 6 от тях са за номер на подмрежа и 10 бита за номер на хост.

За имплементация на подмрежите маршрутизаторите се нуждаят от **мрежова маска**, която определя границата между номера на мрежата + номера на подмрежата и номера на хоста. В горния пример мрежовата маска изглежда по следния начин:



Всеки маршрутизатор има таблица с два вида IP-адреси. Едните са от вида (номер на мрежа, 0) и те указват как се стига до други мрежи, а вторите са от вида (номер на тази мрежа, номер на хост) и те указват как се стига до хостовете в мрежата на маршутизатора. Когато пристигне IP-пакет неговият адрес на получател се преглежда. Ако пакетът е за друга мрежа, той се пренасочва към съответния маршрутизатор, ако е за тази мрежа той се изпраща към съответния хост. Ако пакетът е за неизвестна мрежа, той се предава към маршрутизаторът по премълчаване.

При въвеждане на подмрежи таблицата на маршрутизатора се променя – добавят се IP-адреси от вида (номер на тази мрежа, номер на подмрежа, 0) и (номер на тази мрежа, номер на тази подмрежа, номер на хост). По този начин маршрутизаторът в дадена подмрежа знае как се стига до останалите подмрежи и как се стига до хостовете в неговата подмрежа. Маршрутизаторът извършва логическо & на подмрежовата маска с адреса на получателя и по този начин получава адреса на мрежата и съответната подмрежа.

13. 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.

14. Маршрутни протоколи RIP. Рутери.

RIP (routing information protocol) е 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:

Command	Version	Routing domain
Address family		Route tag
IP address		
Netmask		
Next hop IP addres		
Metric		

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.

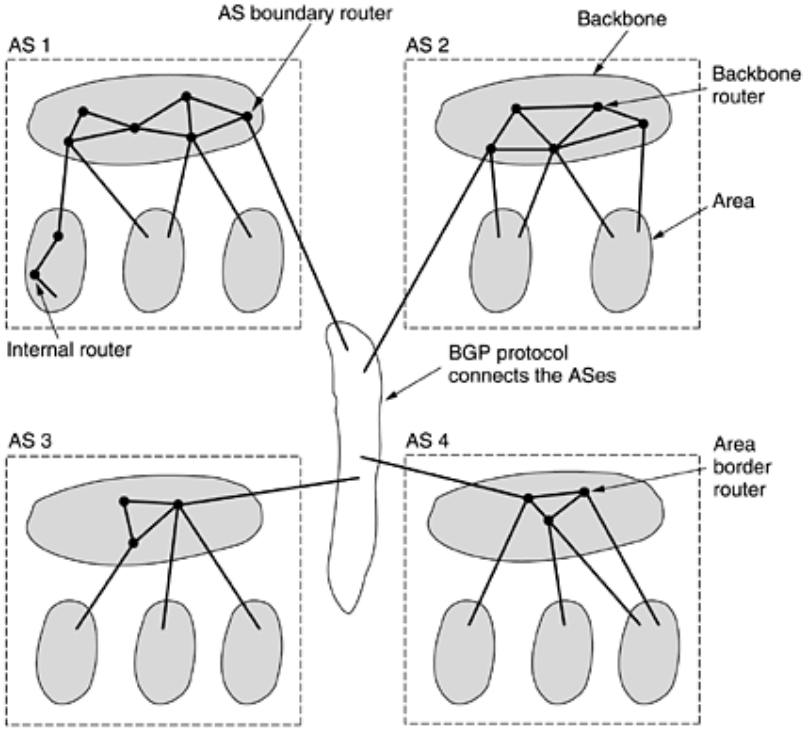
Internet е съвкупност от физически различни мрежи, които са обединени посредством общ протоколен стек, така че логически да се формира една обща мрежа. Най-лесният начин да се изгради такава мрежа е чрез свързване на две или повече мрежи чрез **маршрутизатор** (**router**). Маршрутизаторът представлява 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, пропускателна способност на комуникационните линии, натоварване на маршрутизаторите, цена на комуникационните линии.

15. 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).



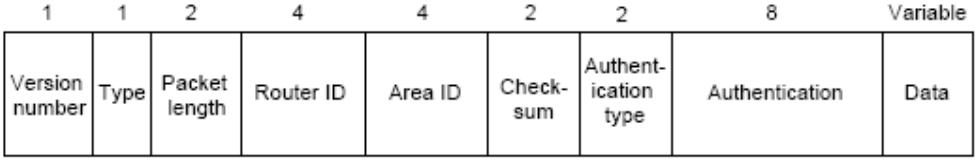
Опорните маршутизатори могат същевременно да бъдат вътрешни, ако всичките им интерфейси са свързани към опорната мрежа.
В рамките на една област всички маршрутизатори имат една и съща топологична база данни. Нейното предназначение е да се изчислят най-късите пътища между дадения маршрутизатор и всички 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.

16. 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.

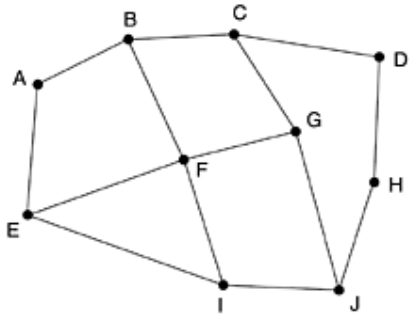
Политическите ограничения се конфигурират ръчно и не са част от BGP протокола.

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.
Като пример да разгледаме следната мрежа от BGP маршрутизатори:



По-конкретно да разгледаме маршрутизатора *F*. Той получава от съседите си следните пътища до *D* – път *BCD* от *B*, път *GCD* от *G*, път *IFGCD* от *I* и път *EFGCD* от *E*. *F* преглежда тези пътища за да определи кой е най-добрия. Той премахва пътищата от *I* и *E*, тъй като минават през него. Изборът е между *B* и *G*. Всеки BGP маршрутизатор съдържа модул, който изследва пътищата до дадено местоназначение и ги оценява. Път, който нарушава политическо ограничение автоматично получава оценка безкрайност. Маршрутизаторът избира пътят с най-ниска оценка. Оценяващата функция не е част от BGP протокола.
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.

17. Групова и безкласова маршрутизация в IP мрежи.

Протоколът IP поддържа групова маршрутизация, като използва IP адресите от клас D. Всеки адрес от клас D идентифицира една група хостове, използват се 28 бита за идентифициране на групите, което дава около 250000000 групи. Когато един процес изпрати пакет към адрес от клас D, се прави опит този пакет да се достави до всички членове на съответната група. Груповата маршрутизация се извършва от специални multicast маршрутизатори, които комуникират с хостовете посредством IGMP протокола.

Големият проблем на IP протокола е недостига на адреси. По принцип съществуват над 2 милиарда адреси, но организацията им по класове е неефективна и заради нея се губят милиони адреси. За повечето организации мрежа от клас A със 16 милиона адреси е твърде голяма, а мрежа от клас C с 256 адреси твърде малка. Затова се избират мрежи от клас B с 65536 адреси. Мрежите от клас B, обаче, са твърде недостатъчно – 16384, докато през 1996 в Internet вече са свързани над 100000 мрежи.

Друг проблем е големината на маршрутните таблици. От гледна точка на маршрутизаторите, IP адресите се състоят от номер на мрежата и номер на хоста. Маршрутизаторите не трябва да знаят всички хостове, но те трябва да имат информация за всички мрежи. Ако се използват, например, половин милион клас C мрежи, то всеки маршрутизатор трябва да има маршрутна таблица с половин милион реда, по един за всяка мрежа, който описва по коя линия се стига до съответната мрежа. Съхраняването на таблици с половин милион реда да кажем е възможно, въпреки че повечето маршрутизатори съхраняват таблиците си в оперативната памет, но друг сериозен проблем е сложността на алгоритмите за маршрутизация. Освен това, маршрутизаторите периодично трябва да изпращат маршрутните си таблици – колкото са по-големи, толкова по-голяма е вероятността части от тях да се загубят при предаването.

Решение на описаните проблеми е използване на безкласова маршрутизация **CIDR** (classless interdomain routing). Основната идея е, че незаетите IP адреси се разпределят по блокове с различна големина (степен на 2), без да се взимат под внимание класовете.

Премахването на класовете усложнява маршрутизацията. При старата система, с класове, тя действа по следния начин: когато един пакет пристигне в маршрутизатор, копие на неговия IP адрес се измества наляво с 28 бита за получаване на четирите сигнални бита, които определят класа на адреса. В зависимост от класа (A, B или C) се извлича номера на мрежата. След това този номер се търси в таблицата, съответна на класа на адреса и когато се намери, пакетът се препредава по съответната изходна линия.

При CIDR този алгоритъм не е приложим. Всеки ред от маршрутната таблица се разширява с 32-битова маска. По този начин маршрутната таблица е масив от тройки (IP адрес, маска, изходяща линия). За удобство IP адресът и маската се записват по следния начин: A.B.C.D/N, където N е броят на единиците в лявата част на маската. Например, 194.24.8.0/22 е мрежа, съставена от 4 мрежи от клас C. Когато пристигне пакет, се извлича IP адресът на получателя. После маршрутната таблица се сканира ред по ред, като за всеки ред съответната маска се прилага към IP адреса и се сравнява получения номер с номера на мрежата на реда. Възможни са няколко съвпадения – в такъв случай се избира реда с най-дълга маска.

18. Транспортно ниво. Протоколи със съединения. Процедури.

Най-общата цел на транспортното ниво е да осигури ефективни и надеждни услуги на своите потребители, които обикновено са процеси в приложното ниво. Транспортното ниво трябва да прехвърля произволно дълги съобщения между два крайни абоната. За да постигне тези цели транспортното ниво използва услугите на по-долното мрежово ниво. Транспортното ниво предлага два вида обслужване – със съединение (надеждни) и без съединение (ненадеждни).
При протоколите със съединение имаме три фази – установяване на съединение, трансфер на данни и закриване на съединението. Транспортното ниво трябва да предоставя интерфейс на приложното ниво. Примерен интерфейс за надеждно обслужване се състои от следните примитиви LISTEN, CONNECT, SEND, RECEIVE, DISCONNECT. За да опишем тяхното действие да предположим, че имаме приложение с един сървър и много отдалечени клиенти. Първоначално сървърът изпълнява примитива LISTEN, който го блокира докато не се появи клиент. Когато клиентът иска да се свърже със сървъра, той изпълнява примитива CONNECT. Това изпълнение блокира клиента и изпраща пакет за установяване на връзка към сървъра. Ако сървъра е в състояние на слушане (блокиран след изпълнение на LISTEN), той изпраща обратно пакет за потвърждение за създаване на връзка, което отблокира клиента и връзката е създадена. След установяване на връзката сървъра и клиента могат да обменят данни, като например всеки от тях изпълнява RECEIVE за изчакване на другия да изпълни SEND. Докато двете страни могат да спазват реда, в който трябва да предават данни тази схема работи добре. Всеки изпратен пакет данни трябва да се потвърждава обратно към изпращача. Когато връзката вече не е нужна, тя трябва да се прекрати за да се освободят заетите от нея ресурси. Освобождаването на връзката има два варианта – симетричен и асиметричен. При асиметричният вариант, само клиентът може да прекрати връзката чрез изпълнение на

примитива DISCONNECT, който води до изпращане на пакет за прекратяване на връзката до сървъра. При симетричния вариант и двете страни трябва да затворят връзката, т.е. да изпълнят DISCONNECT независимо един от друг. Когато едната страна изпълни DISCONNECT, това означава че тя повече няма да предава данни, но не означава че няма да приема данни.

Услугите, предоставяни от транспортното ниво се имплементират от протокола на транспортно ниво, който действа между транспортното ниво на двата крайни абоната. Всъщност транспортните протоколи си приличат с протоколите на канално ниво. И двата типа протоколи се грижат за отстраняването на грешки, за последователно предаване на данните, за управление на потока и др. Съществуват и много разлики. При протоколите от каналното ниво двете точки взаимодействат помежду си през физически канал, докато при транспортното ниво този физически канал се заменя от цялата комуникационна подмрежа. Друга съществена разлика е, че при изпращане на кадър от каналното ниво той или пристига или се изгубва, докато пакетите при транспортното ниво могат да престояват в маршрутизаторите, да обикалят цялата подмрежа, докато достигнат местоназначението.

Когато един процес от приложното ниво иска да се свърже с друг процес, той трябва да може да го идентифицира. Методът, който се използва е да се присвоят адреси на процесите, които слушат за създаване на връзка. В Internet тези адреси се наричат **портове** и са 16-битови номера. Комбинацията от IP адрес и порт се нарича socket и той уникално идентифицира процес, който се изпълнява на някакъв хост.

Установяването на съединение изглежда просто, но не е така. На пръв поглед е достатъчно едното транспортно ниво да изпрати пакет за създаване на връзка, а другото да отговори с пакет за приемане на връзката. Проблемите идват от факта, че пакетите могат да се изгубят, да се забави предаването им или да се дублират.
За преодоляване на тези проблеми при установяване на връзка се използва процедура за **трикратно договаряне** (three-way handshake). Да предположим, че крайните абонати A и B искат да установят връзка. Изпълняват се следните три стъпки:

1. A изпраща заявка, че иска да установи съединение – указва размера на буферите, брой на буферите, скоростта на предаване и др. Тази заявка е специално служебно съобщение на транспортно ниво. За мрежовото ниво заявката е обикновен пакет. Съобщението може да стигне до B, но може и да не стигне. За целта A си включва таймер и ако той изтече преди да се получи отговор от B, A изпраща заявката наново. След три неуспешни опита A решава, че B е недостъпен и се отказва;

2. В отговаря на А, като включва в отговора собствени параметри за връзката;
3. А потвърждава съединението на В. Това потвърждение е необходимо, тъй като В трябва да знае, че А е получил отговора. След успешно получаване на потвърждаване А и В заделят заявените ресурси.

Прекратяването на връзката също се извършва с трикратно договаряне.

19. Протоколи - TCP/IP. Архитектура и основни функции.

Моделът **TCP/IP** за пръв път е използван в мрежата ARPANET, която е предшественик на Internet. Тя е трябвало да свърже стотици университети и правителствени организации посредством арендовани линии. Освен това, към тази мрежа се поставя изискването за включване на сателитни и радио мрежи, за което съществуващите протоколи не са подходящи. Това налага създаването на един нов еталонен модел, който да позволи свързването на различни мрежи по един безболезнен начин, като се запазват основните изисквания, поставени от конструкторите. Подобна гъвкава архитектура е била необходима, тъй като отделните приложения са варирали от изискването за предаване на файлове до предаване на говор в реално време.
Всички тези изисквания са довели до избор на мрежа с комутация на пакети, базирана на обслужване с неустановена връзка, което се реализира от слой, наречен "**интернет слой**". Той е основата на цялата мрежова архитектура. Негово основно предназначение е да позволи на хостовете от дадена мрежа да изпращат пакети, които се предават независимо един от друг до съответното местоназначение, намиращо се обикновено в друга мрежа. Те могат даже да пристигнат в различен ред от този, в който са били изпратени. В този случай задачата на по-високите слоеве е да ги пренареди в реда на изпращане, ако съответното приложение изисква това. Интернет слоя дефинира формат на пакета и съответен протокол, наречен интернет протокол **IP** (internet protocol). Една от неговите функции е да доставя IP пакетите до исканото местоназначение. Комутирането на пакетите е друга важна задача на този слой, както и предотвратяването на претоварвания в мрежата. Поради тази причина може да се каже, че интернет слоя е подобен във функционално отношение на мрежовия слой от OSI модела.
В модела TCP/IP над интернет слоя се намира **транспортният слой**. Той е конструиран, така че да позволи на двойка обекти от едноименни протоколни нива в хоста-източник и хоста-получател да провеждат коректен обмен на съобщения. Неговите функции са същите, като тези на транспортния слой в OSI еталонния модел. Два протокола от типа "край-край" са дефинирани в

транспортния слой на TCP/IP модела. Първият протокол е **TCP** (transmission control protocol), който предоставя надеждно обслужване с установена връзка. Този протокол позволява потокът от байтове, изпратен от една машина да бъде предаден без грешка до произволна друга машина. В хоста-източник той фрагментира потока от байтове в отделни съобщения, които предава на интернет слоя. В хоста-получател транспортният слой реасемблира приетите съобщения в изходящ поток от байтове. TCP също така управлява потока от съобщения по такъв начин, че машината, която изпраща съобщения с по-висока скорост, да не препълни машината, която получава тези съобщения, но работи с по-ниска скорост. Вторият протокол на транспортния слой е **UDP** (user datagram protocol). Той осигурява ненадеждно обслужване с неустановена връзка. UDP се използва при приложения, които не изискват механизмите на TCP протокола за запазване от последователността на съобщенията и управление на потока от информация – например при предаване на звук.
При TCP/IP модела няма сесиен и представителен слой, тъй като няма необходимост от това.
Над транспортното ниво е **приложният слой**, който включва протоколи за различни приложения. Такива са TELNET (виртуалните терминали), FTP (обмен на файлове), SMTP (електронна поща), DNS (съответствие между име на хост и неговия IP адрес), NNTP (трансфер на USENET новини), HTTP (обмен на уеб-страници) и др.
Под интернет слоя в TCP/IP модела има голяма празнина. Споменава се за **слой за достъп до мрежата**, чрез който хостовете трябва да се свържат към мрежата посредством някакъв протокол, така че да могат да изпращат и приемат IP пакети. Този протокол не е дефиниран в TCP/IP модела и той варира в различните мрежи.
Недостатък на TCP/IP модела е, че той не прави хубав паралел между услуги, интерфейси и протоколи – нещо което е добре направено в OSI модела. TCP/IP моделът не е достатъчно общ, той не може да се използва за описание на друг протоколен стек. Освен това той не различава каналното и физическото ниво, които значително се отличават по своите функции.

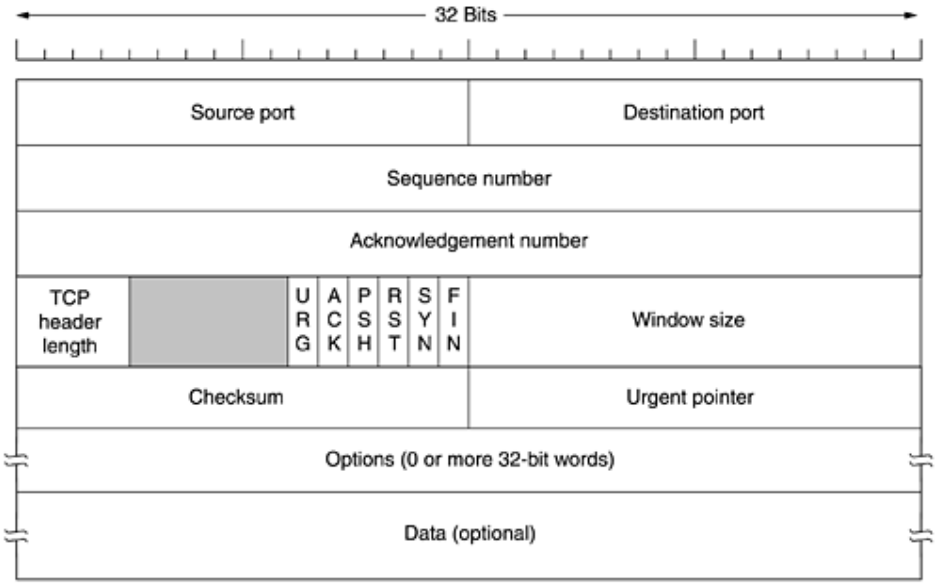
20. Транспортни протоколи TCP и UDP – формати.

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:

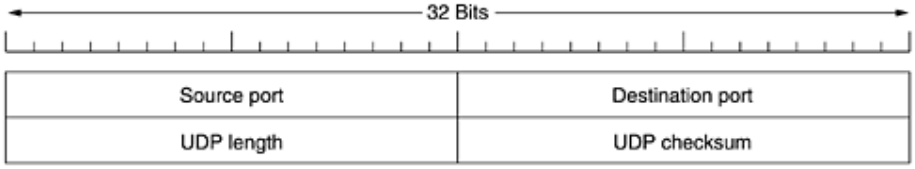
1. Хостът (клиентът), който отваря връзката, изпраща SYN сегмент. В същия сегмент клиентът указва номера на порта на сървъра, с който трябва да се установи връзка, и началният номер x на потока байтове, който клиентът ще предаде към сървъра.
2. Сървърът отговаря със собствен SYN сегмент, включващ началния 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.

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.

21. Домейнна система за именоване и плоски логически имена – DNS и NETBIOS сървъри и клиенти. DNS протокол. ETC файлове.

Чрез IP адресите се осъществява адресирането на дейтаграмите, които носят в себе си данните. Неудобното е, че те са числа и не могат лесно да се използват от човека. Затова се въвеждат системи за именуване. Ще разгледаме две от тях – **NETBIOS** и **DNS**.

NETBIOS се развива в мрежите от персонални компютри и се възприема впоследствие от Microsoft като система за именуване на обектите в мрежата. Имената в NETBIOS са плоски и са с дължина до 15 символа. Те са уникални в рамките на една затворена мрежа. На NETBIOS имената съответстват IP адреси. Те трябва да могат да се регистрират и да се изтриват, т.е. NETBIOS системата има нужда от поддръжка. NETBIOS разчита на broadcast обмен в рамките на една затворена среда – например локална мрежа Ethernet или Token Ring.
Съществуват два начина за съхраняване на NETBIOS имената. Първият начин е всички имена да се съхраняват в таблица в клиента. При търсене на име се проверява в таблицата дали то го има, ако го няма се пуска broadcast в съответния сегмент на мрежата. Този broadcast е на канално ниво и той не минава през маршрутизаторите, но минава през мостовете. С него се запитва дали някой има търсеното име в таблицата си.
Вторият начин е поддържане на специален NETBIOS именен сървър, в който се регистрират имената и към който се пращат заявки дали дадено име е регистрирано. Този именен сървър се нарича **WINS**. Такъв сървър се прави за всеки сегмент.

В ранните дни на Internet в UNIX в директорията etc е имало локален ресурсен файл HOSTS, в който се съхранявали съответствията между имена и IP адреси. Други използвани файлове са PROTOCOL, SERVICES, NETWORKS. В наши дни този файл в UNIX е преименуван на RESOLV.CNF, а в WINDOWS той се намира в поддиректорията SYSTEM32\DRIVERS\ETC и се нарича LMHOSTS. По принцип в този файл се съдържа само записа
127.0.0.1 localhost
, но може да се съдържат и други записи, например
102.54.94.100 rhino
102.54.94.123 popular #PRE
Маркерът #PRE означава, че съответния запис още със стартирането на системата ще се зареди в кеша, което означава че съответните имена ще са първите, които се проверяват при търсене.
Файлът PROTOCOL съдържа описание на протоколите, техните имена и идентификатори:
ip 0 IP
tcp 7 TCP
udp 17 UDP
rdp 27 RDP
Файлът SERVICES описва well-known портовете:
www 80
ftp 21
Файлът NETWORKS съдържа описание на известни мрежи. Практически не се използва.

По отношение на NETBIOS търсенето имаме следните клиенти:

- B-node – търсят само чрез broadcast;
- P-node – търсят само в именния сървър WINS;
- M-node – първо търсят с broadcast, при неуспех в WINS;
- H-node – първо търсят в WINS, при неуспех с broadcast.

Стандартният тип клиент е H-node. При него, ако името го няма в кеша, то се търси в именния сървър WINS. Процедурата е следната: първо се пуска broadcast "има ли именен сървър". Именният сървър трябва да се обади и да си каже IP адреса. H-node записва този адрес в кеша си. След това праща заявка към сървъра дали има регистрирано търсеното име. Ако го има, сървърът връща неговия адрес, ако го няма се пуска broadcast "има ли някой, който да се обади за търсеното име". Всеки компютър си пази таблица на NETBIOS имената. Ако след тези последователни допитвания името не се намери се проверява във файла LMHOSTS, ако и там го няма се счита, че името е невалидно.
NETBIOS е локална система за именуване и позволява динамично регистриране и изтриване на имена, за разлика от DNS.
Ново NETBIOS име се създава по следния начин:

1. Откриваме именния сървър.
2. Подаваме заявка към сървъра за регистриране на име. Сървърът запомня името и адреса на притежателя му.
3. По някое време притежателят може да изтрие името от сървъра.

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 наречения **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.

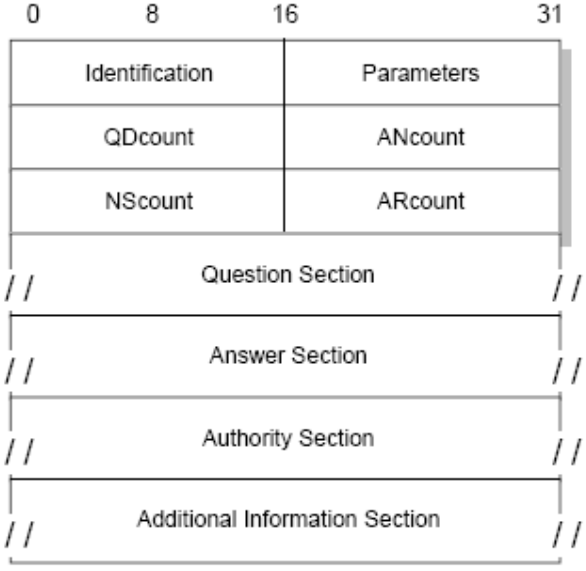
22. Процес на резолвинг на имена. DNS-конфигурационни файлове.

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.

Например, нека имаме URL-то www.fmi.uni-sofia.bg
Първо запитваме root-сървъра за адреса на сървъра, отговарящ за домейна bg. След това запитваме сървъра, отговарящ за домейна bg за адреса на сървъра, отговарящ за домейна uni-sofia. След това запитваме сървъра, отговарящ за домейна uni-sofia за адреса на сървъра, отговарящ за домейна fmi. След това запитваме сървъра, отговарящ за домейна fmi за адреса на обекта www. Именният сървър, който прави итеративна заявка знае поне адреса на root-сървъра.
След като една рекурсивна заявка е превърната в итеративна и итеративната заявка е изпълнена, полученият отговор се връща обратно по веригата на рекурсивната заявка през защитната стена (ако има такава) и се стига обратно до resolve-ра, който слага получения отговор в кеша си.
Инверсните заявки служат за обратен resolve – по IP адрес да се получи URL. В именните сървъри има специални записи, предназначени за инверсни заявки. Инверсните заявки са по-трудни от правите – докато при правата заявка, когато премине в итеративна се знае пътя в дървото, по който трябва да се спуснем, при инверсната трябва да се прави търсене по всички мрежи.

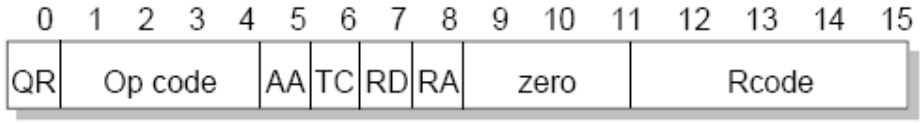
Когато агентът в клиента получи IP адреса, съответен на URL-то, той трябва да се свърже към съответния компютър. За целта се използва полето за протокол в URL, което определя номера на порта, към който да се осъществи връзката. Например, за протоколът http този порт е 80, за протокол ftp портът е 21 и т.н. На клиентската машина агентът върви на произволен порт с номер по-голям от 1024.

Форматът на DNS-съобщенията е следния:



Първите 12 байта са заглавието на DNS-съобщението.

Полето Identification е идентификатор, който се генерира от програмата, образувала запитването. Отговорът съдържа същият идентификатор както въпроса.
Полето Parameters има следния формат:



Полето QR е 0 при запитване и 1 при отговор.
Полето Op code задава типа на заявката:
тип 0 – стандартна заявка
тип 1 – инверсна заявка
тип 2 – запитване за статус на сървъра
тип 5 – съобщение за обновяване
Полето AA е 1, когато заявката е отговор и сървърът, отговорил на заявката е упълномощен за съответното име на домейн.
Полето TC е 1, когато съобщението е прекалено дълго и трябва да се ореже.
Полето RD е 1, когато се изисква заявката да е рекурсивна.
Полето RA е 1, когато сървът е готов за рекурсия.
Полето zero не се използва.
Полето Rcode съдържа код на отговора:
код 0 – в операцията няма грешка
код 1 – грешка на формата
код 2 – грешка на сървъра
код 3 – грешка в името (името не може да бъде намерено)
код 4 – неприложима заявка (сървърът не поддържа такива заявки)
код 5 – отказана заявка (по съображения за сигурност)
Полето QDCount съдържа броя на записите в частта с въпроси (Question Section).
Полето ANCount съдържа броя на ресурсните записи, върнати в частта за отговори (Answer Section).
Полето NSCount съдържа броя на записите за именни сървъри в частта за упълномощени сървъри (Authority Section).
Полето ARCount съдържа броя на записите в частта за допълнения (Additional Information Section).
Един въпрос от частта Question Section съдържа полетата:
QNAME – поле за име с променлива дължина, съдържа в началото си собствената дължина
QTYPE – поле за тип на въпроса
QCLASS – поле за клас на въпроса
Останалите видове записи в другите три секции имат аналогични полета като въпросите, но освен това:
TTL – продължителност на живота на записа
RDLENGTH – 16-битов брояч за дължина на данните
RDATA – данни, зависещи от типа на записа

В класическия DNS ресурсните файлове се формират ръчно. Конфигурирането на един DNS сървър означава правенето на тези файлове или вземането им отнякъде. В класическия DNS файловете са 4: boot, cache, forward-lookup, reverse-lookup.
Файлът boot се нарича още NAMED.boot и е характерен за системите с UNIX. При системите с WINDOWS файлът го няма, а информацията е пръсната в системните регистри. Файлът съдържа:

- директорията, където се намират останалите конфигурационни файлове;
- името на cache-файла, който съдържа DNS сървърите на домейните от първо ниво;
- името на файла, който съдържа ресурсните записи на всички домейни към сървъра;
- имената на всички домейни от второ ниво, за които е отговорен сървъра;
- адресите на вторичните сървъри, за които сървъра е главен.

Когато става дума за вторичен сървър, винаги трябва да се пази адреса на съответния главен сървър (който най-често е първичен).
Файлът cache съдържа адресите на всички сървъри, които са главни за домейните от първо ниво. Този файл ръчно се тегли от сайта на администрацията на DNS поне веднъж седмично. Това е необходимо, тъй като няма сървър за root домейна. С други думи, root домейна е логическа единица, която я няма физически. Ако физически имаше root сървър, към него щеше да има прекалено много обръщения и щеше да е труден за реализация. Затова информацията, която би трябвало да присъства в хипотетичен root сървър се записва във файл, който се разпространява по целия свят.
Файлът forward-lookup съдържа локална конфигурационна информация за домейните, за които сървърът отговаря.
Основен е SOA-записът, който определя кой е първичният сървър и как се обработват данните към него. Има следния формат:
@ IN SOA <source_host>
source_host е пълното име на компютъра, който съхранява главното копие на информацията + серия от параметри, свързани с това как се работи с този компютър.
Вторият тип записи е NS-записът, който съдържа информация кои DNS сървъри са отговорни за този домейн. Имат следния формат:
<domain> IN NS <name_server>
<domain> е име на домейн, а <name_server> е име на сървър, който поддържа домейна.
Третият тип записи е MX-записът, чрез който се указва име на хост, готов да приема електронна поща в рамките на домейн. Неговият формат е следния:
<domain> IN MX <host>
<host> е името на компютър, който може да приема електронна поща в рамките на домейна с име <domain>.

Основният тип записи са адресните записи, които съдържат съответствие между име и IP-адрес. Имат следния формат:
<hostname> IN A <IP address>
<hostname> е пълното име на хост, <IP address> е съответният IP адрес.
В DNS е възможно създаването на прякори, т.е. няколко имена да отговарят на един и същ IP адрес. Това става с помощта на CNAME-записите, които имат следния формат:
<alias> IN CNAME <hostname>
<alias> е новото име (прякорът), а <hostname> е старото име. Допуска се създаване на прякор на прякора, но не се препоръчва. Файлът reverse-lookup се използва за обратно търсене – съдържа информация на кой IP адрес кое име отговаря. Записите в този файл са наопаки, тъй като при търсене индексираме по IP адреса, а не по името.

23. Сесийно ниво в Интернет – файлов трансфер и FTP протокол.

FTP е приложен протокол над TCP/IP, който използва TCP за установяване на съединение и прехвърляне на информация. Основно този протокол се използва за прехвърляне на файлове. Състои се от протоколен интерпретатор (PI) и протокол за прехвърляне на данни (DTP). Протоколният интерпретатор съдържа потребителски интерфейс и през него се обменят команди. Между FTP сървъра и клиента се създават две отделни съединения – едно между DTP и едно между PI. FTP сървърът слуша на порт 20 за DTP и на порт 21 за PI. В клиента съединенията се осъществяват на два порта с номера по-големи от 1024. Поддържат се две паралелни сесии, които могат да се прекратяват и пак да се възстановяват, затова FTP е протокол от сесийно+представително+приложно ниво.
Първо се създава сесия между протоколните интерпретатори. В нея се обменят команди, като клиентът е активният. По принцип клиентите четат файлове от сървърите, ако сървърът позволи клиентът може да качи файл на сървъра. Когато дойде време за прехвърляне на файл се създава нова сесия за прехвърляне на данни. Смисълът на двете сесии е, че се поддържат две различни съединения, които работят паралелно. По време на прехвърлянето на данни могат успоредно да се прехвърлят команди.
Клиентът комуникира с протоколния си интерпретатор посредством потребителски интерфейс.
Командите за FTP са следните видове:

- команди за контрол на достъпа;
- команди за параметри на трансфера;
- команди за прехвърляне на файлове;
- команди за управление на директории и файлове;
- команди за помощ и състояние;
- отговори на FTP сървъра.

Чрез командите за контрол на достъпа се създава контролната сесия между протоколните интерпретатори. Те са OPEN hostname – отваряне на връзка към сървъра hostname. Името hostname се resolve-а и се получава съответния IP адрес. След това се установява TCP връзка към този IP адрес. Следва идентификация на клиента чрез команди USER username и PASS (команда за парола). По-нататък следва команда за зареждане на структура. Тя се използва когато сървърът и клиентът имат различни структури на файловата система. Клиентът казва каква файлова структура очаква и сървърът имитира тази структура, независимо от неговата физическа файлова система.
Повечето FTP сървъри поддържат user anonymous с парола mail-адреса на клиента. Те отварят почти винаги цялото си съдържание като read-only за този user. Останалите user-и може да имат по големи права за създаване или за промяна на вече създадени файлове върху сървъра.
Командите за параметри на трансфера определят клиентските портове, типа на данните които ще се прехвърлят и вида на трансфера по време на прехвърлянето. Вида на трансфера може да е поток, блок или компресиран.
Най-често използваните команди за прехвърляне на файлове са RECV и SEND. RECV има следния синтаксис: RECV [remfile] [locfile] [remfile] е име на файл от сървъра, който ще се тегли, а [locfile] е името под което този файл ще се запише в клиента. Ако няма [locfile] файлът ще се запише под същото име, както е на сървъра. Командата SEND е за обратен трансфер – прехвърляне на файл от клиента към сървъра, в случай че потребителя има права. Командите GET и PUT са аналогични на RECV и SEND.
Когато файлът се трансферира, той се разглежда като цяла съвкупност, а не като състоящ се от записи. Затова FTP сървърите се различават от файловите сървъри. При файловите сървъри файлът се обработва като структура от записи, т.е. във файла се пише и чете по записи, докато при FTP сървъра се работи с трансфер на цели файлове и той не осигурява достъп до части от файла.
Командите за управление на директории и файлове са следните:

- DELETE – изтриване на файл от сървъра, ако потребителя има права;
- CD – смяна на текущата директория на сървъра;
- MKDIR – създаване на нова директория на сървъра;
- RMDIR – изтриване на директория от сървъра;
- DIR – дава списък на съдържанието на текущата директория на сървъра;
- RENAME – преименуване на файл.

Командите за помощ и състояние са ? за показване на всички налични команди и ! за прехвърляне от една FTP сесия в друга.
На всяка команда, издадена от протоколния интерпретатор на клиента, протоколният интерпретатор на сървъра трябва да отговори с някакъв код. Кодовете са трицифрени. Първата цифра

показва общото състояние на изпълнение на командата. Може да е едно от 1, 2, 3, 4, 5.
1 означава положителен подготвителен отговор (командата е започнала да се изпълнява, но не е завършила).
2 означава положителен заключителен отговор (командата е приключила изпълнението си и можем да преминем към следваща команда).
3 означава положителен междинен отговор.
4 означава отрицателен поправителен отговор.
5 означава отрицателен постоянен отговор.
Втората цифра е допълнителна информация за състоянието на сървъра, а третата цифра допълва информацията от втората цифра.

В оригиналния си вид username-а и паролата преминават в чист вид през мрежата. Файлът също минава в явен вид. Това не е желателно – с нов стандарт се добавя authentication и security по отношение на файловия трансфер. Паролата не трябва да минава в явен вид, по време на установяването на сесия по някакъв начин трябва да се установи таен ключ за сесията за шифровано предаване на файлове, като ги получи клиентът ги разшифрова и си ги записва в локалния диск.

24. Приложно ниво в Интернет – електронна поща. Протоколи SMTP и POP3.

Електронната поща (e-mail) означава изграждането на обслужване в мрежата и протоколи, които допускат регламентираното предаване на съобщения.
Основното преимущество на електронната поща пред класическата поща е бързината на доставката. Рядко времето на доставка надвишава един час. Предимството пред телефонните разговори е, че не е нужно получателят да е наличен в момента на изпращане на пощата. Недостатъците на електронната поща са, че подателят не се известява за приемането на пощата от получателя и освен това неустановеното време за доставка – то може да варира от секунди до часове при тежко състояние на мрежата.
Когато се е изграждала електронната поща е следвана идеологията на класическата поща – писмото е едно цяло, то се опакова в плик, който се надписва с адреса на получателя и адреса на подателя. От своя страна самото писмо има заглавна служебна част и тяло, което съдържа съществената част от информацията.
Структурата на адреса е следната: name@domain.
name е името на получателя, domain е името на домейн от най-ниско ниво в DNS нотация.
Важното е, че в DNS сървърът, който поддържа имената има запис от тип MX, който съдържа адреса на mail-сървъра за този домейн. Всеки домейн от крайно ниво би трябвало да има

специален сървър за поддържане на пощенски услуги към този домейн (mail-сървър).
Пощенският сървър съдържа толкова пощенски кутии, колкото имена има дефинирани в него. Пощенската кутия е логическо понятие. Като физическа реализация в някаква файлова система се съхраняват получените и изпратените съобщения от всеки притежател на пощенска кутия.
Тъй като един домейн притежава само един пощенски сървър, в адреса не фигурира името на сървъра, а само името на домейна. IP адреса на сървъра се намира като се resolve-не адресът на домейна и от DNS сървъра за този домейн се прочита MX записа, в който е IP адреса на пощенския сървър.
Протоколът за изпращане на поща е **SMTP** (simple mail transfer protocol). Той се базира на TCP като долустоящ транспортен протокол. При използване на SMTP от клиента от порт с номер по-голям от 1024 се прави заявка за съединение към IP адреса на пощенския сървър на порт 25, т.е. порт 25 стои отворен в пощенския сървър и чака заявка за съединение. Ако сървърът е в състояние да получи заявката, той отговаря с 220, което означава готов. След това клиентът изпраща съобщение HELLO, а при успех сървърът отговаря с 250. След това от клиента се изпращат MAIL FROM (от кого е пощата), RCPT TO (кой е получателя) и накрая се прехвърля самото съобщение, след което връзката се разпада.
Описаното съединение е едно TCP/IP съединение. В неговите рамки се обменят ASCII съобщения, които са с определена структура. Крайният получател на писмото не е SMTP-сървърът – той се отделя и в него се събират изпратените писма до съответния домейн.
Сървърът трупа тези писма на диск при себе си.
Създаден е друг протокол, с който крайният получател изтегля получените писма от пощенския сървър. Това е **POP3** (post office protocol). При него сървърът слуша на порт 110. За разлика от SMTP, протоколът POP3 поддържа автентикация на клиента (username + password), т.е. притежателят на пощенската кутия е регистриран като POP3 потребител. Когато клиентът се свърже към POP3 сървъра, той първо се идентифицира, след което може да извърши други команди за прочитане на получените от него mail-ове.
В клиента SMTP и POP3 агентите се вграждат в една програма – най-широко разпространена е Outlook Express на Microsoft, за UNIX са много.
Писмото се състои от плик, заглавна част и тяло. Всички те са в стандартен ASCII формат (7-битов).
Форматът на плика е прост – той съдържа адресите на подателя и на получателя.
Заглавието съдържа следните полета:

- поле TO – e-mail на получателя;
- поле CC – e-mail на втори получател;
- поле BCC – e-mail на трети получател;

- поле FROM – от кого е писмото;
- поле SENDER – кой изпраща писмото;
- поле RECEIVED – попълва ст от всеки transparent agent, през който е минало писмото;
- поле RETURN PATH – пътят, по който е минало писмото (може да се използва за обратен достъп до подателя).

В по-новия стандарт са включени още следните полета:

- поле DATE – дата на изпращане на писмото;
- поле REPLYTO – e-mail, към който да се изпрати отговора;
- поле KEYWORDS – ключови думи в писмото;
- поле SUBJECT – задължително поле за резюме на писмото.

Първоначално и тялото също трябва да е в ASCII7 формат, и сега също е така. На теория се допускат и други форми за тялото. ASCII7 означава писмото да е написано на латиница. В клиентите, които създават писмото има текстов редактор, чрез който се написва писмото.

За да могат да се предават не само ASCII7 текстове се създава едно разширение **MIME** (multipurpose internet mail extension). Чрез MIME се допуска тялото на писмото да не е ASCII7 текст като се използват различни техники на кодиране. Кодировката се формира при създаване на тялото на писмото, получателят после го връща в оригиналния вид. Кодировката означава специален двоичен код, който трябва да се преработи в ASCII7. Прехвърляните данни може да са само в ASCII7 формат. Допускат се няколко начина на кодиране.

Ако преобладаващата част от тялото на писмото са стандартни ASCII7 символи, но понякога се появяват символи с код по-голям от 127, тогава те се кодират с последователността ==XX, където XX е шестнайсетичния код на символа. Това е в случай, че тези символи се срещат рядко, тъй като един такъв символ се преработва с цели четири и текстът би нараснал твърде много.

Двоични файлове се кодират с така наречената кодировка base64. Тялото на писмото се разбива на групи по 24 бита (3 байта). Групата от 24 бита се разделя на 4 части по 6 бита. Всяка част от 6 бита се кодира с ASCII7 код (кодът е от 0 до 63). На 0 отговаря A, на 1 – B, ..., на 25 – Z, на 26 – a, на 27 – b, ..., на 51 – z, на 52 – 0, на 53 – 1, ..., на 61 – 9, на 62 - +, на 63 - /. Непълните до 24 бита групи се попълват до края със знак =. Когато групата е само 8 бита, тя завършва с ==, а когато е 16 бита завършва с =.

Това е начинът да се прекара двоичен файл през системата на електронната поща, работеща в ASCII7. В крайна сметка всеки три байта се кодират с четири символа, което е най-икономичният начин.

Електронната поща е много разпространена в Internet. Не е трудно да се направи поща във всяка мрежа. Основен недостатък е, че цялото писмо трябва да е в ASCII7 формат, което налага въвеждането на MIME.

25. WEB-технологии в Интернет. Хипертекст и HTTP.

Най-мощното приложение на Internet е **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-а и той я изобразява. Една страница се прехвърля в рамките на една HTTP-сесия. Ако човек кликне върху линк на изтеглената страница, browser-ът установява нова сесия, подава се новото URL, изтегля се страницата от сървъра и отнове се изобразява от browser-а.
Протоколът HTTP се базира на TCP. HTTP клиентът отваря сесия на произволен порт с номер по-голям от 1024. HTTP сървърът слуша за заявки на порт 80.
При HTTP протокола имаме подготвителна фаза – прави се заявка за HTTP сесия към сървъра, след това се прави HELLO към сървъра, потвърждава се от сървъра и се изпращат методи на HTTP. Методите са GET, HEAD, POST, PUT, TRACE, CONNECT. Основният метод е GET. Като аргумент му се подава 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.

26. Секретност и автентичност в мрежите.

В началото е имало изцяло частни, затворени мрежи. Те, обаче, са скъпи и неудобни, затова се преминава към публични мрежи. При тях съобщенията не се знае от къде минават, как минават и така въпросът за защита на информацията става жизненоважен. За да се защити информацията, най-сигурно е тя да се криптира (шифрира), да се предаде криптирана и от другата страна да се декриптира. Схемата е следната: подава се натурален текст P, той се криптира до $C = E_K (P)$ – шифриран текст, предава се C, при получаване следва декриптиране $D_K (C) = P$ до оригиналния натурален текст. E и D са съответно криптираща и декриптираща функция, параметризирани с ключ K. Шифрираният текст C може да се атакува пасивно, при което атакуващият взима текста C, но обикновено не може да го декриптира, тъй като не притежава декриптираща функция или ключа. При активно атакуване, атакуващият взима C и го заменя с друг текст или без да го променя го праща отново след време.
Криптологията е сборно название за криптографията, което е измисляне на шифриращи и дешифриращи функции и криптоанализа, което е разбиване на шифрирани текстове. Задачата на криптографията е да намери такива E и D, че $D_K (E_K (P)) = P$ за всеки натурален текст P.
В началото стремежът е бил както функциите, така и ключът да се пазят в тайна. Днес общият принцип е, че шифриращите и дешифриращите функции са публични, а само ключът се пази в тайна.
От гледна точка на криптоанализа имаме три вида задачи:

- притежаваме само шифриран текст. Разбиването означава да получам оригиналния натурален текст, заедно с ключа;
- притежаваме натурален текст P и съответният му шифриран текст $E_K (P)$. Разбиването означава да получим ключа.
- можем да изберем произволен натурален текст P и да го шифрираме до $E_K (P)$. Разбиването отново означава да получим ключа.

Ще разгледаме някои класически принципи на криптографията. Първият метод за криптиране е чрез така наречените **субституционни шифри**. Най-старият такъв е шифърът на Цезар при който всяка буква се измества с три напред, т.е. на A се съпоставя D, на B се съпоставя E, ..., на Z се съпоставя C. Просто разширение е изместването да става с k позиции напред, а не точно с 3, където k е ключ. Обобщението на този метод е моноалфабетичен шифър, при който на всяка буква съответства друга буква, като на различни букви съответстват различни (на азбуката се съпоставя нейна пермутация). Механичното разбиване на този шифър практически не е възможно чрез изброяване на всичките 26! възможности. Шифърът, обаче, се разбива лесно, ако се използва нещо допълнително. Например, ако натуралният текст е на английски, то се използва честотният анализ на английския език – честотата на появата на букви в произволен текст, а също и на сричките по две и три букви.
Вторият метод за криптиране е чрез така наречените **транспозиционни шифри**. При тях се взима ключова дума в която няма повторение на буквите, например MEGABUCK. След това буквите на думата се номерират по старшинство на азбучния ред. Натуралният текст се изписва по редове под ключовата дума, при това матрицата се допълва за да станат цяло число редове. Шифрираният текст се взима по колони, в реда на определената номерация.

M	E	G	A	B	U	C	K
7	4	5	1	2	8	3	6
p	l	e	a	s	e	t	r
a	n	s	f	e	r	o	n
e	m	i	l	l	i	o	n
d	o	l	l	a	r	s	t
o	m	y	s	w	i	s	s
b	a	n	k	a	c	c	o
u	n	t	s	i	x	t	w
o	t	w	o	a	b	c	d

Plaintext

pleasetransferonemilliondollarsto
myswissbankaccountsixtwotwo

Ciphertext

AFLLSKSOSELAWAIATOOSSCTCLNMOMANT
ESILYNTWRNNTSOWDPAEDOBUOERIRICXB

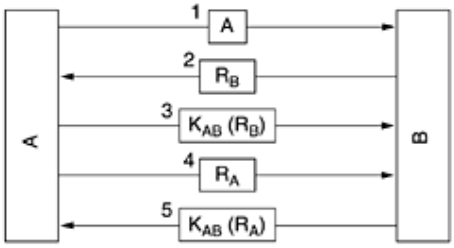
В случая шифрираният текст представлява пермутация на оригиналния текст. Оказва се, че този подход подлежи на сравнително бърза дешифровка.
Един неразбиваем алгоритъм получаваме по следния начин: натуралният текст P се преобразува в битова поредица в съответствие с ASCII кодовете на символите. Криптирането се извършва като получената поредица се сумира по модул 2 със специален битов ключ, който има същата дължина като P.
Оказва се, че за надеждно шифриране трябва да се добави допълнителна информация във шифрирания текст. Също е важно, системата за обмен на съобщения да се обезопаси срещу това да могат да се подхвърлят стари съобщения.
Модерната криптография се състои от комбинация между субституционни и транспозиционни шифри, функциите са известни, надеждността зависи от дължината на ключа.

IBM разработва алгоритмите DES и 3DES при които натуралният текст се разбива на групи и към всяка група се прилагат няколко субституционни и транспозиционни шифри. По-късно е разбработен алгоритъмът IDEA, който по-трудно се разбива от DES и 3DES алгоритмите.
Изброените алгоритми са алгоритми със симетричен ключ – ключът е един и същ при криптиране и декриптиране. Основен проблем при тях е секретността на ключа – той не трябва да преминава в нешифриран вид през мрежата.
През 1976 година Diffie и Hellman създават нов вид криптосистема. При нея има два различни ключа K1 и K2 при криптиране и декриптиране. Тя трябвало да отговаря на следните три условия:

1. D (E (P)) = P за всеки натурален текст P;
2. D не може да се намери от E (K2 от K1);
3. E не може да се разбие чрез избран натурален текст.

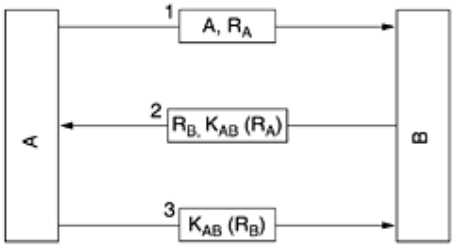
Ключът K1 за криптиране се прави публичен, а ключът K2 за декриптиране е частен и се пази в тайна.
Комуникацията между A и B се извършва по следния начин: A иска да прати съобщението P на B. За целта се сдобива с публичния ключ E_B на B и му праща E_B (P). Всички знаят публичният ключ на B, но по условие 3. не могат да разбият E_B (P). Когато B получи съобщението, той го декриптира със своя частен ключ D_B до D_B (E_B (P)) = P. Аналогично, за пращане на съобщения към A, B използва публичния ключ E_A на A, който ги декриптира със своя частен ключ D_A.
Един алгоритъм, който отговаря на посочените изисквания е RSA. Практически той е най-използвания алгоритъм.

Проблемът за автентикация се състои в следното – когато A и B си говорят, първо A трябва да удостовери себе си пред B и обратно. Първият начин за автентикация е A и B предварително да обменят общ секретен ключ K_{AB}. След това автентикацията се извършва по следната схема:

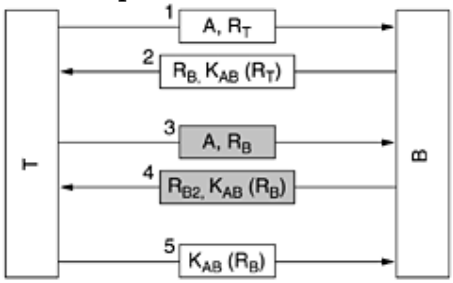


Чрез съобщение 1 A казва на B, че е той. B не е сигурен, че това е съобщение от A, затова генерира случайно число R_B и го праща на A в съобщение 2. A кодира полученото случайно число с общия секретен ключ K_{AB} и го праща на B в съобщение 3. B декодира полученото и по този начин се уверява, че наистина комуникира с A, но в този момент A не е сигурен, че комуникира с B, затова генерира слуйчайно число R_A и го праща на B в съобщение 4. B декодира полученото случайно число с

общия секретен ключ $K_{АБ}$ и праща полученото на А в съобщение 5, след което А вече е сигурен, че говори с В.
Възможна е следната подобрена версия на този алгоритъм:

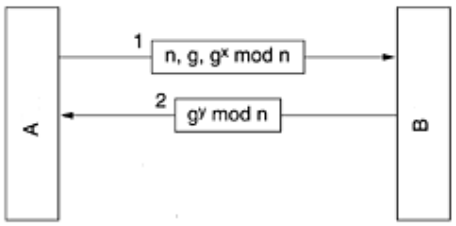


При нея се изпращат само три съобщения, но тя е податлива на така наречената **reflection атака** от Т:



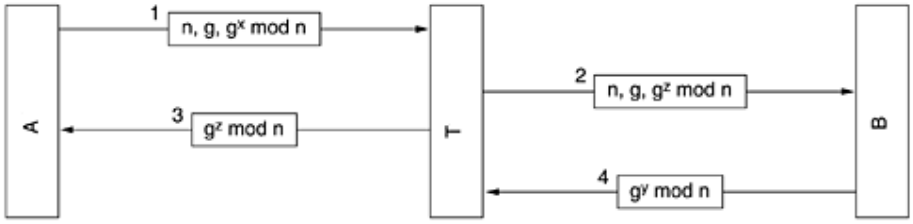
В съобщение 1, Т се преструва на А и изпраща случайно число R_T. В отговаря със своето случайно число R_B и с кодираното с общия секретен ключ K_{AB} случайно число R_T в съобщение 2. В този момент Т трябва да върне K_{AB} (R_B), но той не знае K_{AB} и затова отваря нова сесия към В като отново се преструва за А, но подава случайното число R_B в съобщение 3. В отговаря с ново случайно число R_{B2} и с кодираното с общия секретен ключ K_{AB} случайно число R_B в съобщение 4. Т вече знае K_{AB} (R_B) и го подава в съобщение 5 в първата сесия и след това затваря втората сесия. Оттук нататък Т може да праща съобщения на В и В ще го мисли за А.

Вторият начин за автентикация е без предварително А и В да имат общ секретен ключ. Те установяват общ секретен ключ по следния алгоритъм: А и В се договарят за две големи числа n и g, такива че n и (n-1)/2 са прости и g отговаря на определени условия. Тези числа са публични, т.е. не се пазят в тайна. След това А избира едно голямо число x и го пази в тайна, както и В избира едно голямо число y и го пази в тайна. Схемата на изпращане е следната:



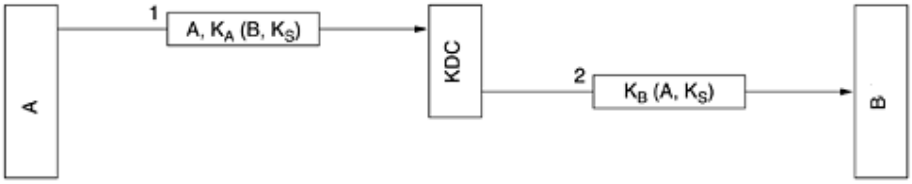
Първото съобщение, което А изпраща на В е (n, g, g^x mod n). В отговаря на А със съобщението g^y mod n. След това А изчислява

$(g^y \bmod n)^x \bmod n = g^{xy} \bmod n$ и B изчислява
$(g^x \bmod n)^y \bmod n = g^{xy} \bmod n$. По този начин A и B имат общ секретен ключ $g^{xy} \bmod n$.
Този алгоритъм има проблем, тъй като е податлив на **бригадна атака**:



Тук освен, че A и B избират по едно случайно число x и y съответно, T избира свое случайно число z. A изпраща първото съобщение (n, g, $g^x \bmod n$), предназначено за B, но T го улавя и изпраща на B (n, g, $g^z \bmod n$). T също така изпраща съобщението $g^z \bmod n$ до A. След това B изпраща към A съобщението $g^y \bmod n$, но T го улавя и го запазва за себе си. По този начин A изчислява секретен ключ $g^{xz} \bmod n$, както и T изчислява същия ключ за съобщения към A и B изчислява $g^{yz} \bmod n$, както и T изчислява същия ключ за съобщения към B. Така всяко съобщение, което A предава се улавя от T, евентуално се модифицира и евентуално се изпраща на B, аналогично за посоката от B към A. По този начин T вижда всичко и може да го променя по свое собствено желание, докато A и B мислят, че комуникират в сигурен канал.

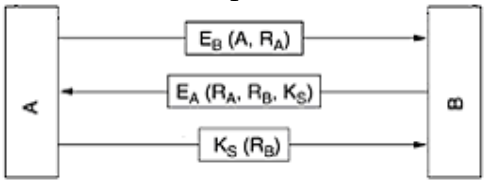
Третият начин за автентикация е чрез център за разпределение на ключове (KDC). При този алгоритъм, всеки потребител има един общ секретен ключ, който се споделя с KDC. Схемата е следната:



A избира ключ за сесия K_S и праща съобщение на KDC, че иска да говори с B, използвайки ключа K_S. Това съобщение се криптира със секретния ключ, който A споделя с KDC – K_A. KDC получава съобщението, декриптира го и разбира, че A иска да говори с B чрез K_S. След това KDC конструира съобщение, съдържащо идентификацията на A и избрания от A ключ за сесията K_S и изпраща това съобщение на B. KDC криптира това съобщение със секретния ключ, който споделя с B – K_B. Когато B получи съобщението, той го декриптира и научава, че A иска да говори с него чрез ключа K_S. Естествено, B е сигурен, че съобщението е от A, тъй като се доверява на KDC.
Недостатъкът на тази схема е, че тя се разбива с **атаката на повторението** (повторение на стари съобщения). T може да прихване съобщение 2 и впоследствие да го изпрати към B.

Един начин за избягване на атаката на повторението е да се въведе timestamp за всяко съобщение. Ако пристигне съобщение с изтекъл timestamp, то се отхвърля. Другият начин е да се използват специални съобщения nonce – A и B се уговарят за списък от nonce за еднократна употреба, всеки употребен nonce се маха и след това се ползва следващия неупотребен nonce.

Четвъртият начин за автентикация е чрез асиметрични ключове. Всеки потребител има публичен ключ, чрез който се изпращат съобщения към него и частен ключ, чрез който той декриптира съобщенията предназначени за него. Схемата е следната:



A научава публичния ключ E_B на B и изпраща съобщение към B, съдържащо идентификацията на A и едно случайно число R_A.
A кодира съобщението с E_B. След като получи съобщението B не знае дали то е от A, но той научава публичният ключ на A и изпраща на A съобщение, съдържащо R_A, случайно число R_B и предложен ключ за сесията K_S. B криптира съобщението с E_A. Когато A получи съобщението, той го декриптира със своя частен ключ и открива в него R_A – тук вече той е сигурен, че полученото съобщение е от B. След това A изпраща на B неговото случайно число R_B, кодирано с предложения ключ за сесия K_S. Когато това съобщение пристигне при B, той знае, че е изпратено от A, тъй като е криптирано с генерирания от B ключ K_S.
Този алгоритъм стои в основата на протокола SSL.
Недостатъкът е, че отново може да се направи бригадна атака. Всичко зависи от това как A и B научават публичните си ключове. Ако A прати публичния си ключ на B, T може да го прихване и да прати на B своя публичен ключ, твърдейки че това е публичният ключ на A. Поради тази причина публичните ключове трябва да се разменят по друг канал, не през мрежата. Друг начин е да има доверен сървър на публичните ключове – при него всеки знае публичният ключ на сървъра, като го е получил по вторичен канал, а не през мрежата.

Последният начин за автентикация е чрез техниката на цифровия подпис. Тя се базира на технологията на кодирането. Замисълът е, че към съобщението се добавя нещо допълнително, което удостоверява кой е подателят. Освен това изпращачът по-късно не може да каже, че не е изпратил съобщението или че е изпратил съобщение с друг текст, както и получателят не може да промени полученото съобщение. Използва се RSA алгоритъма, който освен обичайното свойство D (E (P)) = P за всеки натурален текст P има и свойството E (D (P)) = P за всеки натурален текст P.

А изпраща на В следното съобщение: E_B (D_A (P)). Когато В получи съобщението, той прилага частния си ключ D_B и получава D_B (E_B (D_A (P))) = D_A (P). След това В прилага публичния ключ на А и получава E_A (D_A (P)) = P. В запазва както P, така и D_A (P). D_A (P) служи като подпис, тъй като то може да е съставено единствено от А.

Недостатъкът на този алгоритъм е раздуването на съобщението – ако електронният подпис се приложи върху цялото съобщение, то ще стане много дълго. Затова се използва хеш-функция MD, която "смачква" съобщението в битов низ с фиксирана дължина. MD трябва да има следните свойства:

1. Като е дадено P лесно се изчислява MD (P);
2. Като е дадено MD (P) е невъзможно да получим P;
3. Не съществуват $P \neq P'$, такива че MD (P) = MD (P′).

Схемата на изращане е следната:

Когато В получи съобщението от А, той изчислява MD (P) първо от P, а после като приложи публичния ключ на А към D_A (M (P)) и сравнява получените резултати – те трябва да съвпаднат. По този начин се удостоверява, че съобщението е изпратено от А и от никой друг и че текстът на съобщението е автентичен.

При тази техника на цифров подпис е важно публичните ключове да са сертифицирани, затова те се издават от доверени институции, които подписват публичните ключове със своя частен ключ.

Край

07.12.2006