

## **5. Канално ниво.**

**Кадри, предаване, грешки,  
номерация, прозорци**

# Основни функции

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 информационния обмен.

Данните за каналното ниво представляват последователност от **кадри** (frame).

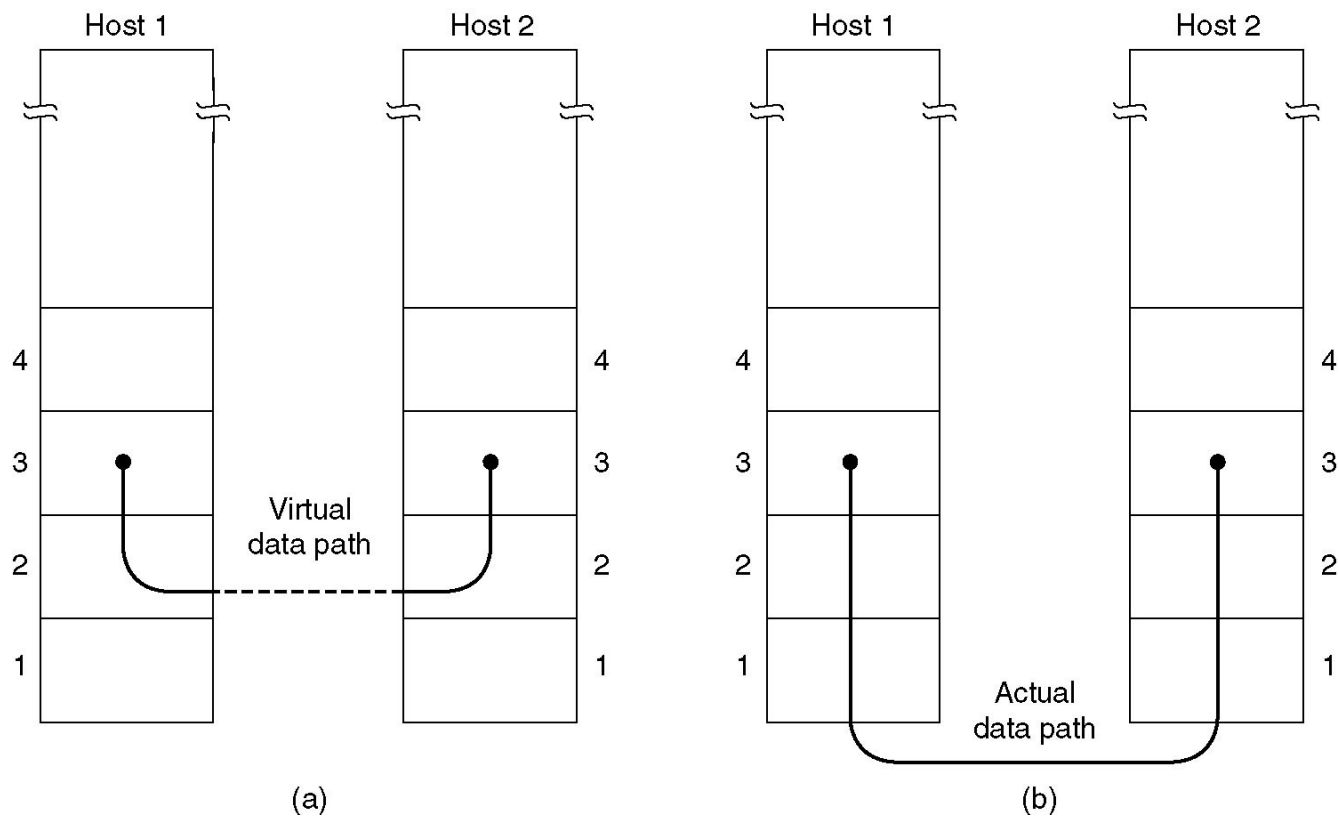
Каналите са три вида - **симплексни, полудуплексни и дуплексни**.

**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.

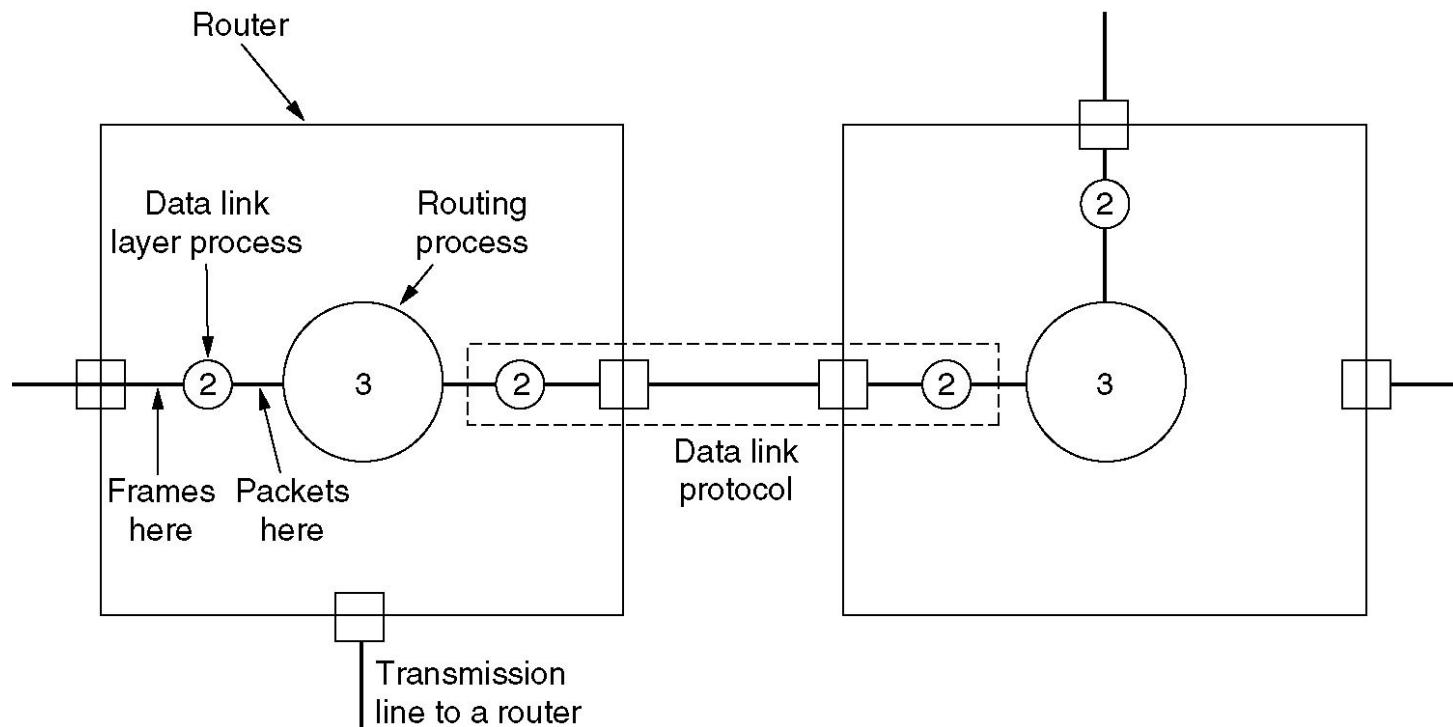
# Основни функции



(a) Логическа комуникация.

(b) Действителна комуникация.

# Основни функции



Ролята на каналния слой между две съседни машини - възли.

# Основни услуги

Най-общата услуга - **прехвърляне на данни (надеждно или best effort, но изчистено от грешки)** 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 (**Unacknowledged connectionless service**),
- потвърдено неустановено обслужване (**Acknowledged connectionless service**) и
- потвърдено установено обслужване (**Acknowledged connection-oriented service**).

# Основни услуги

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.

# Основни услуги

Такова обслужване се реализира в повечето LAN.

То също се използва когато навременното получаване на кадрите е по-важно от тяхната достоверност видео, глас в реално време.

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.

# Основни услуги

Потвърждаването на получената информация е функция на транспортното ниво, но там то се отнася до последователности от сегменти.

**Потвърждаването на каналното ниво има смисъл при ненадеждна комуникационна среда**, каквато е **безжичната**, тъй като повторно ще се предават само непотвърдените кадри.



# Основни услуги. Connection Oriented.

Потвърденото и установено обслужване има три фази.

Първата фаза се установява връзка и се заделят необходимите ресурси (локални буфери, броячи и т.н.).

Втората фаза се изпращат кадрите.

Третата фаза се освобождават ангажираните ресурси.

Гарантира се не само успешното предаване на кадъра, но и последователността в която се предават кадрите.

# Управление на потока (Flow Control)

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 цели масиви от данни, обхващащи последователност от кадри.

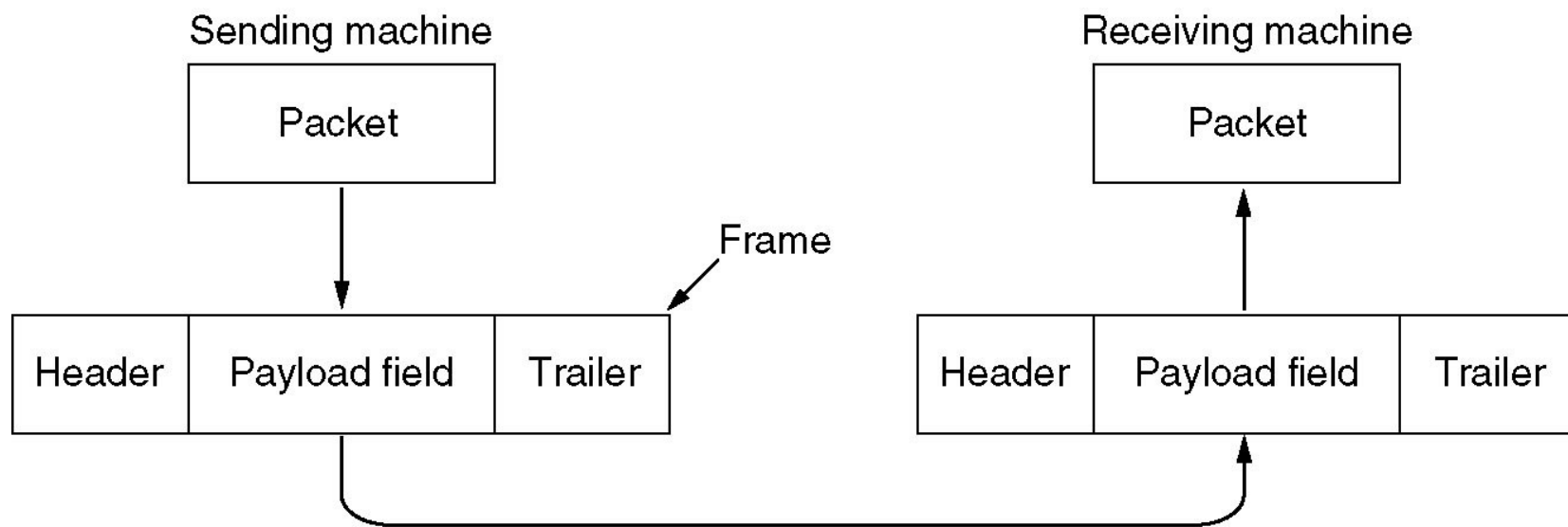
# Формиране на кадри

Каналното ниво взима пакетите, които му се подават от мрежовото ниво и ги опакова в кадри.

Всеки кадър се състои от заглавна част (**header**), поле за данни (**data** или **payload**), което съдържа мрежовия пакет 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.

# Формиране на кадри



Прехвърляне на данни между мрежовите нива на източник и получател (две съседни машини - възли). Пакети и кадри.

# Формиране на кадри

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 при 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.

# Формиране на кадри

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.

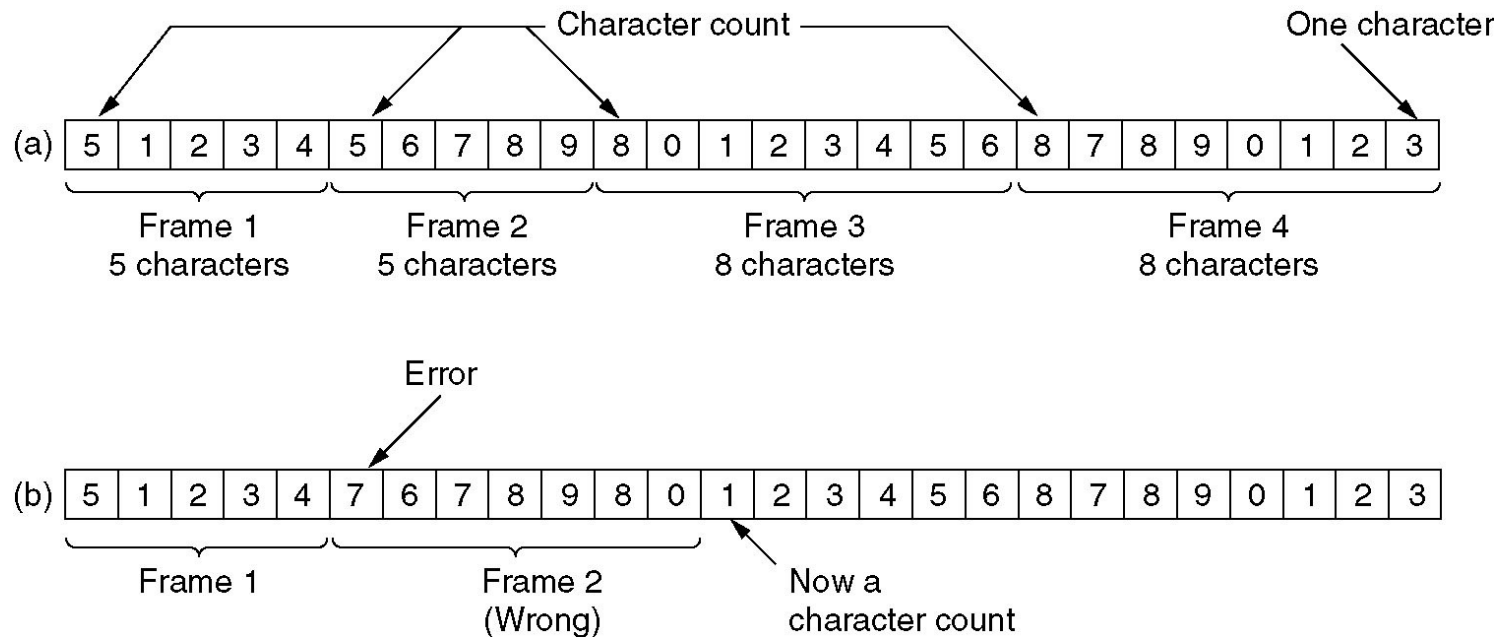
Понастоящем основно се използват **три метода**.

При първия метод е **броене на отделните символи**. В заглавието на кадъра се указва броя на символите в целия кадър.

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. Повторно предаване не помага. Затова **не се използва**.

# Броене на символи



Поток от четири кадъра: 5, 5, 8, 8 символа. (a) Без грешки. (b) Грешка: число 5 във втори става 7. Губи се синхронизация.

# Формиране на кадри

**Втори 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 служебните символи да се срещат като битови последователности в оригиналните данни. За решение на този проблем се въвежда друг служебен символ **ESC (escape)**, който се вмъква преди всяко срещане на служебен символ (STX, ETX, ESC) в данните. Например, ако потокът, предаван от мрежовия слой на източника е **A STX ESC B**, той ще се **преобразува** в **A ESC STX ESC ESC B**.



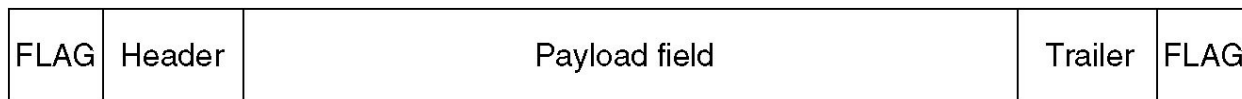
# Формиране на кадри

Каналното ниво на получателя ще премахне символите ESC (като при два последователни ESC, единият се запазва), преди да предаде данните на мрежовото ниво на получателя.

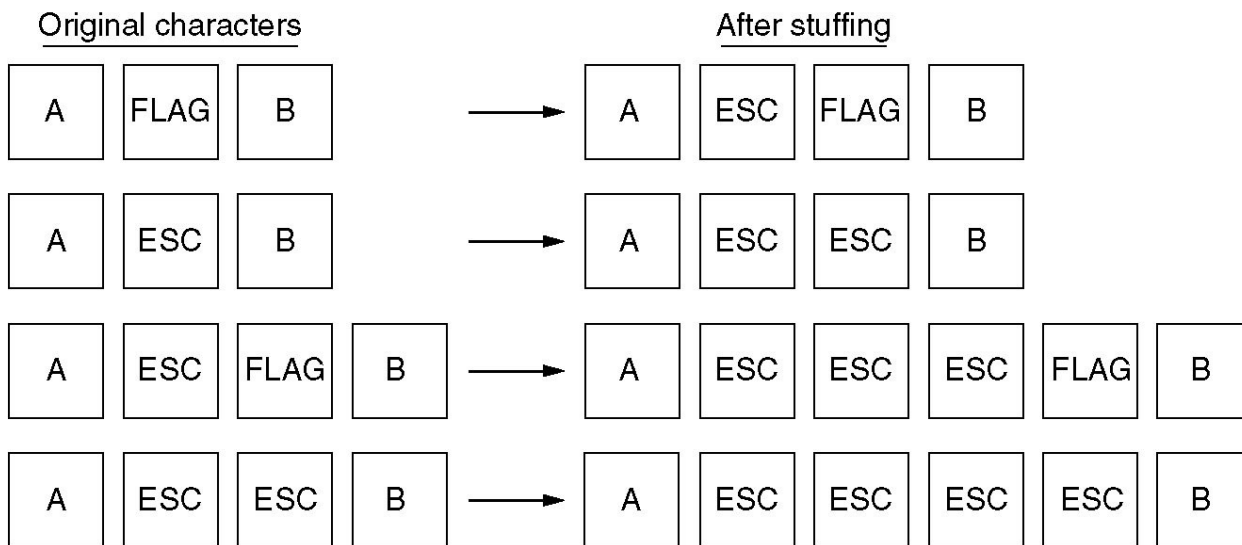
При по-новите протоколи се използва един и същ символ за маркиране на началото и края на кадъра - **флаг**.

Недостатъкът на този метод е, че той се обвързва с 8-битови символи, кодирани в ASCII.

# Вмъкване на символи



(a)



(b)

(a) кадър, ограничен от флагови байтове.

(b) Последователност от байтове преди и след вмъкване.

# Формиране на кадри

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 **0111110**, 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)**.


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.

# Флагов байт

(a) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0



Stuffed bits

The diagram shows three arrows pointing from the text 'Stuffed bits' to the 10th, 14th, and 19th bits of the sequence in (b). These bits are 0, 0, and 0 respectively, which are the stuffed bits.

(c) 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

Bit stuffing

(a) Оригинални данни

(b) Как данните изглеждат по линията

(c) Данните, съхранени в паметта на приемника след destuffing

# Процедури за надеждна работа на канала

Функциите на каналния слой се реализират в **адаптер, предимно хардуерно**: специализирани интегрални схеми за управление (**ASIC**) и програмен код, “прогорен” в EEPROM или записан във Flash памет (firmware).

В адаптера е реализиран **буфер**, в който се записват кадрите, докато изчакват да бъдат предадени нататък.

Кадърът преседява в буфера, докато не се увери, че отсрещната страна го е получила.

Да приемем, че **източник A** изпраща кадър към **B**, но той изобщо не стига до там. Пет причини за това:

- 1) Адаптер *A* дефектен, не излъчва правилен сигнал;
- 2) “Счупен” канал – жица, роми и т.н.;
- 3) *B* не съществува;
- 4) *B* няма свободен буфер;
- 5) Кадърът постъпва в буфера на *B*.

# Процедури за надеждна работа на канала

**A** може да получи отговор единствено при 5). При изпращане на кадъра **A** включва брояч на време - таймер. Чака отговор до определено време – **timeout**.

**timeout** 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.

Възможно е **A** да изпрати кадър към **B**, този кадър да се получи в **B**, но потвърждението да се изгуби.

# Процедури за надеждна работа на канала

При всички положения ***A*** изпраща наново кадъра, ***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.

# Откриване на грешки в кадрите

За целта е измислен **CRC** (*cyclic redundancy check* — *проверка на цикличния остатък*)

Алгоритъм за проверка за грешки при предаване и съхранение на данни чрез използване на **контролна сума** (*контролно число, CRC сума*).

Устройството-източник изчислява CRC-сумата на данните, които следва да бъдат проверявани и я изпраща или записва със самите данни.

Устройството-получател извършва същото изчисление след прочитане на данните и контролната сума, и установява тяхната автентичност чрез сравнение на записаната CRC сума и новоизчислената CRC сума.



# Видове CRC-та

**CRC-16-CCITT** =  $x^{16} + x^{12} + x^5 + 1$  (Bluetooth, XMODEM, HDLC, PPP)

**CRC-32-IEEE 802.3** =  $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

**CRC-64-ISO** =  $x^{64} + x^4 + x^3 + x + 1$  (HDLC — ISO 3309)

Тези кодове са разработени отдавна, оптимизирани и вкарани в хардуерни схеми.

CRC се изчислява в движение и се слага в края на кадъра като **FCS** (Frame Control Sum)

# Пример на CRC изчисление

Подател и получател се уговарят за генератор на контролната сума, **полином  $G(x)$** . Най-старшия и най-младшия бит трябва да са 1.

За да изчислим контролната сума на **кадър с  $t$  бита**, полинома  **$M(x)$** , кадърът трябва да е по-дълъг от полинома на генератора.

**Идеята е** към края на кадъра се да се прибави контролна сума, така че полиномът, който представя **“checksummed”** кадъра, да е делим на  $G(x)$ .

Когато получателят приеме **“checksummed”** кадъра, той се опитва да го раздели на  $G(x)$ . **Ако се получи остатък, значи има грешка.**

# Пример на CRC изчисление

Алгоритъмът за изчисляване на контролната сума е следния:

1. Нека  $r$  е степента на  $G(x)$ . Прикрепяме  $r$  нулеви бита към “младшия” край на кадъра. Така той вече съдържа  $m + r$

Бита и съответства на полинома  $x^r M(x)$ .

2. Делим низа от битовете, съответстващ на  $G(x)$ , на битовия низ, съответстващ на  $x^r M(x)$  с помощта на “сума по модул 2” (modulo 2) делене.

3. Изваждаме **остатъка** (който винаги е  $\leq r$  бита) от битовия низ, съответстващ на  $x^r M(x)$  с помощта на **modulo 2 изваждане**. Резултатът е **checksummed frame**, който ще се предаде, т.е полинома  $T(x)$ .

В следващия пример имаме кадър 1101011011 и генератор:

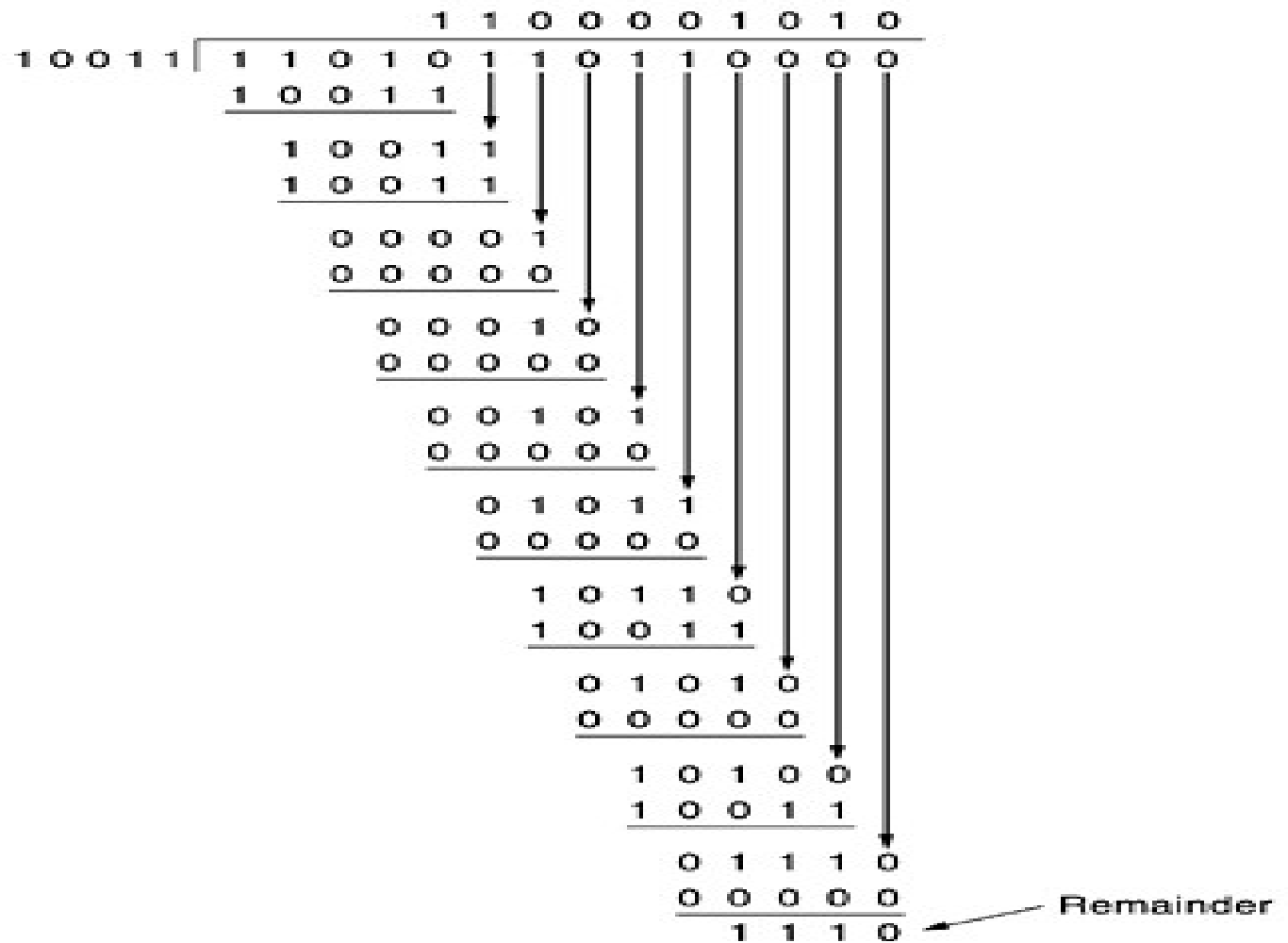
$$G(x) = x^4 + x + 1$$

# Пример на CRC изчисление

Frame : 1 1 0 1 0 1 1 0 1 1

Generator: 1 0 0 1 1

Message after 4 zero bits are appended: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



# Протоколи с прозорци (Sliding Window Protocols)

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 кадър, 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 вмести точно в  $n$  бита.

Протоколът „**спри и чакай**“ (stop-and-wait sliding window protocol) има  $n = 1$ , ограничавайки номера на кадъра до 0 и 1.

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.

# Sliding Window Protocols

Предавател и приемник:

- Последователен номер  $n_t$  и  $n_r$ ; ( $n_t$  следващият пакет за предаване.  $n_r$  предстоящият за приемане пакет);
- Размер на прозорец:  $w_t$  и  $w_r$  ( $> 0$ ).

Приемникът може да следи и още неполучения най-голям номер  $n_s$ :

$= n_r$ , ако  $w_r = 1$

$> n_r$ , ако  $w_r > 1$ .

(Всички пакети преди  $n_r$  са получени, нито един след  $n_s$  не е получен. По средата, между  $n_r$  и  $n_s$ , са получени някои пакети.

# Sliding Window Protocols

Когато приемник получи пакет, обновява променливите си и предава потвърждение (**acknowledgment**) с новото  $\mathbf{n}_r$ .

Предавателят следи най-големия номер на потвърден кадър  $\mathbf{n}_a$ .

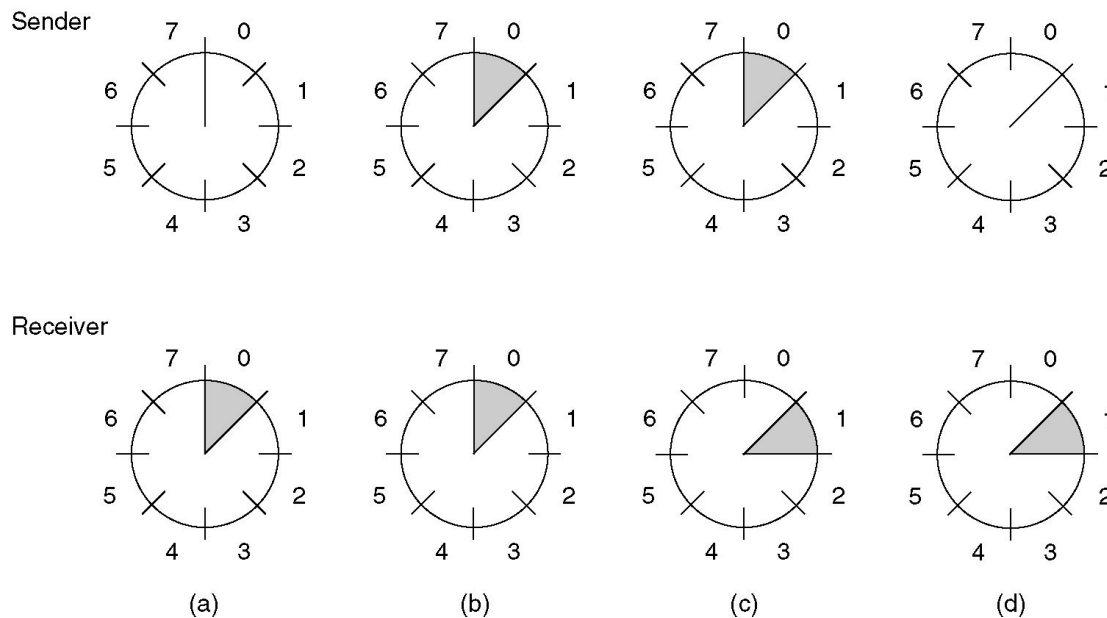
Всички кадри с пореден номер  $< \mathbf{n}_a$  са получени.

Не е сигурен за  $\mathbf{n}_a \leq \mathbf{n}_r \leq \mathbf{n}_s$ .

Последователните номера трябва да следват правилото:

$$\mathbf{n}_a \leq \mathbf{n}_r \leq \mathbf{n}_s \leq \mathbf{n}_t \leq \mathbf{n}_a + \mathbf{w}_t$$

# Sliding Window Protocols



Прозорец с размер 1 и с 3-битов пореден номер.

(a) Отначало.

(b) След изпращане на първи кадър.

(c) След получаване на първи кадър.

(d) След получаване на потвърждение.



# Sliding Window Protocols. go back n

При наличие на сгрешен или изгубен кадър, предавателят ще продължи да предава кадри, преди да разбере че има проблем.

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.

$$w_t > 1, \text{ но } w_r = 1.$$

Приемникът приема единствено и само следващия в последователността.

При загуба на кадър при предаване, **следващите** се изхвърлят (**Discard**), докато се предаде повторно липсващия. Минималната загуба е един round trip time ( $1 * RTT$ ).

# Selective Repeat

Необходим е по-мощен приемник, който да приема кадри с последователни номера  $> n_r$  (текущо) и да ги съхранява, докато се попълни празнината от загубените.

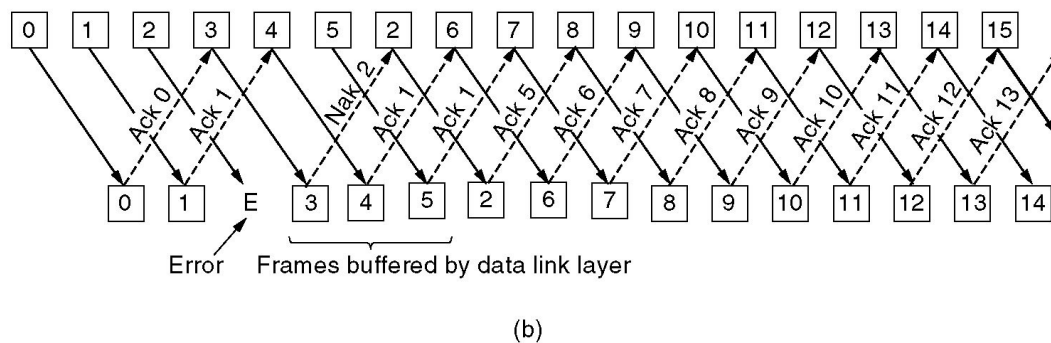
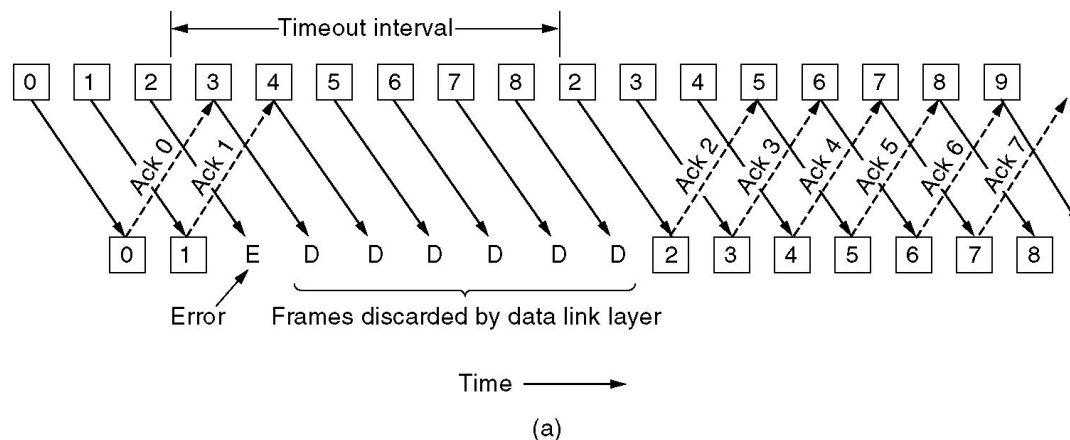
Предимство е, че не е необходимо да се изхвърлят следващите правилно приети кадри за  $1 * RTT$ , преди предавателят да е разбрал, че е необходимо повторно предаване.

Предпочитан за линии с ниска надеждност.

Прозорец  $w_r$  трябва да е по-голям от броя на загубените кадри в поносими граници. Обикновено:

$$w_r = 2$$

# Go Back N & Selective Repeat



Ефект от грешката, когато

(a) Прозорецът на приемника е с размер 1 (**go back n**).

(b) Прозорецът на приемника е голям (**sel. repeat**).

## 6. Управление на канала

HDLC – High-Level Data Link Control

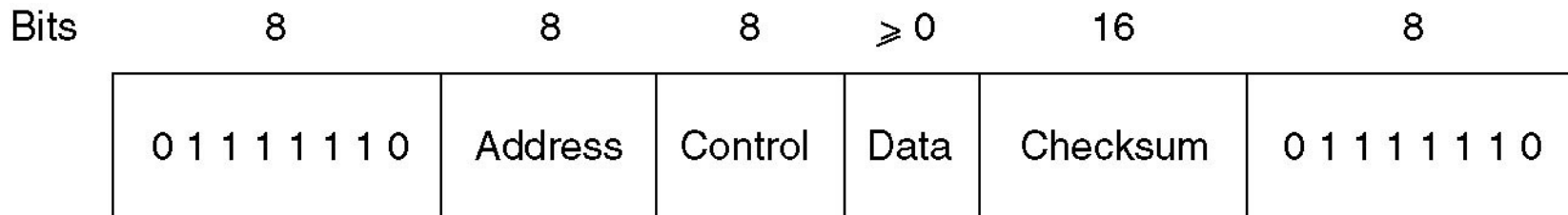
# High-Level Data Link Control

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 протокола **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:



# HDLC

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.

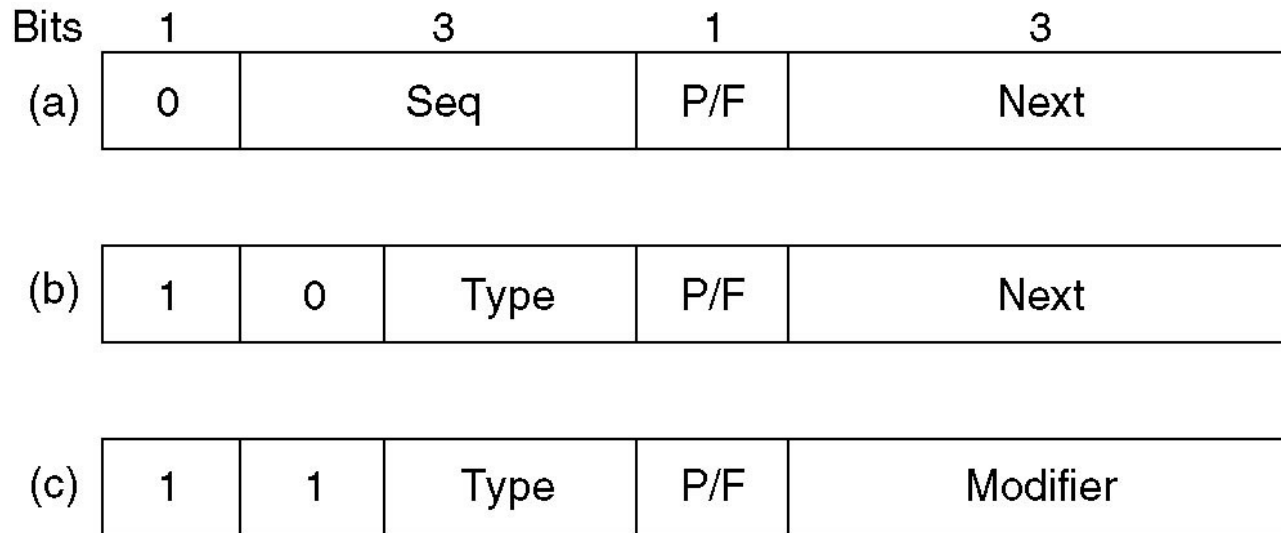
Полето **Control** се използва за номериране на кадрите, за потвърждения и за други цели.

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**.

# High-Level Data Link Control



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

Полето **Control** в:

- (a) information frame.
- (b) supervisory frame.
- (c) unnumbered frame.

# HDLC - Полето *Control* за information-кадрите

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.



# HDLC - Полето *Control* за supervisory-кадрите

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).

# HDLC - Полето *Control* за unnumbered -кадрите

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

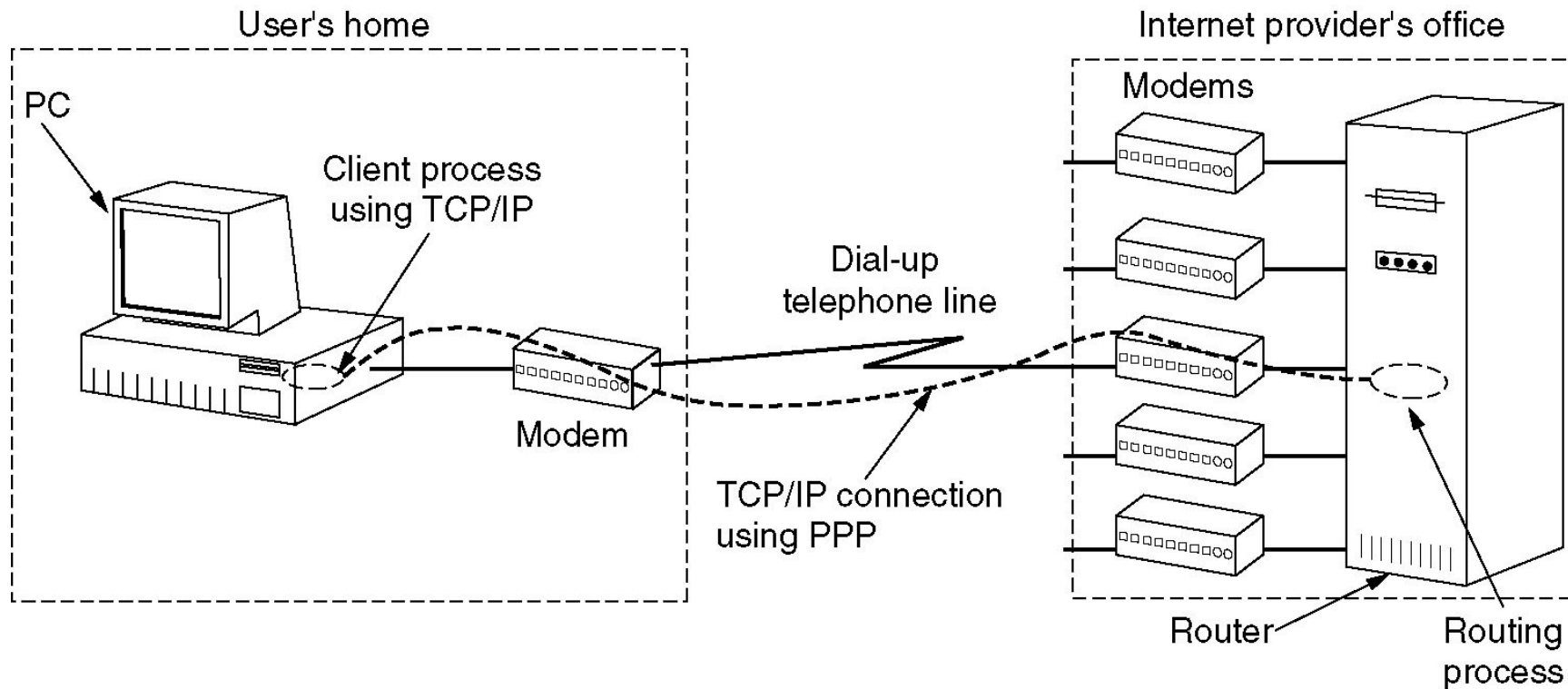
- **DISC** (DISConnect) - команда за разпадане на съединение;
- **SNRM** (Set Normal Response Mode) - команда за установяване на режим, в който едната машина управлява, а другата изпълнява;
- **SABM** (Set Asynchronous Balanced Mode) - команда за установяване на режим, в който двете машини имат еднакво 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.

# HDLC - Полето *Control* за unnumbered -кадрите

Специален unnumbered кадър UA (unnumbered acknowledgement) за потвърждаване на получена команда.

След изпращането на всяка команда се изчаква съответно потвърждение преди да се изпрати друга команда.

# Други Data Link Layer протоколи



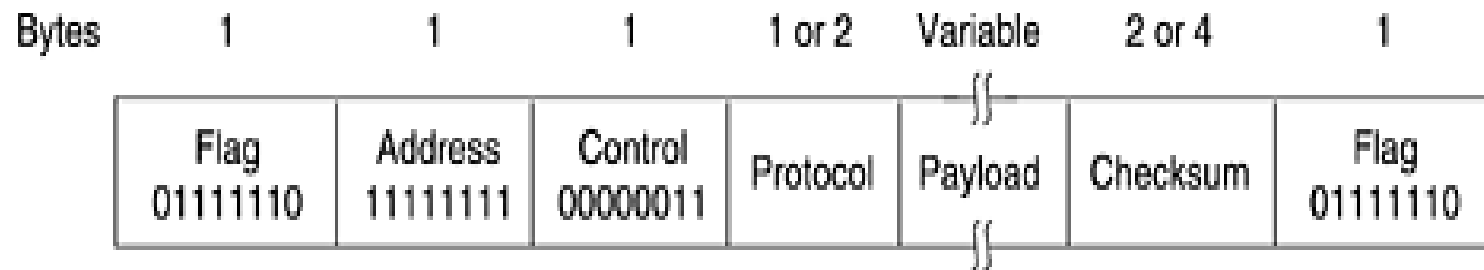
Свързване към internet по PPP.

# 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:



# Протокол PPP

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

Полето *Control* съдържа 00000011, 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 помещава съответния пакет.

Максималната дължина на *Payload* е 1500 байта.

# Протокол RRR

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.

# PPP фази

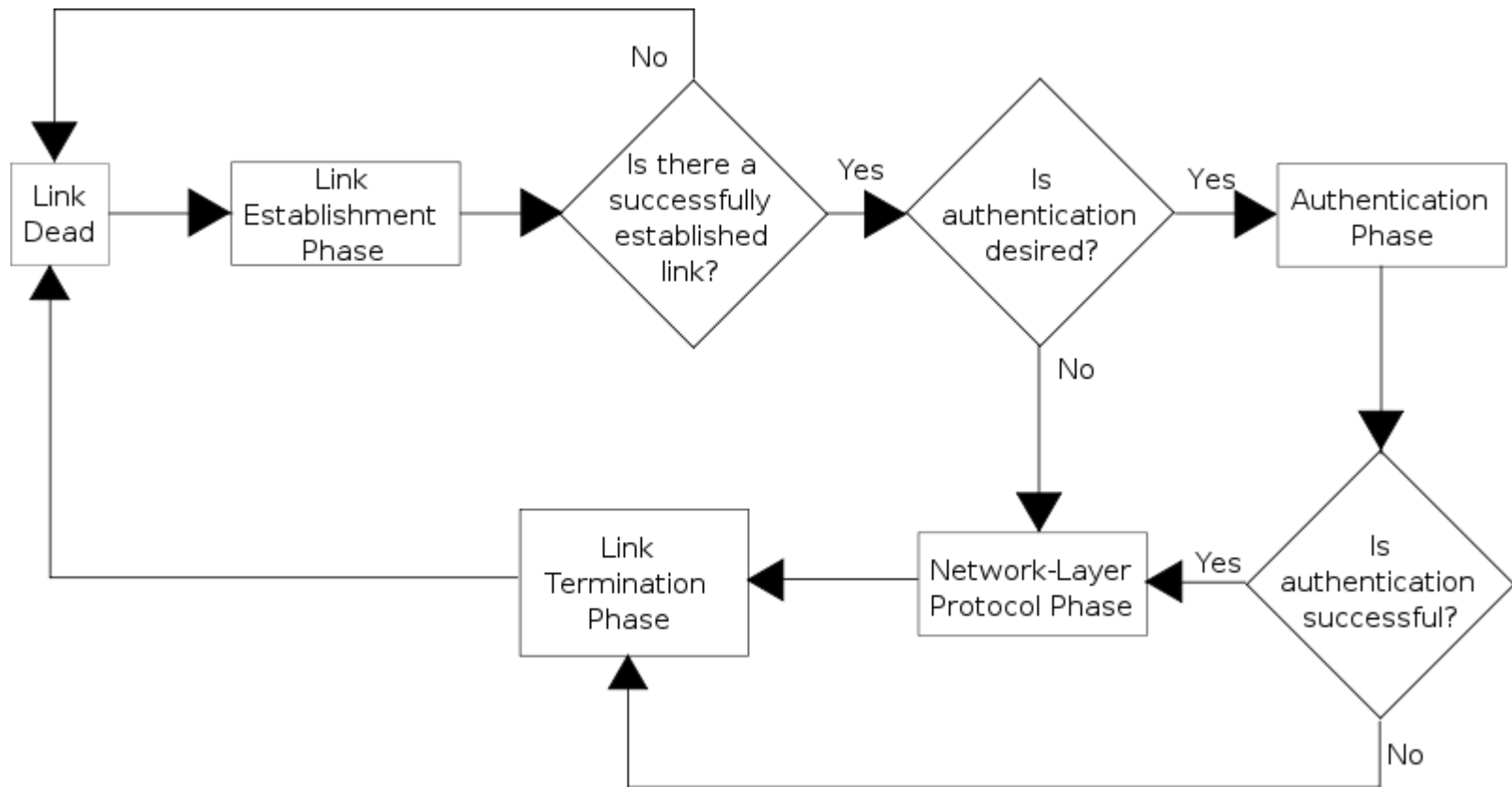


Схема на установяване на връзката.



# LCP

## Line Control Protocol (LCP):

автоматично конфигуриране на срещуположните интерфейси:

дължина на кадъра, ESC символи;

Проверка на линията за грешки с произволни числа (**magic numbers**). Ако линията е дадена накъсо, възелът получава LCP съобщение със своя си magic number, вместо да получи magic number на съседа;

Компресия.

Последвани евентуално от **аутентикация**.

# Аутентикация

Съседите си обменят съобщения за аутентикация.

Имаме два варианта:

Password Authentication Protocol (**PAP**) и  
Challenge Handshake Authentication Protocol  
(**CHAP**).

В безжичните мрежи се ползва **EAP**.

# PAP

**PAP** предава пароли в **явен ASCII текст** по мрежата, затова е **несигурен**.

- \* Клиент изпраща **username** и **password**
- \* Сървърът връща:  
**authentication-ack** (ако е ОК) или  
**authentication-nak** (в противен случай).

# SHAR

SHAR периодически проверява идентичността на клиента чрез **three-way handshake**. При установяване на сесията и през произволни интервали от време след това. Проверката се базира на **споделена “тайна”** (напр. Паролата на потребителя).

1. **Сървърът** изпраща "**challenge**" съобщение към клиента.
2. **Клиентът** отговаря с число, изчислено с помощта на еднопосочна хеш функция, напр. **MD5 checksum hash**.
3. **Сървърът** сравнява този хеш със своя. Ако съвпадат, **с;едва acknowledge**; в противен случай връзката се прекъсва.
4. През **произволни интервали** сървърът изпраща ново предизвикателство: стъпки 1-3 се повтарят.

# Network Control Protocol

Network Control Protocol (NCP) се стартира след LCP.

Уговаря опции за протокола от мрежовия слой, над PPP. NCP са:

Internet Protocol Control Protocol (**IPCP**) за IP,  
Internetwork Packet Exchange Control Protocol  
(**IPXCP**) за IPX и  
AppleTalk Control Protocol за AppleTalk и

**IPv6 Control Protocol (IPV6CP)** за предаване на IPv6 пакети по PPP линии.

# Развитие на PPP – PPPoE

**PPPoE, Point-to-Point Protocol over Ethernet** опакова PPP кадри вътре в Ethernet кадри.

Използва се при свързвания към Интернет чрез ADSL или кабелни модеми, LANs, WLANs или Metro Ethernet мрежи.

Разработена е от UUNET, Redback Networks и RouterWare, стандартизирана е в RFC (Request for Comment) 2516.

Ethernet мрежите са с пакетна комутация, connectionless, нямат механизми за защита срещу IP и MAC конфликти и компрометирани DHCP сървъри.

Чрез PPPoE потребителите виртуално “набират номера” на отдалечен сървър на провайдера през Ethernet и установяват “point to point” връзка.

# PPPoE - стадии

PPPoE се установява на два точно определени стадия:

## PPPoE discovery

Традиционните PPP връзки се установяват между две крайни точки, които са предварително изградени.

Но Ethernet мрежите са multi-access, така че преди обмен на PPP контролни пакети за установяване на връзката върху Ethernet, двете старни ще трябва да си научат MAC адресите, за да бъдат закодирани в контролните пакети.

Също така се установява **Session Id**, която се използва при обмена на пакети.

## PPP session

След като са известни MAC адресите и е установена сесията, двете старни имат всичката информация за изграждане на “point-to-point” връзка по Ethernet и обмен на пакети.

# MPLS

**Multiprotocol Label Switching (MPLS)** е механизъм за реализация на високопроизводителни телекомуникационни мрежи.

Разработен е от **IETF** (Internet Engineering Task Force).

MPLS работи на OSI "**Слой 2.5**".

**Multiprotocol:**

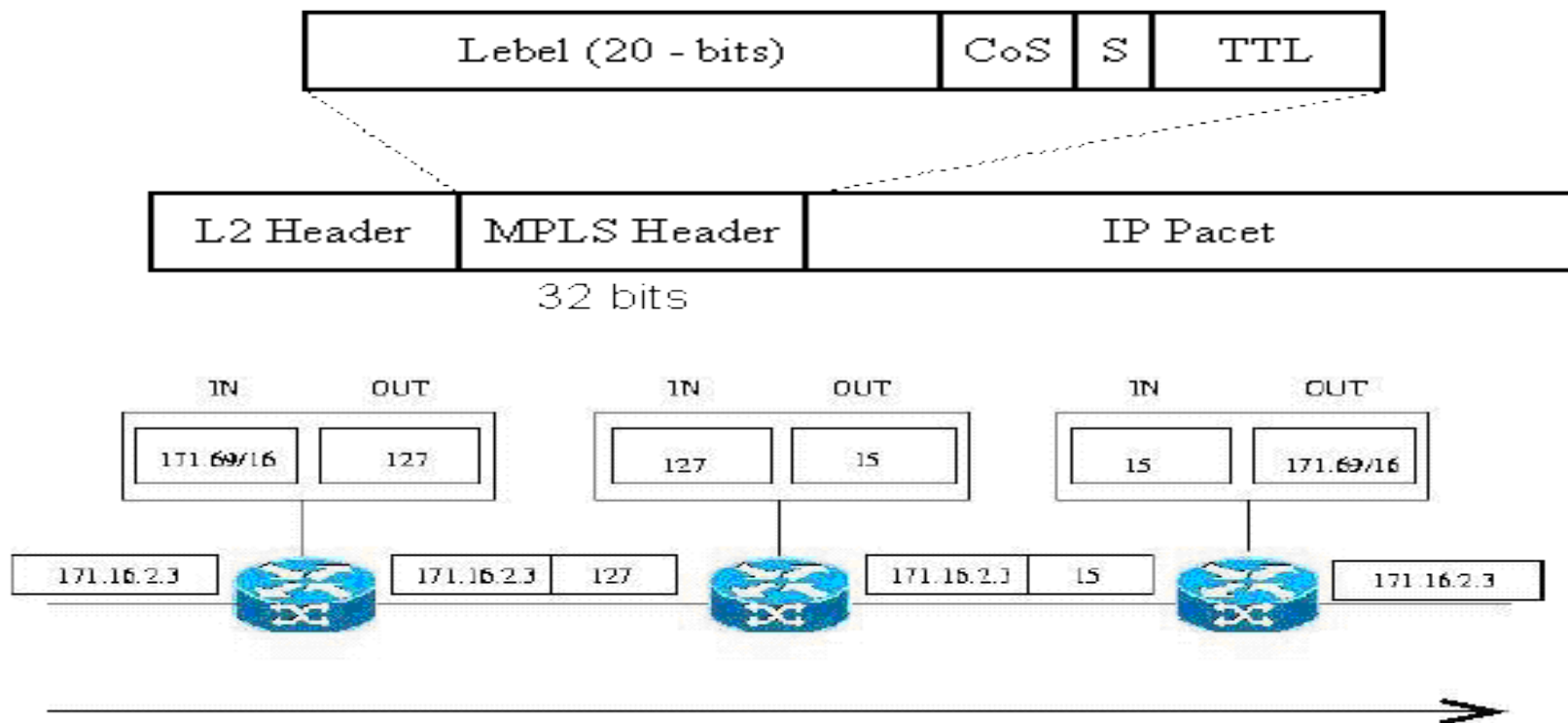
Пренася всякакви видове трафик: **IP, ATM, SONET и Ethernet.**

**Label Switching:**

Данните се направляват от възел на възел с помощта на **етикети.**



# MPLS. Layer 2.5.



Фигура 3: Препращане на пакети чрез етикети в MPLS

Етикетът съответства на Forwarding Equivalence Class – **FEC**;

Прави се проверка в информационната база с етикети (LIB), за да се определи:

- следващия участък от връзката;
- коя връзка да се използва и как да се подреди пакета в опашката.

# Как работи MPLS

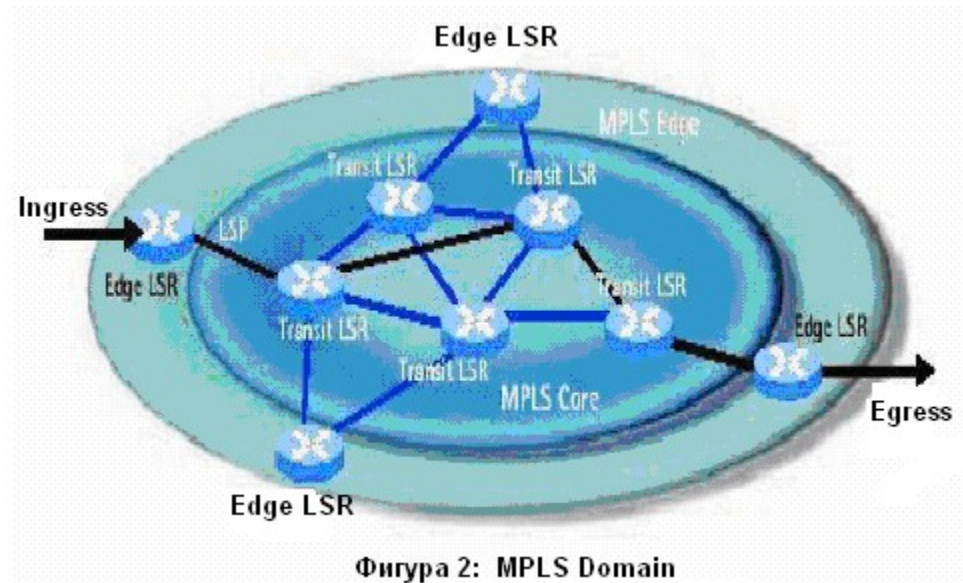
В MPLS маршрутизаторът препраща пакета през MPLS домейн. Възлите по пътя не взимат никакви маршрутизиращи решения. Използват етикета в пакета за определяне на следващата стъпка по пътя.

MPLS домейнът (domain) е подобно на AS (autonomous system – автономна система), в IP маршрутизацията, т.е. група от свързани рутери, които са под единен административен и управленски контрол.

MPLS domain се разделя на MPLS ядро (core) и MPLS периферия (edge).

Възлите в core са LSRs ( Label Switch Routers), а възлите в edge - Label Edge Routers (LER).

# MPLS domain



Когато IP пакет преминава през MPLS domain, той преминава през предварително зададен път, зависещ от FEC, Label Switched Path – LSP.

FEC действа, като филтър: кои IP пакети по кои LSPs да бъдат препратени.

# MPLS и IPv6

MPLS работи и върху IPv4, и **върху IPv6** мрежи. **Ще видим...**

**Митовете** за MPLS <http://delian.blogspot.com/2006/11/mpls.html>

- **QoS в IP** – в IP имаме повече приоритети (64 с DSCP, срещу 8 ако използваме Exp за CoS);

- **Layer3 VPN** - L3 VPN се прави и с **GRE** тунели, и с **IPSec**, и с чист **Ethernet**. и т.н. и т.н.

**MPLS не може да се мери с “чиста” Ethernet мрежа.**

Ethernet е **по-евтин**, същата функционалност, по-висок капацитет и по-висока съвместимост.

Ethernet е достатъчна само заради бързодействието.

# MPLS мрежа OPnet

