

# Úvod do počítačových sítí KIV/UPS

Ing. Petr Včelák

16. prosince 2008

# Obsah

<b>1</b>	<b>Organizační záležitosti</b>	<b>6</b>
1.1	Podmínky absolvování předmětu	6
1.1.1	Zápočet	6
1.1.2	Zkouška	6
1.2	Pokyny pro zpracování semestrální práce	6
1.2.1	Zásady vypracování semestrální práce	6
1.2.2	Co musí obsahovat dokumentace	7
1.2.3	Jak odevzdávat semestrální práce	7
1.3	Postup odevzdání semestrální práce	7
<b>2</b>	<b>Úvod do Linuxu</b>	<b>8</b>
2.1	Základní příkazy	8
2.2	Nástroj Wireshark	8
2.3	helloworld.c	8
<b>3</b>	<b>Protokolový zásobník TCP/IP</b>	<b>8</b>
3.1	Protokoly TCP/IP	8
3.1.1	Vrstvy protokolového zásobníku TCP/IP	9
3.1.2	Přenosová vrstva	9
3.1.3	Síťová vrstva	9
3.1.4	Transportní vrstva	10
3.1.5	Aplikační vrstva	11
3.1.6	Porty	11
3.1.7	Adresování procesu uzlu	12
3.2	IP adresy a adresování	12
3.2.1	Rezervované adresy	12
3.2.2	Privátní adresy	12
3.2.3	Automatická privátní adresa	12
3.2.4	Maska sítě	13
3.2.5	Třídy adres	13
3.2.6	CIDR (Classless InterDomain Routing)	13
3.3	Typy serverů a služeb	13
3.3.1	Model klient-server	13
3.3.2	Spojované služby	13
3.3.3	Nespojované služby	14
3.3.4	Spolehlivá služba	14
3.3.5	Nespolehlivá služba	14
3.3.6	Stavové servery	14
3.3.7	Bezstavové servery	14
3.3.8	Interaktivní zpracování požadavků	14
3.3.9	Paralelní zpracování požadavků	14
<b>4</b>	<b>BSD sockety</b>	<b>15</b>
4.1	Sockety	15
4.1.1	Názvosloví	15
4.1.2	Adresy	15
4.1.3	Systémová volání pro práci se sockety	16
4.1.4	Posílání zpráv do socketu	18
4.1.5	Příjem zpráv ze socketu	19
4.2	Algoritmy - nespojované služby	19
4.2.1	Server	19
4.2.2	Klient	19
4.3	Algoritmy - spojované služby	20
4.3.1	Server	20
4.3.2	Klient	20

4.4	Příklady realizace serverů . . . . .	20
4.5	Select . . . . .	20
4.6	Psaní vlastních programů pod Linuxem . . . . .	21
4.7	Nástroj make . . . . .	21
<b>5</b>	<b>Síťování v Javě</b>	<b>22</b>
5.1	InetAddress . . . . .	22
5.2	SocketAddress . . . . .	23
5.3	Komunikace protokolem UDP . . . . .	23
5.3.1	Klient . . . . .	23
5.3.2	Server . . . . .	24
5.4	Komunikace protokolem TCP . . . . .	24
5.4.1	Klient . . . . .	24
5.4.2	Server . . . . .	25
5.5	Protokol HTTP . . . . .	26
<b>6</b>	<b>Přenosový kanál</b>	<b>26</b>
6.1	Zadání semestrálních prací . . . . .	26
6.2	Přenosy dat . . . . .	26
6.2.1	Reálné vlastnosti . . . . .	26
6.3	Šířka přenosového pásma . . . . .	27
6.3.1	Harmonický signál . . . . .	27
6.3.2	Obecný průběh signálu . . . . .	27
6.3.3	Analogový a digitální přenos . . . . .	27
6.4	Přenos v základním pásmu . . . . .	27
6.4.1	Dvou-úrovňové kódování . . . . .	27
6.4.2	Více-úrovňové kódování . . . . .	27
6.5	Přenos v přeloženém pásmu . . . . .	28
6.5.1	Nosný signál . . . . .	28
6.5.2	Modulace . . . . .	28
6.5.3	Typy modulací . . . . .	28
6.6	Modulační rychlost . . . . .	28
6.6.1	Přenos v základním pásmu . . . . .	28
6.6.2	Přenos v přeloženém pásmu . . . . .	28
6.6.3	Vzorkování přijímaného signálu . . . . .	29
6.7	Přenosová rychlost . . . . .	29
6.7.1	Nyquistovo kritérium . . . . .	29
6.7.2	Shanonovo kritérium . . . . .	29
6.8	Modulační a přenosová rychlost . . . . .	29
6.9	Přenos . . . . .	30
6.9.1	Zapojení vodičů . . . . .	30
6.9.2	Směr přenosu . . . . .	30
6.9.3	Způsob přenosu . . . . .	30
6.9.4	Počet úrovní . . . . .	30
6.9.5	Typy spojů . . . . .	30
6.9.6	Přenosové vedení . . . . .	30
6.10	Kapacita přenosového kanálu . . . . .	30
<b>7</b>	<b>Přenos dat</b>	<b>31</b>
7.1	Synchronizace . . . . .	31
7.2	Synchronní přenos . . . . .	31
7.3	Arytmický přenos . . . . .	31
7.3.1	Start a stop bity . . . . .	31
7.3.2	Paritní bit . . . . .	32
7.3.3	Označení . . . . .	32
7.4	Asynchronní přenos . . . . .	32
7.5	Kódování signálu . . . . .	32

7.5.1	Return to Zero (RZ)	32
7.5.2	Return to Zero Inverted (RZI)	32
7.5.3	Non Return To Zero (NRZ)	33
7.5.4	Non Return To Zero Inverted (NRZI)	33
7.5.5	Manchester	33
7.5.6	Diferenciální Manchester	33
7.6	Vkládání bitů (bit stuffing)	34
7.7	Multiplexování	34
7.7.1	Frekvenční multiplex	35
7.7.2	Vlnový multiplex	35
7.7.3	Časový multiplex	35
7.7.4	Statistický multiplex	36
7.7.5	Synchronní multiplex	36
7.8	Sítě s přepínáním kanálů, paketů a zpráv	36
7.8.1	Přepínání kanálů	36
7.8.2	Přepínání paketů	36
7.8.3	Přepínání zpráv	36
<b>8</b>	<b>Chyby při přenosu</b>	<b>36</b>
8.1	Chybovost	37
8.2	Bezpečnostní kódy	37
8.3	Parita	37
8.3.1	Sudá/lichá parita	37
8.3.2	Příčná parita	37
8.3.3	Podélná parita	37
8.3.4	Křížová parita	38
8.4	Checksum — kontrolní součet	38
8.5	Hammingův kód	38
8.5.1	Postup generování Hammingova kodu	38
8.5.2	Rozšířený Hammingův kód (8,4)	38
8.6	Hammingova vzdálenost	38
8.6.1	Detekce chyb	38
8.6.2	Detekce a korekce chyb	38
8.6.3	Příklad 1	39
8.6.4	Příklad 2	39
8.6.5	Příklad 3	39
8.7	Cyklické kódy (CRC)	39
<b>9</b>	<b>Potvrzování přenosu</b>	<b>40</b>
9.1	Komunikační protokol	40
9.2	Číslování rámců	41
9.3	Potvrzovací schémata	41
9.3.1	Typy potvrzování	41
9.3.2	Stop-and-Wait	42
9.3.3	Sliding window	42
9.4	Stanovení velikosti okénka	45
9.5	Sekvenční příjem	45
9.6	Nesekvenční příjem	45
9.7	Příklady	45
9.7.1	Stop and Wait	45
9.7.2	Sliding Window	46
9.8	Protokoly linkové úrovně	46
9.8.1	BSC	46
9.8.2	HDLC	46
9.8.3	IEEE 802.2	46
9.8.4	PPP	46
9.8.5	SLIP	46

<b>10 Řízení přístupu v lokálních počítačových sítích</b>	<b>46</b>
10.1 Kolize . . . . .	46
10.2 Metody řízení přístupu . . . . .	46
10.3 Centralizované . . . . .	47
10.4 Decentralizované metody . . . . .	47
10.4.1 Aloha . . . . .	47
10.4.2 CSMA . . . . .	47
10.4.3 CSMA/CA . . . . .	48
10.4.4 CSMA/CD . . . . .	48
10.4.5 CSMA/BA . . . . .	50
10.4.6 Použití CSMA/xx . . . . .	50
10.5 Předávání pověření . . . . .	50
10.5.1 Token Ring . . . . .	51
10.5.2 Token Bus . . . . .	52
10.6 Ethernet . . . . .	52
10.6.1 Formát rámce . . . . .	53
<b>11 Směrování</b>	<b>54</b>
11.1 Pojmy . . . . .	54
11.2 Převod mezi MAC a IP adresou . . . . .	55
11.2.1 ARP . . . . .	55
11.2.2 Proxy ARP . . . . .	55
11.2.3 RARP . . . . .	55
11.3 Transparentní mosty . . . . .	56
11.3.1 Source routing . . . . .	56
11.3.2 Spanning Tree . . . . .	56
11.4 Algoritmy směrování . . . . .	56
11.5 Routing x forwarding . . . . .	57
11.6 Záplavové směrování (flooding) . . . . .	58
11.7 Metoda zpětného učení . . . . .	58
11.8 Vector distance routing (DVA) . . . . .	59
11.8.1 RIP (Routing Information Protocol) . . . . .	59
11.8.2 RIP 2 . . . . .	59
11.8.3 RIPng . . . . .	59
11.9 Link state protocol (LSA) . . . . .	60
11.10 OSPF . . . . .	60
<b>12 Transportní protokoly pro přenos dat</b>	<b>60</b>
12.1 Transportní vrstva . . . . .	60
12.2 Transportní protokoly . . . . .	61
12.3 Adresování . . . . .	62
12.4 Základní vlastnosti TCP . . . . .	62
12.4.1 Vlastnosti . . . . .	62
12.4.2 Vlastnosti spojení . . . . .	62
12.4.3 Vytvoření spojení . . . . .	62
12.4.4 Ukončení spojení . . . . .	63
12.4.5 Řízení přenosu dat . . . . .	63
12.5 Základní vlastnosti UDP . . . . .	63

# 1 Organizační záležitosti

## 1.1 Podmínky absolvování předmětu

### 1.1.1 Zápočet

K získání zápočtu je třeba vytvořit program podle vybraného zadání, úspěšně absolvovat zápočtový test a přednést na cvičení referát podle zadané semestrální práce.

**Účast na cvičení** je nutnou podmínkou pro získání zápočtu. Účast musí být minimálně 50%.

**Zápočtový test** je bodován v rozsahu 0 až 20 bodů. Nárok na zápočet budou mít ti, kteří získají alespoň 10 bodů.

Test bude obsahovat otázky bodované od 1 do 3 bodů. Otázky se budou týkat látky probrané do té doby na přednáškách i na cvičeních (základní principy, přenos dat komunikačním kanálem, linková úroveň, úroveň přístupu ke komunikačnímu kanálu). Otázky budou orientovány zejména na výpočty. Zápočtový test se bude psát 47. týden (19. až 23. 11.). Náhradní termín bude určen dodatečně po dohodě.

**Semestrální práce** je bodována stupnicí 0 až 30 bodů. Podmínkou pro získání zápočtu je dosažení alespoň 15 bodů. Mezní termín pro nahlášení vybraného zadání cvičícímu je během cvičení, která budou probíhat 43. týden (22. až 26. 10.). Mezní termín pro odevzdání semestrální práce je 18. 1. 2008. Za každý další pracovní den prodlení se strhává 1 bod. Semestrální práci je nutné odevzdat nejpozději týden před mezním termínem pro získání zápočtu.

**Referát na cvičení** přednese každý student na cvičení. Týkat se bude zadané semestrální práce. Délka referátu je 10 min., prezentace bude doprovázena promítnutím předem připravených předloh datovým projektorem. Termín přednesení referátu bude stanoven cvičícím předem.

Cvičící může zohlednit aktivní přístup studenta na cvičeních nebo nadprůměrnou semestrální práci přidáním až 10 bodů do celkového hodnocení.

### 1.1.2 Zkouška

Viz přednášky.

## 1.2 Pokyny pro zpracování semestrální práce

### 1.2.1 Zásady vypracování semestrální práce

- Úlohu naprogramujte v programovacím jazyku C anebo Java. Pokud se jedná o úlohu server/klient, pak klient bude v Javě a server v C.
- Výstupy serveru budou v alfanumerické podobě, klient může komunikovat i v grafice (není podmínkou).
- Server řešte pod operačním systémem Linux, klient může běžet pod OS Windows XP. Emulátory typu Cygwin nebudou podporovány.
- Realizujte bezstavové a konkurentní (paralelní) servery.
- Server musí být schopen obsluhovat požadavky více klientů souběžně.
- V případě použití nespojovaných služeb (UDP) vyřešte na úrovni aplikačního protokolu problematiku ztráty příp. duplicity dat (např. číslování, metoda okénka, apod.).
- Součástí programu bude trasování komunikace, dovolující zachytit proces komunikace na úrovni aplikačního protokolu a zápis trasování do souboru.
- Každý program bude doplněn o zpracování statistických údajů (přenesený počet bytů, přenesený počet zpráv, počet navázaných spojení, počet přenosů zrušených pro chybu, doba běhu apod.).
- Zdrojové kódy organizujte tak, aby od sebe byly odděleny části volání komunikačních funkcí, které jste vytvořili na základě zadání, od částí určených k demonstraci funkčnosti vašeho řešení (grafické rozhraní).

### 1.2.2 Co musí obsahovat dokumentace

- Dokumentace bude odevzdána pouze v elektronické podobě (jako součást referátu)
- Úvodní stránku se jménem zadání, Vaším příjmením a jménem, univerzitním číslem a emailovou adresou.
- Text zadání spolu s číslem zadání.
- Programátorskou dokumentaci - popis řešení, použitých datových struktur, formáty vyměňovaných zpráv, programů použitých při vývoji a programů potřebných k provozu aplikace.
- Uživatelskou dokumentaci - postup přeložení a sestavení, spuštění, parametry programu, ovládání.
- Odkazy na použité zdroje.
- V případě realizace rozšířených zadání také odkazy na použité RFC dokumenty

### 1.2.3 Jak odevzdávat semestrální práce

- K odevzdání samostatných prací použijte výukový systém KIV, který je dostupný na adrese <http://students.kiv.zcu.cz/vyuka>. Pro UPS je systém nastaven tak, že umožňuje odevzdání pouze 1 souboru.
- K práci musí být odevzdány zdrojové soubory, odpovídající přeložené binární soubory a soubor s dokumentací ve formátu pdf. Pokud pro překlad a sestavení používáte make resp. ant, pak musí k programu být přiložen i příslušný soubor makefile resp. build.xml. Všechny soubory zabalte do zipového souboru s názvem vytvořeným z vašeho příjmení a identifikačního čísla odděleného podtržítkem (<prijmeni>\_<osobni\_cislo>.zip například tedy horak\_A01234.zip). Tento soubor potom vložte do výukového systému. Zipový soubor bude mít následující strukturu:

```
./c_bin/ - přeložené binární soubory pro Linux
./c_src/ - zdrojové soubory a soubory potřebné pro překlad
./java_bin/ - přeložené binární soubory pro Windows
./java_src/ - zdrojové soubory a soubory potřebné pro překlad
./dokumentace.pdf
```

## 1.3 Postup odevzdání semestrální práce

1. Vytvořit soubor <prijmeni>\_<osobni\_cislo>.zip podle postupu výše.
2. Odevzdat soubor do systému KIV výuka (<http://students.kiv.zcu.cz/vyuka>).
3. Připravit si pracovní prostředí v UL402 pod OS GNU/Linux (např. 3 stroje = 1x server + 2x klient, apod. tak aby bylo možno předvést funkci semestrální práce).
4. Ověřit zda lze semestrální práci přeložit, spustit a používat.
5. Domluvit s cvičícím termín předvedení.
6. Na smluvený termín se zastavit u cvičícího, předvést překlad, spuštění a funkci semestrální práce.
7. Jestliže cvičící prezentaci schválí (schválí) čekat až budou výsledky hodnocení semestrální práce.
8. Jsou-li všechny nutné podmínky k získání zápočtu splněny, přijít si pro zápočet v úředních hodinách.

## 2 Úvod do Linuxu

### 2.1 Základní příkazy

- ifconfig — configure a network interface
- arp — manipulate the system ARP cache,
- netstat — print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships,
- ping — send ICMP ECHO\_REQUEST to network hosts,
- traceroute — print the route packets trace to network host,
- nslookup — query Internet name servers interactively,
- dig — DNS lookup utility,
- host — DNS lookup utility,
- ip — show / manipulate routing, devices, policy routing and tunnels,
- route — show / manipulate the IP routing table,
- tcpdump — dump traffic on a network,
- a další.

### 2.2 Nástroj Wireshark

viz cvičení

### 2.3 helloworld.c

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    printf("Hello world!\n");
    return 0;
}

$ gcc -o helloworld helloworld.c
$ ./helloworld
Hello world!
$
```

## 3 Protokolový zásobník TCP/IP

### 3.1 Protokoly TCP/IP

Protokoly TCP/IP umožňují komunikaci mezi uzly na síti. Struktura odpovídá *sérii vrstev* nebo jakéhosi „zásobníku“ mezi aplikací a sítí. Každá vrstva je přítomna na obou komunikujících uzlech.



### 3.1.1 Vrstvy protokolového zásobníku TCP/IP

Vrstvy protokolového zásobníku TCP/IP jsou 4 (rozdíl oproti ISO/OSI modelu) a jsou to:

**aplikační** (application layer)

**transportní** (transport layer)

**síťová** (network layer)

**přenosová** (data link layer)

Vrstva může komunikovat pouze s vrstvou přímo sousedící. Obecně vrstva  $n$  využívá služeb poskytovaných vrstvou  $n-1$  a naopak zase poskytuje služby vrstvě  $n+1$ . Tímto způsobem je docíleno rozdělení složité funkcionality na menší části. Vrstvená architektura umožňuje výměnu protokolů jedné vrstvy bez dopadu na ostatní. Příkladem může být možnost komunikace po různých fyzických médiích - ethernet, token ring, sériová linka a další.

Aplikace z aplikační vrstvy uzlu komunikuje s aplikací na aplikační vrstvě na druhém uzlu. Komunikace mezi stejnými vrstvami na různých uzlech probíhá podle jistých pravidel, která označujeme pojmem *protokol*.

Aplikační vrstva jednoho uzlu je *spojena virtuálně* s aplikační vrstvou uzlu druhého. Komunikující aplikace předává zprávu transportní vrstvě. Podle zvoleného protokolu z transportní vrstvy je vytvořen příslušný paket (TCP, UDP). Paket je z transportní vrstvy předán na síťovou vrstvu. Na síťové vrstvě je paket vložen do IP datagramu a předán na nejnižší úroveň — na fyzické přenosové médium. Přes médium jsou přenesena data na uzel cílový, na který data posíláme. V cílovém uzlu se provádí opak předchozího postupu tak že každá vrstva použije tu část, která jí náleží a ostatní předá na vyšší vrstvu. Výsledkem je že aplikace na cílovém uzlu obdrží zprávu vycházející z komunikačního protokolu aplikační vrstvy — tj. protokolu kterému aplikace rozumí.

### 3.1.2 Přenosová vrstva

Nejnižší vrstva modelu umožňuje přístup k fyzickému přenosovému médium. Je implementačně závislá přímo na fyzickém médium.

Známá přenosová média jsou např:

- Ethernet
- FDDI
- Token ring
- PPP (Point-to-Point Protocol)
- Wi-Fi

### 3.1.3 Síťová vrstva

Již není závislá na médium a využívá služeb přenosové vrstvy. Na úrovni této vrstvy je prováděno adresování, směrování a předávání *datagramů*. Je implementována ve všech prvcích sítě jako jsou směrovače i koncová zařízení (např. počítač).

Protokoly síťové vrstvy jsou:

- IP (Internet Protocol):
  - nespojovaný,
  - nespolehlivý — nepotvrzované služby,
  - princip maximální snahy,
  - přenos paketů a jejich směrování podle cílové adresy,
- ARP (Address Resolution Protocol) - převod síťové adresy na fyzickou,
- RARP ,
- Proxy ARP,

- ICMP (Internet Control Message Protocol) - přenos zpráv o chybách, dosažitelnost uzlu sítě (traceroute),
- IGMP,
- IGRP,
- IPSEC.

V síťové vrstvě jsou tedy zahrnuty:

- síťová adresa (nikoliv linková adresa),
- převodní mechanismus mezi síťovou a fyzickou (linkovou) adresou
- mechanismy fragmentace
- protokol ICMP (informace o nestandardních situacích)
- mechanismy překladu adres (NAT)
- koncept privátních IP adres
- mechanismy adresování (CIDR), dělení a sdružování adres (subnetting, supernetting)
- bezpečnostní mechanismy (IPSEC)
- podpora mobility zařízení (MobileIP)

### 3.1.4 Transportní vrstva

Implementována v koncových zařízeních a umožňuje přizpůsobit chování sítě pro potřeby aplikace. Poskytuje spojované či nespojované transportní služby.

- TCP (Transport Control Protocol)
  - spojované služby (plně duplexní spojení- současný obousměrný přenos dat),
  - potvrzované - spolehlivé (obnova po chybě, kontroluje checksum, sekvenční čísla pro zaručení správného pořadí nebo zjištění ztráty paketu)
  - data z aplikační vrstvy po bytech
  - řízení toku dat (zajištění aby vysílající uzel nevysílal rychleji než je příjemce schopen zvládnout)
  - rozlišování aplikací pomocí portů
- UDP (User Datagram Protocol)
  - nespojované služby,
  - nepotvrzované - nespolehlivé služby — pouze snaha (best effort) o bechybný přenos (mohou tedy být ztraceny, omylem zopakovány, zpožděny, přijaty v jiném pořadí)
  - data z aplikační vrstvy přebírá po blocích
  - nemá fázi navazování a ukončení spojení a už první segment UDP obsahuje
  - použit pro DHCP, TFTP, SNMP, DNS, BOOTP
  - rozlišování aplikací pomocí portů

Programování s využitím spojových služeb (TCP) je snadnější, protože veškeré ošetřování chybových stavů řeší protokol sám o sobě. Nespojované služby (UDP) jsou oproti spojovaným službám (TCP) rychlejší, protože odpadá režie spojená se zaručením spolehlivosti přenosu. Použití nespojovaných služeb UDP umožňuje použití broadcast a multicast zpráv.

### 3.1.5 Aplikační vrstva

Účelem vrstvy je poskytnout aplikacím přístup ke komunikačnímu systému a umožnit tak jejich spolupráci. Jedná se o programy (procesy), které využívají přenosu dat po síti ke konkrétním službám pro uživatele.

Protokoly aplikační vrstvy jsou např.:

- BitTorrent
- DNS
- BOOTP (Bootstrap Protocol) - získání síťového nastavení pro provoz uzlu
- DHCP (Dynamic Host Configuration Protocol) - obdoba BOOTP, modernější - umožňuje dynamickou změnu nastavení uzlu
- FTP
- HTTP
- HTTPS
- IMAP
- IRC
- Ident
- NNTP
- NTP
- POP3
- RTP
- SIP
- SMB
- SMTP
- SNMP
- SSH
- STUN
- X11
- XMPP
- Telnet

Aplikační protokoly používají vždy některou ze základních služeb transportní vrstvy (TCP nebo UDP), příp. obě dvě (používá DNS).

### 3.1.6 Porty

Porty se používají pro rozlišení aplikačních protokolů/komunikujících aplikací. Každá strana TCP spojení má přidruženo 16b číslo portu (65535 portů celkem).

**dobře známé** (well known) jsou porty v rozsahu 0 - 1023 a jsou vyhrazena pro nejběžnější služby,

**registrované porty** (registered) jsou porty z rozsahu 1024 - 49 151 a jejich použití by se mělo registrovat u ICANN

**dynamické a soukromé** (dynamic, private) jsou porty z rozsahu 49 152 - 65 535 a jsou určeny pro dynamické přidělování nebo soukromé využití, tj. nejsou pevně přiděleny žádné aplikaci.

### 3.1.7 Adresování procesu uzlu

Každé síťové spojení aplikace (procesu) je jednoznačně určeno trojicí hodnot:

- adresou uzlu (počítače).
- transportním protokolem
- číslem portu

## 3.2 IP adresy a adresování

Adresa IP se skládá ze dvou částí. První označuje síť, kde se nachází hostitel. Hostitel je druhá část adresy. Dále může existovat jmenné označení adresy.

- IPv4
  - 32b adresa, x.x.x.x kde x je 0 až 255 (0 až FF) tedy celkem  $2^{32}$  adres.
  - adresy dochází, jejich nedostatek se řeší NATem (oddálení problému)
- IPv6
  - 128b adresa (0 až FFFF),
  - podpora bezpečnosti,
  - podpora pro mobilní zařízení,
  - funkce pro zajištění úrovně služeb (QoS - Quality of Service),
  - fragmentace paketů - rozdělování.

Dále budeme mluvit o IPv4.

### 3.2.1 Rezervované adresy

- 0.0.0.0
- 127.0.0.0
- 128.0.0.0
- 191.255.0.0
- 192.0.0.0
- 223.255.255.0
- 255.255.255.255 — všeobecná adresa (přenos všem v daném lokálním segmentu)

### 3.2.2 Privátní adresy

- 10.0.0.0 — 10.255.255.255 (8 bitů masky)
- 172.16.0.0 — 172.31.255.255 (12 bitů masky)
- 192.168.0.0 — 192.168.255.255 (16 bitů masky)

### 3.2.3 Automatická privátní adresa

Automaticky vygenerovaná IP adresa, kterou si počítač přiřadí, pokud nemá implicitně žádnou definovanou a DHCP server není dostupný. Má definován tento rozsah adres:

- 169.254.0.1 — 169.254.255.254 (16 bitů masky)

### 3.2.4 Maska sítě

Oddělení adresy sítě a adresy hostitelského systému. Tyto adresy jsou odděleny pro minimalizaci počtu položek ve směrovacích tabulkách.

Pro masku 255.255.255.0 jsou všechny stroje s IP 147.228.67.\* ve stejné síti.

### 3.2.5 Třídy adres

Síť třídy A je taková síť ve které první číslo (8b z 32b pro IPv4) označuje síť a zbylá tři čísla představují adresy hostitelů. Síť třídy B používá první dvě čísla pro označení síťové části adresy a zbylé dvě pro hostitele. U sítě třídy C jsou použity první tři čísla pro označení sítě a pouze čtvrté je vyhrazeno pro adresy hostitelů. V takové síti je tedy nejmenší prostor pro adresy hostitelských síťových subjektů.

**Třída A** Individuální adresa (1.0.0.0 — 126.255.255.255)

**Třída B** Individuální adresa (128.1.0.0 — 191.254.255.255)

**Třída C** Individuální adresa (192.0.1.0 — 223.255.254.255)

**Třída D** Skupinová adresa (skupina uzlů: 224.0.0.0 - 239.255.255.255)

Původní způsob přidělování adres v síti Internet. Vedl ke špatné efektivitě využití adresního prostoru a rychlý úbytek adres.

Náročné směrování — směrovací tabulky byly velmi rozsáhlé, protože přidělování adres sítí se provádělo náhodně.

Řešením problémů je CIDR.

### 3.2.6 CIDR (Classless InterDomain Routing)

Možnost použít v podsíti adresy, které nejsou na hranici 8. Adresu zapisujeme ve tvaru <adresa>/<pocet\_bitu\_sitove\_casti> tedy např. 147.228.67.0/24.

- délka adresy sítě je libovolná
- adresy se přidělují hierarchicky, což umožňuje agregaci směrování

Aktuální adresní schéma pro přidělování adres v síti Internet a třídy adres byly zrušeny. Adresa sítě může mít libovolnou délku. Není omezení jen na 8, 16 či 24 bitů jako dříve při použití tříd adres. Zapisuje se v podobě prefixu ve tvaru adresa/délka. Délka přidělené adresy se stanoví podle skutečných a předpokládaných potřeb připojované sítě, pravidla pro její hodnotu jsou podstatně přísnější než dříve.

## 3.3 Typy serverů a služeb

### 3.3.1 Model klient-server

**klient** je aplikace, která zahajuje komunikaci (dotaz)

**server** je aplikace, která čeká na komunikaci a na jejím základě provádí požadovanou reakci (odpověď)

Klient využívá služeb serveru. Server služby poskytuje klientům. Server má na starosti veškerou práci spojenou s příjmem a obsluhou požadavků klienta. Serverová aplikace musí být spuštěna uživatelem, aby mohla služby poskytovat, není spouštěna automaticky s příchodem požadavku.

### 3.3.2 Spojované služby

Spojované služby (Connection-oriented Service) jsou založeny na principu, že kontaktujete cílový uzel a ten musí potvrdit (souhlasit), že budete komunikovat spolu. Tím vznikne spojení mezi vaším a cílovým uzlem v síti. Pokud spojení existuje lze komunikovat. Ukončením spojení na libovolné straně dojde k ukončení možnosti komunikovat spolu.

Vhodné pro větší přenosy dat, kdy je potřeba pouze na počátku vytvořit spojení a po té pouze posílat data.

Pro spojované služby je učen protokol TCP z transportní vrstvy.

Fáze spojovaných služeb:

- navázání spojení,
- přenos dat,
- ukončení spojení.

### 3.3.3 Nespojované služby

U nespojované služby (Connectionless Service) se považuje zpráva za jeden celek společně s adresou příjemce. Doručení této zprávy je nezávislé na doručení ostatních zpráv. Každá z odesílaných zpráv může procházet jinou cestou a tedy mohou být příjemci doručeny v různém pořadí (nebo dokonce ztraceny).

Vhodné pro krátké zprávy.

Nespojované služby lze poskytovat protokolem UDP transportní vrstvy.

### 3.3.4 Spolehlivá služba

U spolehlivé služby (Reliable Service) se nikdy neztratí přenášená data. Tj. příjemce vždy dostane uplná a správná data. Příjem dat musí být příjemcem potvrzován.

Také bývá označována jako potvrzovaná služba.

### 3.3.5 Nespolehlivá služba

Nespolehlivá služba (Unreliable Services) je taková služba, která nezaručuje doručení, protože se nepoužívá mechanismus potvrzování.

Označení též nepotvrzovaná služba.

### 3.3.6 Stavové servery

*Stavová informace* je informace, která popisuje spojení klient-server. Jestliže server tuto informaci udržuje, označujeme jej jako *stavový server*.

Pokud server nebo klient přijde o informaci v jakém stavu se komunikace nacházela (z libovolné příčiny) pak samozřejmě nelze pokračovat v komunikaci. Existují způsoby obnovy stavu, ale jejich efektivita je nízká (např. server informuje klienta o rozpadu komunikace a požaduje nové zahájení celé komunikace).

Příkladem stavového serveru je udržování informace o přihlášeném uživateli. Pokud klient požaduje otevření a čtení souboru na serveru, pak stavová informace může být že v prvním kroku dojde pouze k otevření souboru, v dalších krocích probíhá čtení (např. také od místa v souboru, které je definováno stavovou informací). Poslední požadavek klienta zajistí uzavření souboru po tom, co server odeslal místo dat informaci o konci souboru.

Navrhnout stavový server správně je obtížnější v porovnání s bezstavovým serverem.

### 3.3.7 Bezstavové servery

Bezstavové servery jsou takové, které nedrží informaci o stavu spojení klient-server. Kladou menší nároky na spolehlivost přenosových služeb oproti stavovému serveru.

Aby mohl být server bezstavový musí být splněny především podmínky:

- protokol nesmí specifikovat obsah nějaké zprávy na základě zprávy předcházející,
- stejný požadavek vyvolá vždy stejnou reakci.

### 3.3.8 Interaktivní zpracování požadavků

Server interaktivně zpracovává jediný požadavek jednoho klienta. Ostatní musí čekat až se dostanou na řadu.

### 3.3.9 Paralelní zpracování požadavků

Server má sloužit více klientům a to současně, tedy paralelně (více procesů — více procesorů, sdílení času, apod.). Na operačních systémech typu Unix jsou k dispozici následující prostředky, které dovolují provádět paralelní zpracování:

- systémová funkce `fork()` — rozdělí běžící program na dva procesy,
- systémové volání `execve()` — umožní přepsat kód procesu jiným,

- systémové volání `select()` — obsluha několika současných událostí (`stdio`, `socket`, ...).

Těmito způsoby je možné pro každé spojení klienta se serverem vytvořit samostatný proces. Vlastní obsluhu, která bude probíhat paralelně, zajistí operační systém (sdílení času, přidělení procesoru procesu, atd.).

## 4 BSD sockety

### 4.1 Sockety

**TCP** před samotnou komunikací se naváže spojení. Všechna odeslaná data se potvrzují a na konec je nutné spojení ukončit (uzavřít). TCP paket obsahuje svou hlavičku a samotná data, která přenáší. TCP paket bude vložen do IP datagramu (jako data IP datagramu) a odeslán.

**UDP** jedná se tzv. nespojovanou službu. To znamená, že nedochází k navázání spojení. Prostě odešleme data na stanovenou IP adresu a daný UDP port a nevíme, zda data dorazila a zda se nepoškodila nebo nedorazila v jiném spojení.

#### 4.1.1 Názvosloví

**Rodina protokolů** specifikuje a zahrnuje příbuzné protokoly a označuje se prefixem `PF_*` (protocol family). Pro všechny protokoly TCP/IP existuje rodina `PF_INET`.

**Typ služby** se označuje požadovaný protokol. Pro UDP se jedná o službu datagramového typu `SOCK_DGRAM` a pro TCP se jedná o službu typu stream `SOCK_STREAM`.

**Rodina adres** označuje se prefixem `AF_*` a často se plete<sup>1</sup> s konstantami rodiny protokolů (s prefixem `PF_*`). Rozdíl je však v tom, že *rodina protokolů* může využívat jednu nebo více *rodin adres*.

#### 4.1.2 Adresy

Definice struktur a konstant je možné dohledat v hlavičkovém soboru `<sys/socket.h>`. Např. na operačním systému GNU/Linux jej naleznete v `/usr/include/sys/socket.h`.

**SOCK\_STREAM** nejprve se naváže spojení, dále se používá ke spolehlivému přenosu dat, (TCP)

**SOCK\_DGRAM** přenos krátkých bloků dat, není zajištěno doručení ani pořadí, (UDP)

Obecný formát adresy je definován strukturou `sockaddr` následovně:

```
struct sockaddr
{
    unsigned short sa_family;    /* rodina adres AF_* */
    char sa_data[14];           /* 14B prima adresa */
};
```

Konkrétně pro rodinu adres `AF_INET` je formát adresy definován:

```
struct sockaddr_in
{
    short int sin_family;        /* rodina adres AF_* */
    unsigned short int sin_port; /* cislo portu */
    struct in_addr sin_addr;     /* Internetova adresa */
    unsigned char sin_zero[8];   /* nepouzito */
};

struct in_addr
{
    unsigned long s_addr;        /* 32 bitova adresa */
};
```

<sup>1</sup>Naštěstí mají pro TCP/IP obě konstanty (`PF_INET` i `AF_INET`) stejnou hodnotu 2 a jedná se tedy ve výsledku pouze o formální chybu, která vážnější problémy nepůsobí.

Některé architektury používají tzv. big-endian, jiné little-endian. Aby se spolu domluvily počítače s libovolnou architekturou je potřeba převádět adresy do síťového pořadí bytů. K tomu slouží funkce:

```
uint32_t htonl(uint32_t hostlong)    /* host -> network */
uint32_t ntohl(uint32_t netlong)     /* network -> host */

uint16_t htons(uint16_t hostshort)   /* host -> network */
uint16_t ntohs(uint16_t netshort)    /* network -> host */
```

Pro převod adresy z textového tvaru (147.32.86.1) do binárního tvaru ve správném pořadí bytů se používá funkce:

```
int inet_aton(const char *NAME, struct in_addr *ADDR);
```

Pro převod DNS jmenného označení počítače www.zcu.cz na binární IP adresu, použijeme funkci:

```
hostent * gethostbyname (const char *NAME);
```

#### 4.1.3 Systémová volání pro práci se sockety

**socket()** vytvoří socket daného typu a vrátí file-descriptor. Socket není asociován s žádným portem. Používá server i klient.

```
/* Create a new socket of type TYPE in domain DOMAIN, using
   protocol PROTOCOL. If PROTOCOL is zero, one is chosen automatically.
   Returns a file descriptor for the new socket, or -1 for errors. */
int socket (int domain, int type, int protocol);
```

Příklad:

```
#include <sys/types.h>
#include <sys/socket.h>

int sockfd;

sockfd=socket(AF_INET,SOCK_STREAM,0);
if (sockfd==-1)
    perror("Create_socket");
```

**connect()** pro navazování spojení. Používá klient.

```
/* Open a connection on socket SOCKFD to peer at ADDR (which LEN bytes long).
   For connectionless socket types, just set the default address to send to
   and the only address from which to accept transmissions.
   Return 0 on success, -1 for errors. */
int connect(int sockfd, struct sockaddr *addr, int len);
```

Příklad:

**bind()** naváže socket se jménem (vytvoří asociaci socketu a konkrétního portu). Nejčastěji se používá v kombinaci s *listen()* pro určení čísla portu na kterém server poslouchá. Používá server.

```
/* Give the socket SOCKFD the local address ADDR (which is ADDRLEN bytes long). */
int bind(int sockfd, struct sockaddr *addr, int addrlen);
```

Příklad:



```

#include <sys/socket.h>
#include <netinet/in.h>

struct sockaddr_in address;

/* type of socket created in socket() */
address.sin_family = AF_INET;
address.sin_addr.s_addr = INADDR_ANY;
/* 7000 is the port to use for connections */
address.sin_port = htons(7000);
/* bind the socket to the port specified above */
if(bind(socket_desc,(struct sockaddr *)&address,sizeof(address))== -1)
    perror("Bind");

```

**listen()** se volá po *bind()* a slouží k nastavení socketu do stavu čekání na příchozí spojení. *listen()* se okamžitě vrací. Vlastní čekání se děje až ve funkci *accept()*. Použití na server.

```

/* Prepare to accept connections on socket SOCKFD.
   N connection requests will be queued before further requests are refused.
   Returns 0 on success, -1 for errors. */
int listen(int sockfd, int backlog);

```

Příklad:

```

#include <sys/socket.h>

/* there can be up to 3 connections pending */
if(listen(socket_desc,3))
    perror("Do listen");

```

**accept()** čeká na příchozí spojení (když je socket v čekacím stavu). Funkce vrátí parametry prvního spojení ve frontě a nový socket, který slouží ke komunikaci s druhou stranou. Původní socket *sockfd* se tak může opět použít pro čekání na další spojení. Používá server.

```

/* Await a connection on socket SOCKFD.
   When a connection arrives, open a new socket to communicate with it,
   set *ADDR (which is *ADDRLEN bytes long) to the address of the connecting
   peer and *ADDRLEN to the address's actual length, and return the
   new socket's descriptor, or -1 for errors. */
int accept(int sockfd, void *addr, int *addrlen);

```

Příklad:

```

#include <sys/socket.h>

int addrlen; struct sockaddr_in address;
addrlen = sizeof(struct sockaddr_in);
new_socket = accept(socket_desc, (struct sockaddr *)&address, &addrlen);
if(new_socket < 0)
    perror("Accept connection");

```

**shutdown(), close()** Používá server i klient. Uzavření socketu je funkce *close()* a funkce *shutdown()* uzavře část plně duplexního spojení přes socket.

```

/* Shut down all or part of the connection open on socket SOCKFD.
   HOW determines what to shut down:
   SHUT_RD      = No more receptions;

```

```

    SHUT_WR    = No more transmissions;
    SHUT_RDWR = No more receptions or transmissions.
    Returns 0 on success, -1 for errors.  */
int shutdown (int sockfd, int how);

/* The following constants should be used for the second parameter of
   'shutdown'.  */
enum {
    SHUT_RD = 0,      /* No more receptions.  */
#define SHUT_RD    SHUT_RD
    SHUT_WR,      /* No more transmissions.  */
#define SHUT_WR    SHUT_WR
    SHUT_RDWR     /* No more receptions or transmissions.  */
#define SHUT_RDWR SHUT_RDWR
};

```

Příklad:

```

#include <unistd.h>

close(new_socket);

```

#### 4.1.4 Posílání zpráv do socketu

**send()** lze použít pouze pokud je soket ve stavu „připojen“

```

/* pouze pokud je socket spojen, neindikuje doruceni,
   flag=0 je ekvivalentni s write()
   */
int send(int s, const void *msg, int len, unsigned int flags);

/* available flags are: */
#define MSG_OOB      0x00001 /* process out-of-band data */
#define MSG_DONTROUTE 0x00004 /* bypass routing, use direct interface */
#define MSG_EOR      0x00008 /* data completes record */
#define MSG_EOF      0x00100 /* data completes transaction */
#define MSG_NOSIGNAL 0x20000 /* do not generate SIGPIPE on EOF */

#include <sys/socket.h>

char *message="This is a message to send\n\r";
send(new_socket,message,strlen(message),0);

```

**sendto()** může být použit v libovolném stavu socketu. Pošle data zadanému příjemci. Určeno pro nespojovanou komunikaci (není navazováno spojení).

```

int sendto(int s, const void *msg, size_t len, int flags,
           const struct sockaddr *to, socklen_t tolen);

```

**sendmsg()** může být použit v libovolném stavu socketu.

```

/* odeslani zpravy */
int sendmsg(int fd, const struct msghdr *msg, unsigned int flags);

```

#### 4.1.5 Příjem zpráv ze socketu

**recv()**

```
int recv(int s, void *msg, int len, int flags);
```

```
#include <sys/socket.h>
```

```
int bufsize=1024;          /* a 1K buffer */
char *buffer=malloc(bufsize);
recv(new_socket,buffer,bufsize,0);
```

**recvmsg()**

```
int recvmsg(int s, struct msghdr *msg, unsigned int flags);
```

**recvfrom()** přijme data nespojovaným způsobem.

```
int recvfrom(int s, void *buf, int len, unsigned int flags
             struct sockaddr *from, int *fromlen);
```

## 4.2 Algoritmy - nespojované služby

### 4.2.1 Server

1. vytvoří socket
2. navázání (bind) socketu na port
3. příjem/odeslání dat

```
sockfd=socket(...);
bind(sockfd, port);

while(1)
{
    recvfrom(sockfd, buffer, len, from, fromlen);
    ... zpracovani zadosti klienta ...
    ... priprava odpovedi serveru ...
    sendto(sockfd, buffer, len, flags, to, tolen);
}
```

### 4.2.2 Klient

1. vytvoří socket
2. navázání socketu na port
3. příjem/odeslání dat

```
sockfd=socket(...);
bind(sockfd, port);

while(1)
{
    ... priprava zadosti klienta ...
    sendto(sockfd, buffer, len, flags, to, tolen);
    recvfrom(sockfd, buffer, len, from, fromlen);
    ... zpracovani prijate zpravy ...
}
```

## 4.3 Algoritmy - spojované služby

### 4.3.1 Server

1. socket
2. bind
3. listen
4. accept
5. recv
6. send

### 4.3.2 Klient

1. socket
2. bind
3. connect
4. send
5. recv
6. close

## 4.4 Příklady realizace serverů

- <http://cs.baylor.edu/~donahoo/practical/C.Sockets/textcode.html>

## 4.5 Select

Jedná se o funkci, která umožňuje procesu čekat na několik událostí najednou. Např. server čeká na příchozí požadavky. Použitím jiné funkce (recv, send) by obsluha dalších událostí nebyla možná.

```
#include <sys/types.h>
#include <sys/socket.h>
```

```
int select(int nfd, fd_set *readfds, fd_set *writefds,
           fd_set *exceptfds, const struct timeval *timeout);
```

Select kontroluje stav socketů — zda některý není připravený pro čtení (*readfds*), pro zápis (*writefds*) nebo není ve výjimečném stavu (*exceptfds*). Argument *timeout* specifikuje čas do kdy musí být select dokončen. Pokud bude *timeout null pointer*, pak je čekání na nekonečně dlouhou dobu. Pokud je *timeout=0* pak select vrací hodnotu okamžitě.

První argument *nfd* určuje maximální počet socketů, které má kontrolovat. Měla by se do *nfd* uložit velikost největšího pole deskriptorů (maximum z *readfds*, *writefds*, *exceptfds*)

Jestliže select proběhne úspěšně, pak vrací počet připravených socket deskriptorů (file descriptor). Vrací 0 když časový limit vypršel dříve než byl některý socket vybrán. Pokud dojde k chybě vrací select -1.

Pokud kontrolovat některý typ file deskriptorů, pak stačí jako příslušný argument umístit *NULL*.

Po návratu z funkce select jsou všechny file descriptorů modifikovány tak aby bylo poznat, které deskriptory jsou připraveny. Připravenost je indikována makrem *FD\_ISSET*.

```
Initializes fdset to 0, representing the empty set.
FD_ZERO(& fdset)
```

```
Adds socket descriptor fd to fdset.
FD_SET(fd, &fdset)
```

Removes the socket descriptor `fd` from the socket descriptor set `fdset`.  
`FD_CLR(fd, &fdset)`

Returns nonzero if socket descriptor `fd` is a member of `fdset`. Otherwise, it returns a 0.  
`FD_ISSET(fd, &fdset)`

Použití funkce `select` je vidět na následujícím příkladu:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/time.h>
#include <stdio.h>

/* This function calls select to wait for data to read from */
/* one of the sockets passed as a parameter.                */
/* If more than 3 seconds elapses, it returns.                */
/* Return value flags. These indicate the readiness of        */
/* each socket for read.                                       */
#define S1READY 0x01
#define S2READY 0x02

waittoread(int s1,int s2)
{
    fd_set fds;
    struct timeval timeout;
    int rc, result;

    /* Set time limit. */
    timeout.tv_sec = 3;
    timeout.tv_usec = 0;

    /* Create a descriptor set containing our two sockets. */
    FD_ZERO(&fds);
    FD_SET(s1, &fds);
    FD_SET(s2, &fds);
    rc = select(sizeof(fds)*8, &fds, NULL, NULL, &timeout);
    if (rc== -1)
    {
        perror("select failed");
        return -1;
    }
    result = 0;
    if (rc > 0)
    {
        if (FD_ISSET(s1, &fds)) result |= S1READY;
        if (FD_ISSET(s2, &fds)) result |= S2READY;
    }
    return result;
}
```

## 4.6 Psaní vlastních programů pod Linuxem

### 4.7 Nástroj make

- <http://www.linux.cz/noviny/1999-0304/clanek12.html>
- <http://www.opussoftware.com/tutorial/TutMakefile.htm>
- <http://mrbook.org/tutorials/make/>

Účelem nástroje *make* je automatizace překlada celého nebo rekompilace částí programů. Make rozhodne jaké části musí být rekompilovány (změnily se) a které není nutné znovu kompilovat. Rozhodnutí co je třeba zkompileovat make určuje podle závislostí a podle toho zda výsledný soubor není starší než některá část zdrojových souborů. Pokud ano, provede překlad všech změněných částí i včetně vytvoření výsledného souboru.

Chcete-li nástroj make používat, musíte vytvořit soubor *Makefile* (nebo *makefile*), který definuje závislosti částí programu.

Použití je snadné. Pokud máte Makefile stačí spustit příkaz:

```
$ make
```

Všechny nutné překlady tak budou automaticky provedeny.

## 5 Síťování v Javě

Síťování se v Javě nachází v balíku *java.net*. Příklady lze nalézt například na:

- <http://www.exampledepot.com/egs/java.net/pkg.html>

### 5.1 InetAddress

Existuje třída *InetAddress*, která představuje obecnou IP adresu. Třída *InetAddress* je rodičovskou třídou pro *Inet4Address* (IPv4) a *Inet6Address* (IPv6). Obvykle si vystačíme s první zmíněnou (obecnější) třídou. Třídy nemají veřejný konstruktor a používají se statické metody:

- *getByAddress(byte[] addr)* vytvoří instanci IPv4 adresy nebo IPv6 (podle velikosti předaného pole - 4 nebo 16 hodnot). Nejvýznamější byte je jako první. Použijí se přímo předané hodnoty, neprovádí se převody na jméno.
- *getByName(String host)* kde *host* je jmenné označení nebo hodnota IP adresy v podobě textového řetězce. Pokud se jedná o jmenné označení, provede se volání funkce operačního systému (provedení DNS dotazu) pro zjištění IP adresy. Pokud se jedná o IP adresu, tak se přímo použije - obdoba metody *getByAddress(byte[] addr)*. Vrací instanci *Inet4Address* nebo *Inet6Address*.
- *getLocalHost()* vrací lokální adresu. Bohužel nemůžeme ovlivnit kterou, jestliže má počítač více rozhraní.

Získaná instance je neměnná. Může být typu *unicast* či *multicast*. Může být i typu *anylocal* (nespecifikovaná) a takovou adresu nelze použít pro určení cíle. Pouze říká že lze použít libovolnou místní adresu (přidělování lokálních adres) a tedy i libovolné rozhraní.

Třída dále umožňuje provádět reverzní DNS dotazy metodou *getHostName()*, *getCanonicalHostName()*. Lze ověřit dosažitelnost uzlu použitím metody *isReachable()*. Test dosažitelnosti se provádí protokolem ICMP a tedy pokud správce daného uzlu blokuje ICMP protokol, pak neuspějeme.

```
byte ipv4[] = {147, 228, 67, 101};
try {
    InetAddress a1 = InetAddress.getByAddress(ipv4);
    InetAddress a2 = InetAddress.getByName("147.228.67.101");
    InetAddress a3 = InetAddress.getByName("www.zcu.cz");
    InetAddress local = InetAddress.getLocalHost();

    System.out.println(a1.getCanonicalHostName());

    if (!a1.isReachable(5000)) { // 5 sec
        System.err.println("Stroj je nedostupny");
    }
} catch (UnknownHostException e) {
    e.printStackTrace();
}
```

## 5.2 SocketAddress

Jedná se o abstraktní třídu, která je v množství metod jako typ argumentu. Od té je zděděna třída *InetSocketAddress*, která obaluje IP adresu a číslo portu. Přímou umožňuje v konstruktoru různé způsoby inicializace objektu.

```
try {
    // nejprve vytvořime InetAddress (vyhazuje UnknownHostException)
    // ze jména a tu použijeme pro vytvoření InetSocketAddress.
    InetAddress a1 = InetAddress.getByName("www.zcu.cz");
    InetSocketAddress isa1 = new InetSocketAddress(a1, 80);

    // stejná funkce jako u předchozího postupu, rozdíl je
    // v neexistenci výjimky při selhání DNS
    InetSocketAddress isa2 = new InetSocketAddress("www.zcu.cz", 80);

    // adresa je neprelozena
    InetSocketAddress isa3 = InetSocketAddress.createUnresolved("www.zcu.cz", 80);

    // nespecifikovaná adresa - libovolná místní zdrojová adresa
    // specifikován je port
    InetSocketAddress isa4 = new InetSocketAddress(80);

    // adresu nedefinujeme a port je automaticky
    InetSocketAddress isa5 = new InetSocketAddress(0);
} catch (UnknownHostException e) {
    e.printStackTrace();
}
```

## 5.3 Komunikace protokolem UDP

### 5.3.1 Klient

Třída *DatagramPacket*, představuje pole bajtů připravené k odeslání datagramovou službou. Instance tohoto objektu obsahuje také cílovou adresu a číslo portu (nemusí být nastaveny).

```
String s = "Hello_world!";
byte b[];
```

```
try {
    b = s.getBytes("UTF-8");
} catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
```

```
DatagramPacket dp1 = new DatagramPacket(b, b.length);
```

Tímto máme vytvořený paket který můžeme protokolem UDP poslat. Další možností je vytvořit datagram který bude obsahovat i cílovou adresu a číslo portu.

```
DatagramPacket dp2 = null;
DatagramPacket dp3 = null;
try {
    dp2 = new DatagramPacket(b,
                             b.length,
                             InetAddress.getByName("www.zcu.cz"),
                             54321);
    dp3 = new DatagramPacket(b,
                             b.length,
```

```

        new InetSocketAddress("www.zcu.cz"),
        54321);
} catch (Exception e) {
    e.printStackTrace();
}

```

### 5.3.2 Server

Abychom byli schopni poslat datagram, potřebujeme příslušný socket. Ten vytvoříme použitím třídy *DatagramSocket*. Socket je specifikován IP adresou a číslem portu, které obsadí. Tyto hodnoty lze určit, nebo je nechat aby byly přiděleny automaticky.

```

try {
    DatagramSocket sock1 = new DatagramSocket();
    sock1.send(dp1); // FAIL - chybi cilova adresa a port (datagram/socket)
    sock1.send(dp2); // OK
} catch (Exception e) {
    e.printStackTrace();
}

```

Datagramový socket můžeme obrazně řečeno připojit (metoda *connect()*) k adrese příjemce. Odesílat datagramy pak lze pouze tomuto jednomu příjemci. K odeslání slouží metoda *send()*.

```

try {
    DatagramSocket sock2 = new DatagramSocket(55555);
    sock2.connect(InetAddress.getByName("www.zcu.com"), 44444);
    sock2.send(dp1); // OK
    sock2.send(dp2); // FAIL - neshoda adres
} catch (Exception e) {
    e.printStackTrace();
}

```

Na druhé straně by měl logicky být příjemce dat, která jsme právě odeslali. Vytvoříme stejný socket jako při odesílání datagramu a příjem provádíme metodou *receive()*. Volání je blokuující po dobu dokud nevyprší časový limit, pak je vyhozena výjimka *SocketTimeoutException*.

```

try {
    DatagramSocket sock = new DatagramSocket(54321);
    sock.setSoTimeout(600000); // 10 min

    DatagramPacket dp = new DatagramPacket(new byte[1000], 1000);

    while (true) {
        sock.receive(dp);
        sock.send(dp);
    }
} catch (Exception e) {
    e.printStackTrace();
}

```

Socket je tedy vytvořen na portu 54321, s časovým limitem deset minut. Datagram bude mít buffer o velikosti 1000 B. Pokud se data do bufferu nevejdou, pak budou oříznuta.

## 5.4 Komunikace protokolem TCP

### 5.4.1 Klient

Spojová komunikace v Javě funguje stejně jako při práci se soubory. Získáme streamy a jejich použitím provádíme přenos požadovaných dat. Stejně jako v předchozích příkladech potřebujeme vytvořit socket. Ten je reprezentován třídou *Socket*.



```
// vytvoření socketu
Socket sock = new Socket();

// provedeme připojení
sock.connect("www.zcu.cz", 80);

// zavření socketu
sock.close();
```

Další možností je vytvořit přímo připojený socket použitím jiného konstruktoru:

```
Socket sock = new Socket("www.zcu.cz", 80);
```

Komunikace pak probíhá v podobě streamu a tedy stream ze socketu získáme následujícím způsobem.

```
// buffer pro čtení streamu
BufferedReader br = new BufferedReader(
    new InputStreamReader(sock.getInputStream()));
// buffer pro zápis streamu
BufferedWriter bw = new BufferedWriter(
    new OutputStreamWriter(sock.getOutputStream()));

// zapiseme připravený požadavek
bw.write(...požadavek...);
// odeslání z bufferu
bw.flush();

String line = "";
while (line != null) {
    line = br.readLine();
    if (line != null)
        System.out.println(line);
}

// zavřeme socket
sock.close();
```

### 5.4.2 Server

Pokud chceme vytvořit serverovou část, pak použijeme instanci třídy *ServerSocket*. Postup je podobný jako pro sockety v jazyce C. Čekáme na příchozí spojení a použitím metody *accept()* nasloucháme na socketu. Jestliže se klient připojí k našemu socketu, pak metoda *accept()* vrací instanci třídy *Socket*. Z touto instancí se pracuje stejně jako bylo ukázáno výše u klientské části (získáme z ní vstupní a výstupní stream).

```
ServerSocket serverSocket = new ServerSocket(54321);
while (true) {
    final Socket sock = serverSocket.accept();
    Thread t = new Thread() {
        public void run() {
            try {
                InputStream is = sock.getInputStream();
                OutputStream os = sock.getOutputStream();
                ...
                sock.close();
            } catch (IOException e) {
            }
        }
    };
    t.setDaemon(true);
```

```

    t.start();
}

```

## 5.5 Protokol HTTP

Protokol HTTP je velmi oblíbeným protokolem a Java má i pro tento protokol připravené třídy. Jsou to především *URLConnection* a *HttpsURLConnection*, což jsou abstraktní třídy a instanci získáme metodou *openConnection()* nad instancí třídy *URL* a potřebným přetypováním.

```

URL url = new URL("http://www.zcu.cz/");
URLConnection con = (URLConnection) url.openConnection();

```

Z již existující instance *URLConnection* získáme stream a můžeme získat data která nám server zaslal.

```

System.out.println("ResponseCode:␣" + con.getResponseCode());
System.out.println("ContentType:␣" + con.getContentType());

```

```

// získáme buffer pro čtení
BufferedReader br = new BufferedReader(
    new InputStreamReader(con.getInputStream()));

```

```

// čtení dat
String s = "";
while (s != null) {
    s = br.readLine();
    if (s != null) {
        System.out.println(s);
    }
}

```

```

// odpojení
con.disconnect();

```

## 6 Přenosový kanál

### 6.1 Zadání semestrálních prací

Viz cvičení.

### 6.2 Přenosy dat

Přenosy dat se liší podle toho jak jsou přenášena *digitální data* prostřednictvím *analogového signálu*, který *přenosová cesta* v reálu přenáší.

Přenosová cesta (prenosový kanál) je prostředí (prenosové médium) s jehož využitím provádíme přenos dat. Takovou cestou může být telefonní linka, sériová linka, koaxiální kabel, ethernetový kabel, vzduch (WiFi), optické vlákno, apod.

#### 6.2.1 Reálné vlastnosti

Všechny přenosové cesty nejsou vhodné pro přenos všech typů signálů. Je to proto že přenosové cesty nejsou nikdy ideální. Jsou vždy nějakým způsobem ovlivněny signály, které po nich přenášíme. Mezi nejčastější typy ovlivnění signálu patří:

- přeslech (signály z okolních vedení se prolínají s naším signálem)
- útlum (zeslabení signálu)
- zkreslení (deformace signálu)

## 6.3 Šířka přenosového pásma

Přenosová cesta je většinou schopna lépe přenášet *signály o frekvenci z určitého omezeného intervalu*. Signály s jinou frekvencí naopak přenáší špatně (velký útlum, zkreslení, ...).

Šířkou pásma (bandwidth) je *šířka intervalu frekvencí*, které je přenosový kanál schopen přenést s rozumným zkreslením (pokažením). Jednotka šířky pásma je stejná jako jednotka frekvence 1 Hz.

Pro telefonní okruhy je (uměle) vymezeno rozmezí frekvencí od 300 Hz do 3400 Hz, šířka pásma tedy je 3100 Hz (3,1 kHz).

### 6.3.1 Harmonický signál

Jestliže máme v ideálním případě harmonický (sinusový) signál, pak frekvence které odpovídají šířce pásma se přenesou. Ostatní frekvence mimo rozsah neprojdou.

### 6.3.2 Obecný průběh signálu

Pro obecný průběh signálu je situace obtížnější. Pokud pro představu využijeme znalosti že lze libovolný signál rozložit (Fourier) na signály harmonického průběhu a různé frekvence. Pak je situace převedena na předchozí případ. Přenosovou cestou projdou ty části, které odpovídají šířce přenosového pásma a vyšší složky signálu neprojdou.

### 6.3.3 Analogový a digitální přenos

- analogový
  - měříme okamžitou hodnotu (napětí, proud)
  - všechny přenosy v reálném světě jsou analogové
  - nikdy nedosáhneme ideálního přenosu (ztráty, útlum, apod.)
- digitální
  - zjišťujeme do jakého intervalu měřená veličina patří (rozlišení logické 0 a 1)
  - digitální přenos je ideální (hodnoty pro 0 a 1 jsou jasně definovány)
  - odlišení analogových a digitálních dat je pouze věcí interpretace hodnot

## 6.4 Přenos v základním pásmu

Při přenosu v základním pásmu (baseband transmission) se přenášený signál mění přímo podle přenášených dat. Nejedná se o harmonický signál.

Příkladem mohou být impulsy napětí nebo proudu s obdélníkovým průběhem. V takovém případě se projeví vliv omezené šířky přenosového pásma a obdélníkový signál bude značně zkreslený! Čím bude šířka pásma větší tím bude i méně zkreslený signál tohoto obdélníkového průběhu.

Možností jak se vyvarovat zkreslení tohoto typu je použít přenos v přeloženém pásmu.

Z předchozího příkladu je také poznat že pokud budeme mít *větší šířku přenosového pásma, umožní nám přenést více „dat“*, tj. přenesený signál bude přesnější.

Přenos v základním pásmu nemusí být možné v některých případech použít vůbec (transformátory nepřenesou stejnosměrnou složku). Naopak přenos v základním pásmu se používá v sítích LAN na koaxiálním i twist kabelu (Ethernet 10Base2, 10BaseT).

### 6.4.1 Dvou-úrovňové kódování

Máme k dispozici dvě úrovně signálu a tedy můžeme zakódovat stavy 0 a 1 (1 bitovou informaci).

### 6.4.2 Více-úrovňové kódování

Signál rozdělíme na více úrovní (např. 0 V, 1 V, 2 V, 3 V) a těm přidělíme více stavů (00, 01, 10, 11). Zakódujeme dvou bitovou informaci v jediném stavu (úroveň signálu).

Můžeme libovolně jiné značení stavů, stejně tak i mnohem více úrovní signálu. Omezujícím faktorem je aby přijímací strana byla vůbec schopna dané stavy rozlišit.

## 6.5 Přenos v přeloženém pásmu

Podstatou přenosu v přeloženém pásmu (broadband transmission) je přenášet signál, který přenosovou cestou prochází nejlépe. Nejlépe obvykle prochází harmonický (sinusový) signál. Nejhorší na přenos je signál obdélníkový a toho se chceme touto metodou vyvarovat.

### 6.5.1 Nosný signál

Signál, který použijeme jako základ je harmonického průběhu a přenosovou cestou prochází nejlépe (vhodný signál pro přenosovou šířku pásma). Takový harmonický signál označujeme pojmem *nosný signál* (carrier).

### 6.5.2 Modulace

Podle dat, která chceme přenášet, měníme některou z charakteristik nosného signálu (amplituda, frekvence nebo fáze). Této změně říkáme *modulace* signálu. Na přijímací straně je pak prováděn opak — *demodulace*.

Modulací vzniká z analogového nosného a digitálního signálu opět signál analogový. Je nutné rozlišit potřebný počet navzájem odlišných stavů, které mohou reprezentovat právě diskretní logické hodnoty (digitální signál).

### 6.5.3 Typy modulací

$$y = A \cdot \sin(\omega \cdot t + \Phi)$$

**amplitudová** (amplitude modulation, AM)

různá hodnota amplitudy harmonického signálu

**frekvenční** (frequency modulation, FM)

různá frekvence harmonického signálu

**fázová** (phase modulation, PM)

různá fáze (posunutí) harmonického signálu

**kombinace předchozích**

Při modulaci kdy vznikají jen dva navzájem rozlišitelné stavy nosného signálu (např. fázová modulace posunutím signálu o  $0^\circ$  a o  $180^\circ$ ) jde o dvoustavovou modulaci. Dva rozlišitelné stavy nosného signálu mohou reprezentovat dvě diskretní logické hodnoty. Dvě logické hodnoty (0 a 1) jsou pak jednobitovou informací (1b).

Většího počtu rozlišitelných stavů nosného signálu signálu docílíme jinou modulací. Např. čtyřstavová fázová modulace s posunutím fáze nosného signálu o 0, 90, 180 a 270 stupňů může jeden stav nosného signálu reprezentovat jednu ze čtyř možných logických hodnot (00, 01, 10, 11) a tedy nést dvoubitovou informaci (2b).

Způsoby modulace lze také kombinovat (více-bitová informace).

## 6.6 Modulační rychlost

Modulační rychlost (modulation speed) vyjadřuje počet změn nosného signálu za jednotku času (sekundu), a měří se v *Baudech* (zkratkou *Bd*).

Modulační rychlost neříká nic o tom, jaké množství informace nosný signál přenáší (viz přenosová rychlost).

Modulační rychlost nelze libovolně zvyšovat, protože příjemce už by nemusel být schopen jednotlivé stavy od sebe odlišit.

### 6.6.1 Přenos v základním pásmu

Udává jak rychle lze měnit přenášený signál (digitální).

### 6.6.2 Přenos v přeloženém pásmu

Jak rychle lze modulovat nosný signál podle digitálních dat.

### 6.6.3 Vzorkování přijímaného signálu

Aby příjemce měl možnost získat z přenášeného signálu veškerou informaci musí vzorkovat jeho průběh dvojnásobnou frekvencí (2x za každou periodu).

$$\max(v_{\text{modulace}}) = 2 \cdot H$$

Pomalejší změny by nedokázaly využít možností dostupné šířky pásma  $H$ . Rychlejší změny nepřinesou žádnou další informaci navíc, proto je zbytečné vzorkovat s vyšší frekvencí.

*Maximální modulační rychlost* (počet změn nosného signálu za jednotku času) je číselně dvojnásobkem šířky pásma.

## 6.7 Přenosová rychlost

Přenosová rychlost (transmission speed) udává objem informace, přenesený za časovou jednotku. Vyjadřuje se v *bitech za sekundu* (bits per second, zkratkou *bps*).

Přenosová rychlost neříká nic o tom, jak rychle se mění nosný signál.

Platí že čím větší šířka pásma přenosového kanálu, tím větší přenosové rychlosti lze dosáhnout na tomto kanálu.

- *maximální dosažitelná přenosová rychlost* je číselně přímo úměrná šířce pásma, ale konstanta úměrnosti je závislá na „kvalitě“ přenášeného signálu (odstup užitečného signálu od šumu).

### 6.7.1 Nyquistovo kritérium

Nyquistova věta: „Signál, který neobsahuje frekvence vyšší než  $H$  může být plně zrekonstruován ze vzorků snímaných s frekvencí  $2H$  (dvojnásobnou).“

Pro kanál s maximální přenášenou frekvencí  $H$  a v případě že rozlišujeme  $V$  diskrétních úrovní signálu můžeme přenést maximální bitový tok:

$$C = 2 \cdot H \cdot \log_2 V$$

**C** přenosová rychlost [b/s]

**H** šířka pásma [Hz]

**V** počet úrovní signálu [-]

### 6.7.2 Shanonovo kritérium

Jestliže počítáme s kanálem se šumem pak platí Shanonova věta pro maximální bitový tok:

$$C = H \cdot \log_2 \left(1 + \frac{S}{N}\right)$$

Pro telefonní kanál uvedený výše s šířkou pásma 3100 Hz a pokud budeme počítat odstup signálu od šumu 30 dB (tj. 1000/1). Po dosazení do vzorce z Shannonovy věty dostaneme výsledek

$$\maxbps = 3100 \cdot \log_2(1 + 1000/1) = 30,9[kbps]$$

## 6.8 Modulační a přenosová rychlost

Mezi modulační a přenosovou rychlostí platí obecný vztah:

$$v_{\text{prenosova}} = v_{\text{modulacni}} \cdot \log_2(n)$$

kde  $n$  je počet stavů přenášeného signálu.

Modulační rychlost může být rovna rychlosti přenosové v případě dvoustavové modulace. Jestliže použijeme modulaci čtyřstavovou, vyjadřuje jeden stav nosného signálu dvoubitovou informaci a přenosová rychlost je pak číselně dvojnásobná oproti rychlosti modulační.

Přenosová cesta	Přenosová rychlost	Modulační rychlost	Počet stavů
Ethernet	10 Mbps		
modem V.22 bis	2400 bps	600 Bd	16
modem V.32	9600 bps	2400 Bd	16
modem V.32 bis	14400 bps	2400 Bd	64
modem V.34	28800 bps	2400-3200 Bd	512
RS-232	x	x	

## 6.9 Přenos

### 6.9.1 Zapojení vodičů

**paralelní** velký počet vodičů, vhodné pro malé vzdálenosti, vysoká rychlost

**sériový** malý počet vodičů, N krát pomalejší než paralelní

### 6.9.2 Směr přenosu

**simplexní** jedním směrem

**duplexní** přenos oběma směry souběžně (TCP je duplexní protokol transportní úrovně)

**poloduplexní** přenos oběma směry ale střídavě, vždy se přenáší pouze jeden směr (Ethernet je poloduplexní protokol linkové úrovně)

### 6.9.3 Způsob přenosu

Řídící signál používáme k určení okamžiku vzorkování. Způsob přenosu přes přenosový kanál může být:

**synchronní** přesně dané (ekvidistantní) okamžiky vzorkování

**asynchronní** různé okamžiky mezi vzorkováním

**arytmický** přenos je kombinace předchozích; přenos je rozdělen do skupin, bity ve skupině jsou přenášeny synchronně a skupiny asynchronně.

### 6.9.4 Počet úrovní

Jedna úroveň není možná.

- dvouúrovňový
- mnohaúrovňový ( $2^N$ )

### 6.9.5 Typy spojů

- dvoubodové (jeden vysílač, jeden přijímač; jasné impedanční poměry)
- mnohabodové (jeden vysílač, více přijímačů ale i více přijímačů)

### 6.9.6 Přenosové vedení

- symetrické
- nesymetrické

## 6.10 Kapacita přenosového kanálu

Zajímá nás využití přenosového kanálu  $\eta = \frac{N}{M}$ , kde N je počet datových bitů a M celkový počet bitů.

Pro příklad kdy máme 8 b zprávu a přenášíme ji jako 12 b zprávu (včetně pomocných bitů) pak využití kapacity přenosového kanálu bude pouze:

$$\eta = \frac{N}{M} = \frac{8}{12} \doteq 66\%$$

## 7 Přenos dat

- synchronní přenos
- problém synchronizace (synchronní a asynchronní systémy),
- rámce,
- hranice rámců,
- transparentnost přenosu,
- tvary rámců (s délkou, vkládání slabik, vkládání bitů),
- multiplexování — časový a frekvenční multiplex, synchronní a asynchronní multiplex,
- síť s přepínáním kanálů, zpráv a paketů.

### 7.1 Synchronizace

- bity – synchronizace na úrovni bitů (rozlišování jednotlivých bitů). Speciální oddělovací značky (hodně režie) START/STOP bity
- byty – synchronizace na úrovni bytů (znaků), určování hranic jednotlivých bytů (znaků), START/STOP bity
- rámce – synchronizace na úrovni rámců, určování hranic jednotlivých rámců START/STOP znaky (STX, ETX)

<http://www.earchiv.cz/b05/b1100001.php3>

### 7.2 Synchronní přenos

U synchronního přenosu je ekvidistantní vzorkování. Není třeba přenášek řídicí signál? Problém je s fází hodin, protože ta se mění. Fáze mezi vysílaným přijímaným signálem závisí na vzdálenosti. Tento problém řešíme přidáním fázového modulu (může měnit fází).

Problém se synchronizací hodin nemusíme řešit dalším vodičem. Můžeme hodiny synchronizovat na základě změn přijímaného signálu (na začátku bitového intervalu).

$$v = 2 \cdot 10^8 m/s$$

$$s = 1m$$

$$t = \frac{s}{v} = \frac{1}{2 \cdot 10^8} = 0,5 \cdot 10^{-8} = 5 \cdot 10^{-9} s/m = 5ns/m$$

Při synchronním přenosu mezi vysílající a přijímající stranou existuje synchronizace po celou dobu, hodiny jsou zakódovány do přenášených dat (NRZ, diferenciální manchester, ...).

### 7.3 Arytmický přenos

Mezi vysílající a přijímací stranou existuje synchronizace na začátku a na konci přenosu bloku bitů (START/STOP bity). Délka přenosu znaku je pevná, délka přenosu bloku proměnlivá - běžně označován jako asynchronní.

#### 7.3.1 Start a stop bity

Určují začátek a konec přenášeného intervalu bitů. Protože při použití hodin (řídicího signálu) by mohl signál přijít mimo a nebyl by rozpoznán.

Z tohoto důvodu jsou přenášené bity opatřeny start a stop bity, které určí začátek a konec rámce.

Start bit - aktivní úroveň.

Stop bit - neaktivní úroveň.

### 7.3.2 Paritní bit

Paritní bit slouží ke kontrole přenosu.

- Žádná (None) - paritní bit není posílán
- Sudá (Even) - sudý počet jedniček
- Lichá (Odd) - lichý počet jedniček
- 1 (mark) - paritní bit má vždy hodnotu 1
- 0 (space) - paritní bit má vždy hodnotu 0

Parita *mark* a *space* není vhodná pro detekci chyb. Lze ji použít pro 9 bitovou komunikaci prostřednictvím obvodu, který umožňuje maximálně 8 bitovou komunikaci.

### 7.3.3 Označení

Zavádíme značení, kterým popisujeme jaké množství bitů je přenášeno (n), jaká parita se používá (p) a kolik stop bitů je na konci (s). Značka má podobu „nps“, např. *8N1*, *8E2*.

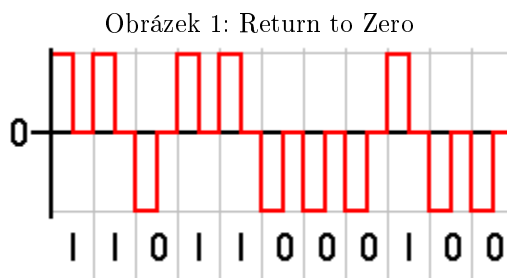
## 7.4 Asynchronní přenos

Mezi vysílající a přijímací stranou neexistuje žádná synchronizace. Používají se speciální značky. Přenos jednoho bitu může trvat libovolně dlouhou dobu (potřebujeme 3 úrovně signálu) - nepoužívá se

## 7.5 Kódování signálu

### 7.5.1 Return to Zero (RZ)

Název kódování *RZ*<sup>2</sup> nebo také kódování s návratem k nule. Při tomto kódování jsou nuly „0“ a jedničky „1“ v nejjednodušším případě reprezentovány kladnými a zápornými pulsy. Přitom je důležité, že po každém pulsu dochází k návratu do neutrální hodnoty (nulové napětí). Díky těmto návratům je možná synchronizace hodin odesílatele a příjemce, není nutné používat samostatný hodinový signál. Zároveň je však toto kódování náročnější na přenosovou kapacitu.



### 7.5.2 Return to Zero Inverted (RZI)

Signál RZI má pouze 2 úrovně. Nula „0“ je reprezentována pulsem, který je kratší než hodinový signál. Jednička „1“ je reprezentována absencí pulsu. Toto kódování se používá u protokolu IrDA.

<sup>2</sup>třístavový, polovina intervalu +1 při bitu 1, -1 při bitu 0, druhá polovina intervalu nulová.



### 7.5.3 Non Return To Zero (NRZ)

Název kódování NRZ<sup>3</sup> pochází z anglického (v překladu znamená bez návratu k nule). V tomto kódování je jednička „1“ reprezentována konkrétní význačnou hodnotou (například kladným napětím). Nula „0“ je reprezentována jinou význačnou hodnotou (například záporným napětím). Žádné další hodnoty se ve výsledném (nezašuměném) signálu nevyskytují, neexistuje zde třetí neutrální hodnota (například nulové napětí) jako je tomu u kódování s návratem k nule.

Kvůli absenci neutrální hodnoty nelze toto kódování v základním tvaru použít pro synchronní přenosy, je potřeba přidat synchronizaci například v podobě RLL (run length limited) nebo přidavného signálu hodin.

Existují další varianty NRZ:

- Unipolární NRZ: hodnota „1“ je reprezentována například kladným napětím, hodnota „0“ je reprezentována menším kladným napětím
- Bipolární NRZ: hodnota „1“ je reprezentována například záporným napětím, hodnota 0 kladným napětím. Například u RS-232 rozsah -5 V až -12 V znamená „1“, +5 V až +12 V znamená „0“.
- NRZ „Mark“: hodnota „1“ je reprezentována změnou, hodnota „0“ je pokud změna nenastává. K přechodu dochází na sestupné hraně hodinového signálu pro daný bit.
- NRZ „Space“: hodnota „0“ je reprezentována změnou, hodnota „1“ je pokud změna nenastává. Podobně jako NRZ „Mark“, pouze je prohozena reprezentace nul „0“ a jedniček „1“. K přechodu dochází na sestupné hraně hodinového signálu pro daný bit.

### 7.5.4 Non Return To Zero Inverted (NRZI)

Invertované NRZ (NRZI<sup>4</sup>): hodnota „1“ je reprezentována změnou, hodnota „0“ je pokud změna nenastává. K přechodu dochází na vzestupné hraně hodinového signálu pro daný bit. Tato varianta je použita v protokolu USB. Opět existuje i varianta s prohozenou reprezentací nul a jedniček.

### 7.5.5 Manchester

Kódování Manchester<sup>5</sup> (PSK) je způsob zakódování dat, který se využívá pro přenos dat počítačovou sítí na fyzické vrstvě ISO/OSI modelu, např. v Ethernetu či Token Ringu.

V případě synchronního přenosu dat mezi odesílatelem a příjemcem je nutný synchronizační signál. Manchesterský kód spojuje původní datový signál se synchronizačním signálem a tedy umožňuje synchronní komunikaci.

Pro vyjádření hodnoty bitu se do poloviny bitového intervalu původního signálu vloží hrana - změna signálu. Pokud signál v této hraně přechází z vysoké úrovně na nízkou úroveň, pak vyjadřuje hrana hodnotu bitu 1. Pokud signál přechází z nízké úrovně na vysokou úroveň, hodnota bitu bude 0.

Protože se hrana vždy nachází uprostřed každého bitového intervalu, může snadno sloužit k synchronizaci.

### 7.5.6 Diferenciální Manchester

Diferenciální kódování Manchester (anglicky Differential Manchester Coding, DPSK<sup>6</sup>) je jedním ze způsobů kódování používaném při synchronním přenosu. Hodinový signál potřebný pro synchronní přenos je přitom začleněn přímo do přenášených dat. Tím pádem dochází k průběžné synchronizaci odesílatele a příjemce a zároveň není potřeba vést hodinový signál po samostatné lince. V názvu je použito slovo diferenciální, neboť narozdíl od klasického manchesterského kódování jsou jednotlivé bity (0 nebo 1) kódovány přítomností nebo absencí přechodu.

Mezi hlavní výhody tohoto kódování patří:

- U zašuměného signálu je detekce přechodů méně náchylná na chyby než srovnání aktuální hodnoty signálu s krajními mezemi
- Důležitý je pouze přechod samotný, nikoli jeho směr. Toto kódování tedy funguje správně i pokud se změní polarita signálu - například prohozením linek spojení.

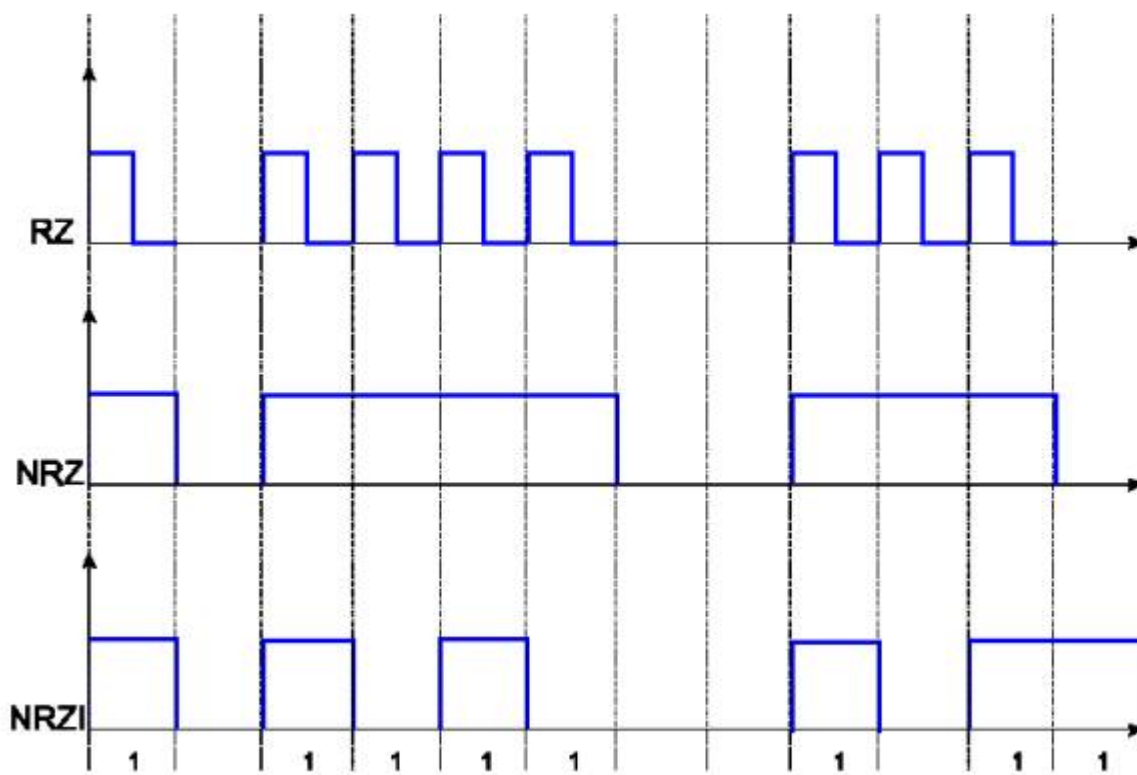
<sup>3</sup>úroveň signálu přímo odpovídá jedničce/nule

<sup>4</sup>1-inverze signálu, 0-úroveň zůstává

<sup>5</sup>fázová modulace, uprostřed intervalu: 0-sestup signálu, 1-vzestup signálu. Každý bitový interval má tedy uprostřed změnu. Dvojnásobné pásmo oproti přímému kódování. Použití v Ethernetu.

<sup>6</sup>1-změna na začátku intervalu, 0-absence změny na začátku intervalu. Uprostřed intervalu změna vždy. Někdy se kóduje změna/zachování úrovně posledního bitu (ne hodnota aktuálního bitu). Použití v Token-Ring.

Obrázek 2: NRZI



Mějme původní signál rozdělený na intervaly, ve kterých nabývá hodnot 0 nebo 1. Bit 1 se zakóduje tak, že první polovina intervalu je stejná jako druhá polovina předchozího intervalu (na začátku intervalu tedy nedochází k přechodu). Bit 0 se zakóduje tak, že první polovina intervalu je opačná než druhá polovina předchozího intervalu (na začátku intervalu dochází k přechodu). Uprostřed intervalu dochází vždy k přechodu, nezávisle na tom, zda se jedná o bit 0 nebo 1. Je také možné prohodit způsob kódování 0 a 1, oba způsoby jsou rovnocenné.

Z hlediska využití přenosové cesty není toto kódování příliš šetrné, neboť na každý přenášený bit (přechod signálu) připadá jeden synchronizační přechod. Je tedy zapotřebí dvojnásobné přenosové kapacity.

Diferenciální kódování Manchester je popsáno ve standardu IEEE 802.5 pro síť Token Ring. Využívá se také v jiných případech, například v magnetických nebo optických záznamových médiích.

## 7.6 Vkládání bitů (bit stuffing)

Počet změn (modulační rychlost) je dvojnásobný oproti tomu co potřebujeme využít pro přenos při synchronizaci kdy pro každý datový bit je vynocena dvojitá změna signálu.

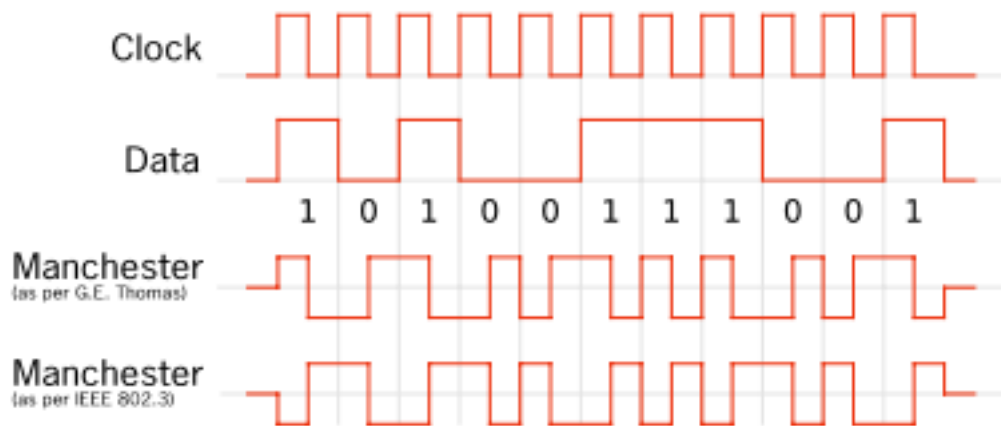
Efektivnější způsob je aby se „jedna změna navíc“, nutná k zajištění synchronizace, nepřidávala ke každému jednotlivému datovému bitu. Jinak máme 100 % režii. Přidáme tedy tuto změnu na víc pouze skupince několika datových bitů. Řešením je umělé zařazení „pomocného bitu“ těsně před možnou ztrátou synchronizace.

Budeme-li uvažovat že hodiny příjemce nevydrží synchronizované více než 5 bitových intervalů. Musíme po každé posloupnosti 5 stejných bitů vložit do přenášeného toku dat jeden opačný bit.

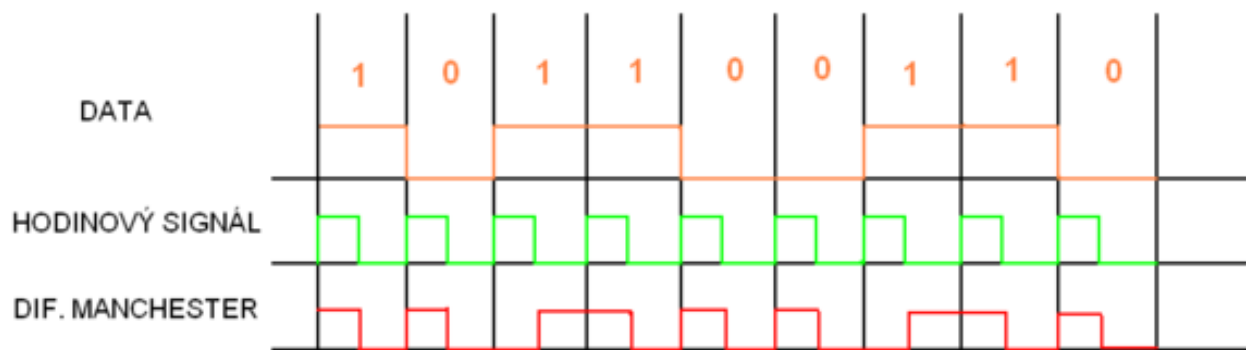
## 7.7 Multiplexování

Multiplexování (<http://www.earchiv.cz/b05/b1000001.php3>) je rozdělení jednoho přenosového kanálu na více samostatně využitelných přenosových kanálů. Rozdělení lze dosáhnout více způsoby (analogovými i digitálními).

Obrázek 3: Kódování Manchester



Obrázek 4: Diferenciální Manchester



### 7.7.1 Frekvenční multiplex

Analogovou metodou multiplexu je *frekvenční multiplex* (anglicky: FDM, Frequency Division Multiplexing). Analogový signál na každém ze vstupních kanálů je posunut do jiného rozsahu frekvencí tak, aby se žádné vzájemně nepřekrývaly. Takto frekvenčně posunuté signály sloučíme do jednoho signálu širšího rozsahu frekvencí. Tento signál, který vznikne sloučením, přeneseme přenosovým kanálem. U příjemce se provede obrácený postup. Jednotlivé signály se zase oddělí do samostatných frekvenčních oblastí a vrátí s do původní frekvenční polohy.

Vzájemné slučování a následné oddělování signálů nejsou nikdy ideální, a vždy určitým způsobem znehodnocují přenášený signál.

### 7.7.2 Vlnový multiplex

Vlnový multiplex (WDM, Wavelength Division Multiplexing) je další variantou analogového multiplexu.

Založen je na přenosu více svazků světla (různé frekvence) přes jedno optické vlákno. Každý svazek světla (o dané vlnové délce - frekvenci) se choval jako jeden přenosový kanál. Podoba s frekvenčním multiplexem není náhodná.

### 7.7.3 Časový multiplex

Časový multiplex (anglicky: TDM, Time Division Multiplexing) je představitelem digitálního způsobu multiplexování.

Přenosový kanál rozdělujeme v čase. Vždy na krátký časový úsek označovaný jako *časový slot* nebo anglicky *timeslot*, přidělí multiplex celý přenosový kanál jednomu vstupnímu kanálu. Multiplex postupně střídá všechny

vstupy a vždy jim přiřadí celý kanál na určité časové období.

#### 7.7.4 Statistický multiplex

Jedná se o modifikaci časového multiplexu. Pro případ kdy se mění objemy dat procházející přes jednotlivé vstupy. Když není dost dat k přenesení a je použito časového multiplexu, pak zůstává nevyužitá část přenosové kapacity (v době kdy je kanál přidělen vstupu, který nemá data). Časový multiplex neumožňuje přidělit přenosový kanál jinému vstupu.

Popsaný problém řeší statistický multiplex (STDM, Statistical Time Division Multiplexing).

Nevýhodou je, že přenášená data obsahují informaci o tom, který vstupní kanál data poslal (režijní data). Je to z důvodu, že není stanoveno pevné pořadí vstupů multiplexu do výstupního přenosového kanálu.

#### 7.7.5 Synchronní multiplex

Stanice jsou očíslovány, každá zná adresu svého následníka. Využívají časového rozdělení kanálu do krátkých úseků. Každá stanice může vysílat čas  $\frac{1}{n}$ , kde  $n$  je počet stanic. Když vyčerpá limit, vyzve další stanici k vysílání. Je nutné brát ohled na počet stanic a na vzdálenost mezi nimi. Potřeba časové synchronizace všech stanic.

Dochází zde k velkému spoždění i při malém počtu stanic.

### 7.8 Síť s přepínáním kanálů, paketů a zpráv

#### 7.8.1 Přepínání kanálů

- existuje kanál mezi 2 body a všechna data tečou jím (virtuální kanál)
- vytvoření kanálu před navázáním spojení (nastavení přepínačů v uzlech sítě)
- kanál se chová jako přímý spoj (např. veřejná telefonní síť, ATM, Frame Relay)

#### 7.8.2 Přepínání paketů

- neexistuje pevný kanál - o cestě každého paketu se rozhoduje zvlášť (na přepínačích)
- přepínání na různých úrovních:
  - linková (přepínání rámců)
  - síťová (přepínání paketů, tj. routování)

#### 7.8.3 Přepínání zpráv

- speciální případ přepínání paketů (předchůdce)
- přepínání jen mezi 2 body současně
- např. e-mail

## 8 Chyby při přenosu

[http://notorola.sh.cvut.cz/~bruxy/nlp\\_test3\\_kody.pdf](http://notorola.sh.cvut.cz/~bruxy/nlp_test3_kody.pdf)

Při přenosu dat může dojít k chybám (zkreslení signálu, chybné rozlišení úrovně signálu, rušení, šum, atd.) a výsledkem budou chybně přečtená data u příjemce.

Možné komplikace způsobené chybami se pokoušíme řešit použitím bezpečnostních kódů.

## 8.1 Chybovost

Uvažujeme pro symetrický (1 nebo 0 se přenáší se stejnou pravděpodobností) binární (přenáší se 1 nebo 0) přenosový kanál bez paměti (nezáleží na tom, co se přeneslo v předešlém kroku).

- Pravděpodobnost bezchybného přenosu jednoho bitu je  $P_1 = p_1$ .
- Pravděpodobnost bezchybného přenosu  $N$  bitů bitu je  $P_N = p_1^N$ .

Př.: Mějme dán symetrický binární přenosový kanál bez paměti. Chceme zjistit kolik bitů můžeme přenést ( $N$ ), aby pravděpodobnost jejich bezchybného přenosu byla  $p_N = 0,9$ , přičemž známe pravděpodobnost přenosu jednoho bitu ( $p_1$ ).

$$N = ?, p_N = 0,9$$

$$0,9 = p^N$$

$$\ln(0,9) = N \cdot \ln(p_1)$$

$$N = \frac{\ln(0,9)}{\ln(p_1)}$$

## 8.2 Bezpečnostní kódy

Princip *bezpečnostních kódů* spočívá v přidání bitů navíc nebo úpravě přenášených dat nějakou konkrétní metodou, tak abychom byli schopni u příjemce rozhodnout zda jsou data v pořádku.

Typy bezpečnostních kódů:

**detekční** umožní rozhodnout (detekovat) zda jsou data přijata správně,

**samoopravné** dokáží chybu detekovat a umožní i odhalenou chybu opravit bez nutnosti opakovaného přenosu dat.

Bezpečnostní kódy vždy k původní zprávě přidávají bity navíc, na jejichž základě jsou založeny detekční i samoopravné metody. Čím více bitů pro účely detekce a opravy chyb máme, tím účinnější metody jsou.

## 8.3 Parita

### 8.3.1 Sudá/lichá parita

Metoda založená na přidání paritního bitu. Rozlišujeme sudou a lichou paritu.

Při sudé paritě je paritní bit 0, pokud zpráva má sudý počet bitů. Paritní bit bude mít 1 jestliže zpráva, kterou zabezpečujeme, má lichý počet bitů. Tím zajistíme že vždy musí být sudý počet 1. Máme tedy možnost detekovat chybu v jednom bitu zprávy, ale nedokážeme ji opravit (nevíme ve kterém bitu je chyba).

Lichá parita je analogická k sudé, jen zajišťujeme aby počet 1 ve zprávě byl lichý.

### 8.3.2 Příčná parita

Ke každému zabezpečovanému slovu je přidán jeden paritní bit v závislosti na tom, zda přenášený znak má lichý nebo sudý počet 1 (sudá nebo lichá parita).

### 8.3.3 Podélná parita

Lze zajistit zabezpečení celého bloku dat. Nekontroluje se sudý/lichý počet jedničkových bitů v jednotlivých znacích, ale sudý/lichý počet jedničkových bitů ve stejnohlých bitových pozicích všech znaků v jednom bloku. Je-li blok dat tvořen např. osmibitovými znaky, přidá se k celému bloku osm paritních bitů. Tj. přidáme jeden znak navíc a každý jeho bit nastavíme aby byla dodržena sudá/lichá parita.

Podélnou paritu lze vyhodnocovat také průběžně.

### 8.3.4 Křížová parita

Je kombinací zabezpečení bloku dat příčnou a podélnou paritou.

## 8.4 Checksum — kontrolní součet

Metoda kontrolního součtu je metoda pro zabezpečení celého bloku dat. Pro tento účel chápeme jednotlivé znaky zprávy jako celá dvojková čísla bez znaménka. Čísla sčítáme *modulo*  $2^8$  nebo *modulo*  $2^{16}$ .

Výsledkem je opět číslo v délce jeden až dvou bitů. Výpočet probíhá průběžně při přijímání znaků. Po přijetí kontrolního součtu se ověří s vypočteným a pokud souhlasí blok dat je přijat. V opačném případě je nutné si vyžádat opakované poslaní bloku.

## 8.5 Hammingův kód

- <http://www.karlin.mff.cuni.cz/~tuma/2002/NLinalg8.pdf>
- [http://cs.wikipedia.org/wiki/Hamming%C5%AFv\\_k%C3%B3d](http://cs.wikipedia.org/wiki/Hamming%C5%AFv_k%C3%B3d)

### 8.5.1 Postup generování Hammingova kodu

1. Všechny bitové pozice, jejichž číslo je rovné mocnině 2, jsou použity pro paritní bit (1, 2, 4, 8, 16, 32, ...).
2. Všechny ostatní bitové pozice náleží kódovanému informačnímu slovu (3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, ...).
3. Každý paritní bit je vypočítán z některých bitů informačního slova. Pozice paritního bitu udává sekvenci bitů, které jsou v kódovém slově zjišťovány a které přeskočeny.
  - Pro paritní bit  $p_1$  (pozice 1) se ve zbylém kódovém slově 1 bit přeskočí, 1 zkontroluje, 1 bit přeskočí, 1 zkontroluje, atd.
  - Pro paritní bit  $p_2$  (pozice 2) se přeskočí první bit, 2 zkontrolují, 2 přeskočí, 2 zkontrolují, atd.
  - Pro paritní bit  $p_3$  (pozice 4) se přeskočí první 3 bity, 4 zkontrolují, 4 přeskočí, 4 zkontrolují, atd.

### 8.5.2 Rozšířený Hammingův kód (8,4)

Rozšíření Hammingova kodu spočívá v přidání bitu na začátek každého kódového slova. Určen bude pro kontrolu parity. Paritní bit je volen jako sudá parita.

Rozšířený kód dovoluje, tak jako předchozí Hammingův kód, opravit jednu chybu a navíc je schopen detekovat dvě chyby.

## 8.6 Hammingova vzdálenost

Hammingova vzdálenost  $\rho$  dvou kódových slov je *počet míst, v nichž se kódová slova liší*. Charakterizuje odolnost kódů proti poruchám a schopnost identifikovat, případně i opravit chyby.

Minimální Hammingova vzdálenost (vzdálenost kodu) = minimum ze vzdáleností mezi všemi možnými páry vektorů.

### 8.6.1 Detekce chyb

Pro detekci  $n$  bitových chyb musí být  $d_{min} \geq n + 1$ , tj.  $n \leq d_{min} - 1$

### 8.6.2 Detekce a korekce chyb

Pro opravu  $n$  bitových chyb musí být  $d_{min} \geq 2n + 1$ , tj.  $n \leq \frac{(d_{min}-1)}{2}$

### 8.6.3 Příklad 1

Je-li kód bez redundance (tj. vzdálenost dvou slov = 1) pak nelze počítat s identifikací chyby.

$$\rho(000, 001) = \rho(000, 010) = \rho(000, 100) = 1$$

Pokud minimální vzdálenost dvou kódových slov je 2, lze zjišťovat chyby v jednom symbolu.

$$\rho(000, 101) = 2$$

Při minimální vzdálenosti 3 je již možné opravovat chyby v jednom řádu.

$$\rho(000, 111) = 3$$

Hammingova vzdálenost je počet jedniček v součtu kódových slov modulo 2. V následujícím příkladě jsou dvě kódová slova a jejich Hammingova vzdálenost.

$$1011101010 \quad (1)$$

$$0011001001 \quad (2)$$

$$1000100011 \quad (3)$$

Hammingovy vzdálenosti se využívá v samoopravovacích kódech.

### 8.6.4 Příklad 2

Máme Hammingovu vzdálenost kódu 2. Jednobitová chyba jde detekovat, ale nelze opravit.

$$\begin{array}{c} \langle -? - \rangle \\ *-----* \\ 00 \quad 01 \quad 11 \\ \quad \quad 10 \end{array}$$

### 8.6.5 Příklad 3

Máme hammingovu vzdálenost kódu 3. Jedno a dvoubitová chyba lze detekovat. Opravit lze pouze jednobitovou chybu.

$$\begin{array}{c} \langle \leftarrow \quad \rightarrow \rangle \\ *-----* \\ 000 \quad 001 \quad 011 \quad 111 \\ \quad \quad 010 \quad 110 \\ \quad \quad 100 \quad 101 \end{array}$$

## 8.7 Cyklické kódy (CRC)

Cyklický redundantní součet (Cyclic Redundancy Check, CRC) se používá k detekci chyb během přenosu či ukládání dat. CRC je vypočten před operací, u níž jsou předpokládány chyby. Je odeslán či uložen spolu s daty. Po převzetí dat je z nich nezávisle spočítán CRC znovu. Pokud vyjde různý CRC, je přenos prohlášen za chybový. V určitých případech je možné chybu pomocí CRC opravit.

Generující polynom  $G(x) = x^4 + x + 1$  což je  $(10011)_2$  a zpráva  $M(x) = 1101011011$ . Délka zabezpečení je rovna stupni generujícího polynomu, tj.  $k = 4$ . Vypočteme zbytek po dělení  $R(x) = \frac{M(x)}{G(x)}$

$$\begin{array}{r} 11010110110000 : 10011 = 1100001010 \\ 10011 \\ \hline 010011 \end{array}$$

$$\begin{array}{r}
10011 \\
\hline
0000010110 \\
\phantom{00000}10011 \\
\hline
\phantom{00000}0010100 \\
\phantom{00000}\phantom{0}10011 \\
\hline
\phantom{00000}\phantom{0}\phantom{0}1110 = R(x)
\end{array}$$

Posíláme tedy zprávu  $T(x) = M(x) + R(x)$  což je  $T(x) = 1101011011 \mid 1110$ . Nyní příjemce přijme zprávu i se zabezpečením  $T(x)$  a vydělí ji generujícím polynomem  $G(x)$ . Bude-li zbytek nula, zpráva byla doručena bezchybně.

$$\begin{array}{r}
11010110111110 : 10011 = 1100001010 \\
10011 \\
\hline
010011 \\
\phantom{010011}10011 \\
\hline
\phantom{010011}0000010111 \\
\phantom{010011}\phantom{00000}10011 \\
\hline
\phantom{010011}\phantom{00000}\phantom{001}0011 \\
\phantom{010011}\phantom{00000}\phantom{001}\phantom{0}10011 \\
\hline
\phantom{010011}\phantom{00000}\phantom{001}\phantom{0}\phantom{0}000000 = R(x)
\end{array}$$

Takže žádná chyba nebyla detekována.

## 9 Potvrzování přenosu

V reálných sítích dochází k problémům při komunikaci:

- ztráta paketu,
- poškození paketu,
- duplikace paketu,
- změna pořadí paketů (v sítích s alternativními cestami).

Proto je nutné zavést možnost *zpětné vazby* k přenosu. Příjemce informuje odesílatele jak přenos dopadl. Musíme mít na paměti, že chyby mohou nastat nejen v datech která přenášíme, ale mohou se ztratit i potvrzení. Zpětná vazba může být:

- potvrzovací (ACK/NACK)
- detekční (např. CRC)
- informační (celý potvrzovací rámec)

### 9.1 Komunikační protokol

Soubor (syntaktických a sémantických) pravidel včetně definice časových poměrů pro komunikaci dvou nebo více zařízení.

**Nespolehlivá služba** příjemce nemá povinnost explicitně reagovat, poškozený rámec může zahodit.

**Spolehlivá služba** jestliže příjemce detekuje poškozený rámec, musí se postarat o nápravu (vyžádání opakovaného přenosu rámce). Z předchozího cvičení víme, že detekce chyb není nikdy na 100 %.



Budeme se zabývat zajištěním spolehlivé komunikace pro dvě stanice.

- protokol stop a wait,
- využití kapacity přenosového kanálu,
- kladné a záporné potvrzování,
- jednoduchý program pro příjem a vysílání.

## 9.2 Číslování rámců

Zavádí se sekvenční čísla pro zajištění správného pořadí rámců. Zajistí ochranu proti:

- chybnému pořadí paketů,
- duplicitám při opakovaném vysílání.

## 9.3 Potvrzovací schémata

### 9.3.1 Typy potvrzování

#### potvrzování s časovým limitem

- Volba vhodného časového limitu (timeout) řeší problém ztráty pozitivního potvrzení.
- Komplikuje formální popis protokolu protože záleží i na časovém kontextu.
- problém volby velikosti timeoutu: brzké zjištění nutnosti retransmise vs. předčasné retransmise

Podle toho co potvrzujeme:

**pozitivní** (ACK) potvrzuje správné přijetí

**negativní** (NACK) informuje o přijetí rámce s chybou (urychluje detekci chyby)

**kombinované** použití ACK i NACK

Negativní potvrzení nejsou nutná, jen urychlují detekci neúspěšného přenosu rámce (která by se jinak zjistila až při vypršení timeoutu)

Podle způsobu jak je potvrzení posíláno:

**samostatné** potvrzení je přenášeno jako samostatný rámec speciálního typu (větší režie protože potvrzení je malé)

**nesamostatné** potvrzení je zasláno jako součást datových rámců (piggybacking<sup>7</sup>)

Potvrzovací schémata (protokoly):

**stop-and-wait** vysílač vyšle jediný rámec a čeká na potvrzení. Na kanálech s velkým zpožděním velmi neefektivní.

**skupinové** je efektivní pro spoje s velkou dobou zpoždění

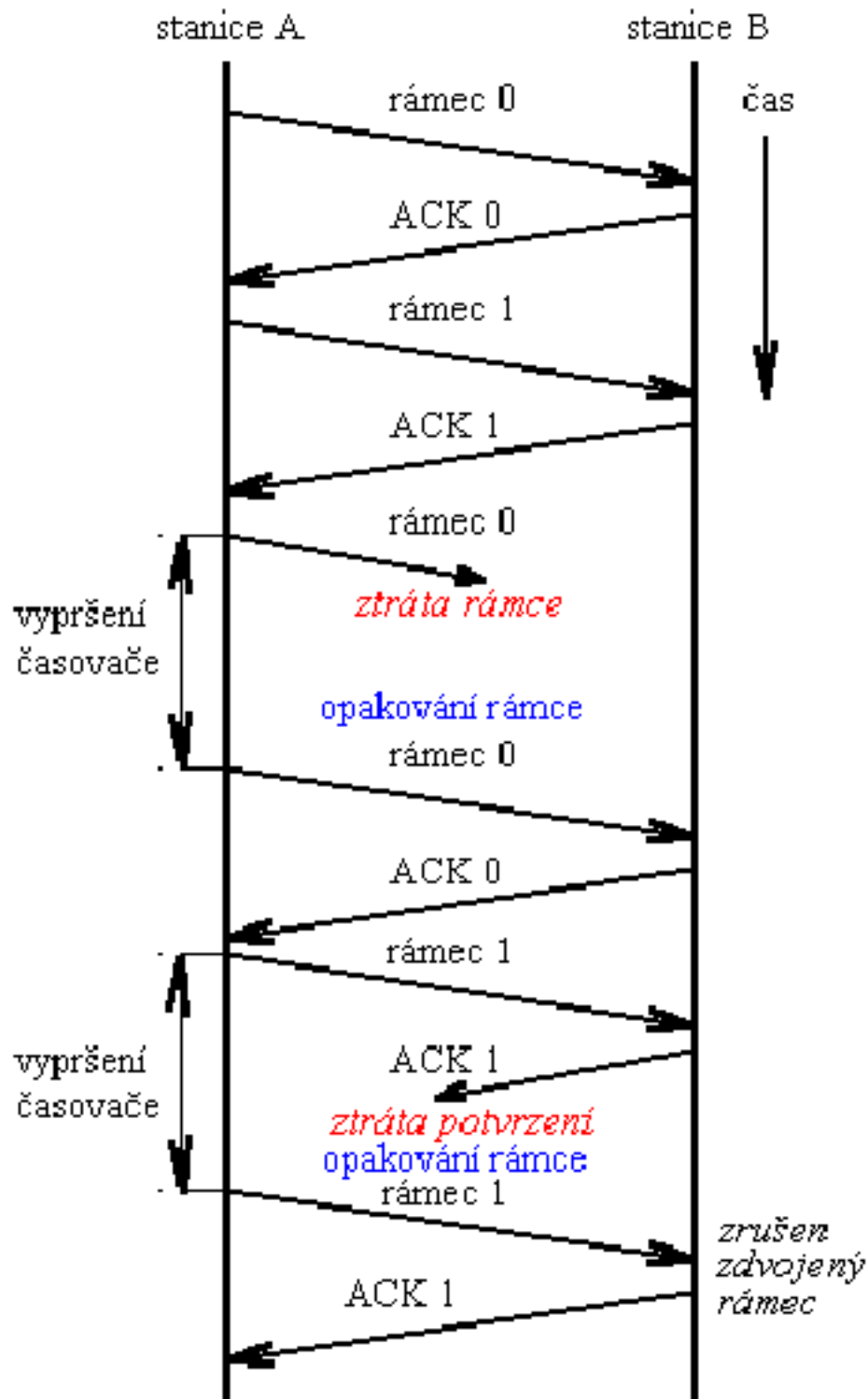
- Continuous ARQ (Automatic Retransmission Request): na full-duplex kanálu, efektivita až 100 %.
- potvrzení zpravidla inkluzivní (potvrzuje vše až do uvedeného sekvenčního čísla), čímž se chrání před ztrátou předchozího potvrzení

---

<sup>7</sup> *Piggybacking* je připojování potvrzení k informačním rámcům ve druhém směru.

### 9.3.2 Stop-and-Wait

Obrázek 5: Stop and wait protokol



### 9.3.3 Sliding window

Metody klouzavého okénka (sliding window) mají tato pravidla:

- Stanice smí vyslat více rámců (počet dán šířkou vysílacího okna).

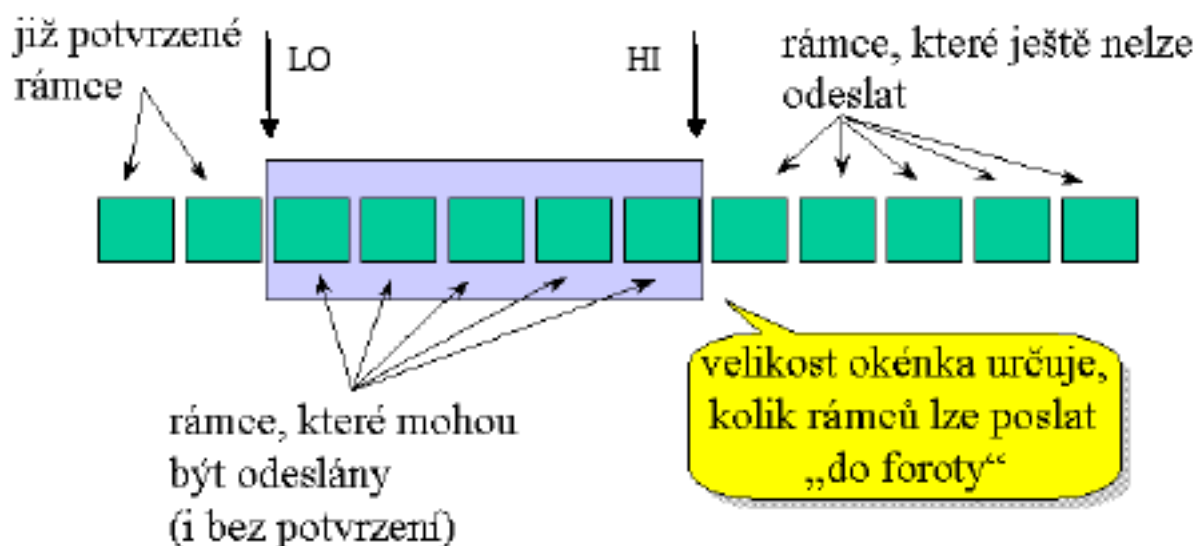
- Při odeslání se pro každý rámeček nastavuje časovač pro potvrzení.
- ACK se posílá od každého přijatého rámce, popř. do časového limitu od času příchodu prvního dosud nepotvrzeného rámce (ušetření některých ACK).
- Při přijetí rámce s chybou ACK nevyšleme nebo vyšleme NACK.

Obě okénka „kloužou“ po sekvenčních číslech.

**Vysílací okno** Buffer s vyslanými rámci, které dosud nebyly potvrzeny a možná budou muset být vyslány znovu.

**Přijímací okno** Buffer na přijímané rámce, které ještě nemohly být doručeny vyšší vrstvě přijímače, protože dosud chybí některý z předchozích rámců v řadě.

Obrázek 6: Obecný princip metody Sliding window

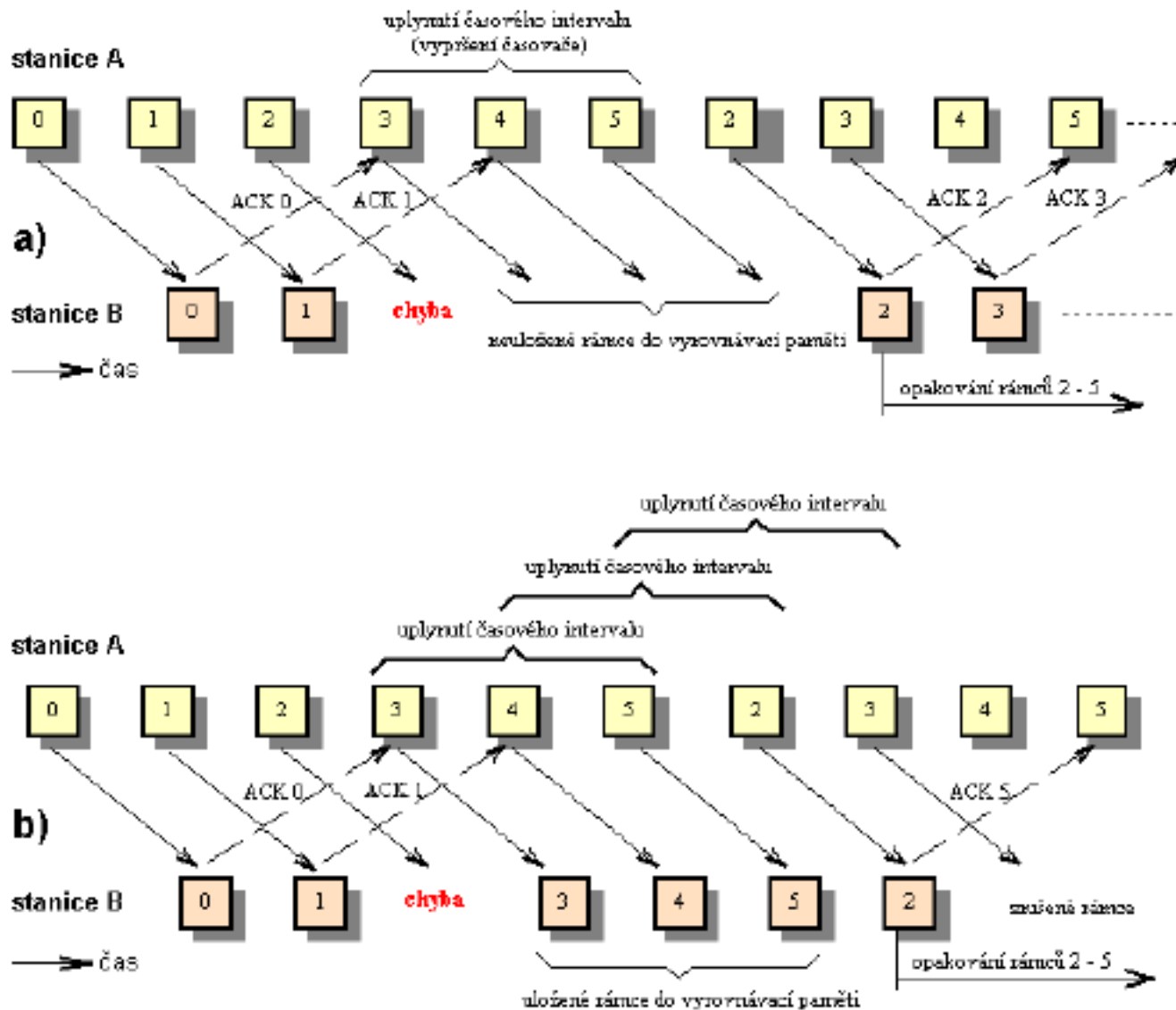


**Metoda Go-Back-N** <http://lerci.tagus.ist.utl.pt/applets/go-back-n/go-back-n.html> (Go-Back-N demo)

Šířka okna musí být aspoň o jednu menší než počet použitelných sekvenčních čísel, jinak bychom nepoznali ztrátu všech rámců okna.

Při timeoutu (nebo explicitním odmítnutí) rámce se *zopakuje nepotvrzený rámeček i všechny následující* (předpokládá se, že přijímač s přijímacím okénkem o jedné pozici je stejně odmítne a zahodí).

Obrázek 7: Princip metody Go-Back-N pro šířku okna přijímače 1



Chování při chybě v přenosu rámců je následující:

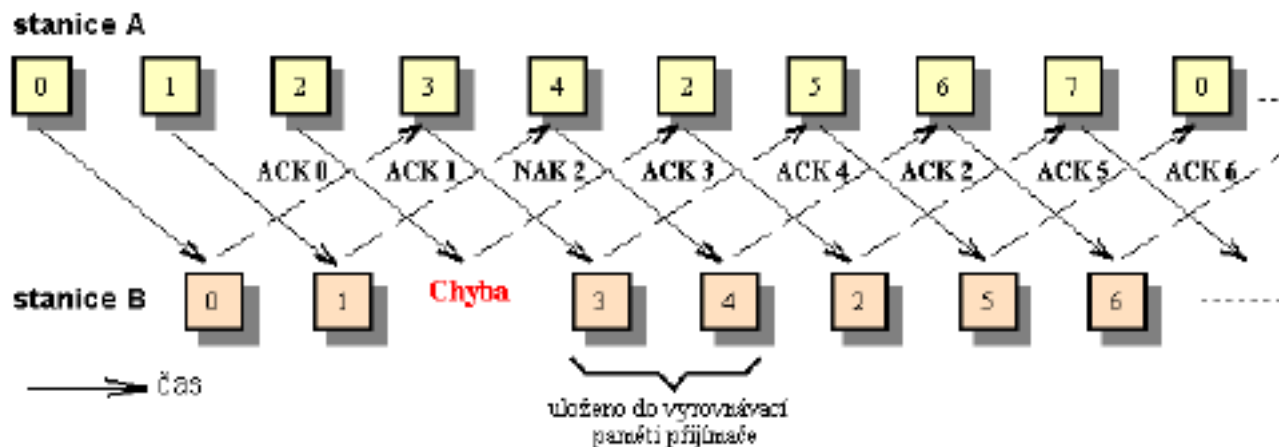
1. Přijímač detekoval chybu v rámci  $i$ :
  - (a) přijímač pošle NAK nebo mlčí (převod na ztrátu rámců v síti),
  - (b) vysílač při příjmu NAK opakuje všechny rámce vysílacího okna od  $i$  znovu.
2. Rámec se ztratí a přijímač očekává rámec  $i - 1$ , ten se ztratil, ale přišel jeho následník  $i$ :
  - (a) přijímač odešle NAK  $i$  (nebo mlčí, čímž se převede na ztrátu rámců).
3. Rámec  $i$  se ztratí a další se nevysílají:
  - (a) vysílač po timeoutu rámec  $i$  pošle znovu.

**Metoda Selective Repeat** [http://media.pearsoncmg.com/aw/aw\\_kurose\\_network\\_3/applets/SelectRepeat/SR.html](http://media.pearsoncmg.com/aw/aw_kurose_network_3/applets/SelectRepeat/SR.html) (Selective Repeat Demo)

Selektivní zopakování jen těch rámců, které vytimeoutují nebo jsou přijímačem explicitně odmítnuty (NACK)

Přijímač musí obsahovat buffery na rámce, které přijdou za chybovým a uchovávat je do doplnění celé sekvence (a jejího poskytnutí aplikaci). Šířka okna přijímače je tedy více než jeden rámec.

Obrázek 8: Princip metody Selective Repeat



Maximální šířka okna je polovina sekvenčních čísel (z důvodu překrývání vysílacího a přijímacího okna).

- Negativní potvrzování se často nepoužívá, řeší se timeouty na straně vysílače. Může však urychlit opravu chyb.
- Maximální šířka přijímacího okna může být stanovena pevně. Inzerování aktuální šířky přijímacího okna v potvrzeních však můžeme protokol současně použít pro řízení toku dat od vysílače (flow control). Toho je využito např. v protokolu TCP. Vysílací okno se pak dynamicky upravuje tak, aby nepřesahovalo šířku přijímacího okna.
- Sliding window nalezneme např. na spojové nebo transportní vrstvě modelu OSI-RM.

## 9.4 Stanovení velikosti okénka

[http://www2.rad.com/networks/2004/sliding\\_window/](http://www2.rad.com/networks/2004/sliding_window/)

## 9.5 Sekvenční příjem

## 9.6 Nesekvenční příjem

## 9.7 Příklady

### 9.7.1 Stop and Wait

Modemová linka (pomalá a LAN se spíše větším zpožděním)

$$l_m = 80 \text{ B}, l_a = 1 \text{ B}, c = 14400 \text{ bps}, T = 1 \text{ ms}, ef = 94.56 \%$$

pro družicový spoj:

$$l_m = 80 \text{ B}, l_a = 1 \text{ B}, c = 14400 \text{ bps}, T = 270 \text{ ms}, ef = 7.6 \%$$

**Po prodloužení rámce 8x** Modemová linka (pomalá a LAN se spíše větším zpožděním):

$$l_m = 640 \text{ B}, l_a = 1 \text{ B}, c = 14400 \text{ bps}, T = 1 \text{ ms}, ef = 99.28 \%$$

a pro družicový spoj

$$l_m = 640 \text{ B}, l_a = 1 \text{ B}, c = 14400 \text{ bps}, T = 270 \text{ ms}, ef = 40.38 \%$$

Prodloužení rámce efektivitu zlepší, ale v případě výskytu chyby v rámci se pak zahazuje celý (dlouhý) rámec. To je rozdíl oproti jednomu ze dvou polovičních.

### 9.7.2 Sliding Window

Lze dosáhnout až 100 % efektivity.

## 9.8 Protokoly linkové úrovně

### 9.8.1 BSC

### 9.8.2 HDLC

### 9.8.3 IEEE 802.2

### 9.8.4 PPP

### 9.8.5 SLIP

## 10 Řízení přístupu v lokálních počítačových sítích

V lokálních počítačových sítích (LAN), jsou uzly spojeny ke společnému přenosovému médiumu. Vysílání jednoho uzlu mohou tedy přijímat i ostatní uzly na stejné síti současně. Problém však nastane v případě, že chce více uzlů používat přenosové médium současně pro vysílání. V tomto okamžiku je potřeba uzel nebo algoritmus, který vybere jeden uzel a ten bude mít možnost použít přenosové médium k vysílání.

### 10.1 Kolize

Kolizí se označuje situace kdy k současnému vysílání přeci jen dojde. Kolize je tedy současné (nebo s nepatrným zpožděním) vysílání kdy dojde na přenosovém médium ke „smíšení“ různých vysílání. Zpětně je již oddělit nelze a proto je potřeba se kolizím vyhýbat. V lepším případě jím předcházet nebo je zcela vyloučit za pomoci přístupových metod k přenosovému médium.

### 10.2 Metody řízení přístupu

<http://www.earchiv.cz/b06/b0100001.php3>

Podle typu detekce kolizí:

- zcela vylučující kolize (CA, Collision Avoidance)
- detekující kolize (CD, Collision Detection)
- bez detekce kolizí

Podle charakteru řízení:

- řízené (deterministické chování)
- neřízené (nedeterministické) — existuje náhodný faktor ovlivňující výsledek

Podle existence arbitra:

- centralizované (s centrálním prvkem)
- distribuované (bez centrálního prvku)

Dalším způsobem předcházení kolizí je že se vysílající uzel snaží nejprve detekovat provoz na přenosovém médium. V případě že je médium volné, začne vysílat. Tento způsob ale neřeší případ kdy dvě stanice naslouchají ve stejný okamžik a rozhodnou že je přenosové médium volné.

### 10.3 Centralizované

Existuje centrální prvek, arbitr. Ten přiděluje přenosový kanál uzlům na základě:

- výzvy (pooling — „chceš vysílat? *ano x ne*“)
- žádosti (request — „chci vysílat! Můžeš...“)

Veškerý provoz na síti je v režii arbitra, na jehož algoritmu závisí. Algoritmus se může samozřejmě s časem měnit. Může také počítat s prioritami apod.

- CMTS (Cable Modem Termination System) pro kabelové sítě. Existuje speciální rezervační rámec podle něhož uzly označí svůj zájem o vysílání.
- Demand Priority využívá stromovou strukturu sítě a každý uzel má jednu přípojku k nadřazenému hubu (rozbočovači). Tento hub rozhoduje v součinnosti s ostatními huby v celé síti jako arbitr a uděluje právo vysílat.

Neřízené metody nemají u centralizovaného způsobu řízení význam.

Nevýhodou centralizovaného způsobu řízení je výpadek sítě při selhání centrálního bodu.

### 10.4 Decentralizované metody

U decentralizovaných nebo také distribuovaných metod centrální arbitr neexistuje. Uzly se musí domluvit mezi sebou právě distribuovaným způsobem. Proto musí všechny uzly striktně dodržovat stejná pravidla.

Tyto metody mohou být řízené i neřízené.

#### 10.4.1 Aloha

Aloha je neřízená distribuovaná metoda, která vznikla na univerzitě na Havajských ostrovech (proto Aloha) v roce 1970.

- Využívá rádiového přenosu éterem.
- Nekontroluje stav přenosového média — nedívá se, zda už někdo jiný nevysílá. Prostě pošle zprávu.
- Když do určité doby nedostane potvrzení, tak pošle zprávu znovu.
- Potvrzování musí být řešeno na vyšších vrstvách, třeba na aplikační.
- Vykazuje velmi nízké procento celkové kapacity kanálu asi 20 % (s výjimkou limitní situace, kdy vysílá pouze jedna stanice velké množství dat a ostatní „mlčí“).

Další varianta je *Slotted Aloha*

- koncová stanice může zahájit vysílání pouze v pevně stanovených okamžicích (čas je rozdělen do slotů).
- Maximální dosažitelné využití kapacity se tak téměř zdvojnásobí.

#### 10.4.2 CSMA

CSMA což je zkratka Carrier Sense Multiple Access.

**Carrier Sense** (naslouchání nosné) vysílač naslouchá nosné vlně před pokusem vysílat. Pokouší se detekovat přítomnost signálu přenášeného z jiné stanice před pokusem o vysílání. Je-li nosná detekována, uzel před pokusem vlastního vysílání počká, než probíhající vysílání skončí.

**Multiple Access** (vícenásobný přístup) na médiu vysílá a přijímá více uzlů. Vysílání jednoho uzlu je obecně přijímáno všemi ostatními uzly užívajícími médium.

Současné vysílání více uzlů vede ke kolizi rámců. Všechny rámce jsou pak zkresleny a přijímače nejsou schopny rozlišit překrývající se přijaté signály jeden od druhého. Není možné zcela zabránit kolizím, ale lze se s nimi vypořádat.

CSMA má jako ochranu proti kolizím pouze naslouchání nosné.

Pokud se dva uzly pokusí vysílat v téměř stejném čase, žádný nedetekuje nosnou a oba začnou vysílat. Vysílající nezjišťují kolize, takže přenesou celý rámec (čímž plýtvají šířkou pásma). Přijímače nemohou rozlišit mezi kolizemi a jinými zdroji chyb rámců, takže obnova z kolize závisí na schopnosti komunikujících uzlů detekovat chyby rámců a vyvolat proceduru obnovy z chyb. Například přijímač nepošle požadované potvrzení, což přiměje vysílače myslet si, že došlo k vypršení časového limitu a opakovat odeslání.

Existují tři typy kolizí:

- Lokální kolize
- Vzdálená kolize
- Pozdní kolize (jakmile je přeneseno 64 bajtů z rámce a poté nastane kolize, tzn. že nějaká jiná stanice začne vysílat)

#### 10.4.3 CSMA/CA

Jedná se o metodu CSMA s předcházením kolizím — Carrier Sense Multiple Access With Collision Avoidance.

Každý uzel musí informovat ostatní uzly o úmyslu vysílat. Jakmile byly ostatní uzly informovány, informace je vysílána. Zabrání se částečně kolizím, protože všechny uzly vědí o vysílání dříve, než k němu dojde. Kolize jsou nicméně stále možné ale nejsou stejně jako v čistém CSMA detekovány.

CSMA/CA se využívá v bezdrátových sítích, protože účastníci bezdrátového přenosu nejsou schopni zároveň vysílat a přijímat.

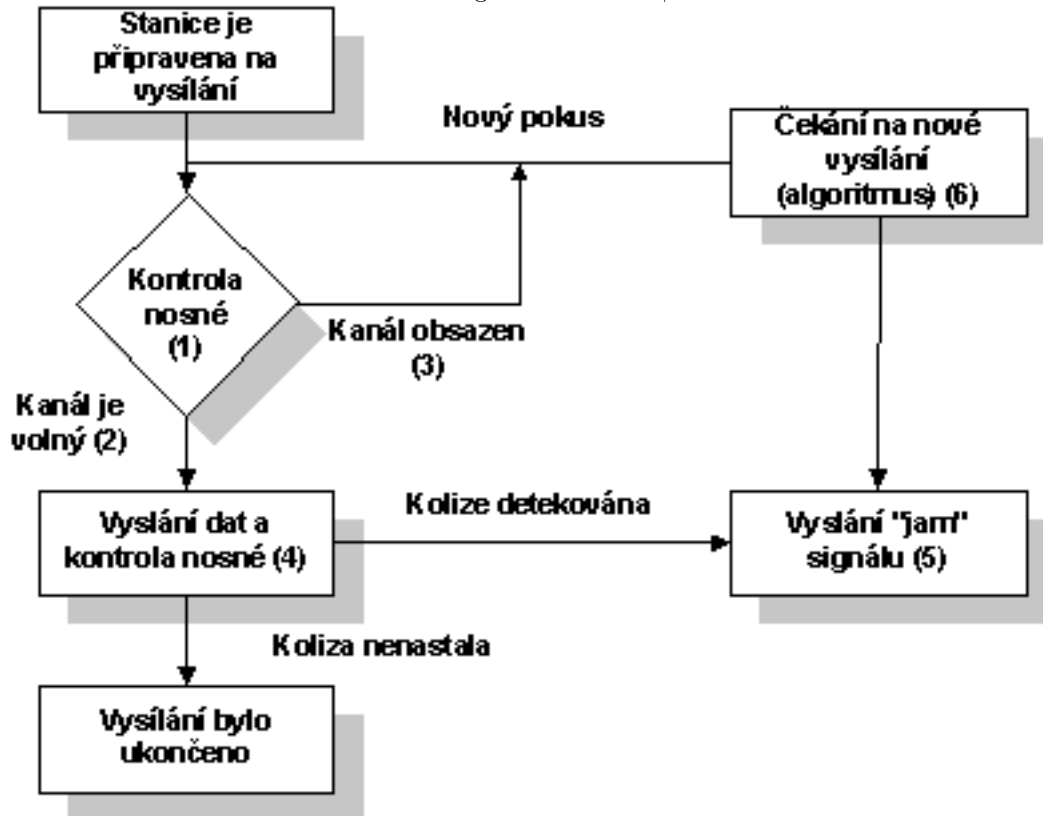
#### 10.4.4 CSMA/CD

Jedná se opět o CSMA a tentokrát s detekcí kolizí — Carrier Sense Multiple Access With Collision Detection.

- Vysílající uzly jsou schopny detekovat výskyt kolize a zastavit vysílání okamžitě.
- Počkat náhodnou dobu před dalším pokusem o odeslání. Exponenciální čekání znamená odložený pokus o vysílání. Stanice si náhodně vybere dobu z intervalu, jehož velikost se během opakovaných pokusů zdvojnásobuje. Tuto náhodnou dobu čeká a zároveň sleduje, zda nezačal vysílat někdo jiný. Pokud ano, čeká na uvolnění. Zůstalo-li médium volné, odvysílá rámec. Jestliže vysílání neuspěje, opakuje exponenciální čekání. Náhodný výběr čekací doby z rychle rostoucího intervalu má za cíl snížit pravděpodobnost, že kolidující stanice spolu budou při dalším pokusu kolidovat znovu.
- Mnohem efektivnější využití média, protože se neplýtvá časem na vysílání celých kolidujících rámců.



Obrázek 9: Algoritmus CSMA/CD



Vlastnosti:

- distribuovaná přístupová metoda.
- neřízená (nedeterministické) přístupová metoda.
- využívá příposlech nosné (CS, Carrier Sense). Jednotlivé uzly tedy před začátkem vlastního vysílání poslouchají, zda právě nevysílá někdo jiný.
- přístupové metody s detekcí kolize (CD, Collision Detect). Takže v Ethernetu může ke kolizím docházet, ale tyto jsou následně detekovány, a jsou řešeny.
- CSMA/CD nelze použít pro všechna média (rádio) a vyžaduje přídatnou elektroniku (detekce kolizí).

Oproti čistému CSMA u CSMA/CD stanice při svém vysílání současně kontroluje přenosové médium, zda nezachytí jiné vysílání, které koliduje s jejím. Proto „s detekcí kolizí“ (with Collision Detection). Pokud stanice zjistí kolizi, zastaví vysílání, počká náhodnou dobu a opakuje svůj pokus znovu.

CSMA/CD je efektivnější než samotné CSMA či CSMA/CA — v nich se kolize nezjišťují a dojde-li k nim, zbytečně se odvysílá celý datový rámec, který bude beztak nutno opakovat.

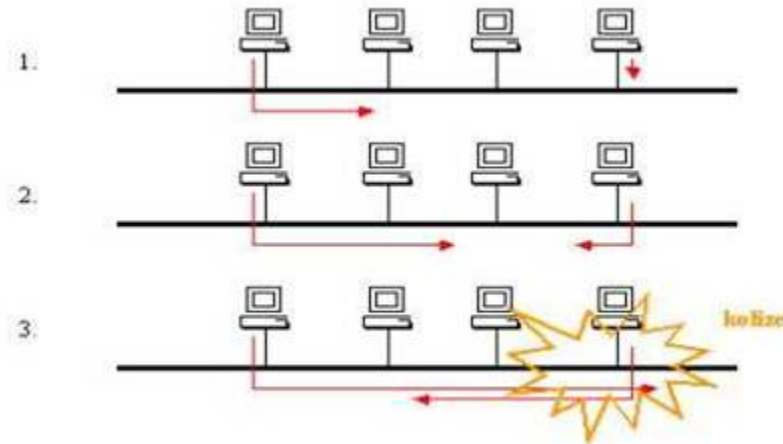
Modernější varianty Ethernetu již používají přepínače s plně duplexním režimem provozu a metoda CSMA/CD u nich není nadále uplatňována.

**Kolizní okénko** Ke kolizi může dojít jen pokud dvě či více stanic zahájí vysílání téměř současně. Jakmile se stanici podaří obsadit svým signálem médium, ostatní už zjistí probíhající vysílání a počkají na jeho ukončení.

Doba, která uplyne od okamžiku začátku vysílání stanice do okamžiku, kdy se její signál rozšíří do celého média, je označována jako kolizní okénko. Pouze během této doby může dojít ke kolizi. Signál se šíří prakticky rychlostí světla a velikost kolizního okénka je proto dána délkou média a zpožděním signálu v aktivních prvcích (opakovačích) na cestě.

Kolizní okénko musí být menší než doba vysílání minimálního rámce. Jedině tak je zajištěno, že signál obsadí celé médium dříve, než stanice ukončí vysílání rámce. Jinak by mohlo docházet k nejsporným kolizím.

Obrázek 10: Detekce kolize



Zásadní důsledky jsou tyto:

- Rámec nesmí být příliš krátký. Proto má datová část ethernetového rámce minimální délku 46 B, což společně s hlavičkami představuje minimální rámec velikosti 64 B.
- Maximální délka média a počet opakovačů na trase jsou omezeny.
- Komplikuje se zvýšení přenosové rychlosti. Vede ke zkrácení doby vysílání minimálního rámce, která vynucuje příslušné zkrácení kolizního okénka (zpravidla zkrácením média).

#### 10.4.5 CSMA/BA

Opět CSMA a tentokrát s bitovou arbitráží — Carrier Sense Multiple Access With Bitwise Arbitration. Rovněž známo jako CSMA/CR, Carrier Sense Multiple Access with Collision Resolution, CSMA s rozlišením kolizí).

Všem uzlům na propojovacím vedení je přiřazeno identifikační číslo či kód priority. Při výskytu kolize jeden z uzlů pokoušejících se vysílat současně dostane prioritu vysílat podle identifikačního čísla či kódu priority (oproti počkání náhodnou dobu a znovuvyslání jako v CSMA/CD).

Používáno v CAN komunikacích, často k nalezení na vozidlech.

#### 10.4.6 Použití CSMA/xx

- Bezdrátová síť ALOHAnet používala čisté CSMA
- 802.11 DCF používá formu CSMA
- Ethernet používá CSMA/CD v half duplex módu (ale ne ve full duplex módu)
- LocalTalk používá CSMA/CA
- IEEE 802.11 (bezdrátová LAN) používá CSMA/CA
- CAN používá CSMA/BA
- IEEE 802.15 (bezdrátová PAN) používá CSMA/CA

### 10.5 Předávání pověření

Pověření je v podobě *tokenu*. Tento token si uzly předávají v určitém pořadí a token určuje „kdo je na řadě“.

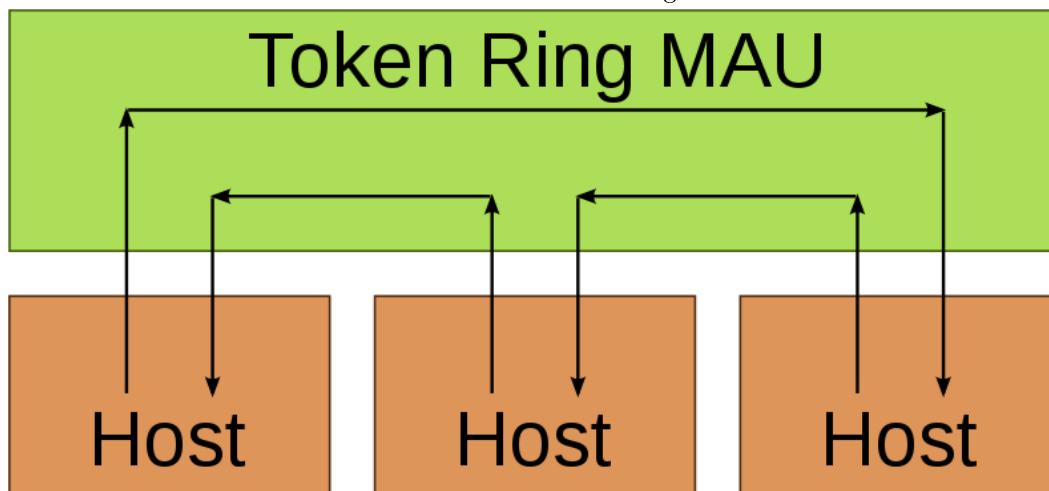
Uzel, který má token může přistupovat k přenosovému médiu a po skončení vysílání předá token dalšímu uzlu v pořadí. Je tedy potřeba uspořádání do *logického kruhu*. Uspořádání do fyzického kruhu (Token Ring) není potřeba, což je případ existence Token Bus.

### 10.5.1 Token Ring

Distribuovaná a řízená metoda využívá předávání pověření (token) v kruhu.

Řešení vyvinuté firmou IBM bylo standardizováno společností IEEE (802.5). Od toho je pak odvozen i název takto vzniklého řešení IEEE 802.5. IBM vyrábí vlastní verzi sítě Token Ring (IBM Token Ring), která je standardu IEEE 802.5 opravdu velmi blízká, je s ním plně kompatibilní, ale není identická. Rozdíl je například v tom, že IBM Token Ring specifikuje konkrétní topologii (do hvězdy), zatímco IEEE 802.5 žádnou fyzickou topologii explicitně nepředepisuje (ale v praxi jde téměř vždy o zapojení do hvězdy). Další rozdíl je např. v přenosovém médiu. IBM Token Ring předepisuje kroucenou dvoulinku, IEEE 802.5 nepředepisuje žádné konkrétní přenosové médium.

Obrázek 11: Token Ring



- Pracovní stanice v lokální síti Token ring jsou zapojeny do logického kruhu.
- Data se přenášejí postupně z jedné stanice na obvodu kruhu do druhé.
- Právo vysílat, tzv. „Token“, koluje mezi stanicemi.
- Předávání tokenu má výhodu nad stochastickým přístupem použitým v Ethernetu, a to zejména při vyšším zatížení sítě. Využívá ho například také ARCNET, Token Bus a FDDI.
- Fyzické uspořádání sítě je do hvězdy — uprostřed kruhu je stanice, která je spojena dvěma linkami s každou stanicí na obvodu kruhu. Spoj mezi dvěma stanicemi na obvodu je realizován přes tuto prostřední.
- Každá stanice předává tento Token rámeček svému sousedovi. Toto předávání tokenu slouží k rozhodnutí, která stanice má právo vysílat na sdílenou sběrnici. Stanice, které chtějí vysílat, musejí počkat než k nim Token dorazí. Token ring sítě obvykle používají k reprezentaci dat diferenciální kódování Manchester.
- Pokud žádná stanice nevysílá data, smyčkou stanic koluje speciální rámeček „token“. Je stanicemi přeposílán k dalšímu sousedovi, dokud se nedostane do stanice, která potřebuje vysílat data. Tato stanice, která chce data vysílat, přemění token na datový rámeček a data odvysílá. Jakmile stanice, která dříve vysílala, přijme data přemění je zpátky na Token a pošle ho dále.
- Pokud někde nastane chyba přenosu a v síti nekoluje žádný nebo naopak více tokenů, zasáhne k tomuto účelu vyčleněná stanice, tzv. „aktivní monitor“, která vloží nový nebo odstraní přebytečné tokeny.

**MAU** MAU zkratka z Multistation Access Unit je zařízení, které slouží jako koncentrátor nebo rozbočovač a zajišťuje propojení do kruhu.

Ven ze zařízení vychází přípojky k jednotlivým koncovým uzlům.

**Aktivní monitor** Může docházet k problémům při ztrátě tokenu, přerušení logického kruhu (výpadkem některého z uzlů v logickém kruhu). Dále je potřeba řešit situaci s přidáním a odebráním uzlu z kruhu (zapnutí/vypnutí stanice).

Tyto administrační funkce vykonává jedna ze stanic v síti. Taková stanice má roli *aktivního monitoru*. Funkčnost však musí být implementována ve všech uzlech sítě, tak aby roli aktivního monitoru mohl v případě potřeby vykonávat libovolný uzel

Bývá jím zařízení s nejvyšší MAC adresou v síti nebo první zapnuté zařízení v síti.

### 10.5.2 Token Bus

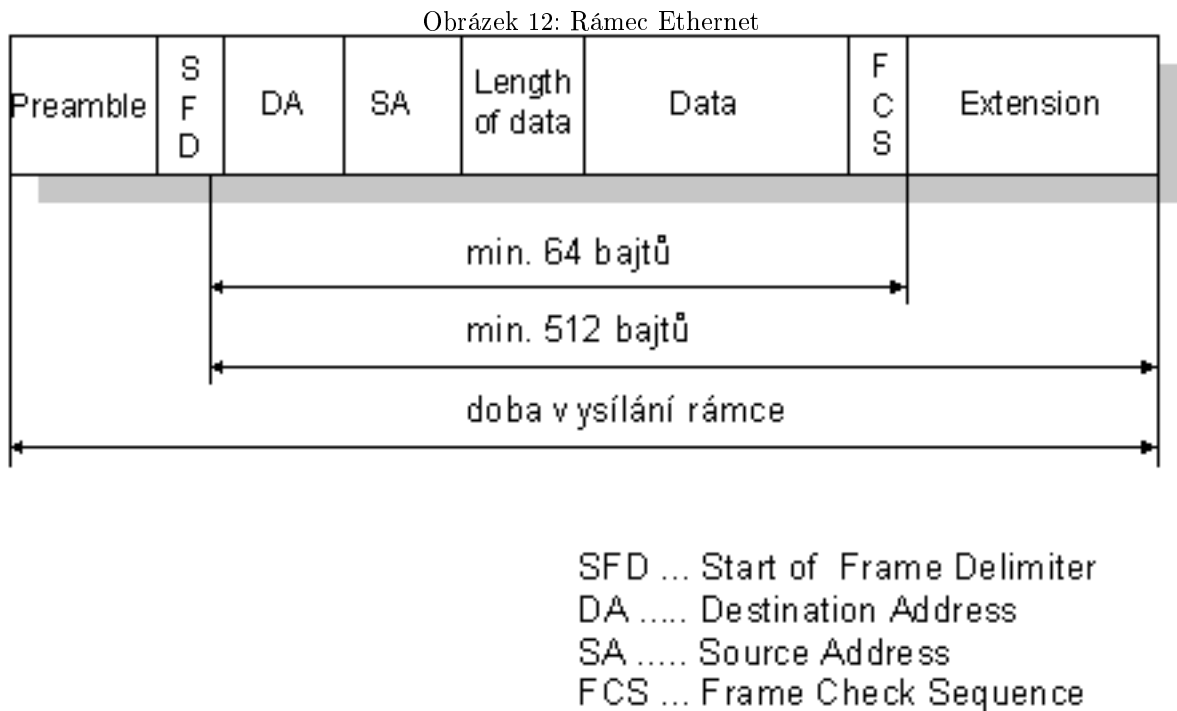
Využívá metody předávání pověření (token) na sběrnici. Fyzická topologie sítě je sběrníková a kruhová je pouze logická topologie (systém předávání oprávnění mezi jednotlivými uzly).

## 10.6 Ethernet

Distribuovaná a neřízená (nedeterministická) metoda přístupu.

- Stanice, která potřebuje vysílat, naslouchá co se děje na přenosovém médiu. Pokud je v klidu, začne stanice vysílat.
- Může se stát (v důsledku zpoždění signálu), že dvě stanice začnou vysílat přibližně ve stejný okamžik.
- Vysílající stanice kolizi poznají podle toho, že během svého vysílání zároveň zjistí příchod cizího signálu. Stanice, která detekuje kolizi, vyšle krátký signál (jam o 32 bitech). Poté se všechny vysílající stanice odmlčí a později se pokusí o nové vysílání.
- Mezi opakovanými pokusy o vysílání stanice počká vždy náhodnou dobu.
- Interval, ze kterého se čekací doba náhodně vybírá, se během prvních deseti pokusů vždy zdvojnásobuje.
- Pokud se během šestnácti pokusů nepodaří rámec odvysílat, stanice své snažení ukončí a ohlásí nadřazené vrstvě neúspěch.
- Ke kolizi může dojít jen v době, která uplyne od začátku vysílání do okamžiku, kdy signál vysílaný stanicí obsadí celé médium.
- Kolizní okénko musí být kratší, než je doba vysílání nejkratšího rámce (dvě vzdálené stanice odvysílají krátké rámce, které se na kabelu protnou a zkomolí, ale obě stanice ukončí vysílání dříve, než k nim dorazí kolidující signál).
- Tato metoda přístupu k médiu je velmi efektivní při nižším zatížení sítě (cca 30 % šířky pásma).
- Efektivita přístupu k médiu klesá při větším počtu zájemců o vysílání (možnost exponenciálního nárůstu kolizí).
- Efektivita CSMA/CD je vyšší pro delší rámce, protože při jejich přenosu je výhodnější poměr mezi trváním kolizního okénka a vysílání dat.

### 10.6.1 Formát rámce



Formát rámců lokální sítě Ethernet II a IEEE 802.3 se skládá z následujících polí:

**Preamble** skládá se z 8 byte, střídavě binární 0 a 1. Poslední byte má tvar 10101011 a označuje začátek vlastního rámce. Preamble slouží k synchronizaci. Poslední byte se někdy nazývá omezovač počátku rámce (Starting Frame Delimiter, SFD).

**Příznak začátku rámce** 1 B

**Cílová adresa** fyzická MAC adresa o délce 48 bitů (v rámci LAN pro všechny stanice stejné délky). Adresa může být individuální (unicast), skupinová (multicast) a všeobecná (broadcast).

**Zdrojová adresa** fyzická adresa stejného typu jako cílová, ale je to vždy individuální adresa konkrétní stanice (rozhraní).

**Typ protokolu nebo délka**

- Pro Ethernet II je to pole určující typ vyššího protokolu.
- Pro IEEE 802.3 udává toto pole délku pole dat.

**Data** Pole dlouhé minimálně 46 oktetů a maximálně 1500 oktetů (46 B - 1500 B). Minimální délka je nutná pro správnou detekci kolizí.

**Datová výplň** vyplní rámec pokud je přepravovaných dat méně, než 64 B

**Kontrolní součet** (Frame Check Sequence, FCS) 32b CRC, který se počítá ze všech polí s výjimkou preamble a FCS. Slouží ke kontrole správnosti dat - příjemce si jej vypočítá z obdrženého rámce a pokud výsledek nesouhlasí s hodnotou pole, rámec zahodí jako vadný.

Vysílaný paket musí mít takovou délku aby kolizní situace byla vyhodnocena dříve všemi prvky sítě, než skončí vysílání paketů, která kolizi způsobila. Vezmeme-li v úvahu rychlost šíření a maximální délku sběrnice dostaneme pro spolehlivou detekce kolize minimální velikost paketu 64 B (někdy se uvádí v literatuře 72 B). Maximální zpoždění vedení je 25 µs.

# 11 Směrování

## 11.1 Pojmy

- Routing, routování, směrování
  - Je proces volby cesty v síti pro přenos síťového provozu.
  - Dopravit datový paket určenému adresátovi (pokud možno co nejefektivnější cestou).
  - Nezabývá se celou cestou paketu, ale řeší vždy jen jeden krok (komu data předat jako dalšímu).
  - Realizováno je v síťové vrstvě.
  - Jako cesta autem: „Jedete a podle návěstí se rozhodnete kam jet.“
- Source Routing
  - Způsob průchodu datových rámců skrz jednotlivé mosty se určí předem u odesílatele (proto „source“).
  - O směrování (proto „routing“) rozhoduje odesílatel.
  - Lépe „source route bridging“, .
  - Potřebné pokyny k průchodu takto zvolenou trasou se vloží do každého jednotlivého rámce.
  - Pokyny pro směrování mají formu lineárního seznamu mostů, přes které má datový rámec postupně projít.
  - Odesílatel pošle na všechny strany „průzkumný rámec“, který se sám šíří do všech existujících směrů, dokud nedojde k hledanému cíli. Od něj se rámec vrací zpět ke svému původnímu odesílateli a nese v sobě informaci o trase, kterou přitom prošel.
  - Vyvinuto IBM pro síť Token Ring.
  - Jako cesta vlakem: „Naplánujete si cestu, přestupní stanice předem a pak podle tohoto seznamu jedete k cíli.“
- Repeater (opakovač)
  - zařízení, které zajišťuje propojení na fyzické vrstvě
- Switch (přepínač)
  - zařízení, které zajišťuje propojení na linkové vrstvě
- Router (směrovač)
  - zařízení, které zajišťuje propojení na síťové vrstvě
  - propojuje dvě a více sítí
  - přepojování paketů (manipulace s pakety)
  - volba směru (směrování - routing) i předávání (forwarding) paketů
- Routing tables (směrovací tabulky)
  - Ve směrovací tabulce si router (směrovač) udržuje informaci o topologii sítě.
  - Mohou být:
    - \* statické — naplněny jednorázově a dále jsou neměnné,
    - \* dynamické — upravují se podle aktuálního stavu sítě, směrovače si vyměňují informace mezi sebou a podle toho aktualizují směrovací tabulky.
  - Obsahuje trojici údajů:
    - \* cílová síť
    - \* vzdálenost
    - \* odchozí směr

- Manipulace s tabulkou v linuxu:
  - \* route
  - \* ip route show
- Default route (výchozí cesta)
  - pro všechny cílové adresy, které nejsou uvedeny ve smerovací tabulce se použije default route, tj. výchozí směr kudy budou posílány
  - cílem je snížit počet záznamů ve směrovací tabulce
  - route add default gw 147.228.67.1
- Schéma doručování:
  - *unicast* doručí zprávu jednomu specifickému uzlu (uni=jedno),
  - *broadcast* doručí zprávu všem uzlům v síti (broad=úplný),
  - *multicast* doručí zprávu skupině uzlů, které projeví zájem o tuto zprávu (multi=více),
  - *anycast* delivers a message to any one out of a group of nodes, typically the one nearest to the source.

## 11.2 Převod mezi MAC a IP adresou

### 11.2.1 ARP

- Address Resolution Protocol
- známe cílovou IP adresu, zjišťujeme MAC - abychom mohli IP paket umístit do rámce (Ethernet — linková vrstva) a odeslat
- vytváří ARP tabulku, MAC<->IP
- 1 MAC může mít více IP adres
- vytváří dynamický záznam (dynamické záznamy mají omezenou životnost - sekundy až desítky minut)

### 11.2.2 Proxy ARP

- router se při ARP odpovědích vydává za PC, které leží za ním
- vytvoření 1 logické podsítě z více fyzických sítí

### 11.2.3 RARP

- známe svoji MAC, ale neznáme svoji IP adresu
- pro konfiguraci bezdiskových stanic — dnes DHCP (Dynamic Host Configuration Protocol)
- umí předat jen IP (omezené možnosti)
- v síti je server se statickou ARP tabulkou
- následník BootP, zpětně nekompatibilní
- dynamické konfigurace stanic pro TCP/IP
- server přiděluje: staticky (dle MAC přidělí IP) nebo dynamicky (stanice dostanou volnou IP z definovaného rozsahu)
- stanice si „pronájem“ obnovují podle požadovaného serveru — evidence aktivních stanic
- stanice požádá pomocí broadcastu (se svojí MAC), server odpoví

### 11.3 Transparentní mosty

- sám se učí, neovlivňuje pakety
- tabulku si buduje z příchozích rámců (podle zdrojové MAC adresy a příchozího portu)
- rámce, jejichž cílová adresa v tabulce dosud není, se rozesílají na všechny porty (tzv. flooding)
- záznamy v tabulce mají časově omezenou platnost (při příchodu rámce s jistou zdrojovou adresou z jistého portu se časovač příslušné položky tabulky resetuje)
- rámce s cílovou adresou typu broadcast se rozesílají na všechny porty
- zasílá rámce podle tabulky se záznamy ve tvaru <MAC, port>

#### 11.3.1 Source routing

- Každý paket nese informaci kudy má projít.
- Cestu určuje odesílatel (průzkumný paket se šíří záplavově).
- Použití na linkové vrstvě.
- Využití v sítích Token Ring.
- <http://support.novell.com/techcenter/articles/ana19910501.html>

#### 11.3.2 Spanning Tree

- [http://www.ciscosystems.com/image/gif/paws/10556/spanning\\_tree1.swf](http://www.ciscosystems.com/image/gif/paws/10556/spanning_tree1.swf)
- potřebujeme předejít vzniku cyklu v síti (teorie grafů)
- způsob jak odstranit cykly je zablokovat port, který by způsobil vznik cyklu
- po výpadku portu/linky se strom aktualizuje a může blokováný port opět povolit
- algoritmus:
  1. volba kořenu stromu (root bridge) se provede podle nastavených priorit nebo podle unikátního ID
  2. vytvoření stromu s nejnižším ohodnocením
    - můžeme určit ceny linek
    - implicitně se vychází z přenosové rychlosti linky
    - aktivní (forwarding) porty jsou součástí stromu, ostatní jsou blokovány (blocked)
  3. root bridge generuje každé 2 sekundy BPDU zprávu a šíří ji stromem; všechny mosty ověřují, zda tuto zprávu na svém root portu slyší
- existují přechodové stavy portů, pro situaci kdy dochází k rekonfiguraci stromu (learning, listening)
- po výpadku se ustálí nový strom do 50 sekund

### 11.4 Algoritmy směrování

- Potřebujeme od směrovacího algoritmu/protokolu následující vlastnosti:
  - jednoduchost
  - robustnost
  - stabilita
  - optimalita
- Snaží se hledat optimální cestu:



- nejkratší (počet přeskoků, délka kabelu, přenosové cesty, ...)
- nejrychlejší (přenosové zpoždění, délka front, ...)
- nejlevnější (náklady, poplatky, ...)
- *Metrika*
  - určuje ohodnocení spojů v síti,
  - může být učena podle libovolného „měřitelného“ kritéria (přeskoky, celková doba přenosu, délka front, ...)
  - směrovací algoritmy hledají optimální cestu podle této metriky.
- Neadaptivní algoritmy
  - optimální cestu určí předem
  - nedochází k aktualizaci směrovací tabulky
  - výpadky některých spojů v síti mohou ochromit provoz sítě (nedojde k úpravě směrování přes jiné dostupné uzly)
- Adaptivní algoritmy
  - reagují na aktuální stav na síti
  - pravidelná aktualizace směrovacích tabulek

## 11.5 Routing x forwarding

- Centralizované směrování
  - rozhodování o směru provádí jeden uzel centrálně (route server)
  - route server vypočítává optimální cesty sám (statické/dynamické algoritmy),
  - forwarding provádí jednotlivé směrovače (edge-switches), které když neví jak mají směrovat, zeptají se route serveru,
  - výpadek route serveru ochromí provoz na síti,
  - v praxi není tento typ rozšířen.
- Distribuované směrování
  - Směrovače provádí routing i forwarding.
  - Uzly vzájemně spolupracují a hledají optimální cesty a aktualizují směrovací tabulky.
  - Výpočet může provádět každý uzel sám, nebo si mohou distribuovaně předávat výsledky.
  - Zásadní je jak často (pravidelně/po změně) a jak mnoho informací se přenáší.
  - Příklady:
    - \* vector distance routing (RIP, ...)
    - \* link state routing (OSPF, ...)
- Izolované směrování
  - uzly nespolupracují při hledání optimálních cest (každý uzel se rozhoduje sám)
  - méně efektivní než když uzly spolupracují
  - směrovače provádí routing i forwarding
  - Příklady:
    - \* záplavové směrování,

- \* metoda „horké brambory“ („hot potato algorithm“ je co nejrychleji se zprávy zbavit a to jakkoliv — libovolný směr; doplňková metoda např. když základní metoda selže a „teče do bot“),
  - \* náhodné směrování (paket pošleme libovolným směrem; cesta nemusí vést k cíli; použití v případě nouze jako metoda „horké brambory“)
  - \* metoda zpětného učení
  - \* source routing,
  - \* ...
- Hierarchické směrování
    - logickým důsledkem složité problematiky směrování je rozdělení (dekompozice) problému na menší části.
    - soustava samostatných oblastí (area)
    - uvnitř oblasti se řeší směrování samostatně
    - mezi různými oblastmi se směruje jen „obecně“
    - do každé oblasti je vymezený počet vstupních bodů

## 11.6 Záplavové směrování (flooding)

- varianta izolovaného směrování
- každý mezilehlý uzel rozešle paket do všech směrů, které existují s výjimkou směru, odkud paket přišel.
- existující cestu najde algoritmus vždy (robustní)
- snadná realizace (neexistují směrovací tabulky, není tedy ani nutnost žádných aktualizací směrovacích informací)
- nevýhodou je existence nadbytečných paketů (plýtvá přenosovou kapacitou)
  - řešení 1: vložení čítačů do paketů (TTL - time to live) určujících životnost paketu (vynulování čítače znamená zahození paketu)
  - řešení 2: pamatování si paketů, které prošly (ID) a eliminace duplikátů
- využití:
  - „obyčejná data“ jen ve speciálních sítích (vojenské)
  - pro „speciální data“ je běžnější (aktualizační informace, hledání cesty, ...)
  - pro speciální účely: distribuované databáze, distribuované vyhledávací služby
- Použito je obvykle *selektivní záplavové směrování* (selective flooding), při kterém není každý paket znovu vyslán všemi směry, ale pouze těmi, které jsou alespoň přibližně orientovány ke konečnému příjemci paketu.

## 11.7 Metoda zpětného učení

- Směrovač si získává potřebné informace o topologii sítě podle paketů které skrz něj prochází.
- Na počátku směrovač má prázdné směrovací tabulky a směruje záplavově.
- Když přijme směrovač paket od uzlu A ze směru 1 odvodí si že A leží ve směru 1. Když přijme paket od uzlu B pro uzel A ze směru 2, odvodí si, že uzel B leží ve směru 2 a ví, že je nutné jej poslat směrem 2. :-)
- Použití na linkové vrstvě u ethernetových mostů a přeppínačů, pro větší sítě není tato metoda vhodná.

## 11.8 Vector distance routing (DVA)

- každý směrovač si udržuje tabulku svých nejmenších vzdáleností od všech ostatních uzlů (vektor)
- směrovače si vyměňují přibližně informace o dosažitelnosti a ceně uzlů (uzel A za cenu X)
- výměna probíhá jen mezi přímými sousedy
- pro velké sítě, jsou velké objemy dat
- problém s počítáním do nekonečna (při každém nalezení neprůchodné cesty se zvyšuje cena o 1; trvá dlouho než hodnota signalizuje neprůchodnost)
  - Částečným řešením je neinformovat o ceně uzly od nichž dostal uzel informaci o ceně.
  - Částečným řešením je poslat uzlu X záměrně chybnou informaci (poisoned reverse) z uzlu Y „o nekonečně vysoké ceně cesty z uzlu X“. Uzel X pošle správnou cenu uzlu Y.
- problémy na síti (zánik cesty) se šíří pomalu, existence lepší cesty se šíří rychle

### 11.8.1 RIP (Routing Information Protocol)

- Metrikou je počet přeskoků s maximem 15 (16 má význam „nekonečno“).
- Vektor vzdáleností je rozesílán každých 30 s všem sousedním směrovačům.
- UDP na port 520
- Jestliže není vektor vzdáleností přijat do 180 s, je spoj označen za mrtvý a použije se metoda
- snadná konfigurace a jednoduchost
- nevhodný pro velké sítě
- <http://www.cisco.com/en/US/docs/internetworking/technology/handbook/RIP.html>

### 11.8.2 RIP 2

Oproti RIP 1 má navíc:

- zabezpečení komunikace mezi routery pomocí šifrovaného hesla,
- přenos síťových masek ve zprávách mezi routery umožňuje implementovat také subnetting.

### 11.8.3 RIPng

- Routing Information Protocol next generation
- Rozdíly oproti RIP 2:
  - RIPng does not need to implement authentication on packets.
  - There is no support for multiple instances of RIPng.
  - There is no support for RIPng routing table groups.
- UDP na port 521

## 11.9 Link state protocol (LSA)

- stabilnější než vector distance protokoly
- menší režijní zátěž sítě (navíc stačí posílat jen při změně)
- každý uzel monitoruje průchodnost cesty ke svým sousedům (změnu distribuuje po celé síti)
- každý uzel má kompletní informaci o topologii sítě a průchodnosti všech spojů
- každý uzel si počítá optimální cesty sám a chyba tedy neovlivní ostatní uzly
- není vhodné pro opravdu velké sítě
- příkladem je OSPF protokol.

## 11.10 OSPF

- Open Shortest Path First (OSPF)
- link state protokol + hierarchické směrování
- po zapnutí si každý uzel zjistí přímé sousedy (HELLO protokol)
- průběžně zjišťuje dobu odezvy svých sousedů (ECHO pakety)
- každý uzel pravidelně (stačí po změně pro snížení zátěžení sítě režijními informacemi) záplavově rozesílá všem ostatním uzlům své naměřené hodnoty dostupnosti svých přímých sousedů
- každý uzel postupně sbírá informace od ostatních uzlů a zjišťuje tak topologii sítě
- předpoklad hierarchie:
  - AS — autonomní systém (AS boundary router)
    - \* použití EGP (Exterior Gateway Protocol, externí směrovací protokol; nutná je stabilita; např. BGP)
  - backbone area — páteřní systém v rámci AS (backbone router)
    - \* použití IGP (Interior Gateway Protocol, interní směrovací protokol; nutná pružnost a rychlá reakce na změny; např. RIP, ...)
  - area — oblast v rámci AS, připojenou k backbone area (area border router a podřízené routery jsou internal router)
- detailní směrovací informace neopustí příslušnou oblast (area)
- mezi oblastmi se komunikace směruje jen přes vymezené body:
  - z area do area přes backbone area
  - z backbone area do backbone area přes autonomní systém
- <http://www.abclinuxu.cz/clanky/site/dynamicke-routovani-ospf-3-topologie-zabezpeceni>
- <http://ist.marshall.edu/ist362/pics/OSPF.gif>

## 12 Transportní protokoly pro přenos dat

### 12.1 Transportní vrstva

- transparentní, spolehlivý přenos dat s požadovanou kvalitou,
- vyrovnává různé vlastnosti a kvalitu přenosových sítí,
- převod transportních adres na síťové,
- nestará se o směrování.

## 12.2 Transportní protokoly

Transportní protokoly jsou definovány na transportní (4.) vrstvě.

- AEP
- AMTP (náhrada za SMTP)
- CUDP
- IL
- NBP
- NetBEUI (NetBIOS Extended User Interface)
  - vznik z původního NetBIOS,
  - nepodporuje směrování, proto jen pro lokální síť,
  - obsahuje prvky vrstvy síťové i transportní.
- RTMP
- SMB (Server Message Block)
- IPX/SPX (Internet Packet Exchange/Sequenced Packet Exchange)
  - Novell
- TCP (Transmission Control Protocol)
- UDP (User Datagram Protocol)
- SCTP (Stream Control Transmission Protocol)
  - přenos telefonní signalizace (SS7) po IP
- RTP (Real-time Transport Protocol)
  - paketový formát pro doručování zvukových a obrazových dat po internetu
  - určení užitečného zatížení
  - číslování sekvencí
  - časové razítkování
  - sledování přenosu
  - nepřerušovaný přenos hlasových balíčků (paketů) na internetu
  - RTP Control Protocol (RTCP) slouží k řízení RTP sezení a pro sledování kvality toku. Protokol RTCP obvykle využívá port o jedno číslo větší než RTP.
- RUDP (Reliable User Datagram Protocol)
  - spolehlivý protokol pro přenos dat
  - vyvinut v Bellových laboratořích pro operační systém Plan 9
  - rozšiřuje UDP:
    1. potvrzení o doručení paketu
    2. ovládání přetížení sítě
    3. přeposlání ztracených paketů
    4. overbuffering
  - není standardizován

## 12.3 Adresování

Na adresování aplikačních procesů se používají, podobně jako na adresování v protokolu UDP, celočíselné identifikátory, tzv. aplikační porty (sekety).

Protokol TCP a protokol UDP používají nezávislý systém číslování portů. Proto číslo portu TCP 80 není totéž co UDP 80. Jedná se o dva různé porty.

## 12.4 Základní vlastnosti TCP

- <http://www.cs.vsb.cz/grygarek/SPS/> Počítačové sítě Petr Grygárek (VSB)
- <http://www.cs.vsb.cz/grygarek/SPS/lect/TCP/tcp.html> Informace k TCP
- <http://www.cpress.cz/knihy/tcp-ip-bezp/CD-0x/9.html> Informace k TCP

Protokol TCP (Transport Control Protocol) pracuje podle modelu OSI v transportní vrstvě. Hlavní funkcí protokolu je vytváření, udržování a rušení transportních spojení, prostřednictvím kterých komunikují aplikační procesy v koncových uzlech sítě IP. Pro protokol TCP jsou charakteristické následující vlastnosti:

### 12.4.1 Vlastnosti

- poskytování spolehlivé spojové transportní služby koncovým procesům
- multiplexní režim práce pro koncové procesy uzlů sítě (několik procesů současně)
- duplexní režim přenosu údajů mezi koncovými procesy (piggybacking)
- služba přenosu dat v podobě proudu přenášených bytů, stream segmentů protokolu TCP
- potvrzování přenosu dat (ACK) a opakování přenosu segmentů
- adaptivní režim přizpůsobení parametrů protokolu podle stavu spojení
- řízení toku systémem klouzavého okna (Sliding window) a pomocí přijímacích a vysílacích bufferů

### 12.4.2 Vlastnosti spojení

Spojení je definováno jako dvojice soketových adres komunikujících koncových procesů. Aplikace si vyměňují data prostřednictvím vytvořeného spojení formou proudu bytů.

- koncepce klient–server
- aplikační procesy odevzdají vrstvě TCP údaje pro vytvoření spojení — číslo zdrojového a cílového aplikačního portu.

Při vytváření spojení rozlišujeme dva režimy otevření:

**pasivní režim** přijímá na veřejně známém aplikačním portě volání klientů, kteří požadují vytvoření transportního spojení se serverem.

**aktivní režim** je aktivován klienty, kteří vysílají požadavky na aplikační port serveru, otevřený v pasivním režimu

### 12.4.3 Vytvoření spojení

Při vytváření spojení se vykonává synchronizační procedura *Three - Way Handshake* (SYN, SYN/ACK, ACK). Během této doby si obě strany vymění a potvrdí výchozí sekvenci čísla.

Pakety TCP mají v poli řízení spojení nastavené bity SYN a ACK, které druhé straně signalizují žádost o vytvoření spojení.

Procedura je dostatečně robustní a odolná proti výpadkům spojení. Stejně tak je schopná zamezit duplicitám sekvencí čísel.

#### 12.4.4 Ukončení spojení

K ukončení transportního spojení dochází po přerušení přenosu dat zapříčiněného libovolnou stranou spojení. Uzel, který provádí ukončení spojení TCP vyšle zprávu s nastaveným bitem FIN řídicího pole. Po přenesení druhá strana přerušení spojení potvrdí zprávou FIN.

#### 12.4.5 Řízení přenosu dat

Pro aplikaci je spojení jako proud bytů. Zabezpečený přenos TCP protokolem musí umožnit přenos proudu bytů, tj. přenášená data musí být přeneseny ve správném pořadí a nesmí dojít k jejich ztrátě nebo zahlcení příjemce.

Protokol TCP zabezpečuje řízení přenosu následovně:

- data aplikačního procesu jsou ukládána do vysílací paměti (Output Buffer), z které jich TCP odebírá postupně po segmentech, které se doplňují o hlavičku řídicích údajů do TCP paketu.
- integrity přenosu se zabezpečuje číslováním přenášených bytů proudu prostřednictvím sekvenčních čísel *SN* (pořadové číslo prvního bytu v segmentu).
- příjemcí strana potvrzuje příjem segmentu pozitivním potvrzením ACK prostřednictvím sekvenčního čísla nejbližšího očekávaného bytu v segmentu *AN*.
- aby nedošlo k zahlcení příjemce vyčerpáním příjemcího bufferu, příjemcí strana odesílá při potvrzení příjmu vysílací straně informaci o dostupné velikosti bufferu, tzv. příjemcího okna. Tak ovlivní velikost odesílaných segmentů metodou Sliding Window.
- výpadek, poškození nebo ztráta segmentu při přenosu se ošetří opakováním přenosu segmentu. Po vypršení časového intervalu na příjem potvrzení segmentu (Retransmission Timeout) od příjemce strany, musí vysílač znovu zopakovat přenos ztraceného segmentu.

Hodnota časového intervalu na příjem potvrzení se mění dynamicky podle aktuálního stavu sítě adaptivním výpočtem (Karn-Jacobsonův algoritmus) na základě předpokládaného zpoždění SRTT (Smoothed Round Trip Time) a předpokládané průměrné odchylky od skutečného zpoždění SDEV (Smoothed Deviation).

### 12.5 Základní vlastnosti UDP

User Datagram Protocol, tj. UDP je protokol určený pro aplikační protokoly a procesy.

- rychlý a nezabezpečený přenos paketů
- bez potvrzování příjmu a opakování přenosu (datagramová služba)
- protokol se přenáší v datové části IP paketů
- Na adresaci aplikačních procesů se používají celočíselné identifikátory (aplikační porty - socket) přidělené aplikacím před vlastní komunikací. Kombinace IP adresy s adresou aplikačního portu tvoří úplný identifikátor koncového procesu v IP síti a nazývá se socketová adresa (IP-port).