# 1 Teorie informace

**Informace** je soubor údajů. Sdělování informace zahrnuje vznik, transformaci, přenos a uchování těchto údajů. Informace určuje i vztah údajů a okolního světa.

**Pojmy:**

- *Zpráva* = posloupnost rozlišitelných prvků prezentujících informaci
- *Symbol* = rozlišitelný prvek zprávy (grafický symbol je *znak*)
- *Délka zprávy* = počet symbolů ve zprávě
- *Abeceda* = konečná neprázdná množina všech možných symbolů zprávy
- *Kódování* = převod zprávy v jedné abecedě na zprávu v abecedě druhé nesoucí stejnou informaci
- *Signál* = fyzikální nositel zprávy

**Množství informace ve zprávě** $I(a)$ je založeno na pravděpodobnosti výskytu dané zprávy. Pokud se zpráva objevuje pouze někdy, pak je její výskyt mnohem důležitější. Matematicky řečeno $p(a) > p(b) \Rightarrow I(a) < I(b)$. Množství informace nezávislých zpráv je dáno součtem $I(ab) = I(a) + I(b)$.

**Shanonova věta:** $I(a) = -log_2\, p(a)$ definuje funkci, která vyhovuje definici $I(a)$ a je obecně používána.

**Střední hodnota očekávané informace** udává nejpravděpodobnější hodnotu informace zprávy (nejpravděpodobnější zprávu). $I = -\sum_{i=1}^{n} p(a_i) log_2\, p(a_i)$

**Entropie** $H(X)$ je míra neurčitosti náhodného jevu ($n$ možných výsledků). O zprávě uvažujeme také jako o náhodném jevu, proto
$I = H(X) = -\sum_{i=1}^{n} p(x_i) log_2\, p(x_i)$ a platí:

- $0 \leq H \leq log_2(n)$
- $p(x_{i=k}) = 1 \wedge p(x_{i\neq k}) = 0 \Rightarrow H = 0$ (není neurčitost, jediný výsledek)
- $\forall i \in \{1, \ldots, n\} :\ p(x_i) = \frac{1}{n} \Rightarrow H = -\sum_{i=1}^{n} (\frac{1}{n} log_2 \frac{1}{n}) = log_2 n$ (maximální entropie, rovnoměrné rozložení pravděpodobnosti)

Množství informace není závislé na entropii, entropie je relativní k očekávanému výsledku (např. padne šestka nebo nepadne – 2 stavy, proto entropie klesá, ale množství informace padnutí šestky je stejné jako když hodnotíme padnutí jakékoli hodnoty).

**Diskrétní zpráva** obsahuje symboly abecedy v daném pořadí. Pravděpodobnost následujícího znaku závisí na $n$ znacích předchozích (0 pro nezávislé znaky).

**Relativní entropie** je měřena vůči maximální, tedy $h = \frac{H}{H_{max}}$. Pokud je entropie menší než maximální ($h < 1$), zdroj plně nevyužívá svou abecedu a dochází k *redundanci* $r = 1 - h$. Čím větší redundance, tím větší délka zprávy, což se využívá při redundantním kódování.

**Kódování** je převod zprávy v jedné abecedě na zprávu v abecedě druhé nesoucí stejnou informaci.

**Kódování pro zvýšení redundance** zvyšuje počet nevyužitých znaků abecedy, tím se zvětší možnosti opravy a detekce chyb.

**Kódování k potlačení redundance** se snaží co nejvíce využít danou abecedu.

**Optimální kódování** potlačuje redundanci nejvíce jak je to možné, nejznámější je *Huffmanovo kódování*.

**Huffmanovo kódování** je prefixový kód (žádné kódové slovo není prefixem jiného, není potřeba oddělovačů), který častým symbolům přiřazuje krátká slova. Postup: seřadit symboly podle pravděpodobnosti, po dvou vždy nejnižších pravděpodobnostech spojit a sečíst, slovo se pak čte od kořene.

# 2 Bezpečnostní kódy, bitové chyby

**Bezpečnostní kódy** jsou zástupci kódování pro zvýšení redundance. Redundance je využita k detekci a opravě chyb.

**Rozdělení kódů:**

- *Systematické kódy* – při přenosu lze rozlišit bity informační a bity zabezpečovací (kódy $n : k$)
- *Nesystematické kódy* – bity informační a zabezpečovací nelze rozlišit
- *Detekční kódy* – zjišťují přítomnost chyby
- *Korekční kódy* – zjišťují a samy opravují chyby (znají pozici chyby)

**Hammingova vzdálenost** je vzdálenost slov abecedy (tedy v praxi počet bitů, které se musejí změnit, abychom jedno slovo změnili na druhé). Důležitá je nejmenší taková hodnota v abecedě, $d_{min}$ (tedy dvě nejbližší slova). Příklad: 000 a 001 mají vzdálenost 1bit, 010 a 101 mají vzdálenost 3b.

**Detekce chyb:** $d_{min} = x + 1$, počet chyb musí byt o jedničku menší než hammingova vzdálenost.

**Oprava chyb:** $d_{min} = 2x + 1$, hammingova vzdálenost musí být větší než dvojnásobek počtu chyb, které se mají opravit.

**Elementární kódy:** $k$ z $n$ (používají $k$ jedniček z $n$ bitů), příčná a podélná parita.

**Lineární kódy** jsou založeny na Galoisových polích a vektorových prostorech, lineární kód pak vytváří podprostor v GF(2). Generující matice obsahuje bázi podprostoru doplněnou o jednotkovou matici.

Generující matice $G = \left(\begin{array}{cc|ccc} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 \end{array}\right)$ zakóduje slovo [11] (násobením) na slovo [11|000] (dva datové a tři kontrolní bity).

Kontrolní matice vznikne použitím báze (pravé části) a jednotkové matice v opačném pořadí:
$H = \left(\begin{array}{cc|ccc} 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \end{array}\right)$, která se musí ještě transponovat pro násobení kódovým slovem zleva, tedy $\begin{pmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$. Pokud tedy kontrolujeme slovo [11000], pak je výsledkem násobení [000], což je správně. Pro slovo [11100] však výsledek [100] nulový není.

**Hammingovy kódy** jsou lineární kódy sestaveny tak, aby kontrolní *syndrom*, pokud není nulový, obsahoval pozici chybného bitu. Pro hammingovu vzdálenost 3 je pak polynom $c_1c_2a_3c_4a_5a_6a_7$.

**Cyklické kódy** využívají generující polynom a primitivní prvky. Zpráva se vyjádří jako polynom, ten se vynásobí nejvyšším prvkem generujícího polynomu a pak se jím vydělí, syndrom je zbytek po dělení. Typickým zástupcem je CRC.

**Konvoluční kódy** nejsou blokové, ale průběžné. Reprezentace formální mocninnou řadou.

# 3 Bitová synchronizace

**Bitové chyby** na linkové vrstvě se kontrolují pomocí detekčních a korekčních kódů.

**Checksum** (kontrolní součet) je využíván v hlavičkách některých protokolů (většinou se jedná opravdu jen o součet).

**Bitová synchronizace** znamená synchronizaci komunikace pouze pomocí bitů, např. *start-stop bits*.

**Linkové kódy** jsou způsob přenosu digitální informace pomocí analogového nosiče (jak namodulovat nuly a jedničky na změny amplitudy signálu).

- Externí hodiny (synchronní přenos) – můžeme přenášet data pouze jako maximální/minimální amplitudu
- Jediný nosič (asynchronní přenos) – hodiny musejí být součástí signálu, nelze přenášet $n$ nul pouze tak, že signál se nebude měnit, je potřeba synchronizace.
- Unipolární vs bipolární kódy
- Binární vs. ternární kódy
- Přímé vs diferenční kódy (změny sgnálu probíhají za přítomnosti jedničky, jinak se signál nemění)

**Manchester** kóduje jedničku jako změnu z MAX na MIN a nulu jako změnu opačnou, hodiny pak mají dvojnásobnou frekvenci oproti datům.

**Kódy mBnB** – mapují $m$ bitů na $n$ bitů (Machester je 1B2B), používají se i 4B5B, 8B10B nebo 64B66B a to pro vyšší rychlosti přenosu (10Gbit), protože je snaha odstranit redundanci, ale při zachování synchronizace.

**Zrychlení** je také možné zavedením více stavů napětí na lince (dvojice 00, 01, 10, 11 mají rozdílná napětí).

# 4 Paketová synchronizace a správa chyb

**Linková vrstva** detekuje a opravuje pouze bitové chyby, nejčastěji na základě CRC (detekuje všechny jednoduché nebo liché chyby, většinu dvojitých). Vyšší vrstvy pak mají pro bity jen kontrolní součet.

**Paketové chyby** zahrnují ztrátu paketu (většinou zahozen kvůli přeplnění, nebo timeout) a duplikaci paketu (chyba v komunikaci). Také se mohou objevit pakety cizího spojení nebo jsou pakety přijaty ve špatném pořadí.

**Pakety ACK nebo NACK** potrvrzují nebo oznamují chybu předchozího doručení.

**Sekvenční čísla** (čísla z posloupnosti, která se posouvá pro každý paket) lze použít pro detekci ztráty, duplikaci nebo doručení ve špatném pořadí. Cizí paket je rozpoznán pouze, pokud se číslo liší diametrálně. Je důležitý počet bitů, jelikož omezuje délku sekvence.

**Chyba vložení paketu** neexistuje u spojovaných sítí. V internetu je možné přijmout paket z cizího spojení. Možná řešení:

- *Číslování spojení* – zabírá místo v hlavičce a musí se pamatovat poslední číslo
- *Znovupoužití portu až po MPL* (Maximum Packet Lifetime) – příliš dlouho, co restart serveru?
- *Přidělování portů postupně* – používá se, ale restart nebo příliš mnoho spojení může porušit
- *Sekvenční číslo závislé na čísle portu* – používá se, stále problém restartu
- *Po restartu počkat dobu MPL* – používá se

**Trojcestný handshake** je typický postup pro TCP, které využívá sériové přiřazení portů, sekvenční čísla se generují z hodin a po pádu se čeká. Při navázání spojení se posílá $SYN(x)$, odpovědí je $SYN(y), ACK(x+1)$ a poté ještě $SYN(x+1), ACK(y+1)$ s odpovědí $ACK(x+2)$. Pak se mohou posílat data.

**Detekce ztráty paketu** je založena na pozdním (nebo žádném) doručení paketu ACK a také na mezeře v sekvenčních číslech.

**Znovuzaslání paketu** je potřeba, pokud druhá strana neoznámí úspěšné přijetí do časového intervalu *timeout*.

**ECW** (Error Control Window) je množina paketů, které ještě nebyly potvrzeny (množina proto, že pakety se neposílají po jednom ale souběžně kvůli rychlosti).

**Go-back-N algoritmus** pošle ACK pro všechny pakety v ECW, které jsou správně v pořadí, ostatní se pošlou znovu (i když některé byly v pořádku), větší zátěž sítě, ale používá se v TCP.

**Selective Retransmit** zasílá v ACK sekvenční číslo, po které jsou pakety v pořádku, ale také bitovou mapu o jisté délce, která udává stav dalších paketů. Tak se server dozví o již dobře přijatých paketech a zašle pouze některé.

**Parametr** $a$ je poměr doby, jak dlouho jde zpráva od zdroje k cíli a doby, jak dlouho trvá vyslání zprávy. $a$ je tedy malé, pokud se dlouho vysílá, ale přenos je rychlý, $a$ je velké, pokud se zpráva vyšle rychle, ale k cíli jde dlouho. Malé $a$ je u LAN, velké třeba u satelitního připojení.

$a$ **v kolizních protokolech** – pokud je $a$ malé, pak se lehce zjistí kolize, protože se hlídá linka při vysílání zprávy. Pokud je však $a$ velké, pak může nastat problém na lince po odeslání zprávy a to je problém pro detekci a korekci kolize.

# 5 Metody řízení vícenásobného přístupu a protokoly lokálních sítí

**Vícenásobný přístup** je metoda přístupu na sdílené médium. Musí se řešit kolize a doručování správnému příjemci. Snaha maximalizovat propustnost a minimalizovat čekání, férovost přístupu.

**Centralizované řešení** – existence moderátora, který řídí přenos a určuje, kdo bude komunikovat, jednoduché, ale omezuje podřízené (neznají všechny sousedy, nemají takový přístup k lince jako nadřízený).

**FDMA** (Frequency Division Multiple Access) – rozdělení pásma podle frekvence, ale frekvencí je málo, pásmo není využito na 100%.

**TDMA** (Time Division Multiple Access) – přidělení pásma na časovou jednotku, lepší využití, ale potřeba synchronizace (statické rozdělení by mrhalo časem).

**Frequency Hopping** – FDMA + TDMA, každé přidělení jednotky je spojeno i s danou frekvencí, nepoužije se celé pásmo pro jednoho uživatele.

**CDMA** (Code Division Multiple Access) – zakóduje bit delším kódem, který je odlišitelný od kódu ostatních uživatelů pásma, může využívat frequency hopping, složitější, malé využití pásma.

**Duplexing** – využití kanálu pro komunikaci oběma směry, používá se FDD a TDD, důležité pouze pro bezdrátové médium (dráty jsou položeny v obou směrech).

**Přepínání okruhů** je centralizovaná metoda řízení, podřízený požádá o vytvoření cesty, moderátor ji alokuje a umožní komunikaci. Přiděleno na danou dobu přenosu, nesoutěží se o medium.

**Výzva (polling)** – řídící stanice se ptá, kdo chce vysílat. Při malém počtu aktivních stanic je vysoká režie.

**Sondování (probing)** – každá stanice má své číslo a řídící jen označí číslem, kdo je na řadě, stanice se pozná a komunikuje. Řídící stanice hledá nové stanice.

**Rezervační schéma** – stanice mají rezervovány sloty pro přenos dat, rezervační zprávy mají speciální minisloty a bojuje se pouze o tyto (urychlení).

**Distribuované řešení** – neexistuje centrální prvek, je třeba se dohodnout, více spolehlivé, menší zpoždění, složité.

**Decentralizovaná výzva** – místo výzvy od centrální stanice je zde přidělený časový slot, ale při neaktivní stanici se mrhá prostředky.

**Decentralizované sondování** – založeno na stromu, nejprve zkusí poslat levá strana, bez kolize pak zasílá pravá strana, při kolizi se posílá znovu s omezením na některé uzly.

**CSMA** (Carrier Sense Multiple Access) – stanice testují, zda je médium volné, pokud ano vyšlou data, pokud ne čekají náhodnou dobu.

**CSMA/CD** (Collision Detection) – při kolizi je navíc poslán *jam* signál a ostatní stanice počkají déle.

**CSMA/CA** (Collision Avoidance) – kolizi se předchází potvrzováním povolení k přenosu, využití v bezdrátové komunikaci, kde existují skryté stanice a nechráněné stanice, proto rozšířeno o *busy-tone*, kdy stanice vysílá oznámení o své komunikaci.

**Ethernet** – používá CSMA/CD, dnes ale není potřeba kvůli topologii za použití přepínačů.

**Token-passing** – jediná stanice má token a proto může komunikovat, po dokončení jej pošle následníkovi v logickém kruhu. Pokud dlouho nebyl token a nekomunikuje se na síti, vytváří se nový (asi se ztratil). Kruh může být propojen dvojitě kvůli možnosti chyby jedné stanice.

**Aloha** – starý protokol, bez testování se vyšle paket a čeká se na ACK, pokud nepřijde nebo došlo ke kolizi opakuj po nějaké náhodné době. Jednoduché, ale používá jen 18% kapacity, při zátěži neefektivní. *Slotted aloha* určuje okamžik, kdy se může vysílat a jak dlouho, zlepšuje vlastnosti. *Reservation aloha* ustanoví sloty, ve kterých mohou stanice vysílat.

# 6 Řízení toku dat

**Řízení toku dat** zajišťuje komunikaci optimální rychlostí. Nízká rychlost plýtvá prostředky, vysoká tvoří chyby přetečením a zbytečnou režii.

**Otevřená smyčka** – stanice popíše požadavky na rychlost (deskriptor) a odešle, síť a druhá strana přijmou/odmítnou, dohodnutou rychlostí se odesílá

**Uzavřená smyčka** – stanice monitoruje možnou rychlost na síti a posílá (určí z doby čekání), jde o zpětnou vazbu

**Hybridní smyčka** – dohodne se minimální rychlost, ale může se zasílat i rychleji a ověřují se chyby

**Deskriptor** musí dobře reprezentovat chování v síti, ale také musí být uchovatelný a použitelný. Používá se špičková/průměrná rychlost apod.

**Metody řízení toku:**

- *On/Off* – zpětná vazba obsahuje signál On/Off (vysílej plnou rychlostí/nevysílej), sériové linky
- *Stop and Wait* – čeká na ack, pak teprve pošle další paket
- *Static window* – čeká na ack, ale neposílá po jednom paketu, nýbrž po okně paketů
- *DECbit* – jeden bit paketu označuje, že je okno příliš velké, routery si jej přeposílají

**TCP** využívá dynamické okno DECbit ale bez podpory routerů, zvětšuje se aditivně když nedojde k chybě, zmenší se multiplikativně při timeoutu

# 7 Propojování sítí

**Vrstvy Internetu:**

1. *Linková* – někdy rozdělena na linkovou a fyzickou, ARP protokol, MAC adresy
2. *Síťová* – IP protokol, IP adresy
3. *Transportní* – TCP/UDP protokol, adresy procesu (porty)
4. *Aplikační* – samotné aplikace

**Vrstvy ISO/OSI** jsou obecnější:

1. *Fyzická* – přenos bitů
2. *Linková* – spojení počítačů mezi sebou
3. *Síťová* – spojení koncových bodů cesty
4. *Transportní* – spojení jednotlivých programů
5. *Relační* – správa sezení
6. *Prezentační* – prezentace dat
7. *Aplikační* – samotné aplikace

**IP adresa IPv4** se skládá z adresy sítě (a popř. subsítě) oddělené pomocí masky od adresy počítače.

**Propojovací jednotky v paketových sítích:**

- *Opakovač* – L1, pouze zesiluje signál novým vysláním
- *Hub* – L1, víceportový opakovač (propojí několik počítačů, ale všechny dostávají pakety všech)
- *Most* – L2, propojuje oddělené sítě, podle MAC adresy přeposílá pakety jen na jednu stranu
- *Switch* – L2, je to vlastně most, jen jiný název, switch je obecný název (používá se L1 switch apod.)
- *Směrovač* – L3, podle IP adresy přeposílá na výstup, umí směrování
- *Aplikační brána* – L7, poskytuje služby, které přeposílá do jiných sítí

**DHCP, DNS, ...** jsou pojmy notoricky známé

# 8 Techniky přepínání a architektury přepínačů

**Přepínač** je zařízení, které přepravuje signály ze svých vstupů na své výstupy. Výstupy vybírá podle řídící informace (např. HW adresa nebo IP v počítačové oblasti).

**Přepínání okruhů** – Před zasíláním dat je vytvořena cesta v síti, po které se pak posílají. Není potřeba hlaviček, řízení je dáno cestou.

**Přepínání paketů** – Data se dělí na malé části (pakety) a každá je přenášena nezávisle. Je potřeba hlavička, podle které se data sestaví v cíli.

**Spojované přepínání** vytváří cestu na úrovni přepínačů, přepínání okruhů tak funguje vždy, přepínání paketů umí emulovat takové chování (ATM).

**Nespojované přepínání** je možné pouze u přepínání paketů, je běžné v internetu.

**Kapacita přepínače** je maximální rychlost přenosu, pokud jsou všechny cesty přepínačem aktivní. Snaha maximalizovat (vyvarovat se blokování a ztrátám).

**Generická struktura přepínače:**

1. *Input Buffers* – každý port má vstupní frontu
2. *Port Mappers* – říkají, kam data ze vstupů mají jít
3. *Switching Fabric* – posílá data ze vstupu na výstup
4. *Output Buffers* – fronty na výstupu

**Switching Fabric pro okruhy:**

- *Multiplexor a demultiplexor* – $n$ vstupů a propojení $n$-krát rychlejší, TDM, kde vzorky se cyklicky střídají, jedná se jen o skládání za sebe, není třeba hlaviček
- *Křížový přepínač* – řádky jsou vstupy a sloupce výstupy, křížení se může spojit a tak vést data, nastavení ale potřebuje $n^2$ časových jednotek
- *Vícestavový křížový přepínač* – zapojuje se hierarchicky pro snížení složitosti, ale horší hledání cesty

**Mapovače portů** jsou pro přepínání okruhů jednoduché, pouze se ustaví jednou spojení, pro pakety složitější. Je potřeba najít nejdelší prefix sítě, kam paket patří, využívá se struktura *trie*.

**Generace přepínačů:**

1. CPU a NIC, mapování běží jako proces, nestíhá procesor
2. CPU a specializované karty, které již mají mapování v sobě, nestíhá sběrnice
3. paralelní přepínací síť s kontrolním procesorem

**Přepínací sítě:**

- *Křížový přepínač* – spojení řádku a sloupce na potřebnou dobu k přenosu paketu
- *Bufferovaný křížový přepínač* bufferuje body křížení kvůli kolizím
- *Broadcast* – vstupy a výstupy označeny tagem, paralelní porovnání, spíše pro malé switche
- *Složená síť* rozdělená na malé elementy, které přepínají mezi dvěma možnostmi a mají buffer, skládají se hierarchicky
- *Banyan* – instance složené sítě, výstupní porty označeny číslem, každý bit určuje přepnutí jednoho elementu, nejjednodušší samosměrující síť; buffery u elementů jsou příliš drahé, většinou se kontroluje dostupnost předem (např. výzvou), dražší možnost je více paralelních sítí

**Buffery:**

- *Na vstupu* – jednoduché, ale první paket blokuje ostatní (HOL), malá účinnost; vylepšení: bufferuje se nejen pro každý vstup ale v něm pro každý výstup, složité a je potřeba řízení
- *V síti* – každý element má buffer (nákladné a složité), zrychlení podle počtu vstupů
- *Na výstupu* – není blokování, ale buffery musejí být velmi rychlé oproti výstupní lince, proto se některé sdílejí
- *Sdílená paměť* – paměť přístupná odevšad, ale pak je pomalé čtení a zápis

# 9 Směrování

**Směrování** se oproti přepínání nezabývá jedním přepínačem, ale dopravou dat mezi koncovými uzly. Proto je možné až na síťové úrovni. Směrovače udržují směrovací tabulky, které obsahují cílovou adresu a adresu sítě, do které se má paket odeslat, aby cíle dosáhl nejoptimálnější cestou. Tyto tabulky je zapotřebí dynamicky měnit. Je tedy zapotřebí algoritmů pro nalezení *optimální cesty*.

**Lokální rozhodování** je nevýhodné, jelikož optimální cesta je globálním problémem, ale směrovač má jen lokální informace. Je zapotřebí protokol, který tyto informace nastaví dle globálního kontextu (komunikace mezi směrovači).

**Algoritmus Distance Vector** udržuje na každém směrovači vektor $[cil, cena]$ pro všechny cíle. U sousedů je cena známá, u zbytku je zpočátku nekonečno, ale výměnou tabulek se sousedy se cena doplní. Varianta Path Vector nese i celou cestu ve vektoru. Má problémy se zacyklením a rychlou adaptací ve větší síti.

**Algoritmus Link-State** posílá po síti vektory vzdáleností ke všem cílům. Jednotlivé směrovače upraví chybnou nebo chybějící informaci. Při budování směrovač dodá své sousedy a ostatní jsou v nekonečnu, při předání dál, si tyto informace další směrovač doplní a opět je rozešle. Databáze může být poměrně veliká.

**Metriky směrování** udávají cenu cesty mezi dvěma uzly. Je možné použít statické (v praxi počet skoků) nebo dynamické (např. délka fronty směrovače, propustnost apod.) metriky. Dynamické i když jsou adaptabilnější mohou způsobovat oscilace.

**Hierarchické směrování** je zapotřebí ve velkých sítích, jelikož tabulky by byly příliš veliké a směrování pomalé, navíc režie směrovačů by síť zahltila. Síť se dělí na domény a počítače uvnitř domény nevědí o počítačích vně, používají brány pro komunikaci s okolím. Brány pak vědí o ostatních branách a přeposílají pakety. V internetu je hierarchie podle čísla sítě (popř. subsítě).

**Interní a externí protokoly** – jelikož v internetu jsou tři typy domén (backbone, autonomní oblasti a LAN), používají se specializované prokoly pro směrování uvnitř a vně těchto domén. Vznikají pak ale problémy s propojením různých protokolů.

**Externí protokoly** méně důvěřují informacím, jelikož ostatní externí směrovače jsou cizí. Mohou mít zadní vrátka pro ruční směrování. Protokoly EGP (DV) nebo dnes více BGP (Path Vector).

**Interní protokoly** počítají s důvěryhodnými sousedy, jsou jednodušší. Protokoly RIP (DV) – jednoduché, daemon, pro menší sítě; OSPF (LS) – hierarchické, i větší sítě.

**Směrování s pravidly** zasahuje do metriky na základě různých pravidel (úroveň služeb, nedůvěra, denní doba apod.)

# 10 Multicastové směrování

**Multicast** – účastníci skupinové komunikace jsou členy multicastové skupiny, paket poslaný na adresu skupiny je rozeslán všem členům. Směrovače musejí dynamicky měnit tabulky členů a směrovací tabulky při přidání nebo odebrání člena.

**Kostra grafu** zaručuje nejúspornější rozesílání paketů. Je možné multicast simulovat násobným unicastem, ale to je mrhání zdroji.

**LAN** – jednodušší situace dá se buďto využít broadcast nebo mapování multicast na MAC

**WAN** – problém, časté přepojování klientů přináší rychlé změny topologie

**Používané protokoly:**

- *MDVRP* (Multicast Distance Vector Routing Protocol) – používá záplavu a poté prořezání
- *MOSPF* (Multicat OSPF) – mnohem složitější, udržuje velké tabulky
- *CBT* (Core Based Tree) – obsahuje centrální člen, je rychlejší správa, ale úzké hrdlo v core
- *PIM* (Protocol Independent Multicast) – využívá více strategií podle topologie (hustá skupina je lepší záplavou, řídká CBT)

# 11 Plánování zdrojů

**Plánování zdrojů** je postup, jak přidělovat sdílené médium tak, aby se zachovala spravedlnost a zamezilo se zbytečnému soupeření na úkor výkonu. Určuje se pořadí přístupu a spravují se fronty čekajících.

**Typy plánování:**

1. *Best-effort* – možné ztráty, realtime, neadaptivní, zaručena propustnost
2. *Guaranteed* – beze ztrát, non realtime, adaptivní, zaručen výsledek

**Plánovací pravidla** zaručují různé stupně QoS (Quality of Service) různým uživatelům. Mohou alokovat šířku pásma, zaručit zpoždění nebo limitovat ztráty. Také určují spravedlivost nebo preferenci. Musejí být jednoduchá, účinná, dobře implementovatelná a rychlá.

**Plánovací algoritmy:**

- *Priorita* – nespravedlivá, možné vyhladovění, jednoduchá, v rámci úrovně FCFS (First Come, First Served), kolik úrovní priority?
- *Non-work-conserving* – uchová paket, dokud není požadován, vyšší zpoždění, menší kolísání, složitý, zmenší buffery v síti
- *Best-effort* – cyklicky prochází fronty a z každé bere paket (ideálně nekonečně malý, ale nelze)

**GPS** (Generalized Processor Sharing) je obecná myšlenka, ze které vyšlo Best-effort plánování, tedy každý proces je obsloužen po nekonečně malých intervalech a tak se dosáhne spravedlnosti.

# 12 Správa provozu

**Správa provozu** znamená dohled, kontrolu a koordinaci v sítích pro usměrňování provozu. Základními oblastmi je správa poruch, konfigurace, účtování, výkonu a bezpečnosti. V sítích TCP/IP je nejrozšířenějším protokolem pro správu protokol *SNMP*.

**SNMP** (Simple Network Management Protocol) je standardizovaný protokol integrovaný i v HW pro TCP/IP nebo IPX sítě. Modeluje manager objekt a agenty.

**SNMP Manager** je program, který běží na jedné stanici a dotazuje se agentů, následně prezentuje data.

**SNMP Agent** je prostředek, který na daném zařízení shromažďuje informace a ukládá je do své informační báze MIB.

**Protokol** je postaven nad UDP 161/162, asynchronně zasílá krátké zprávy na požádání (s výjimkou Trap):

- *Get-request* – zažádání o informace
- *Get-next-request* – další informace v dotazu
- *Set-request* – nastavení hodnoty
- *Get-response* – odpovědní zpráva od agenta
- *Trap* – zpráva od agenta ve výjimečné nebo nečekané situaci

**MIB** je informační báze SNMP, syntaxe a sémantika je definována normou ISO ASN.1 (Abstract Syntax Notation).

**SNMPv1:** nutnost pravidelného dotazování ubírala propustnost, manageři nemohou komunikovat, jedním dotazem pouze jeden typ dat, trap obsahuje málo informací, malé zabezpečení protokolu

**SNMPv2:** rozšířen o požadavky Get-bulk a Inform, kryptografická identifikace (ale složitá konfigurace)

**SNMPv3:** nový formát zpráv a řízení přístupu

**RMON** (Remote Monitoring) je postaven nad SNMP ale agenti mají více činností. Sondují a získávají statistické informace ve své síti. Původně jen na linkové vrstvě a starých protokolech, RMON2 již lepší.

# 13 Protokolové inženýrství

**Validace protokolu** je ověření, zda plní požadované funkce a parametry.

**Verifikace protokolu** ověřuje správnost implementace vzhledem ke specifikaci (hlavně bezpečnost, úplnost, souběžnost, jednoznačnost).

**Výkonnostní analýza** zkoumá vliv parametrů protokolu na celkovou výkonnost v různých prostředích.

**Testování** je experimentální ověřování:

- *Statické* – existence a dostupnost funkcí, rozsahy parametrů, komunikace s nižšími vrstvami
- *Dynamické* – sledování komunikace, časování, spolehlivosti
- *Standardizované (konformance)* – zda splňuje standardy
- *Interoperability* – schopnost komunikace, rozpoznání chyb, starších verzí protokolu apod.

**Syntéza protokolu:**

- *Analyticky* – návrh, specifikace protokolu, validace, verifikace; pak možné změny a znovu
- *Synteticky* – částečně specifikovaný protokol se podle požadavků validity a korektnosti konstruuje rovnou (interaktivně a inkrementálně, popř. automaticky), výsledek již třeba není verifikovat nebo validovat

**Implementace protokolu** může probíhat automaticky (z formální specifikace) nebo klasicky (vývojové prostředí a nástroje, prototypy apod.)

**Formální modely** používané při vývoji: konečné automaty a regulární výrazy, formální gramatiky, temporální logika, petriho sítě, teorie front