Jan Staudek

Počítačové sítě

Zápisy z přednášky zpracoval
Jan Šerák

23. května 1995

Obsah

Část I
Distribuované algoritmy

1 Úvodní slovo autora

2 Distribuované algoritmy

2.1 Kde se s distribuovanými algoritmy můžeme setkat?

Jsou to následující oblasti:

1. počítačové sítě,
2. paralelní zpracování,
3. spolehlivé systémy,
4. operační systémy,
5. programovací jazyky,
6. distribuované databáze,
7. sítě automatů.

2.1.1 Počítačové sítě

WAN,LAN. Model OSI[1] má tyto vrstvy:

- fyzická vrstva,
- linková vrstva,
- síťová vrstva,
- transportní vrstva,
- relační vrstva,
- prezentační vrstva,
- aplikační vrstva.

2.1.2 Paralelní zpracování

- kooperace více procesorů
- Flynnova klasifikace:
  - ⋆ `SIMD` ... Simple Instruction Multiple Data
  - ⋆ `MIMD` ... Multiple Instruction Multiple Data

[1] OSI = Open System Interconnection

Základní model paralelního stroje (SIMD) ∞ propojených sekvenčních synchronních strojů s náhodným přístupem (Random Access Mashine = RAM). Během výpočtu je konečné množství RAM aktivních. Synchronizace je prováděna tak, že jsou synchronizované instrukční cykly.

Všechny procesory mají týž program a týž čítač, centrální procesor nabízí instrukci a každý procesor si na základě hodnoty svého lokálního čítače zvolí jednu z alternativ: provede nebo neprovede nabízenou instrukci $\Longrightarrow$ různé procesory dělají v různém čase jinou část programu

Uniformní přiřazení — každý stroj má končené vyjádření (tzn. program a procesorové omezení).

MIMD: *neuniformní přiřazení*: nekonečně mnoho konečných vyjádření.

Alternativa: SIMDAG ... SIMD s globální pamětí

MIMD:

$\longrightarrow$ multiprocesory (sdílená paměť), dataflow

$\longrightarrow$ multicomputers, LANs, WANs $\longrightarrow$ distribuované systémy

Distribuovaný systém: několik autonomních procesorů, které nesdílejí paměť a spolupracují zasíláním zpráv prostřednictvím komunikačního média. Druhy DS ukazuje následující tabulka:

Těsně vázané DS	**Volně vázané DS**
rychlé a spolehlivé spoje - transputerové sítě (fyzicky blízko)	spoje s poruchami - LAN - WAN
jemný (instrukce) paralelismus, střední(procedury) paralelismus	hrubý (procesy) paralelismus large–grain

2.1.3 Spolehlivé systémy

- zálohové kopie paměti v daných intervalech
- zálohování hardware, hlasování
- softwarové řešení:
  - ⋆ zotavení (roll back)
  - ⋆ snapshot problem – zjištění stavu systému

2.1.4 Operační systémy

Podobné problémy při multitaskingu a sdílené paměti (semafory, monitory).

2.1.5 Distribuované programování

1. Využití více procesorů – odezvy na zprávy od jiných
2. Spolupráce procesorů
3. Možnost vzniku poruch

- Strategie přidělování procesů procesorům $\Longrightarrow$ výkonnost systému (rozdělení programu na paralelní sekce);
- Komunikace, synchronizace (daň z distribuce);
- Aplikace odolné proti poruchám.

Podpora: OS a/nebo programovacího jazyka (vhodnost pro aplikaci, možnost implemetace jazyka) Z tohoto hlediska budeme rozlišovat tyto typy:

1. Logicky distribuovaný SW a Fyzicky distribuovaný HW (send, receive, LAN, WAN);
2. Logicky distribuovaný SW a Fyzicky nedistribuovaný HW (zprávy přes sdílenou paměť);
3. Logicky nedistribuovaný SW a Fyzicky distribuovaný HW (sdílená paměť simulovaná zasíláním zpráv);
4. Logicky nedistribuovaný SW a Fyzicky nedistribuovaný HW (sekvenční programování).

2.1.6 Programovací jazyky

- Algol68, Concurrent Pascal – sdílená paměť;
- procedurální (imperativní model): Modula, DP, Ada, Linda, Occam;
- deklarativní (neimperativní model):
  - ⋆ funkcionální: ParAlfl;
  - ⋆ logické: Concurrent Prolog, PARLOG

2.1.7 Distribuované databáze

- zamykání datových položek $\Longrightarrow$ deadlock;
- distribuovaná data: segmentace, replikace dat;
- atomické operace: commit protokoly.

2.1.8 Sítě automatů

Umělá inteligence, neuronové sítě. Proces $\sim$ automat.

2.2 Procesově orientovaný model paralelního systému

V tomto modelu rozlišujeme *proces* a *program*. Rozlišujeme tyto stavy procesu:

- běžící,
- připravený,
- čekající.

Řídící blok procesu (Process Control Block) uchovává tyto informace:

- jméno
- stav procesoru (PC, registry, maska přerušení,...)
- stav procesu
- priorita
- definice užívaných prostředků
- účtování

Paralelní procesy potom spolu komunikují:

- za účelem dohody
- z důvodu soupeření o prostředky

Naše snahy směřují k nalezení *determinované množiny* paralelních procesů $P = \{p_1, \ldots, p_n\}$ takovou, že na stejné vstupy dává stejné výstupy bez ohledu na:

- relativní rychlost běhu procesů (nevím nic o rychlosti běhu procesů)
- připustné překrývání běhu procesů v čase

Další podrobnosti paralelního zpracování úloh najde laskavý čtenář v literatuře ([1], odst. 1.6 na stranách 14 – 16).

2.3 Vyjádření komunikace v modelu

* sdílené paměti:
  - ⟶ semafory (proměnné) a monitory (volání)
  - ⟶ multiprocesory
- synchronní komunikace, kapacita komunikačního kanálu je nulová, nutná časová koincidence instrukcí: `write(m)` ⟶ `read(m)`
- volání vzdálených procedur
- asynchronní komunikace, kapacita komunikačního kanálu není nulová, předávání zpráv

Poznamenejme, že se nebudeme zabývat typem komunikace označeným $*$, protože to není předmětem této přednášky.

2.4 Rozdíly mezi paralelním a distribuovaným modelem

Hlavní rozdíly mezi modelem paralelního a distribuovaného systému ukazuje následující tabulka:

Algoritmus	**Sekvenční, Paralelní**	**Distribuovaný**
řídící informace	dostupné v jediném uzlu	rozložené v různých uzlech
pořadí operací	dané předem	předem neurčené
provádění operací	v jednom uzlu	v různých uzlech
měřítko výkonnosti	činnost procesor(ů)	doba přenosu a počet zpráv

2.5 Distribuovaný model

U distribuovaného modelu výpočtu se budeme zajímat o následující vlastnosti:

- kapacita kanálu,
- synchronnost komunikačních cest,
- poruchy komunikačních cest,
- topologie komunikační sítě,
- identifikovatelnost uzlů.

1. Odolnost proti poruchám
   - Komunikační chyby (změna pořadí zpráv, duplikace a ztráta zprávy, výpadek linky). Pozor na problém rozdělení sítě!
   - chyby procesů (dočasný a trvalý výpadek procesu, Byzantské chyby)
2. Kapacita komunikačního kanálu

0 ⟶ synchronní komunikace sousedů,
n ⟶ reálná asynchronnost (FIFO?),
∞ ⟶ teoretická asynchronnost.

3. Topologie sítě
   - úplné propojení,
   - strom,
   - hvězda,
   - kruh.

   Uvědomme si rozdíl mezi *síťovým grafem* a *komunikačním grafem*.

2.6 Centralizovaný a Distribuovaný algoritmus

V tomto odstavci si uvedeme rozdíly mezi *centralizovaným* a *distribuovaným* algoritmem z hlediska implementace v počítačové síti.

2.6.1 Centralizovaný algoritmus

1. Rozhodnutí o postupu se provádí v *centrálním* uzlu.
2. Všechny informace jsou soustředěny v *centrálním* uzlu.
3. Centrum může:
   - migrovat po síti,
   - stát se „úzkým profilem",
   - vypadnout z činnosti.

2.6.2 Distribuovaný algoritmus

1. Všechny uzly mají stejný díl informace.
2. Každý uzel se rozhoduje podle své lokální informace.
3. Za konečné rozhodnutí má každý uzel stejný díl odpovědnosti.
4. Všechny uzly vynakládají stejné úsilí k dosažení konečného rozhodnutí.
5. Porucha uzlu nezpůsobí zhroucení systému.
6. Vždy platí:
   - Každý uzel zná jen část stavu systému.
   - Neexistují hodiny distribuované do uzlů.

2.7 Složitost distribuovaných algoritmů

Časová složitost je doba potřebná k přenosu nejdelší posloupnosti zpráv před ukončením algoritmu. Časová jednotka je u:

- synchronní sítě *takt* komunikační linky.
- asynchronní sítě *interval* mezi vysláním a přijmutím zprávy

Komunikační složitost je počet zpráv potřebných pro realizaci algoritmu (počet bitů).

Složitost algoritmů budeme udávat vzhledem k počtu uzlů nebo hran v síti.
Blíže se laskavý čtenář dočte o složitosti distribuovaných algoritmů ve skriptu [1] v odst. 1.7 na straně 16.

2.8 Analýza distribuovaných algoritmů

Analýza DA odpovídá na otázku, zda DA provádí to, co se předpokládalo, že má dělat.

Specifikace DA: je definice vlastností DA

Verifikace DA: dokázání dosažených vlastností na základě proměnných a akcí DA.

2.8.1 Formalismus pro analýzu DA

Distribuovaný systém je trojice $DS = (S, A, \Sigma)$, kde $S = \{s_i\}$ je množina stavů distribuovaného systému DS, $A = \{a_i\}$ je množina akcí DS a $\Sigma = \{s_{i-1} \xrightarrow{a_i} s_i\}$ je množina chování (reakcí) na akce.

- s_i jsou hodnoty proměnných DS.
- a_i jsou vnitřní operace DS.
- Σ je popsána:
  - ⋆ konstruktivně
    - ◦ popis programem
    - ◦ popis čtivou formou
    - ◦ formálně (t.j. tak, jak jsme popsali výše)
  - ⋆ axiomaticky, tj. výčtem axiomů a formulí. Tento postup je vyhovující pro logický popis, ale už ne pro praxi.

2.8.2 Cíle formální analýzy DA

- *bezpečnost* ... nestane se nic špatného.
- *životnost* ... v konečném čase se stane něco pozitivního (skončení, výhra v soupeření, dohoda).
- *spravedlivost* ... každý někdy zvítězí.

Bezpečnost dokazujeme tak, že dokazujeme, zda se distribuovaný systém nedostane ze správné situace (stavu) do nesprávné žádnou reakcí na akci.

Blíže o dokazování vlastností DS je popsáno v literatuře [1] v odstavci 1.9.1 na stranách 17 – 18.

3 Procházení sítí

Poznámka o složitosti: Složitosti algoritmů budeme značit takto:

- $O(f(n))$ je asymptoticky lepší než $f(n)$, tzv. *horní odhad*,
- $\Omega(f(n))$ je asymptoticky horší než $f(n)$, tzv. *dolní odhad*,
- $\Theta(f(n))$ je asymptoticky rovna $f(n)$, tzv. *oboustranný odhad*.

Na úvod této kapitoly si jen pro oživení v paměti „v duchu" zopakujme, význam pojmů: *graf, kostra grafu, kořenový strom, kruh, řídící a pomocná struktura, dynamické vytváření řídících struktur.*

Důležité odstavce této kapitoly pojednávají zejména o konstrukci kořenového stromu *paralelním* a *sekvenčním prohledáváním* a o konstrukci kostry grafu a kruhu.

3.1 Procházení grafu sítě

Při diskusi o výše zmíněných algoritmech nás budou zajímat zejména tyto aspekty:

- způsob navštívení všech uzlů grafu
- efektivnost grafových a synchronizačních algoritmů, závisející na způsobu procházení.

Základní strategie procházení grafu jsou tyto:

- obecná
- do hloubky (užívá se pro konstrukci silných komponent, dvojitých silných komponent,...)
- do šířky (užívá se pro konstrukci stromu nejkratších cest,...)

3.2 Distribuované grafové algoritmy

Počítačovou síť můžeme chápat jako orientovaný nebo neorientovaný (ohodnocený) graf $G = (V, E)$, kde V jsou procesory a E linky. Řízení distribuovaného grafového algoritmu je rozděleno mezi procesory:

- iniciátory
- výměna řídících informací
- terminátory

Pro konstrukci grafových algoritmů v následujících odstavcích budeme uvažovat model s částečnou spolehlivostí (model s linkami bez poruch během provádění algoritmu), asynchronní sítí (různá spoždění linek), lokální znalostí topologie sítě a lokální komunikací.

Algoritmy můžeme popsat *globálně* nebo *distribuovaně*; my budeme popisovat distribuovaně.

Formát zpráv:

```
send | receive message (type, parameter)
```

Dynamické distribuované vytváření v uzlech: při inicializaci platí pro každý uzel:

- uzel *nezná* svoji pozici v síti,
- uzel *nezná* topologii sítě,
- uzel *zná* svoje sousedy.

a výsledkem je, že v lokálních proměnných uzlu jsou speciální pozice v síti, lokální topologie apod.

3.3 Konstrukce stromu paralelním hledáním

Algoritmus, který si zde za okamžik uvedeme konstruuje strom, který obsahuje nejkratší cesty z každého uzlu do kořene. Je potřebná synchronizace (zpětná vazba na kořen). K tomu použijeme zprávy `explore` (vysílaná uzlem směrem k potomkům, *lokální* synchronizace k uzlu) a `return` (vysílaná uzlem směrem k předkovi, *globální* synchronizace ke kořeni).

Množství řídících informací ve zprávách je:

`počet zpráv` $\times$ `délka zprávy`

3.4 Konstrukce kořenového stromu paralelním hledáním se zpětnou vazbou (NTF)

Zkratka NTF znamená `Network Traversal with Feedback`. Časová složitost algoritmu je lineární ($O_T(n)$) a komunikační složitost je kvadratická ($O_C(n^2)$).

Délka větví, které algoritmus vytvoří, závisí na rychlosti komunikačních linek (neexistuje globální synchronizace), algoritmus nebere ohled na dobu přenosu zpráv, zprávy jsou konstantní délky.

Lokální kontext v uzlu i:

- `marked`$_i$ ... uzel je označen (boolean)
- `father`$_i$ ... hrana, od níž přijde první zpráva `explore`
- `sons`$_i$ ... hrany, od nichž uzel i obdržel zprávu `return(yes)`

Formát zpráv:

- `explore`
- `return(no)`
- `return(yes)`
- `init` (tato zpráva reprezentuje pokyn k zahájení algoritmu)

Algoritmus: zde uvádíme z pohledu *iniciátora* a ostatních uzlů. Poznamenejme ještě, že algoritmus užívá synchronizační primitivum

```
await(<condition>)
```

které zajistí čekání procesu na splnění podmínky `condition`.

A nyní již přikročme ke znění algoritmu:

```
#pohled iniciátora

when received init do                          #příchod init z vnějšku
  marked := TRUE;
  father := i;                                 #i je můj identifikátor
  sons := {};
  waited := card(neighbours);                  #kardinalita množiny sousedů
  for all k in neighbours do
    send explore to process_k;
  end do;
  await(waited = 0);
end do;
```

Pozor! O ukončení algoritmu se dozví jen kořen (iniciátor). Musí-li to vědět všechny uzly $\longrightarrow$ broadcastová zpráva před ukončením algoritmu v iniciátorovi.

```
#pohled ostatních uzlů

when received explore from process_j do
  case
    marked => send return(no) to process_j;
    !marked => marked := TRUE;
               father := j;
               sons := {};
               waited := card(neighbours - {j});
               for all k in neighbours - {j} do
                 send explore to process_k;
               end do;
               await(waited=0);
               send return(yes) to father;
  end case;
end do;

#pohled všech uzlů na události return

when received return(b) from process_j do
  waited := waited - 1;
  if b = yes then
    sons := sons + {j};                    #množinové sjednocení
  end if;
end do;
```

Důkaz algoritmu: K dokázání algoritmu stačí dokázat tato dvě tvrzení:

1. Každý proces dostane alespoň jednu zprávu `explore` a v konečném čase odpoví zprávou `return`.

2. Po skončení algoritmu má každý uzel p_i svého otce a pro uzly p_i, p_j, $i \neq j$ platí, že $\texttt{sons}_i \cap \texttt{sons}_j = \emptyset$.

Tato tvrzení jsou tak zřejmá, že si dovolíme jejich ověření ponechat na čtenáři jako cvičení.
Komunikační složitost je $O_C(e)$, kde $e = |E|$, tedy $O_C(n)$, kde $n = |V|$. Časová složitost je $O_T(n)$.
Algoritmus pracuje pomocí tzv. *vln*, což je rozeslání zprávy `explore` na všechny strany a přijetí zpráv `return` ze všech stran.

3.5 Prohledávání do šířky (paralelní)

Algoritmus BFS (prohledávání do šířky) je variantou algoritmu NTF. I zde budeme rozesílat zprávy `explore` a `return`. Parametrem zprávy `explore` bude délka vlny, protože BFS vyžaduje synchronizaci do pater. Zpráva `return` bude mít dva typy:

- `return(c)` bude vysláno k otci v případě, že uzel má nějaké následníky,
- `return(t)` bude vysláno v případě, že uzel nemá žádného následníka.

Blíže se může laskavý čtenář o algoritmu BFS dočíst ve skriptu [1] v odstavci 3.2.1 na stranách 35 – 38.

3.6 Prohledávání do hloubky (sekvenční)

Při provádění algoritmu DFS počítá nejvýše jeden uzel. Je to ten, kterému je právě přidělen příznak oprávnění, který putuje sítí. Činností tohoto algoritmu vzniká nevyvážený strom. Jakmile se příznak oprávnění vrátí k iniciátorovi, algoritmus končí. Více o konstrukci algoritmu DFS je napsáno ve skripti [1] v odstavci 2.3.1 na stranách 29 – 33.
Popišme si ještě jak jsou udržovány struktury, popisující stav sítě:

Lokální synchronizace: Příznak oprávnění putuje sítí prostřednictvím zpráv `explore` (shora dolů) a `return` (zespoda nahoru). Uzel, který poprvé obdržel `explore` rozešle všem svým sousedům zprávu `attained`, kterou zjišťuje, které uzly ještě neměly příznak oprávnění. Vybere nějaký z uzlů, které na tuto zprávu příslušně odpoví a jemu pošle svůj příznak oprávnění (`explore`). V případě, že všichni sousedi uzlu byly již zasaženy příznakem oprávnění, pošle uzel příznak oprávnění svému otci (`return`).

Stav sítě je tedy udržován distribuovaně ve všech uzlech.

Součást příznaku oprávnění: Příznak oprávnění putuje sítí prostřednictvím zpráv `explore` a `return`, struktura popisující stav sítě však putuje s ním jako parametr zpráv. Uzel vyšle nezasaženému sousedu zprávu `explore(z)` a obdrží od něj zprávu `return(z')`, kde `z` a `z'` jsou množiny navštívených uzlů.

Stav sítě je tak udržován centralizovaně v příznaku oprávnění. Tímto způsobem se šetří počet zpráv (ruší se kvantum zpráv `attained` a odpovědí na ně), ale roste délka zpráv (ukládáním stále většího počtu identifikátorů uzlů).

3.7 Konstrukce kostry grafu sítě

Tento problém se pro nás stane zřejmým, když si uvědomíme, co to je kostra grafu. Je to kořenový strom bez orientace předek – potomek a bez označeného kořene. Příslušnou modifikací některého z předchozích algoritmů tak dostaneme algoritmus pro konstrukci kostry grafu sítě.

Kdo přece jen nevěří tomu, co je zde uvedeno, nechť si přečte ve skriptu [1] odstavec 2.4 na straně 33.

3.8 Konstrukce kruhu

Kruh je relace úlného uspořádání na množině vrcholů grafu. Tento pojem je však známější pod označením Hamiltonova cesta. Zjištění existence popřípadě nalezení Hamiltonovy cesty je NP-úplný problém. Proto bývá v praktických síťových aplikacích nahrazován tzv. *virtuálním kruhem*, tedy kruhem z logických spojů (tj. takových spojů, které jsou realizovány jedním a více fyzickými spoji). Virtuální kruh lze najít dobře modifikací DFS, protože příznak oprávnění nutně putuje po virtuálním kruhu (stačí tedy zaznamenávat jeho „stopu"). Více se lze dočíst ve skriptu [1] v odstavci 2.5 na stránkách 33 a 34.

Výsledný virtuální kruh je reprezentován směrovacími tabulkami, které určují směr, kterým virtuální kruh v daném uzlu pokračuje včetně informace, zda je tento směr regulérní, či má být příslušný uzel přeskočen.

3.9 Konstrukce kořenového stromu nejkratších cest

V tomto odstavci si uvedeme tzv. SPA (`Shortest Paths Algorithm`) paralelním hledáním. Tento algoritmus používá zprávy pevné délky, nepoužívá synchronizaci v kořeni, pracuje na principu NTF s tím rozdílem, že si v lokálním kontextu $\texttt{level}_i$ „hlídá" vzdálenost od kořene. Zprávy jsou tohoto formátu:

- `explore(r:NATURAL)`, kde `r` představuje `level` vysílajícího,
- `return(b:(yes,no),r:NATURAL)`.

Na rozdíl od NTF se bude kontext $\texttt{father}_i$ měnit dynamicky. U NTF se nastavil v případě, že uzel nebyl ještě zasažen. U SPA se to provede taky, ale provede se tato akce také v případě, že přijde `explore(r)` taková, že uzel zasažen již byl a zároveň má aktuální kontext `level` větší než $\texttt{r} + 1$ (tedy ozval-li se bližší potenciální otec než otec dosud aktuální).

```
#pohled iniciátora

when received init do
  marked := TRUE;
  level := 0;
  father := i;
  sons := {};
  waited := card(neighbours);
  for all k in neighbours do
```

```
    send explore to process_k;
  end do;
  await(waited = 0);
end do;

when received return(b,r) from process_j do
  waited := waited - 1;
  sons := sons + {j};                        #množinové sjednocení
end do;

#pohled ostatních uzlů

when received explore(r) from process_j do
  if !marked or (marked and r+1 < level) then
    marked := TRUE;
    father := j;
    sons := {};
    level := r+1;
    waited := card(neighbours - {j});
    for all k in neighbours - {j} do
      send explore(level) to process_k;
    end do;
    await(waited=0);
    send return(yes,level) to father;
  else
    send return(no,level) to father;
  end if;
end do;

when received return(b,r) from process_j do
  waited := waited - 1;
  if b = yes then
    sons := sons + {j};                      #množinové sjednocení
  end if;
end do;
```

Komunikační složitost je $O_C(n^3)$ a časová složitost je $O_T(n)$.

3.10 Zhodnocení prohledávání sítě

Přehled algoritmů uvedených v předchozích odstavcích shrnuje následující tabulka:

Algoritmus	O_C	O_T
Pevná délka zpráv		
BFS, synchronizace v kořeni	n^2	n^2
NTF	n^2	n
SPA na bázi NTF	n^3	n
Proměnná délka zpráv		
SPA na bázi NTF s proměnnou délkou zprávy `explore`	n^{2*}	n
BFS, vlny s proměnnou délkou zprávy	n^2	n

* Limitně k úplnému propojení konverguje k n.

4 Synchronizace

V následujících kapitolách si uvedeme tyto typy synchronizačních algoritmů:

- soupeření (vzájemné vyloučení),
- kooperace (produkce $\longrightarrow$ konzumace),
- dohoda (na hodnotě, provedení, zahájení,...).

K realizaci těchto algoritmů však potřebujeme jisté synchronizační nástroje, jejichž stručný přehled si uvedeme v této kapitole. Nutno však hned v úvodu poznamenat, že tato kapitola obsahuje pouhý stručný, schematický a zdaleka ne úplný přehled synchronizačních nástrojů. Daleko více je uvedeno ve skriptu [1] ve 3. kapitole na stranách 35 – 49.

4.1 Synchronizátory

Příkladem synchronizátoru je Awerbuchova implementace synchronizátoru. Proces je v *bezpečném* stavu, pokud jsou přijaty a zpracovány všechny zprávy, které rozeslal v čase `t`. V procesu (uzlu) lze generovat `inc(t)` tehdy, když provedl všechno, co měl udělat podle zpráv přijatých v `t` a když ví, že jeho sousedé jsou v bezpečném stavu.

4.2 Synchronizace procesů

K synchronizaci procesů potřebujeme jisté *synchronizační nástroje*. Jsou to:

- semafory
- monitory
- časová razítka
- čítače událostí
- AND-synchronizátory (skupinové P,V)
- OR-synchronizátory (nedeterminismus)

4.3 Synchronizace semaforem

Semafor je speciální proměnná, na níž lze aplikovat operace `P(s)` a `V(s)`, u nichž je zaručena atomičnost takto:

- u jednoprocesorových počítačů to lze provést maskováním přerušení
- u multiprocesorových počítačů maskování technicky selhává, proto se zavádí instrukce `TST x` se sémantikou:

```
if (x == 0) {x = 0; <PC> += 2;}
else <PC> += 1;
```

- u sítí počítačů je implementace semaforů velikým otazníkem, protože nebylo dosud nalezeno uspokojivé řešení.

4.4 Synchronizace monitorem

Monitor je objektovým modelem semaforu (proměnné a operace v jednom objektu). Je při tom zajištěno, že v jednom okamžiku se neprolnou instrukce víc než jedné akce monitoru.

4.5 Synchronizace čítači událostí

Serializátor je reprezentován proměnnou `S` a čítač událostí proměnnou `E`. Jsou definovány tyto operace:

- `ticket(S): return S++;`
- `read(S): return S;`
- `await(E,v)`: je-li `E<v`, pak volající proces je překlopen do stavu čekající, přičemž E_0=`v`.
- `advance(E): E += 1` a čekající procesy s E_0 `= E` jsou překlopeny do stavu připravený.

4.6 Synchronizace AND–synchronizátorem

Vyčkání na výskyt dvou a více událostí lze provést explicitním vnořováním semaforů:

```
... P(X); P(Y); ... V(Y); V(X); ...
```

U tomto případě se však může zlehka stát, že napíšu nový program a prohodím `X` a `Y`, čímž se dá způsobit *deadlock* se starými programy.

Řešením z této prekérní situace jsou skupinové operace `P` a `V`, takže předchozí fragment zdrojového textu je nahrazen sekvencí:

```
... SP(X,Y); ... SV(X,Y); ...
```

5 Vzájemné vyloučení

Na úvod si uvedeme ilustrativní řešení vzájemného vyloučení procesů:

- řešení pomocí semaforu:

```
proces i;
  ...
  P(S);
  # Kritická sekce
  ...
  V(S);
  ...
end;
```

- řešení pomocí serializátoru:

```
proces i;
  ...
  await(E,ticket(S));
  # Kritická sekce
  ...
  advance(E);
  ...
end;
```

Nás však bude nejvíce zajímat distribuované řešení vzájemného vyloučení. Shrňme si tedy v úvodu všechny požadavky na distribuovaný systém, o nichž budeme předpokládat, že platí:

1. Distribuovaný systém tvoří N uzlů, v každém z nich je jeden procesor.
2. Síť je plně propojena (alespoň virtuálně).
3. Propojení zachovává pořadí zpráv.
4. Doprava zprávy se uskutečňuje v konstantním čase.

Obecné řešení musí splňovat:

- každá dvojice žádajících procesu má alespoň jednoho arbitra;
- každý proces musí dávat stejné úsilí k dosažení úspěchu;
- každý proces musí mít stejnou odpovědnost.

Lamportovo řešení pomocí časových razítek je uvedeno ve skriptu [1] na straně 60.

5.1 Algoritmus BAKERY

V algoritmu BAKERY vystupuje proces jako zákazník v pekařství. Při vstupu do pekařství (při žádosti o vstup do kritické sekce) obdrží zákazník „pořadové číslo". Zákazníci jsou obsluhováni (je jim povolen vstup do kritické sekce) v pořadí jejich pořadových čísel.

Uveďme si nyní přesný postup při žádosti procesu `proces_i` o vstup do kritické sekce:

1. od všech procesů `proces_j`, `j`$\neq$`i`, `j` $= 1, 2, \ldots, N$ získá jejich pořadová čísla p_j;
2. $p_i = \max(p_j) + 1$;
3. `proces_i` klade dotazy, zda existuje proces `proces_j` s pořadovým číslem $p_j < p_i$ čekající na vstup do kritické sekce; když žádný takový neexistuje, vstoupí proces `proces_i` do kritické sekce, po výstupu z kritické sekce se nastaví p_i na nulu.

Algoritmus BAKERY implementovaný pány Ricarta a Agrawaly je uveden ve skriptu [1] na straně 61. Snižováním počtu arbiterů se zabývali pánové Maekawa a Raymond. K jakým výsledkům se dopracovali si může laskavý čtenář přečíst na straně 62 téhož skripta.

5.2 Algoritmus „čtenáři a písaři"

Jak vypadá tento algoritmus implementovaný pro počítač se sdílenou pamětí si může laskavý čtenář najít v literatuře [2]. Připomeňme si, co tento algoritmus musí splňovat:

- *priorita čtenáře:* přicházející písař čeká, dokud nějaký čtenář čte.
- *silná priorita čtenáře:* platí priorita čtenáře a navíc má přednost čekající čtenář.
- *silná priorita písaře:* přicházející čtenář čeká na skončení písaře.

5.2.1 Řešení pomocí serializátoru a čítače událostí

```
# sdílený modul
var  řada_p : sequencer;
     který_p: eventcounter;

# čtenář i
var  řada_č : sequencer;
     který_č: eventcounter;
     počet_č: integer := 0;

loop await(který_č,ticket(řada_č));
     if počet_č = 0 then
       await(který_p,ticket(řada_p));
     end if;
     počet_č++;
     advance(který_č);
     # Kritická sekce
     ...
     await(který_č,ticket(řada_č));
     počet_č−;
     if počet_č = 0 then advance(který_p);
     end if;
     advance(který_č);
     # Konzumace
     ...
end loop;

# písař j
loop
  # Produkce
  ...
  await(který_p,ticket(řada_p));
  # Kritická sekce
  ...
  advance(který_p);
end loop;
```

5.3 Problém pěti filosofů

S problémem pěti filosofů jsme se také seznámili již dříve (viz [2]). Použití semaforů hrozí deadlockem, skupinové operace `P` a `V` jsou řešením. My si zde uvedeme implementaci procedury pro dvojitou operaci `P` pomocí sequenceru:

```
procedure Double_P(x,y);
  r,s:NATURAL;
begin
```

```
  – nejprve zamčít koordinované generované pořadí
  await(zámek.E,ticket(zámek.T));
  – nýní získej koordinovanou množinu pořadí
  r := ticket(X.T);
  s := ticket(Y.T);
  – odemknu generátory pořadí
  advance(zámek.E);
  – a konečně počkám, až nastane X a Y
  await(X.E,r);
  await(Y.E,s);
end;
```

Lepší je však rozbít symetrii a použít semafory.

5.4 Algoritmus „kuřáci cigaret"

Kuřáci mohou kouřit až když má všechny prostředky (tedy cigaretový papír, tabák a zápalku), dodavatelé však neposkytují všechno. Přikročme tedy k algoritmu:

```
            var  p,t,z : semaphore := 0,0,0;
 dodávka: semaphore := 1;

#dodavatelé:  X          Y                Z
loop P(dodávka);    loop P(dodávka);    loop P(dodávka);
  V(t);               V(z);               V(p);
  V(z);               V(p);               V(t);
end;                end;                end;

#kuřáci:
# A má t              B má z              C má p
loop SP(z,p);       loop SP(t,p);       loop SP(t,z);
  Kouří;              Kouří;              Kouří;
  V(dodávka);         V(dodávka);         V(dodávka);
end;                end;                end;
```

6 Dohoda

6.1 Algoritmus producenti a konzumenti

Zde je na místě opět odkaz na literaturu [2]. My si zde uvedeme tento algoritmus implementovaný pomocí serializátorů:

```
#globální část

var ulož,vyber : eventcounter;

#producent
var řada_p : sequencer;
loop
  t := ticket(řada_p);
  await(vyber,t - N + 1);
  #zápis do bufferu
  ...
  advance(ulož);
end loop;

#konzument
var řada_k : sequencer;
loop
  n := ticket(řada_k);
  await(ulož,n+1);
  #čtení z bufferu
  ...
  advance(vyber);
end loop;

#pokud je více producentů a konzumentů
#producenti
var řada_p : sequencer;
loop
  t := ticket(řada_p);
  await(ulož,t);                  #serializace producentů
  await(vyber,t-N+1);             #serializace s konzumenty
  #zápis do bufferu
  ...
  advance(uloz);                  #serializace producentů i s konzumenty
end loop;

#konzumenti
var řada_k : sequencer;
loop
  n := ticket(řada_k);
  await(vyber,n);                 #serializace konzumentů
  await(ulož,n+1);                #serializace s producenty
  #čtení z bufferu
  ...
  advance(vyber);                 #serializace konzumentů i s producenty
end loop;
```

6.2 Dohoda dvou generálů

Dohoda dvou generálů je prakticky neřešitelná. Úvahu „Ví, že vím, že ví?" můžeme rozvíjet až do nekonečna.

Odsouhlasení dohody je také netriviální problém, uvážíme-li, že většinou víme o linkách, že jsou spolehlivé, ale o uzlech už to říci nemůžeme, protože uzly mohou:

- dočasně vypadnout
- trvale vypadnout
- lhát

O způsobu zajištění hovoří odstavec 5.4 skripta [1].

6.3 Volba koordinátora

Tento problém je nutný řešit v architektuře klient/server. Z důvodu nesymetričnosti je malá odolnost proti poruchám, což také způsobuje potřebu volby servera. K tomu je nutná jednoznačná identifikace procesu. Předpokládaná topologie je jednosměrný kruh; procedura `sendL` pošle následující svůj výběr a svůj identifikátor dalšímu uzlu. Časová složitost je $O_T(n)$ a komunikační je $O_C(n^2)$.

Strategie volby je dvojí, buď začnou volit všichni zároveň, nebo začne proces vlevo od procesu s maximálním identifikátorem.

```
#inicializace
  participant := TRUE;
  sendL(election,my_number);

#vlastní algoritmus dohody
when received (election,j) do
  if j>my_number then
    sendL(election,j);
    participant := TRUE;
  end if;
  if j<my_number and !participant then
    sendL(election,my_number);
    participant := TRUE;
  end if;
  if j=my_number then
    sendL(elected,j);
  end if;
end do;

when received (elected,j) do
  co_ordinator := j;
  participant := FALSE;
  if j!=my_number then
    sendL(elected,j);
  end if;
end do;
```

Část II
Počítačové sítě

- co to je?
- důvody
- typy zpracování dat v síti
- služby
- jak jsou realizovány
- kategorizace

7 Co je počítačová síť

Síť:

- *uzly* – číslicová zařízení, která umějí data + procesy
- *hrany* – komunikační prostředky

Propojení sítí:

- se stejným / příbuzným systémem přenosu dat
  - ⋆ opakovač (repeater)
  - ⋆ most (bridge)
- s odlišným způsobem adresování a provozování síťových služeb
  - ⋆ směrovač (router)
  - ⋆ brána (gateway)

Repeater: prodlouží (upravuje přenosové podmínky) fyzický dosah přenosu dat (zesilovač).
Bridge: opět prodlužuje dosah, je však inteligentnější. Rozezná totiž komunikaci v rámci jedné nebo druhé sítě a mezi sítěmi.
Síťový hardware:

- spoje
  - ⋆ telefonní síť (veřejná / soukromá)
  - ⋆ lokální síť
  - ⋆ satelitní spoje
- počítače

Síťové programové prostředky:

- operační systémy
  - ⋆ Unix, TCP/IP (InterNet)
  - ⋆ MS DOS, Novell
  - ⋆ VMS, DecNet
- síťové aplikační služby
  - ⋆ vzdálené relace (telnet, rlogin)
  - ⋆ přenos souborů (rcp, ftp, NFS, FTAM)
  - ⋆ distribuovaný adresář (X.500)
- provozní pravidla

Důvody zřizování počítačových sítí

- zvýšení spolehlivosti, tj. zálohování v aplikačních systémech
- sdílený přístup ke kapacitám, jako jsou programy, koprocesory, software
- sdílený přístup k údajům

Relace v síti je reprezentace provozu. Typy relací jsou:

- terminál × terminál
- terminál × vzdálený program
- program × program
  - ⋆ kooperující procesy (`P1` × `P2`)
  - ⋆ klient / server (více klientů × jeden server)

Realizace sítí

- ISO standardizace
- OSI (Open System Interconnection)

Vrstvy OSI:

7. aplikační procesy + služby (`rlogin, rcp, ftp, ...`)
6. jednotná reprezentace dat, kryptografie
5. relace mezi procesy (Unix Sockets, RPC)
4. transport dat mezi koncovými uzly (TCP)
3. směrování toku dat mezi dvěmi uzly (IP)
2. tok dat mezi uzly (Ethernet)
1. ovládání fyzického přenosového média (RS 232, X.24)

Druhy služeb:

- vrstvy
- protokoly pro komunikaci ve vrstvě
- služby na rozhraní mezi vrstvami

8 Aplikační vrstva

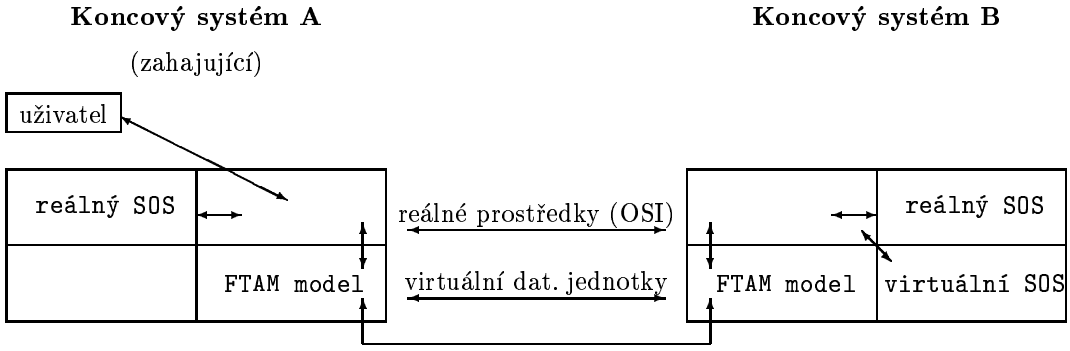
- ftp, rfa (remote file access), dohromady FTAM
- E-mail (MHS – Message Handling System, X.400)
- telnet
- adresář (X.500)
- rlogin, RJE
- telltex, videotex

8.1 Komunikační služby

Vrstva	Systém přenosu souborů	Distribuovaný adresář	Elektronická pošta
aplikační	FTAM (ISO 8571) VT (ISO 9040)	DA (ISO 9504)	MHS (X.400)
transportní	virtuální kanály	datagramy	
síťová	virtuální kanály	datagramy	

Tabulka 1: Komunikační služby počítačových sítí

Princip FTAM ukazuje následující obrázek 1, kde SOS znamená systém ovládání souborů



Obrázek 1: Princip FTAM

Co se týče elektronické pošty, v budoucnu se počítá s plným uplatněním normy X.400 (což je soubor dokumentů a protokolů). Lidé i aplikace mohou používat MHS a přitom nemusí být on-line. Store & forward, až to bude možné. Lze přenášet cokoli, není omezena délka, není omezen koncový terminál. Uživatelský agent je buď zvlášť, nebo v rámci poštovního úřadu. Minimum toho, co musí umět uživatelský agent zvládnout je komunikace s poštovním úřadem a se vzdáleným uživatelským agentem. Navíc může umět manipulace se zprávami, okýnka, grafika,...

Bližší informace o principech X.400 jsou v [1] v kap. 9.5.1.

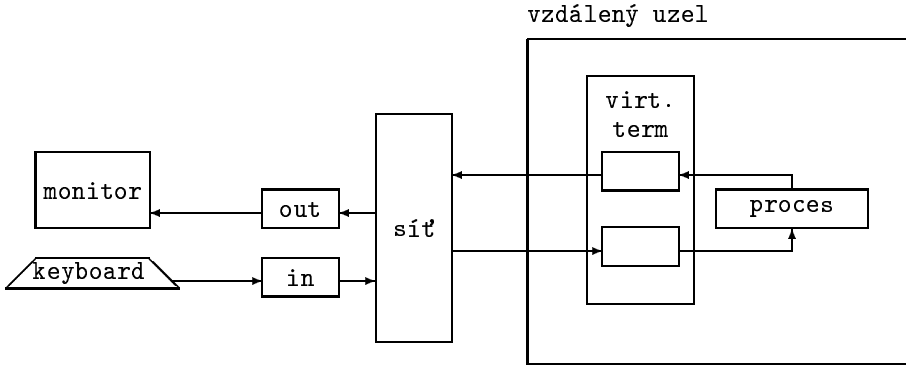
8.2 Virtuální terminál

Z důvodu mnoha typů terminálu a protože chci aplikaci nezávislou na terminálu, definuje se tzv. *virtuální terminál*. Rozeznáváme tyto druhy terminálů:

- řádkový terminál
- stránkový terminál

- formulářový terminál

PAD – Packet Assembler / Disassembler
Na koncovém uzlu sítě je virtuální terminál, jak ukazuje obrázek 2.



Obrázek 2: Princip virtuálního terminálu

Každý terminál obsahuje μ-procesor.

Normy vztahující se k virtuálnímu terminálu

X.3: doporučení vzhledu tabulky charakterizující terminál;

X.28: doporučení ovládání virtuálního terminálu softwarem (pravidla provozu, signálové ovládání, tj. jak se ovládá virtuální terminál fyzicky, pro veřejné datové sítě)

X.29: doporučení, jak lze vzdáleně tuto tabulku nastavovat

8.3 Adresář

Chci identifikovat každý element sítě, např. `jméno_1`, `jméno_2`, ... což jsou symbolická jména. Z tohoto jména se odvodí *síťová adresa* X.Y.Z.W., z níž lze snadno odvodit cestu poslání zprávy na danou adresu.

Transformace mezi jménem a síťovou adresou je buď známa, nebo ji zná někdo jiný (analogie telefonního seznamu). Jsou však zde problémy s tříděním. Je definována norma X.500, která udává, jak implementovat distribuovanou databázi pro styk počítačových sítí (data jsou distribuována).

Adresář je distribuovaná množina otevřených systémů, plnících funkci databázových informací o objektech reálného světa.

Princip distribuovaného adresáře je popsán obrázkem na straně 177 skripta [1], přičemž uživatel může být osoba nebo program. *DIB* distribuovaná databáze atributů objektů.

Implementace distribuovaného adresáře:

- Unix (BSD, V)
- nad X.25, TCP/IP

Řešení dotazu uživatele U přes jeho agenta DUA. Zná-li odpověď dotazovaný systémový agent (DSA), pak odpoví. Pokud nezná, položí dotaz „sousedícím" DSA, a to:

- řetězově (do hloubky grafu „sousednosti" DSA)
- multicasting (do šířky grafu „sousednosti" DSA)

Lze doplňovat, definovat nové položky. X.500 umožňuje aliasing, aby nebyly duplicitní informace na více místech.

9 Šifrování

Šifrování probíhá v těchto fázích:

`srozumitelný text M` → $\boxed{\text{šifrování } C = E_k(M)}$ → `šifrovaný text C` → $\boxed{\text{dešifrování } M = D_k(C)}$ → `srozumitelný text M`.

Šifruje se podle klíčů:

- symetrické × asymetrické
- tajný klíč × veřejný klíč

Kryptografie tajným klíčem

- šifrování a dešifrování se provádí stejným (tajným) klíčem. Používá se pro utajení (digitální obálka).
- uživatelé se dohodnou na tajném klíči.
- užívá se symetrického klíče
- norma DES (nesmí mimo USA), 56-bitový klíč

9.1 Symetrické šifrování

Viz [1] kap. 10.8.2 str. 122 a další.

9.2 Asymetrické šifrování

`zpráva`

→ šifrování veřejným klíčem → `šifrovaný text` → dešifrování soukromým klíčem → `zpráva`

Kdokoli může tímto způsobem veřejným klíčem poslat tajnou zprávu. Často se kombinuje s kryptografií tajným klíčem.

Bližší informace ve skriptech [1] kap. 10.8.3 str. 124.

Asymetrie je pomalejší, ale lépe se distribuuje.

9.3 Ověření pravosti zprávy

Toto ověřování je pro zjištění, zda zprávu při dopravě nikdo nezměnil. Pro zjištění „otisku prstu" zprávy – jednocestová rozptylovací funkce.

Velikost charakteristiky zprávy (message digest) je 12 nebo 16 bitů.

Skripta 10.8.4 str. 125.

9.4 Digitální podpis

Kódování podpisu se provede soukromým klíčem a autentizace odesílatele veřejným: `zpráva` → podpis kódovaný soukromým klíčem → `podpis` → verifikace veřejným klíčem → `platný / neplatný`.

Digitální podpis bývá kombinován s charakteristikou zprávy.

Více viz skripta 10.8.5 str. 126.

9.5 Posílání tajného klíče

`zpráva` → šifrování tajným klíčem TK → `šifra1` a potom `TK` → šifrování veřejným klíčem VK → `šifra2`.

Dešifrování zprávy se provede tak, že adresát svým soukromým klíčem dešifruje klíč TK a pomocí něj dešifruje zprávu.

9.6 Dvojí šifrování

Nechť O je odesílatel a A adresát, nechť $i \in \{A, O\}$ a nechť je SK_i je soukromý klíč subjektu i a VK_i je veřejný klíč i. Pak průběh zprávy proběhne tímto způsobem:

`zpráva` → šifrování pomocí SK_O → `mezikód` → šifrování pomocí VK_A → `šifra` → průchod šifrované zprávy sítí → `šifra` → dešifrování pomocí SK_A → `mezikód` → dešifrování pomocí VK_O → `zpráva`.

10 Bezpečnost informačních systémů

Informační systém je:

- Hardware (procesor(y), paměti, terminály, ...)
- Software (aplikační programy, operační systém,...)
- data (údaje uložené v databázích, výsledky výpočtů)
- lidé (uživatelé, personál působící při zpracování dat)

Příklady: výzkum a vývoj, burza, banka, knihovna,...

Aktiva informačního systému jsou HW + SW + data, což má tržní cenu.

Bezpečnostní politika se souhrn norem, pravidel a praktik definujících způsob zpracování, ochrany a distribuce citlivých informací v rámci činnosti informačního systému.

Objekt informačního systému je pasivní entita, která obsahuje / přijímá informace zpřístupňované subjekty informačního systému (udělováním práv přístupu subjektům).

Subjekt IS je aktivní entita (osoba, proces nebo zařízení činné na základě příkazu uživatele) autorizovatelná pro:

- získání informace z objektu
- změnu stavu objektu
- vydání příkazů ovlivňujících udělení práv přístupu k objektu

Autorizace je určení, zda subjekt je důvěryhodný z hlediska jisté činnosti.

Důvěryhodný IS (subjekt / objekt) je takový, o kterém se věří, že je implementovaný tak, že splňuje svou specifikaci vypracovanou v souladu s bezpečnostní politikou.

Bezpečný IS (subjekt / objekt) je ten, na který se můžeme plně spolehnout a který se chová tak, jak očekáváme, že se bude chovat.

Zranitelné místo je slabina IS využitelná ke způsobení škod / ztrát. Je důsledkem chyb v analýze, návrhu, implementaci. Je hrozbou, jejíž existence představuje jisté riziko (pravděpodobnost).

Protiopatření jsou administrativní, fyzická, logická opatření, odstraňující zranitelná místa nebo snižující zranitelnost danou hrozbou.

Zranitelnost IS je dána:

- vysokou hustotou informací
- složitostí SW
- skrytými vazbami a kanály
- elektromagnetickým vyzařováním (fyzikální rizika)

Hrozba, nebezpečí je akce nebo událost, která může ohrozit bezpečnost způsobením ztráty, škody na aktivech IS, např.:

- útok lidského činitele
- neautorizované zpřístupnění dat
- ...

Čím je dána bezpečnost IS?

1. zajištěním důvěrnosti, tj. k aktivům IS mají přístup pouze autorizované subjekty
2. zajištěním integrity, tj. aktiva IS smí modifikovat pouze autorizované subjekty
3. zajištěním dostupnosti, tj. aktiva IS jsou autorizovaným subjektům dostupná (do určité doby)

IS absolutně utajující svá aktiva zachovává sice důvěrnost, ale je k ničemu.

Typy útoků (projev hrozeb):

1. přerušením, tj. ztrátou, znepřístupněním, poškozením aktiva – porucha periferie, vymazání programu, dat, porucha OS
2. odposlechem, tj. neautorizovaná strana si zpřístupní aktiva, okopírování programu, dat
3. změnou, tj. neautorizovaná strana zasáhne do aktiva
4. přidáním hodnoty, tj. neautorizovaná strana něco vytvoří (podvržení transakce, ...)

Útok na hardware

- živelné / bezděčné havárie
  - ⋆ neúmyslné škody
  - ⋆ záplava, požár,...
  - ⋆ obsluha (drobky, tekutiny,...)
  - ⋆ prach, kouř,...
  - ⋆ otřesy, údery,...
- úmyslné havárie

Útoky na software

- vymazání
  - ⋆ náhodné zrušení programu
  - ⋆ archivace nesprávné kopie
- změna
  - ⋆ logická bomba (datově destrukční program)
  - ⋆ trojský kůň (dělá to co má + něco navíc)
  - ⋆ tajné vstupní body
  - ⋆ prosakování informací
- krádež
  - ⋆ neautorizované okopírování

Útok na data Data mají hodnotu

- dočasnost hodnoty dat
- cena rekonstrukce dat
- tržní hodnota obsahu
- cena opětovného vytvoření
- porušení důvěrnosti, utajení
- porušení integrity (salámový útok, modifikace / generování transakcí)

Kdo může útočit?

- počítačový zločin
  - ⋆ zločin provedený pomocí IS
  - ⋆ zločin provedení proti IS
- amatéři
- hackeři, koumáci
- profesionální zločinci (potenciálně neomezené prostředky)

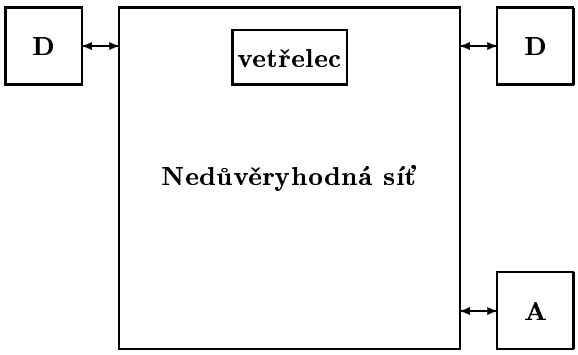
Bezpečnostní cíle a jejich dosažení Dílčí přínosy k bezpečnosti, kterou dosahuje IS z hlediska udržení:

- důvěrnosti
- pohotovosti
- integrity

Funkce IS bezprostředně přispívající ke splnění bezpečnosti:

- identifikace a autentizace
- řízení přístupu
- účtovatelnost
- utajování
- audit (kdo kdy pracoval se systémem, musí být off-line přístupný arbitrovi)
- řízení opakovaného užívání
- implementace funkcí prosazujících bezpečnost

Bezpečnostní mechanismus je logika nebo algoritmus, kterým hardware nebo software implementuje funkce prosazující bezpečnost (opírá se o kryptografii).



Obrázek 3: Schéma bezpečné komunikace

10.1 OSI Security Architecture

OSI Security Architecture popisuje obrázek 3. **D** jsou důvěryhodné aplikační systémy, **A** je v této architektuře nutná důvěryhodná certifikační autorita, což může být arbitr, třetí strana v komunikaci, která posuzuje autorizaci. Tato autorita však nemusí být nutně on-line dostupná.

Hrozby podle OSI SA:

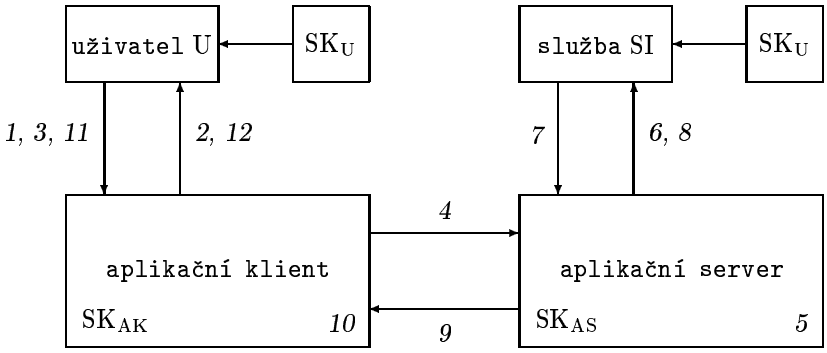
- maškaráda (narušení autentizace, subjekt si hraje na jiný, než je)
- odmítnutí služby
- zapuzení zprávy (uzel si zprávu přečte a pak ji nechce předat)
- únik informace
- modifikace dat v DB (změna / zrušení)
- modifikace toku zpráv (změna pořadí)
- změna SW (viry, trojský kůň)
- dedukce informace
- neautorizovaný přístup
- porušení autentizační a autorizační politiky

Bezpečná komunikace v OSI SA:

- oba komunikující partneři komunikují se skutečně oznámeným partnerem
  - ⋆ vzájemná autentizace
  - ⋆ autentizace zdroje dat
- nikdo nemůže provádět odposlech (důvěrnost)
- nikdo nemůže přenášenou informaci měnit (integrita)
- služby jen pro autorizované uživatele (řízení přístupu)
- nikdo nemůže popřít komunikace (nepopiratelnost vyslání[2]).

Bezpečná spojová relace podle OSI SA:

1. Navázání spojení se zaručenou vzájemnou autentizací (PKC). Získá se klíč pro symetrické šifrování (SKC) zabezpečující integritu a důvěrnost fáze 2.
2. Proběhne přenos informace se zabezpečenou integritou a důvěrností. Slabá integrita zabezpečuje jen proti náhodným poruchám; Silná integrita chrání i proti vetřelcům.
3. Spojení se zruší (v síti už nejsou žádná data).
4. Ověří podepsané části informace (fáze nepopiratelnosti).



Obrázek 4: Princip vzájemné autentizace

10.2 Vzájemná autentizace dle OSI SA

Princip vzájemné autentizace je uveden na obr. 4.
Vysvětlivky k obrázku 4:

1. žádost o službu SI → AK.
2. aplikační klient vypracuje token TU = {U, AK, SI, AS, t}, kde t je logický čas (ochrana proti duplikacím).
3. uživatel udělá podpis {TU} *rightarrow* AK
4. Podpis$_{AK}$ { Podpis$_U$ {TU}} *rightarrow* nedůvěryhodná síť *rightarrow* AS
5. AS ověří identitu AK (podle podpisu)
6. předá to SI, SI ověří identitu uživatele a vyhodnotí t (kvůli maškarádě).
7. AS ověří aktuálnost (veřejné kódování) a vytvoří token služby TS = { SI, AS, U, AK, $t+1$, PKC,(SKC$_{AB}$)}
8. Podpis$_{SI}${ TS }
9. Podpis$_{AS}${ Podpis$_{SI}$ { TS }} → AK
10. AK ověří identitu AS
11. uživatel ověří identitu služby, vyhodnotí $t+1$
12. AK ověří aktuálnost

10.3 Funkce a mechanismy OSI SA

1. autentizace (PKC)
   - vzájemná (spojová komunikace)
   - původu dat (nespojová komunikace)
   - 7. vrstva
2. integrita (SKC^i)
   - slabá / silná
   - s opravami / bez oprav

[2]právně se prokazuje u arbitra, silnější požadavek než předchozí

- zpráv / vybraných polí zpráv (rozhraní standardních služeb OS; integrita zpráv na transportní úrovni)
- nejníže ve 4. vrstvě, vybraná pole zpráv v 7. vrstvě (jen aplikace umí rozebrat record na položky)

3. utajení ($SKC^u \neq SKC^i$)
   - zpráv / vybraných polí zpráv
   - toku zpráv
   - nejníže ve 4. vrstvě, vybraná pole zpráv v 7. vrstvě
4. neodmítnutelnost
   - s prokázáním původu
   - s prokázáním doručení
   - notář
5. řízení přístupu

Autentizace je to, že vím, s kým komunikuji, kdežto neodmítnutelnost je to, že vím, s kým komunikuji, a mohu to dokázat (certifikační autoritě).

11 Komprese dat

Počítačové sítě jsou prostředí, vyžadující minimalizaci „objemu" zobrazení dat.

- přenos souborů po telefonních linkách (sítí)
- archivace SW (pevný disk, pružný disk)
- distribuce SW (3MB balík SW na 1,4MB pružné disky)
- on-line zpřístupnění rozsáhlých informačních celků („help" soubory v integrovaném vývojovém / pracovním prostředí).

Upozornění! Bližší informace o všem, co je uvedeno v této kapitole obsahuje skriptum [1]

11.1 Jak informace měřit?

Jak souvisí počet bitů zprávy s množstvím informace, kterou zpráva nese?
Míra:

$$mi = I(Z)$$

mi je množství informace, I je míra, Z je zpráva.
mi roste se snižováním pravděpodobnosti výskytu zprávy Z.
mi v nezávislých zprávách $Z_1, \ldots, Z_n$ je rovna

$$\sum_i mI(Z_i)$$

mi je vždy kladné. $I(Z) = -\log p(Z)$.
$Z = \{Z_1, \ldots, Z_n\}$ jsou možné podoby zprávy Z.
Střední hodnota očekávané informace

$$I_s(Z) = -\sum_i p_Z \cdot \log p_{Z_i}$$

Např. $Z = \{Z_1, \ldots, Z_n\}$, kde Z je žádost o službu FTAM a Z_1 je open file, ..., Z_n delete FTAM, přičemž Z je jev a $Z_1, \ldots, Z_n$ jsou výsledky jevu.

Entropie H zprávy o jevu Z je míra střední informace, kterou získáme ze zprávy.

$$H(Z) = -\sum_{i=1}^{n} (p_{Z_i} \cdot \log p_{Z_i})$$

. Maximální entropie je, když všechny $p_{Z_i} = \frac{1}{n}$, tedy

$$H_{max}(Z) = -\sum_{i} \frac{1}{n} \cdot \log \frac{1}{n} = n\frac{1}{n} \cdot \log \frac{1}{n} = \log n$$

Entropie dvou nezávislých jevů se rovná součtu entropií obou jevů.

Objem zobrazení dat je počet symbolů použitých pro zápis zprávy (nebo taky počet bitů použitých pro zakódování zprávy). Je to diskrétní veličina.

- zpráva (paket)
- symbol zprávy (slabika)
- prvek abecedy (znak)

Zpráva je posloupnost n symbolů, z nichž každý může nabýt hodnoty prvku použité abecedy o m prvcích.

Střední množství informace nesené jednou zprávou (měří se logaritmickou mírou neurčitosti rozdělení pravděpodobnostního výskytu jednotlivých prvků abcedy):

$$Hn = -n \cdot \sum_{i=1}^{m} p_i \cdot \ln p_i$$

kde p_i je apriorní pravděpodobnost výskytu prvků abecedy ve zprávě.

Střední množství informace připadající na jeden symbol $\frac{Hn}{n}$ je

- ln $\sim$ přirozené jednotky
- $\log_2 \sim$ bity

Maximální množství informace nese zpráva o n symbolech tehdy, když $p_i = \frac{1}{m}$ (tj. všechny prvky abecedy na místě symbolů se vyskytují se stejnou pravděpodobností).

$$(Hn)_{max} = n \cdot \ln m$$

Čím větší je rozptyl pravděpodobností výskytu prvků použité abecedy jako symbolů zprávy, tím méně informace zpráva s daným počtem symbolů nese.

Zpráva vyjádřená $n' < n$ symboly může nést stejné množství informace jako zpráva vyjádřená n symboly, pokud rozptyl pravděpodobností výskytů prvků abecedy A' ve zprávě je menší než v případě abecedy A.

$$-n \cdot \sum_{i}^{m} p_i \ln p_i = -n \sum_{j}^{m'} p_j \ln p_j$$

Pravděpodobnost výskytu prvku závisí na jeho okolí – kontextu (na prvcích v okolních symbolech).

- v textu se vyskytují určité posloupnosti symbolů s různou pravděpodobností (`"the"`)
- vyskytují se posloupnosti symbol-u s určitou vlastností (`1274.7 SLOVO`)
- obsah zprávy závisí na obsahu předchozí zprávy

S využitím těchto závislostí na kontextu se sníží délka zprávy.

Nechť Z_A je zpráva vyjádřená prvky abecedy A a $Z_{A'}$ je zpráva vyjádřená prvky abecedy A' a f kódování. Pak $Z_{A'} = f(Z_A)$, přičemž je zachováno množství informace.

Proč kódování:

- změna mohutnosti abecedy (např. { A,..., Z} $\longrightarrow$ { 0,1 })
- utajení (kryptografie)
- minimalizace redundandnosti zpráv
  - ⋆ snížení rezdílu $H - H_{max}$
  - ⋆ odstraňování kontextové závislosti

11.2 Frekvenčně závislé kódování

Nechť jsou zprávy tvořeny symboly z prvků abecedy $\mathcal{A} = \{A, B, C, D\}$ a $p_A = 0,5$, $p_B = 0,3$, $p_C = 0,15$,$p_D = 0,05$ jsou pravděpodobnosti výskytu. Zpráva je např. ABAABC.

Střední množství informace připadající na jeden symbol (v bitech) je:

$$\begin{aligned} \frac{Hn}{n} - 0.5\log_2 0.5 &= 0.5 \\ -0.3\log_2 0.3 &= 0.48 \\ -0.15\log_2 0.15 &= 0.41 \\ -0.05\log_2 0.05 &= 0.22 \end{aligned}$$

Tedy $\frac{Hn}{n} = 1.61$ bitů/symbol

Máme překódovat abecedu $\mathcal{A}$ do abecedy $\mathcal{A}' = \{0, 1\}$.

Jak to provést, aby se průměrný počet bitů/symbol zprávy blíží teoretické mezi (1.61 bitů/symbol), tj. 2 bity, viz následující tabulka:

A	B	C	D
00	01	10	11

A tedy:

$$\begin{aligned} 0.5 \cdot 2 &= 1.0 \\ 0.3 \cdot 2 &= 0.6 \\ 0.15 \cdot 2 & \quad 0.3 \\ 0.05 \cdot 2 &= 0.1 \end{aligned}$$

tedy po sečtení 2.0.

Zvolíme-li však kódování podle této tabulky:

A	B	C	D
0	10	110	111

dostáváme hodnoty:

$$\begin{aligned} 0.5 \cdot 1 &= 0.5 \\ 0.3 \cdot 2 &= 0.6 \\ 0.15 \cdot 3 & \quad 0.45 \\ 0.05 \cdot 3 &= 0.15 \end{aligned}$$

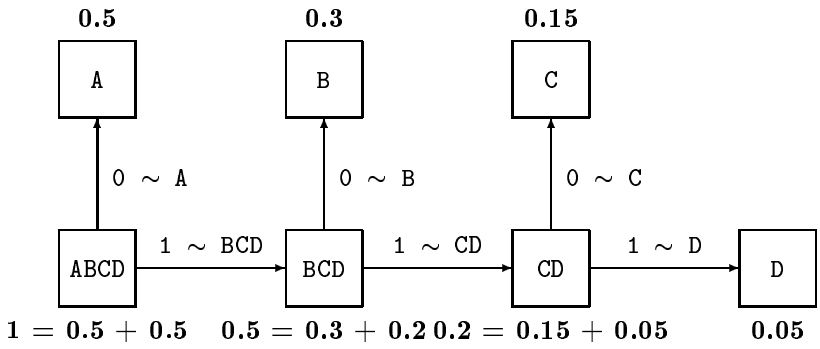
tedy po sečtení 1.7 bitů/symbol.

Tedy z toho plyne ponaučení, že prvky s větší pravděpodobností výskytu $\Longrightarrow$ kratší bitové řetězce.

11.3 Huffmannovo kódování

Je to nejčastější kompresní metoda. Vytvoří se kódovací strom podle pravděpodobnosti výskytu prvků. Náš příklad by měl tento kódovací strom:

Typická komprese pomocí Huffmannova kódování je na 80% délky. Je však nutné se souborem přenášet i kódovací strom (0.5 ÷ 1KB).



Obrázek 5: Příklad kódovacího stromu Huffmannova kódování

11.4 Aritmetické kódování

Každou zprávu zakódujeme jako číslo.

Nechť máme abecedu { I, J, K, L} a příslušné pravděpodobnosti 0.4, 0.3, 0.1, 0.2. Výsledná „abeceda" je interval $< 0, 1)$. Dejme tomu, že chceme kódovat slovo LIK. Postup zakódování je tento:

1. rozdělím interval $< 0, 1)$ podle pravděpodobností I, J, K a L.
2. vezmu 1. znak slova LIK, čili L.
3. vezmu interval, který přísluší L (tj. $< 0.8, 1)$) a rozdělím jej podle pravděpodobností I, J, K a L.
4. vezmu 2. znak slova LIK, čili I.
5. vezmu interval, který přísluší I (tj. $< 0.8, 0.88)$) a rozdělím jej podle pravděpodobností I, J, K a L.
6. vezmu poslední znak slova LIK, čili K.
7. vezmu interval, který přísluší K (tj. $< 0.856, 0.864)$) a vezmu jeho střed (0.86) a to je zakódované slovo LIK.

Dekódování:

1. $0.86 \in < 0.8, 1.0) \Rightarrow$ L
2. $0.86 \in < 0.80, 0.88) \Rightarrow$ LI
3. $0.86 \in < 0.856, 0.864) \Rightarrow$ LIK

Nutno je však ještě zvolit způsob označení délky zprávy, protože v našem případě bychom mohli dekódovat dále.

znak	p_i	dolní mez	horní mez
I	0.4	0.0	0.4
J	0.3	0.4	0.7
K	0.1	0.7	0.8
L	0.2	0.8	1.0

Kódování

```
dm := 0;
hm := 1;
while getch(ch) != '$' do
  interval := hm - dm;
  hm := dm + interval * hmi(ch);
  dm := dm + interval * dmi(ch);
enddo;
pnt(dm);
```

Dekódování

```
get(cislo);       – zakodovana zprava
repeat decod(cislo,ch)        – vrati symbol do jehoz intervalu cislo patri
  interval := hmi(ch) - dmi(ch);
  cislo := cislo - dmi(ch);
  cislo := cislo / interval;
until ch = '$';
```

11.5 Kontextově závislé kódování

Informační hodnotu dané zprávy zachováme v jiné zprávě, ve které připadá na jeden symbol větší střední množství informace.

Přepínače / přeřazovače / znaky / změny / *. Dvě různé části zprávy jsou vyjádřeny stejným prvkem abecedy.

Escape posloupnosti.

11.6 Čítače opakovaných prvků (RLE)

Místo posloupnosti `rrrrrrrrrrrrrrrrrrrrXYZ=A` bude vytvořena posloupnost `#20#r#XYZ=A#`. Lze tímto způsobem dosáhnout až 60% úspory místa.

Analýza na úrovni bitového zobrazení zprávy, převažuje-li jeden z prvků abecedy. 111 *equiv* 7x + hodnota další trojice bitů. 1111111000 $\sim$ 111111000 tedy $7 + 7 + 0 = 14$. Takto až 36% úspory.

11.7 Vysílání diferencí zpráv

Opakované vysílání „téměř totožných zpráv" na aplikační vrstvě. Lze tedy vysílat pouze změny v dané zprávě proti předcházející zprávě.

Řešení komprese na vrstvě

- aplikační: při návrhu aplikace (s konečnou množinou zpráv) např. textový soubor položek.
- služba jádra systému (pro ukončenou množinu zpráv)
  - ⋆ frekvenčně závislé kódování
  - ⋆ kontextově závislé kódování

11.8 Statistické metody vyšších řádů

0. řádu – frekvenčně závislé kódování

1. řádu – kontextově závislé kódování

Paměťová náročnost kódování je m^{o+1}, kde o je řád kódování, tedy 0. řád 256B, 1. řád 64KB, 2. řád 16MB. Proto se používá v omezené verzi, tzv. Follower Sets Method (metoda následníků).

11.9 Klouzající slovník

Nekóduje se explicitní slovník, což je poslední úsek komprimačního textu (asi 4KB). Opakované řetězy prvků abecedy se dynamicky nahrazují dynamicky vytvářenými prvky nové abecedy. Na tomto principu funguje program ARC. Abeceda {0, 1, 2, ... , 255} se při kompresi změní na {0, 1, 2, ... , 255, 256 (kód 1: opakovaný řetězec), 257 (kód 2: opakovaný řetězec), ... }

11.10 LZW komprese

Původní abeceda A: { 0,..., 255 }, kde je 8 bitů/prvek. Nová abeceda A' bude { 0,..., 4095 }, což je 12 bitů/prvek. Slovník je budován asi takto:

Nechť přenášíme zprávu `This is a ...`:

Vstup	**Slovník Buffer**	**Výstup**
T		
h	T 256 = Th	T
i	h 257 = hi	h
	i	
	...	
	258 = is	

Výstupní sekvence tedy bude `This␣.258`.

Některé problémy této implementace:

- Rychle narůstá délka tabulky prvků nové abecedy (opakovaných řetězců).
- Neuchovávat textové formy, ale kódové formy.
- Při vyčerpání kapacity slovníku lze zapomínat existující formy.
- Prodlužuje se doba hledání řetězů v tabulce (hash)
- Volba bitové šířky prvků nové abecedy ovlivňuje poměr komprese
- ARC mění šířku prvků dle rozsahu původní abecedy

Kontextovou a frekvenční kompresi lze kombinovat.

Komprese:

```
GET(STRING);
while not EOF
loop  GET(CHARACTER);
      if IN(STRING & CHARACTER, LIST)          - LIST je nova abeceda
      then STRING := STRING & CHARACTER;
      else OUTPUT_CODE(STRING,LIST);
           ADD(STRING x CHARACTER, LIST);
           STRING := CHARACTER;
      endif;
endloop;
OUTPUT_CODE(STRING, LIST);
```

12 Nepřímá asynchronní komunikace mezi procesy (BSD sockets)

virtuální okruhy, datagramy – spojované, nespojované služby, nastavení pomocí směrovacích tabulek – pevná cesta. Datagramy – samostatné jednotky, směřují se v uzlech.

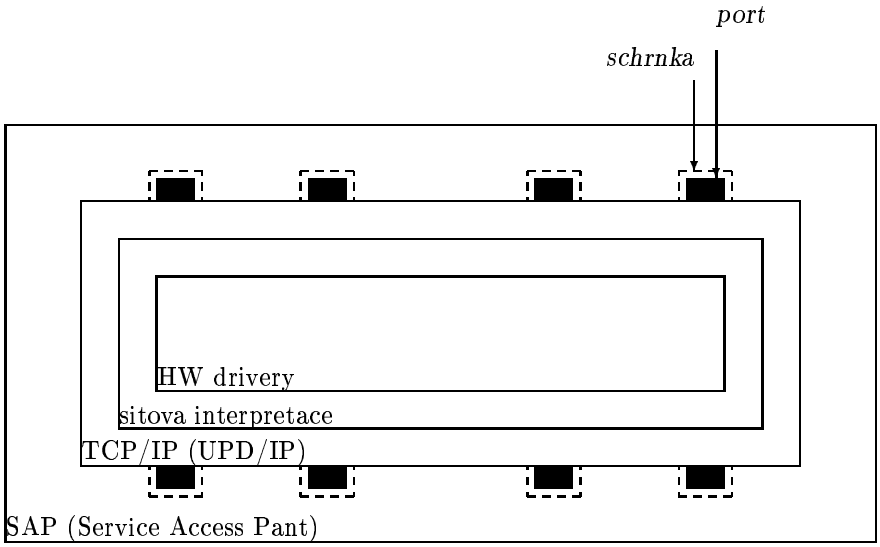
Socket – BSD metoda komunikace mezi procesy.

InterProcess Communication (IPC) ukazuje obrázek 6.

Obrázek 6: Princip IPC

BSD obsahuje knihovnu `libsocket.a`, která obsahuje systémová volání (služby jádra) a knihovní podprogramy pro obsluhu BSD soketů.

Schéma interprocesové komunikace uvádí obrázek 7.



Obrázek 7: Schéma IPC

Port – adresovací místo „sítě".

Schránka – obousměrný koncový komunikační bod v procesu. Ekvivalentní označení je *socket*.

Proces si obhospodařuje soubory, speciální soubory (zařízení) a schránky.

Domény – např. Darpa, Xerox, Appletalk jsou adresové prostory se stejným způsobem identifikace.

Rodiny protokolů musí definovat:

- vytváření paketů
- směrování
- adresace

Sémantika komunikace – způsob zabezpečení služby. Rozlišujeme:

- datagramy (nespojová transportní služba)

- virtuální okruhy (spojová transportní služba)
- RAW (nástroj pro přímý přístup k driverům, určený pro tvorbu nových rodin protokolů)

Mohou spolu komunikovat pouze dvě schránky téhož typu. RAW umožňuje přeskočit transportní vrstvu a pracuje s nimi přímo IP.

Atributy komunikace jsou domény a typy schránek.

Mohou spolu komunikovat pouze schránky vytvořené ve stejné adresové vrstvě a se stejným typem. Nutnost shody na obou koncových bodech.

paragraph`socket()` podle formátu adresy (address family)

- AF_UNIX adresa je unixová přístupová adresa (IPC v jednom uzlu)
- AF_INET adresa je InterNetová adresa (Darpa, TCP/IP). Jedná se o IPC mezi uzly sítě.

Podle charakteru přenosu se určí typ:

- SOCK_STREAM spojová služba
- nespojová služba (viz man)

Dále je třeba zadat použitý protokol.

Schránka je tedy vytvořena voláním:

```
s = socket(AF_INET, SOCK_STREAM, 0)
```

přičemž `s` je vrácený deskriptor schránky. Datový typ, který funkce `socket` vrací je stejný jako file descriptor, ale `s` není svázáno se souborem ani zařízením, nýbrž portem (TSAP), tj. adresovatelným místem v lokálním uzlu sítě.

Získání protokolu / výběr protokolu:

```
pp = getprotetyname("tcp");
s = socket(AF_INET, SOCK_STREAM, pp -> p_proto);
```

Bind – svázání schránky s portem. Port je identifikován číslem portu (1... 50 000 portů v každém uzlu), číslem uzlu a číslem sítě. Přitom porty čísel 1 ... cca 1000 jsou privilegované porty určené pro standardní služby a ostatní jsou volně využitelné procesy (např. pro výstup z klienta).

Volání jádra `bind()` vyžaduje specifikaci lokální adresy:

```
struct sockaddr_in sin;
bind(s,sin,sineof(sin));
```

kde `s` je deskriptor schránky, `sin` je struktura, v níž je uložena adresa, `sineof(sin)` je velikost adresy.

Volání `bind` vrací nulu při úspěšném provedení, záporné číslo, pokud nastala chyba.

Určení SAP (místa poskytování služby): lze z něj získat jméno služby (číslo portu) voláním `getverbyname`, jméno uzlu (číslo uzlu) voláním `gethostbyname`, které vrátí síťovou adresu uzlu.

Volání služeb jádra při komunikaci Klient/Server uvádí přehledně tabulka 2

Server:	**Klient:**
`socket`	`socket`
`bind`	`bind` (nemusí už volat, protože se zavolá explicitně v rámci volání `connect`)
`listen`	
`accept` (ukončí se až přijde `connect`)	`connect`
`read/write`	`read/write`

Tabulka 2: Volání jádra při komunikaci klient/server

Pasivní × aktivní koncový bod spojení, proto asymetrická komunikace.

Definice schránky v serveru, na které se poskytuje služba typu virtuální kanál:

1. požádá se o navázání spojení (klient → server)
2. server akceptuje žádost a potvrdí navázání klientovi
3. klient používá službu, násobné žádosti se řadí do fronty

`ftpd` – může poslouchat na všech schránkách připojených na porty současně.
`establish()` je knihovní funkce na vytvoření schránky, která:

1. vyprázdní adresové struktury
2. zeptá se, kdo jsem?
3. jakou mám síťovou adresu?
4. existuji? (jsem zapojen do sítě?)
5. toto je adresa mého uzlu
6. toto je číslo portu, na který se vytvořená schránka připojí
7. vytvoření schránky
8. svázání schránky s portem
9. určení maximálního počtu žádostí o spojení ve frontě.

12.1 Klientova žádost o navázání spojení

Navázání spojení ≡ vytvoření virtuálního okruhu.

```
schranka = socket ...
connect(schranka, text_adresy, vzd_port, delka_textu);
```

Při klientově žádosti dojde k implicitnímu svázání na port, tudíž nemusím vkládat volání `bind` mezi `socket` a `connect`.

Ustavení virtuálního okruhu se provede následovně: server si vytvoří ještě pracovní schránku, volání přesměruje do ní, tam je obsluha, sám se vrátí a přijme další požadavek (a ten opět přesměruje).

Spojení je uspořádaná pětice:

1. lokální uzel
2. lokální port
3. vzdálený uzel
4. vzdálený port
5. protokol

12.2 Potvrzení přijetí žádosti klienta serverem

Je provedeno voláním `accept`, jehož parametry jsou: obsluhovaná schránka, text adresy klientova portu a délka tohoto textu.

12.3 Obsluha komunikace klient/server

Samostatná obsluha komunikace klient/server obsahuje

- inkludy
- náhodně zvolené číslo portu (něco musíme mít)
- odstranění duchů
- při `accept()` může nastat chyba `EINTR`
- příprava masek (`select(..., ..., line_out)`)
- analýza

12.4 Datagramová komunikace (UPD)

Formálně je shodná s virtuálním okruhem (`connect`, `read`, ...), ale obsahuje speciální volání jádra `sendto` a `recefrom`, které pošle / přijme přímo ze/do schránky na/z port(u).

13 Internet

13.1 Struktura Internetu

- služby ftp, telnet, E-mail (aplikační vrstva)

– (BSD sockets)

- TCP (spolehlivý transport dat – transportní vrstva)
- IP (nespojovaná doprava datagramů – směrování toku dat)
- hostitelská síť (dopravuje datagramy Internetu ve svých rámcích – tok dat)

13.2 Internet protocol (IP)

Struktura IP datagramu:

- záhlaví: verze, délka, čas, síťové adresy odkud a kam se přenáší, ladící volby
- data: transportní packet

Data jsou „obalena" transportní službou, kvůli detekci duplikací apod.
Identifikace objektů v síti se děje čtveřicí dvoumístných hexadecimálních čísel: XX.XX.XX.XX, přičemž první dvě dvojice identifikují síť (např. 135.10) a druhé dvě dvojice uzel v této síti, tedy 135.10.27.14 je uzel 27.14 ležící v síti 135.10.

Klady a zápory takové identifikace

+ snadno lze algoritmicky zpracovat při směrování (kudy se má jít)

- dlouhá čísla

- těžce pamatovatelé identifikace

- nic neříkají o objektu (kdo to je, co dělá)

Proto Internet zavádí symbolická jména uzlů:

+ je (mělo by být) synonymem pro objekt

+ je snadno zapamatovatelné

Proto Internet musí definovat transformační funkce:

$$\texttt{address} = f(\texttt{name})$$

a

$$\texttt{name} = f^{-1}(\texttt{address})$$

Internet však dnes již obsahuje 2.600 sítí s celkem 1.500.000 počítačů a proto je nutné zajistit jednoznačnost funkce f. Proto přidělování jmen organizuje SRI NIC. Ta to organizuje stromovou strukturou, kdy koordinuje správce adres v jednotlivých zemích, ti zase koordinují správce jednotlivých sítí a ti nakonec správce uzlů. Tím se tento problém stává lokální záležitostí v jedné síti.

Je-li nainstalován nový uzel { jméno, adresa }, pak jméno dává určení místa a adresa dává identifikaci uzlu. Proto znám-li jméno, pak Internet už ví, kam má dopis, soubor, ... poslat.

Problém: Pojmenovaný uzel změnil adresu. Jak informovat ostatní uzly o této změně?

1. „hosts" soubory (každém uzlu). Je to soubor tvaru:

```
adelard.dcs     192.47.125.1
artemis.dcs     192.47.125.24
```

   - každý uzel × každý uzel $\Longrightarrow$ kvadratická složitost změny. V síti sítí nemožné.
   - změna se musí zanést do všech uzlů

2. „name server", tedy je definována „podsíť" uzlů, které „ovládají" určité území a do nich se zanášejí informace o jménech a adresách. Ostatní uzly znají adresu „nejbližšího" name serveru. V případě, že uzel nezná adresu nějakého jména, zeptá se svého name serveru (ten to buď ví, nebo se zeptá jiného name serveru v podsíti name serverů). Možná hierarchie name serverů je v rámci domén (nevím-li něco, zeptám se nadřazeného servera).

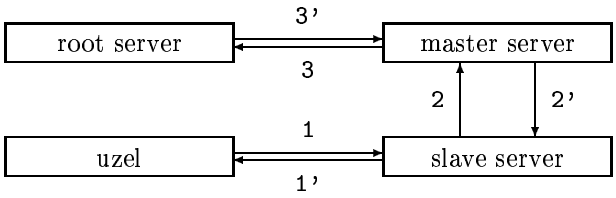
13.3 Domény

doménové jméno: uzel(.poddoména)*.vnější doména.

Pro každou (pod)doménu se zavádí name server. Vnější doména je v Evropě stát (`cz` Czech, atd.), v USA ekonomická sféra (`com` Commerce, `edu` Education, `mil` Military, `gov` Government) nebo přechod do jiné sítě (`bitnet`, `arpa`). Poddomény (např. `muni` Masaryk University, `vutbr` VUT Brno) jsou již organizovány podle potřeb organizací do libovolné hloubky. Name server pro vnější doménu se nazývá *root server* (`cz`), pro nejvyšší doménu *master server* (`muni.cz`), další už jsou jen name servery (`dcs.muni.cz`).

13.4 Root server

Root server zná vše o nejvyšších doménách. Např. `cz` neví nic o `dcs.muni.cz`, ale ví, který name server to ví. Postup při hledání této informace je na obrázku 8.



Obrázek 8: Postup práce root servera

Požadavky na provoz root servera:

- pohotovost
- po celých 24 hodin
- spolehlivý uzel, přístupný
- v centru sítě, ve výkonném uzlu

13.5 Master server

Je autorita v dané doméně. Může být master pro více domén. Pro jednu doménu je primární server (zavádí svoji databázi), pro ostatní je sekundární (jeho znalosti nahrávány z primárních serverů (dynamicky). Při výpadku primárního MS funguje sekundární MS jako záloha.

13.6 Cashing server

nemá svou databázi, má cache a dotazuje se ostatních serverů (primárních a sekundárních master serverů).

13.7 Forwarding server

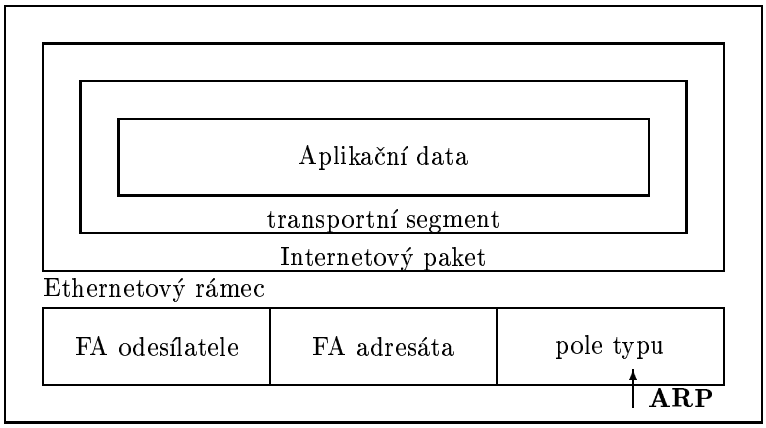
Tento server zprostředkuje dotazy od slave serverů, které mají přístup pouze v rámci jedné sítě. ptají se kterýchkoli PMS, SMS, CNS. Pokud FNS v síti chybí, je tato síť izolovaná

13.8 Fyzická adresa

Fyzická adresa je šestislabičná a je vypálená v Ethernetové kartě. Je přidělována realizátorem sítě, Internetová adresa je dána administrátorem. Po Ethernetu nesmí najednou vysílat více než jeden uzel, na každý uzel se musí v konečném čase dostat. Vysílání je určeno všem, ale data přejímá pouze adresát (uzel s příslušnou fyzickou adresou).

ARP (address resolution protocol) určuje transformace síťových adres na fyzické a dynamické udržování tabulek v každém uzlu.

Hierarchii přenosu dat po Ethernetu ukazuje obr. 9.

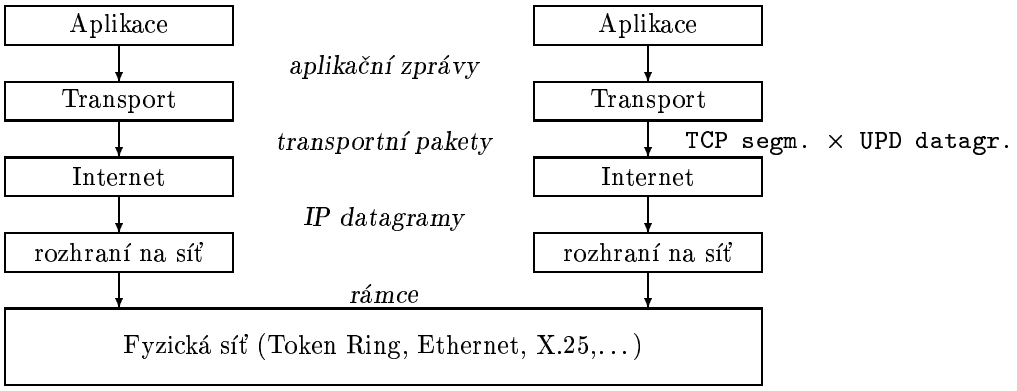


Obrázek 9: Princip ARP

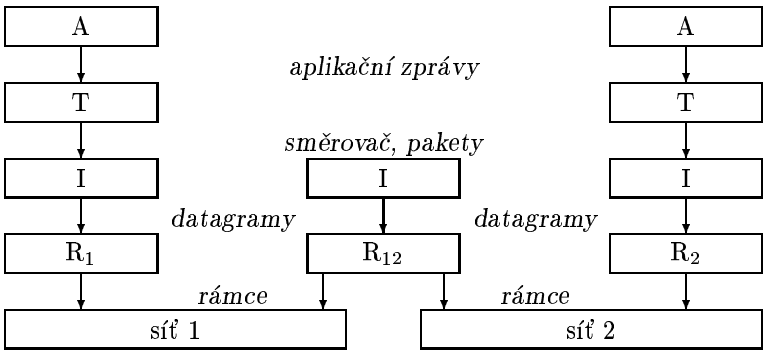
RARP (reverse ARP) transformace fyzické adresy na Internetovou. Stává se to např. je-li v síti disk-less station, která nemá kde uložit IP adresu. Proto jsou zřizovány v této síti RARP (primary a backup) servery, které tyto informace udržují v databázích (`/dev/inet/arp`) a které mají software na manipulaci s těmito databázemi (`arp`).

IP adresy se ještě rozlišují podle 1. čísla identifikace sítě:

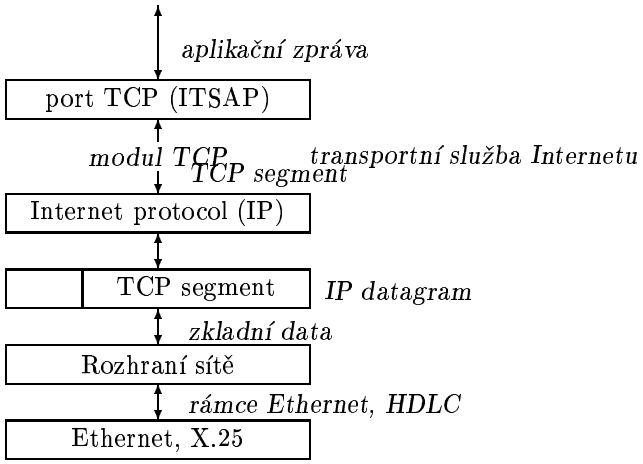
- Kategorie A: 0 – 127 velmi rozlehlé sítě
- Kategorie B:128 – 191
- Kategorie C:192 – 255 lokální sítě
- existují také kagegorie D a E



Obrázek 10: Hierarchie přenosu zprávy přes TCP/IP



Obrázek 11: Hierarchie přenosu zprávy přes TCP/IP mezi sítěmi



Obrázek 12: Detailnější pohled na TCP/IP

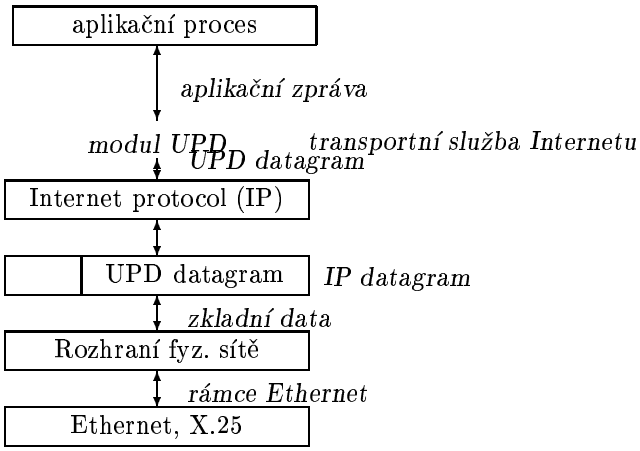
13.9 Hierarchie protokolů TCP/IP

Pohled na hierarchie TCP/IP poskytují obrázky 10, 11 a 12.

Základní data (obr. 12) se skládají ze záhlaví a dat (data jsou zde TCP segment), kde záhlaví obsahuje:

- odesílací port
- cílový port
- pořadová čísla segmentu
- pořadová čísla potvrzení
- typ segmentu
  - ⋆ obsahuje potvrzení
  - ⋆ reset spojení
  - ⋆ synchronizaci spojení

13.9.1 Nespojovaná služba transportu dat (UPD)

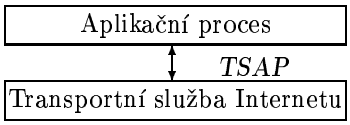


Obrázek 13: Transport dat nespojovanou (nespolehlivou) službou

V záhlaví je uložen (data jsou UPD datagram):

- odesílací port
- cílový port
- délka
- kontrolní součet

Po přijetí zprávy se provádí tzv. *demultiplicing*.



Obrázek 14: Transport dat spojovanou (spolehlivou) službou

13.9.2 Spojovaná služba transportu dat (TCP)

1. proud bitů dělený do slabik
2. spojování
3. zrychlování vysílání
4. řádková strukturalizace proudu slabik
5. plný duplex

13.10 Potvrzování

Vysílač	Přijímač	Poznámka
vyslání paketu1	příjem paketu1 a vyslání ACK1	Mezi těmito událostmi je časová prodleva
příjem ACK1 a vyslání paketu2	příjem paketu2	atd.
vyslání paketu X	příjem paketuX a odeslání ACKX	Nyní dojde ke ztrátě ACKX
čekání na ACK		
retransmise paketu X	vznik duplikace paketu X	

Průběžné okno Abychom nemuseli po odeslání každého paketu čekat na potvrzení, nepotvrzuji samostatné paketu, ale nechám odvysílat množství paketů podle šířky okna (např. 4):

Např. $\boxed{1 \mid 2 \mid 3 \mid 4}\ 5 \mid 6 \mid 7 \mid 8 \mid \ldots$ je posloupnost paketů k odeslání, rámeček tvoří okno. Tedy vyšlu 1 – 4 a čekám. Číslo příchozího ACK mi sděluje, který poslední paket z okna je potvrzován. Přijde-li ACK4, potvrzují se všechny vyslané. Podle obdrženého ACK posunu okno a vyšlu další čtveřici paketů.

Klady:

- vyšší propustnost sítě
- nižší komunikační složitost
- velikost okna podle spolehlivosti sítě (zpravidla 8)

13.11 TCP protokol

Uzel je adresován 4-místnou síťovou adresou, port je adresován 2-místnou adresou v uzlu.

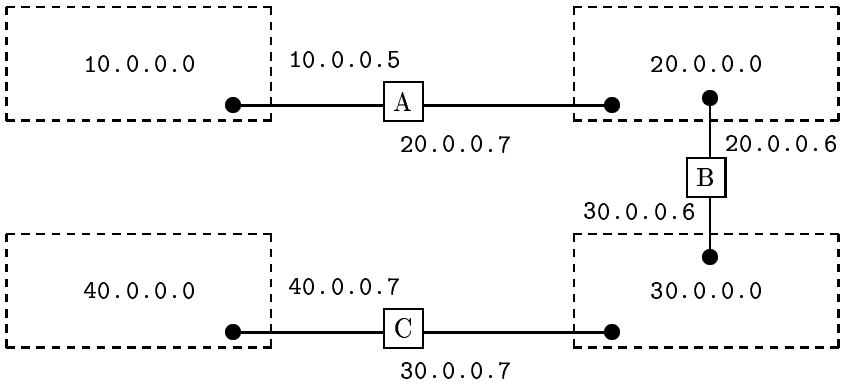
Protokol TCP definuje:

- formáty dat a potvrzování
- postupy (procedury) pro získání dat
- rozlišování adresovatelných míst v cílovém uzlu
- reakce na chyby (ztráta, duplikát)
- jak zahájit TCP proud dat mezi počítači

- jak uzavřít TCP proud dat
- není předepsáno rozhraní na aplikační proces (BSD sockets, TLID (ISO 8072))
- není předepsáno, na kterém komunikačním podsystému má běžet (služba přenosu paketů IP, na Ethernetu,...)

13.12 Směrování IP datagramů (routing)

A) mezi sítěmi Nechť máme síť na obrázku 15.



Obrázek 15: Příklad sítě

Pak směrovací tabulka v routeru B může vypadat takto:

Cílová síť	Pošli na síť	Poznámka
20.0.x.x	20.0.x.x	Přímá viditelnost
30.0.x.x	30.0.x.x	Přímá viditelnost
40.0.x.x	20.0.0.5	Nepřímá viditelnost
40.0.x.x	30.0.0.7	Nepřímá viditelnost

V IP datagramu je uložena informace, odkud a kam je uskutečňován přenos, např. 20.0.0.0 do 40.0.0.0 (obr. 15), ale také Ethernetové adresy odesílatele a adresáta, které se v routerech na pomezí přepisují:

$$O_{ET} = \texttt{arp(30.0.0.6)}$$
$$A_{ET} = \texttt{arp(30.0.0.7)}$$

Implicitní cesty: Když směrovač X nezná cestu k požadovanému cíli, pak automaticky směruje IP datagram na směrovač Y.

B) v rámci sítě Udržují se směrovací tabulky:

- v rámci Ethernet (prázdná směrovací tabulka)
- mezi směrovači
  - ⋆ periodická výměna směrovacích tabulek mezi sousedy
  - ⋆ počáteční nastavení (≡ sousední sítě)
  - ⋆ periodická korekce (vypočítávání nejkratších vzdáleností mezi směrovači)

Použitý protokol pro komunikaci mezi směrovači: ICPM , GGP (gateway to gateway protocol).
Vzdálené směrovače: remote router, half router, exterior gateway spojují dvě fyzicky nesouvislé sítě do jedné (EGP exterior gateway protocol).

14 Transport dat

- transportní služby
- ustanovení / ukončení transportního spojení
- opravné postupy

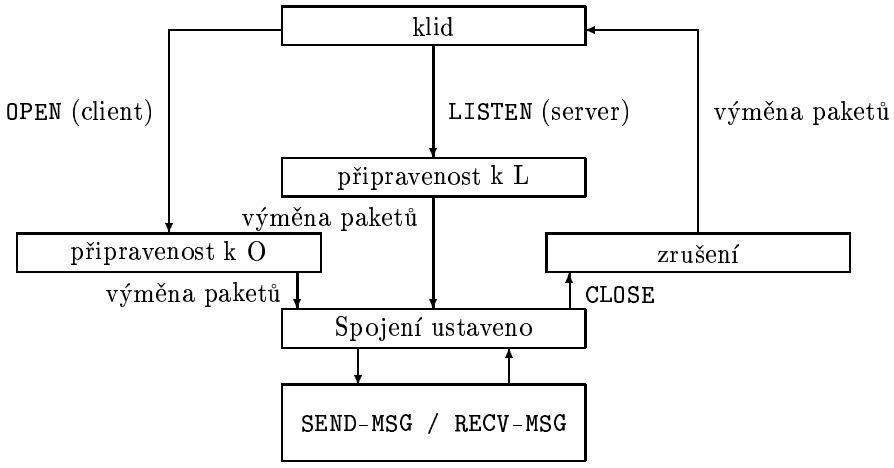
U sítě s přepojováním paketů nebo u sítě dvoubodových paketů může dojít k:

- ztrátě paketu
- zduplikování paketu
- zpoždění paketu
- změna pořadí paketů
- výpadek uzlu, sítě

Z tohoto důvodu požadujeme spolehlivý transportní protokol z hlediska dopravení paketů sítí.

14.1 Protokolový stroj

Protokolový stroj je stavový diagram na obr. 16



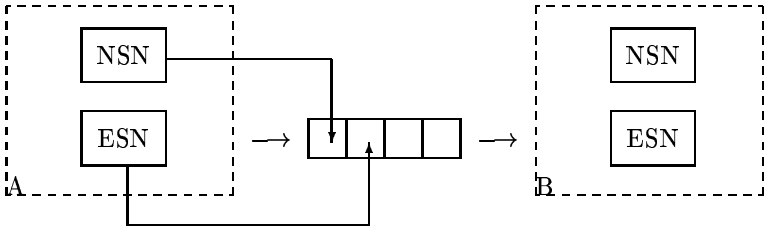
Obrázek 16: Protokolový stroj

14.2 Zabezpečení spolehlivosti

Je nutné uspořádat pakety při příjmu. Packet obsahuje:

- `SN#` – pořadové číslo paketu (zde pro jednoduchost číslo prvního přenášeného byte)
- `C` – řídící informace pro partnera
- `ACK#` – očekávané pořadové číslo příště přijatého paketu
- `D` – data
- plus ještě adresy od koho, komu, ...

next sequence number (NSN) je číslo vysílaného paketu.
expected sequence number (ESN) je číslo očekávaného paketu.
`C` a `D` jsou posílány v datovém poli paketu, je proto nutné je od sebe rozlišit. Schéma komunikace uzlů A a B je na obrázku 17.



Obrázek 17: Schéma komunikace pomocí `SN#` a `ACK#`

Jsou tu však 2 problémy:

1. problém: Inicializace čítačů NSN a ESN.
Triviální řešení je nulovat čítače při ustanovení spojení. Jenže díky tomu, že:

- síť spožďuje pakety
- transportní vrstva neumí rozlišit pakety stávajícího a starých spojení

může nastat situace popsaná tabulkou 3.

Fáze	Uzel A	ISN	ESN	C/D	Data	Uzel B
	open					listen
1.	⟶	0	0	D	ABC	⟶
2.	⟵	0	3	C	'ack'	⟵
3.	⟶	3	1	D	DE	⟶*zpoždění*
	timeout					
3.1.	⟶	3	1	D	DE	⟶
4.	⟵	1	5	C	'ack'	⟵
	close					close
			...			
	open					listen
5.	⟶	0	0	D	GHI	⟶
6.	⟵	0	3	C	'ack'	⟵
7.	⟶	3	1	D	JK	⟶*zpoždění*
3.1.	⟶	3	1	D	DE	⟶
8.	⟵	1	5	C	'ack'	⟵

Tabulka 3: Spoždění a následné míchání paketů

Možné řešení 1. problému:

1. pakety starých spojení nesmí přežít do doby nového spojení. Tedy lze zavést ještě konstantu L, která bude obsahovat maximální dobu platnosti paketu. Pak ji lze využít takto:
   - k sebedestrukci packetu (přijde-li paket v čase time > časvzniku + L, pak jej uzel zapomene).
   - (a) čekat s connectem až do uplynutí času L
     (b) totéž pro každou dvojici procesů

2. pakety starých spojení lze eliminovat těmito způsoby:
   - rodné číslo paketu (globální čítač, spěje však k prodloužení paketu)
   - pro každé spojení nová adresa portu (nelze použít pro standardní služby)
   - rozlišení pomocí ISN (initial sequence number):
     (a) pamatovat si poslední NSN a ISN založit s hodnotou NSN+1. NSN je nutné pamatovat si po dobu delší než L. Pozor na výpadek proudu!
     (b) odvození ISN z hodin, např. ISN := číslo uzlu + okamžitý čas, NSN = ISN + pořadí paketu ve spojení. To však skrývá problém s resynchronizací hodin (v každém uzlu jiné) i čísel ISN (kdy odečíst čas z hodin?).

2. problém po ustavení spojení správně nastavit ESN

Každý uzel volí ISN a paketem `sync` jej oznamuje partnerovi. Ten nastaví svoje ESN a potvrdí `ack`, jak ukazuje tabulka 4

Fáze	Uzel A	ISN	ESN	C/D	Data	Uzel B
	`open`					`listen`
	`volí ISN = x`					
1.	`⟶`	`x`	`?`	`C`	`'sync'`	`⟶`
						`nastavuje ESN = x+1`
2.	`⟵`	`?`	`x+1`	`C`	`'ack'`	`⟵`
						`volí ISN = y`
3.	`⟵`	`y`	`?`	`C`	`'sync'`	`⟵`
	`nastaví ESN = y+1`					
4.	`⟶`	`x+1`	`y+1`	`D`	`ABC`	`⟶`
Spojení je ustaveno						

Tabulka 4: Nastavení ESN

Uváznutí při ustavování spojení Oba uzly jsou `listen`. Do B dojde zpožděný `sync` packet – žádost o zřízení spojení z předchozího pokusu o spojení. B na ni odpoví `ack` paketem a `sync` paketem. Do A dojde `ack` packet (ten A zapomene) a `sync` packet, který pochopí jako žádost o navázání spojení a do B pošle `ack` packet. B jej pochopí jako potvrzení navázání spojení a začne posílat data. A však čeká na potvrzení svého `sync` paketu – jeho žádosti o navázání spojení. Protože spojení nebylo korektně navázáno (A a B jsou o fázi posunuty), nesedí NSN a ESN a tedy A odmítá / zapomíná pakety z B a B pakety z A $\Longrightarrow$ deadlock.

14.3 Průběh ustavení TCP spojení

Užívá se metoda 3-way-handshaking, tj. zhruba tento postup:

chci spojení $\longrightarrow$ rozumím,
myslím, $\longleftarrow$ myslíš to vážně?
pokračuj $\longrightarrow$

Ustanovení spojení

Three-way-handshake ukazuje tabulka 5.

Odmítnutí `sync` pomocí paketu `rst` Princip odmítnutí ukazuje tabulka 6.

Fáze	Uzel A	ISN	ESN	C/D	Data	Uzel B
	open volí ISN = x					listen
1.	—→	x	?	C	'sync'	—→ zapamatuje si x volí ISN = y
2.	←—	y	x+1	C	'sync_ack'	←—
3.	nastaví ESN = y+1 —→	x+1	y+1	C	'ack'	—→ nastaví ESN = x+2
Spojení je ustaveno						

Tabulka 5: Three way handshake

Fáze	Uzel A	ISN	ESN	C/D	Data	Uzel B
Starý packet						
1.	—→	z	?	C	'sync'	—→ zapamatuje si z volí ISN = y
2.	←—	y	z+1	C	'sync_ack'	←—
3.	zjistí nesmysl —→	z+1	y+1	C	'rst'	—→ dále jen naslouchá

Tabulka 6: Odmíntutí starého požadavku `sync`

15 Směrování toku dat

Tato vrstva OSI poskytuje tyto služby:

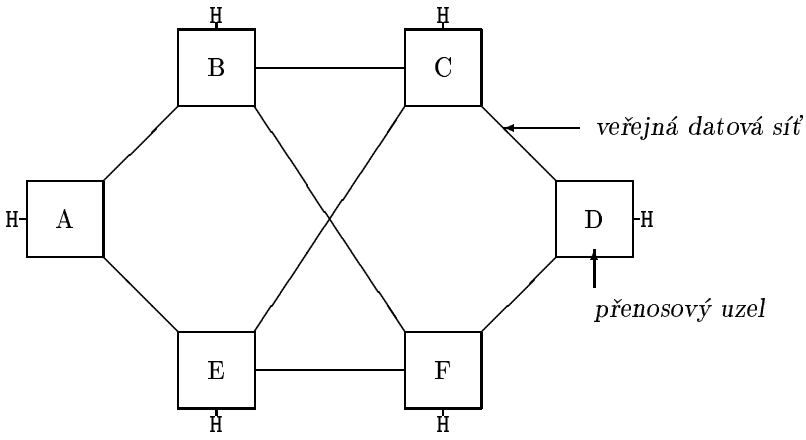
- síťové služby
- statické směrování
- dynamické směrování
- broadcasting

2. vrstva (linková) má za cíl detekci a opravu chyb fyzické vrstvy, koordinaci provozu na médiu, řízení rychlosti toku dat, nás však bude zajímat 3. vrstva (síťová), která obhospodařuje směrování toku dat.

Packet v této vrstvě je elementární informací.

Služby:

1. *spojované* – po celou dobu přenosu je zaručena spolehlivost (proti duplikacím, ztrátám, změnám pořadí) – virtuální kanály
2. *nespojované* – dnes používanější, protože je vyšší fyzická spolehlivost – datagramy.



Obrázek 18: Příklad sítě

Mějme síť na obrázku 18 (H je tzv. *host*, tj.počítač s vlastní aplikací) a na této síti virtuální okruhy:

1. ABCD
2. BCD
3. AEFD
4. BAE
5. ABFD

Směrovací tabulky, které jsou tvořeny uspořádanými dvojicemi `(in,out)`, kde každá s položek je uspořádanou dvojicí `(jméno linky,číslo virt. kruhu)`.

Uzel A	
in	out
H,0	B,0
H,1	E,0
B,0	E,1

Uzel B	
in	out
A,0	C,0
H,0	C,1
H,0	A,0

Uzel C	
in	out
B,0	D,0
B,1	D,1

Uzel D	
in	out
C,0	H,0
C,1	H,1
F,0	H,2

Uzel E	
in	out
A,0	F,0
A,1	H,0

Uzel F	
in	out
E,0	D,0

Virtuální kanál nemusí mít stejnou identifikaci na obou koncích. Nové cesty se vyhledávají podle zatíženosti hran sítě. Tabulky musí být v konzistentním. Výpadek uzlu způsobuje výpadek spoje (u datagramů to neplatí).

Datagramy × Virtuální okruhy			
-	musí být úplná síťová adresa (delší)	+	do packetu stačí dodat číslo virtuálního okruhu (menší bitová komunikační složitost)
+		-	musím udržovat v mezilehlých uzlech směrovací tabulky
+		-	časová prodleva při zřizování okruhu
+	výpadek uzlu $\Longrightarrow$ jde jinou cestou	-	výpadek uzlu znamená výpadek celého okruhu
-	obtížně regulovatelné zahlcení sítě	+	snadno regulovtelné zahlcení sítě
-	směrování se řeší pro každý datakram v každém uzlu znovu	+	směrování se řeší jen při ustanovování okruhu

15.1 Směrovací algoritmy

- na základě čeho rozhodovat o cestě
- algoritmy:
  - ⋆ korektnost (musí se dostat do cíle),
  - ⋆ jednoduchost,
  - ⋆ robustnost (odolnost proti poruchám),
  - ⋆ stabilnost
  - ⋆ spravedlivost
  - ⋆ optimálnost

Algoritmy rozeznáváme statické (neadoptivní) a dynamické (adoptivní). Trend je provádět směrování dynamicky s přístupem:

- centralizovaným
- izolovaným
- distribuovaným

Zahlcení Ideální by bylo, kdyby se vzrůstajícím počtem vyslaných zpráv rost počet přenesených zpráv. Ve skutečnosti však od jisté meze se vzrůstajícím počtem vyslaných zpráv, počet přenesených klesá. Tomuto jevu se říká *zahlcení*, jehož příčiny jsou:

- vysoká režie v uzlu
- rychlost přicházejících paketů >> kapacita výstupního spoje

15.2 Statické směrovací algoritmy

Provádí se směrování nejkratšími cestami, přičemž míra pro to může být:

- počet uzlů na cestě
- geografická vzdálenost
- rychlost / propustnost linek

Změní-li se topologie, musí se znovu nastavit všechny potřebné směrovací tabulky.

15.3 Dynamické směrovací algoritmy

15.3.1 Centralizované směrování

Existuje centrální uzel, který v sobě shromažďuje všechny informace o topologii. Stavová informace mapuje:

- nové sousedy
- délky front na přenos

Nevýhody:

- nestabilní stav $\Longrightarrow$ vysoká režie
- možnost výpadku centra
- různá vzdálenost uzlu od centra $\Longrightarrow$ časová nekonzistence směrování

Tento způsob směrování se už nepoužívá.

15.3.2 Izolované směrování

Každý uzel se rozhoduje sám, bez ohledu na topologii na základě nějaké heuristiky:

Horký brambor: (Hot Potato) – datagram, který mi nepatří, šoupnu co nejrychleji na nejméně zatíženou linku. Dává se do kombinace se statickým směrováním, bere se do úvahy jak váha tak i fronty na přenos.

Excerpce zpráv cestovatelů: Každý packet uvádí, kde vznikl a v každém paketu se počítá počet skoků přes uzly. Mohu tak identifikovat nové, kratší cesty.

Záplava: každý příchozí packet pošli na všechny linky vedoucí z uzlu, kromě příchozí. Přitom musím ještě restringovat počet paketů takto:

- vymazáním packetů po `x` skocích, kde `x` je průměr sítě.
- hubením duplikátů (tím dojde první packet po nejkratší cestě do cíle.

Selektivní záplava: nevysílá se na linky, o kterých „se ví", že k cíli nevedou.

Náhodná procházka: tento způsob má vynikající robustnost. Packet poskakuje, dokud nedorazí na cílový uzel.

15.3.3 Distribuované směrování

V každém uzlu je směrovací tabulka, která říká kudy kam vedou nejkratší cesty, informace je nutné udržovat distribuovaným algoritmem.

Počáteční stav: znám sebe a své komunikační porty. Po časovém intervalu rozešlu tuto informaci svému okolí, v dalším intervalu všechny své získané znalosti všem sousedům. Získané údaje si zaznamenávám pouze v případě nových uzlů nebo kratší cesty (v dalším kroku se posílá už novelizovaná tabulka).

16 Lokální sítě

LAN (Local Area Networks) řízení přístupu k přenosovému médiu.

WAN (Worldwide Area Networks) rozloha 10^2 – 10^3 km, čas t = 10 μs (řádově).

Připojení v/v zařízení na vzdálenosti jednotek metru, čas t = 10 ns (řádově).

Rychlost dodávání bitů do přenosového média je 1 – 100 Mb.s^{-1}.

Doba generování zprávy o n bitech: Je-li rychlost přenosu 10 Mb.s^{-1} a délka zprávy 1000 b, pak tato doba T je 100 μs.

LAN dělíme:

- je-li $\frac{t}{T} < 1$, pak nazýváme soustředěnou LAN. Tedy v době, kdy ještě vysílám zprávu, je již přijímána.
- je-li $\frac{t}{T} > 1$, pak hovoříme o rozsáhlé LAN $\sim$ WAN. Tady zprávu odvysílám, mohu začít další vysílání a pak teprve dorazí do dalšího uzlu. Tato věc se však může stát i u rychlých lokálních sítí.

LAN má neomezenou topologii:

- *dvoubodové spoje* – soupeření o spoj se odehrává pouze mezi těmito dvěma „konci" spoje.
- *princip sběrnice* – informace vysílaná na síť se rovnoměrně šíří po celé síti (všichni dostanou informaci), jak ovšem řídit přístup?
- *kruh* – uzavřené médium, host počítače připojeny jednotkami, kdo získá právo vysílat, kruh rozpojí a vysílá zprávu, kterou se šíří po kruhu jedním směrem, pouze připojené místo, kterému patří, ji pustí do host počítače.
- *hvězdicové sítě* – budované na principu dvojpodového spoje.
  - ⋆ HUBNET = LAN s kombinovanou metodou přístupu ke hvězdicové topologii
  - ⋆ přenosový podsystém generován na jediný uzel, klasická metoda přístupu ke hvězdicové topologii.
  - ⋆ polling – centrální uzel se pořád ptá, jestli někdo nechce vysílat.

Sdílení přenosového média

- v prostoru (křížové přepínače)
- v čase
  - ⋆ synchronizované přístupy
    - ∘ neřízené metody přístupu
    - ∘ řízené metody přístupu (TDM, STDM)
  - ⋆ nesynchronizované přístupy
    - ∘ řízené metody přístupu: centrální arbitr (vícebodová spojení) a decentralizovaný arbitr (token)
    - ∘ neřízené metody přístupu (ALOMA, CSMA)
- ve frekvenčních pásmech (nepoužívá se)

16.1 Časové sdílení přenosového média

A) synchronizované, řízené Čas musí běžet ve všech uzlech stejně rychle, což se děje vysíláním puslů, například po zvláštním vodiči. Uzel počítá tiky, pozná svůj puls.

- je zaručena jedinečnost vysílání
- problémem je zpožďování signálu
- tiky může šířit centrální stanice, ale její výpadek znamená zhroucení sítě
- znám počet a rychlosti uzlů $\Longrightarrow$ znám časové intervaly vysílání jednotlivých stanic (vhodné pro real time systémy).

- prodlevy (prázdná místa ve vysílání)
- nemusím zprávy opatřovat adresou odesílatele
- vzniknou-li dva požadavky v jednom okamžiku:
  - ⋆ arbitr
  - ⋆ na základě heuristik se spolu musí domluvit

Kruh Piercova typu Časové zdrže v bodech připojení uzlů. Po kruhu se šíří po taktech „vagónky", které buď mají zprávu nebo jsou prázdné. Chce-li uzel vysílat, detekuje první prázdný vagónek a naplní ho.

- je-li vagónek se zprávou, všichni do něj koukají, jestli to není pro ně.
- prázdný / plný – jednobitový semafor
- je-li vagónek pro mě (adr. info.) vezmu si obsah a nastavím na prázdný.
- uzly musí být uspořádány do kruhů.

B) synchronizované, neřízené Neurčuje, který časový interval je komu přidělen (je to řešení v sítích typu taktovaná ALOHA).

- více uzlů v jednom okamžiku $\Longrightarrow$ zbytečné vysílání
- začne-li uzel vysílat, musí poslouchat, co se na síti děje
- po odvysílání musím zachytit to, co jsem vyslal
- dojde-li ke kolizi, nemá smysl vysílat znova v dalším taktu, protože by určitě došlo k nové kolizi
- vysílám tedy po uplynutí náhodně dlouhé doby, čímž se sníží pravděpodobnost nové kolize
- prodlužuje se doba úspěšného vysílání
- nevhodné pro real time

Také se u tohoto typu používá **prioritní metoda**:

- stanice, která skončila vysílání, začne vysílat časové pulsy
- syncrhnní vysílání priority před vysláním paketu (např. podle identifikace uzlů, které chtějí vysílat – postupným srovnáváním bitů)
- tato metoda je velmi náročná na interface (řadiče)

C) nesynchronizované, neřízené

- kdokoli může kdykoli vysílat, po libovolnou dobu, musí být určeno kdo a komu, musí se potvrzovat
- vysoká pravděpodobnost kolizí
- u alohy při více zprávách velké zhoršení situace.
- zabránění kolizím: tím, že není centrální arbitr, může každá stanice, než začne vysílat, zkontrolovat médium, je-li volné: metoda CSMA/CD (Carrier Sense Multiple Access / Collision Detect). Zjistí-li, že není volné, mlčí, zjistí-li opak, vysílá. To sice nazabrání kolizím, ale hodně sníží jejich pravděpodobnost.
- uzel poslouchá, co vysílá a je-li to špatně, přestane vysílat a za náhodnou dobu začne znovu.

1-persistentní CSMA

- obsazený kanál, za náhodnou dobu znovu

- volný kanál, ihned vysílá, kolizí nedopravený rámec, znovu zahájí vysílání.

V tomto případě mohou uzlu strádat; je vhodné pro kancelářské sítě, nevhodné pro real time
p-persistentní CSMA

- obsazený kanál, za náhodnou dobu znovu
- volný kanál:
  - ⋆ s pravděpodobností p vysílá
  - ⋆ s pravděpodobností $1 - p$ musí zkusit znovu

Tento způsob má velmi vysokou propustnost.
Ethernet používá 1-persistentní CSMA (technicky realizovatelné), rozbitý rámec už nevysílá.
Vkládání registru

- metoda zabránění kolizí
- uzly v kruhu
- zpráva v cache
- je-li volno rozpojení kruhu, zapojení cache do kruhu
- cizí zprávu, která se takto do uzlu dostane (může se dostat), chovám se k ní jako k vlastní, nejde-li už další, spojení kruhu, host může znovu případně plnit cache.

16.2 Ethernet

Jeden přenosový kabel (koaxiál) a sběrnice. Nelze posílat libovolně krátké zprávy. Kvůli CD (detekci kolizí) musí mít nějakou minimální délku.

16.2.1 Dosah

Uzly jsou napojeny na Ethernetovou sběrnici maximálně po 50 metrech kabelu. Sběrnice může být dlouhá do 500 metrů, pak musí být zařazen repeater. Tedy vzdálenost mezi stanicemi je maximálně 1500 metrů (přes 2 repeatery a 3 sběrnice). 1 repeater se však dá nahradit 2 repeatery spojenými maximálně 500 metrů dlouhým kabelem. Tedy celková největší vzdálenost mezi dvěma uzly na Ethernetu je

$$3 \cdot 500 + 2 \cdot 500 + 6 \cdot 50 = 2800$$

(repeater je v podstatě stanice a na nejdelší cestě jsou 4 ve vzdálenosti maximálně 50 m plus naše dva uzly, také maximálně 50 m vzdálené od sběrnice).

Šíření signálu v ideálních podmínkách je na Ethernetu 300 000 $\mathrm{km.s^{-1}}$ (rychlost světla), tedy zhruba 4.5 $\mathrm{s.km^{-1}}$. Mezi nejvzdálenějšími stanicemi dráhu 2800 m urazí signál za 11 – 13 μs, tam a zpět za asi 26 μs. 20 μs činí ztráta na elektrických zařízeních, tedy celkem cca 46 μs. Za tuto dobu se přenese při rychlosti 10 $\mathrm{Mb.s^{-1}}$ asi 464 bitů, se 48 bity na kolize takže minimální délka zprávy činí 512 bitů.

16.2.2 Rámce

Rámec:

- 8B preambule
- 6B adresát
- 6B odesílatel
- 2B typ pole (TCP/IP – hodnota je 800, Novell, ..., řádově stovky typů)
- 46B datová část

- 4B FCS (Frame Control Sequence)

Horní hranice pro velikost rámce je 1500 slabik.

Preambule slouží k tomu, že stanice, které nepracují, poslouchají a ladí rytmus podle této preambule (ta má tvar `1010101010...101011`).

Je definovaná prodleva před vysíláním další preambule (aby byl rozpoznatelný konec). Eternetovská deska má 6-ti slabičnou adresu, horní 3 slabiky přiděluje IEEE. Je-li adresa ze samých jedniček, je určena pro libovolnou desku, je-li 1. bit 1, 2. bit 0 a ostatní libovolné, jde o skupinovou adresu, je-li 1. bin nulový, jde o individuální adresu.

16.2.3 Tlustá Ethernet

Tlustý koaxiál, po minimálně 2,5 m MAU (odbočka), sběrnice do 500 m, vzdálenost AUI (u stanice) od své MAU do 50 m. Původní řešení, je však drahé (kabel i MAU), proto se používá pro těžké průmyslové provozy. Maximálně 99 kabelů na MAU.

16.2.4 Tenká Ethernet

Logicky a technicky se chová stejně jako tlustá, odbočky přímo do počítače, vzdálenost odboček aspoň 0,5 m, celková vzdálenost pouze 900 m, sběrnice jen 185 m. Užívá se proto jen v rámci budovy. Maximálně 30 kabelů na MAU.

16.2.5 Bridge

Je to inteligentnější opakovač. Ví, jaké uzly má na kterém svém konektoru (udržuje si tabulku). Až má dostatek informací, nepouští zprávy mezi uzly z prvního segmentu do druhého. Stará se o fyzické adresy. Existuje programovatelná bridge, mnohaportová bridge, multiprotokolová bridge (mezi různými prostředími). V bridgi je zdrž. Bridge může být i počítač (znamená to větší zdrž).

D) nesynchronizovaný, řízený přenos

1. Token ring (decentralizovaný arbitr – pešek), kruh Newhallova typu.
   - mohu vysílat pouze s příznakem oprávnění (rozpojím síť)
   - token cirkuluje po kruhu, uzel, který chce vysílat, musí na token počkat
   - problém: ztráta paketu s tokenem. Lze jednu stanici považovat za řídící, která kontroluje, zda token obíhá. Při timeoutu generuje nový token.
2. Token bus (logický kruh, decentralizovaný arbitr)
   - každý uzel si pamatuje logický kruh
   - jedna stanice inicializuje přenos, nemá-li co vysílat, pošle peška sousedovi. Všechny stanice poslouchají, chodí buď balíky se zprávami nebo pešek s adresou komu se má poslat. Stanice, která peška vyšle, kontroluje, jestli další stanice reaguje. Když nereaguje, pošle to další stanici po logickém kruhu.
   - musí být definována monitorovací stanice s výsadním právem generovat po časovém limitu nový token.
3. Centralizovaný arbitr není pro LAN typický. Ale funguje na principu Výzva – Odpověď
   - postupně se ptám všech uzlů, jestli mají něco k vyslání nebo je ochotna přijímat
   - multipant na sběrnici
   - typické pro terminály
   - historicky: 60. léta
   - centralizované arbitrovo řízení

17 Přenos dat (fyzická vrstva)

Přenos dat ve fyzické vrstvě OSI je přenos informace vodičem změnou vhodné fyzikální veličiny.

Signál je fyzikální vyjádření informace ve formě změn fyzikální veličiny v čase. Formální vyjádření signálu je funkce jedné proměnné (času). Je to periodická funkce $g(t)$ s periodou T.

Fourierovy posloupnosti:

$$g(T) = \frac{c}{2} + \sum_{n=1}^{\infty} a_n \cdot \sin(2\pi n f t) + \sum_{n=1}^{\infty} b_n \cdot \cos(2\pi n f t)$$

kde $f = \frac{1}{T}$ je základní kmitočet (frekvence), a_n a b_n jsou amplitudy harmonických složek. Známe-li T, a_n, b_n a c, můžeme zrekonstruovat $g(t)$ následujícím způsobem:

$$\begin{aligned} a_n &= \frac{2}{T} \cdot \int_0^T (g(t) \cdot \sin(2\pi n f t))\, dt \\ b_n &= \frac{2}{T} \cdot \int_0^T (g(t) \cdot \cos(2\pi n f t))\, dt \\ c &= \frac{2}{T} \cdot \int_0^T g(t)\, dt \end{aligned}$$

Každý signál má konečnou délku trvání (T); lze tedy chápat každý datový signál za periodický s periodou T.

Ztráta výkonu při přenosu signálu je frekvenčně závislá. Jak rychle máme tedy vysílat?

Vyměření šířky pásma (vlnové délky) je buď vrozené médiu nebo vymezené. Rychlost přenosu dat se uvádí ve znacích za sekundu:

- znaky jsou reprezentovány bity – rychlost v bitech za sekundu
- znaky jsou změny hodnot signálu – jednotka je baud = počet změn signálu za sekundu

Uvědomme si, že rychost v baudech a v bitech za sekundu není totéž. Rovnost platí pouze pro 2-hodnotový signál. Pro 8-hodnotový signál (8 hladin signálu) se s každou hodnotou signálu přepravují 3 bity.

Hlasový telefonní spoj: šířka pásma $0 \div 3000$ Hz, přenosová rychlost je rychlost světla (horní mez), znak je 8 bitů. Doba přenosu jednoho znaku je tedy $\frac{8}{c}$ s (T), kmitočet jedné harmonické je $\frac{c}{8}$ Hz. Číslo nejvyšší propouštěné harmonické je

$$\frac{3000}{\frac{c}{8}} = \frac{24000}{c}$$

v [b.s−1]	**T** [ms]	**f** [Hz]	**n**
300	26.7	37.5	80
600	13.3	75	40
1200	6.7	150	20
2400	3.3	300	10
4800	1.7	600	5
9600	0.8	1200	2
19200	0.4	2400	1
38400	0.2	4800	?

V předchozí tabulce znamená **n** počet vysokých harmonických. Poslední řádek tabulky ukazuje, že při 3000 Hz nemá binární signál na telefonních spojích šanci.

Jaká je nejvyšší rychlost přenosu dat kanálem, který funguje jako dolní propust kmitočtů a je bezporuchový (bez šumů), aby bylo možné restaurovat původní signál, říká Nygistova věta:

$$v_{max} = 2H \log_2 V$$

kde V je počet úrovní signálu.

Tedy na frekvenci 3 kHz lze posílat binární signál rychlostí maximálně 6000 b.s^{-1}.

Jak se projeví existence šumu v kanále? Na tuto otázku odpověděl Shannon:

$$v_{max} = H \log_2 1 + \frac{S}{N}$$

, kde S je výkon signálu a N je výkon šumu (termální, Gaussův,...). $\frac{S}{N}$ se měří v dB. Je-li typický telefonní signál H rovno 3000 Hz a $\frac{S}{N}$ rovno 30 dB, pak v_{max} je 20 000 b.s^{-1} bez ohledu na V,...

17.1 Analogový × číslicový (digitální) přenos

Přenos se uskutečňuje:

- počítač — modem — analogová telefonní síť — modem — počítač
- počítač — LAN (dvoubodové spoje) — počítač
- počítač — digitální síť — počítač

U digitálního přenosu se vysílá přímo digitální signál do přenosového média, kdežto při analogovém přenosu se digitální signál transformuje v modemu na analogový, ten projde sítí a druhý modem jej transformuje zpět do digitální podoby.

Digitální signál má diskrétní úrovně, analogický signál má spojitý průběh.

Digitální signál:

- základní pásmo (baseband) – přímý přenos – změna ss signálu
- přeložené pásmo (broadband) – širokopásmový přenos – změna vf signálu

17.2 Kódování signálů

Při vysílání v základním pásmu se používá NRZ. Je výborné využití šířky pásma (bit ~ baud) a není nutné zvláštní kódování / dekódování, avšak potřebuje externí synchronizaci, tj. zvláštní drát pro detekci intervalů, protože jednička je horní hladina, nula je dolní hladina a nevíme, kolik jedniček a nul protéká. Ethernet má vnitřní synchronizaci (v každém bitovém intervalu dojde ke změně – buď se ustálí nebo změní, musím tedy zase přenést více harmonických)

Způsoby přenosu digitální informace:

Modulace: konverze digitální informace na analogovou – vložení digitální signálu do kmitání

Demodulace: konverze analogové informace na digitální – sejmutí digitálního signálu z kmitání

Druhy modulací:

Amplitudová: (AM – Amplitude Modulation) digitální signál se nasadí na vlnění tak, že je-li digitální signál na vyšší hladině, vezme se nosné kmitání, je-li na dolní hladině, je ve výsledném signálu klid.

Frekvenční: (FSK – Freqence Shift Koding) spočívá v tom, že kde byla digitální nula, výsledný signál kmitá jinak, než kde byla digitální jednička.

Fázová: posouvá signál v čase. Obvyklý způsob fázové modulace používá posuvy o 45, 135, 225 a 315 stupňů. Každý fázový posun přenáší 2 bity (4-hodnotový signál)

Kvadraturní amplitudová: posun ve fázi a amplitudě (8-hodnotový až 16-hodnotový výsledný signál)

17.3 Datový (2-bodový) spoj

Datový spoj se odehrává na poli:

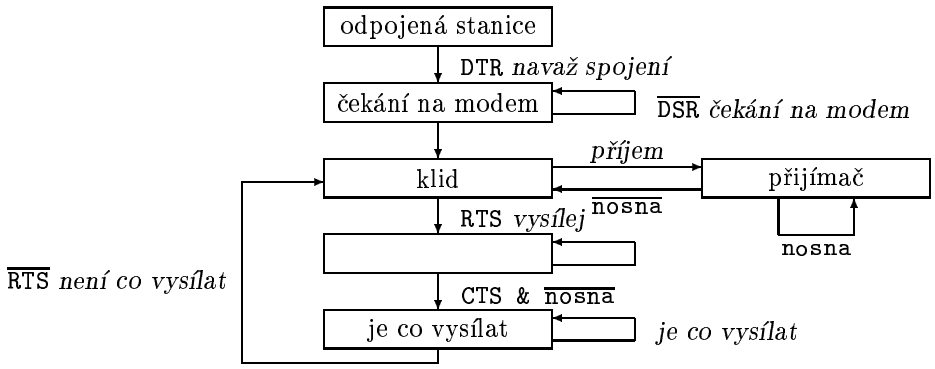
`data terminal equipment (DTE)` ⟶ `data circuit equipment (DCE)` ⟶ `DTE`

bez ohledu na to, zda je DTE účastnická stanice a DCE telekomunikační spoj nebo je-li DTE koncové zařízení přenosu a DCE je spojení modem — telefonní síť — modem.

Mezi DTE a DCE je definovaný interface, např. RS 232 C (V.24).

RS 232 C má 25 pinů, konektor typu CANNON (DB 25), které jsou přesně specifikovány (země, data, řízení, časování, synchronizace). Přenosová rychlost je 0 – 20 000 b.s^{-1}, dovoluje synchronní i asynchronní přenos, DTE má samce DCE samici, délka spojení max. 50 stop (15 m).

Princip modemu Obrázek 19 ukazuje stavový diagram činnosti přenosu dat po modemu.



Obrázek 19: Stavový diagram práce modemu

17.4 Digitální síť

Šířka pásma média používaného pro digitální síť je 4 kHz. (tj. V = 2, H = 4000, rychost přenosu je 8000). Analogový signál téhož pásma je nutné vzorkovat 8000 krát za sekundu, jinak dojde ke ztrátě informace. Médium je možné realizovat: telefon $\longrightarrow$ lokální smyčka $\longrightarrow$ CODEC $\longrightarrow$ digitální signál.

CODEC vzorkuje analogový signál po 125 μs a vysílá digitální informaci vyjadřující okamžitou hodnotu signálu.

17.4.1 Systém T1 (Bell)

- kroucená dvoulinka, opakovače po 1.5 km
- slučování 24 hlasových kanálů
- vzorkování po 125 μs
- vzorek transformuje do osmibitového vzorku (7b dat a 1b řídící), rámec má 192b, poslední bit je kontrolní (střídá se 1 a 0, přijdou-li 2 stejné, indikuje se ztráta rámce
- 7b dat, tj. 128 úrovní analogového signálu, $\frac{193b}{125\mu s}$ je 1.544 Mb.s^{-1}.

17.4.2 CCITT PCM (1.544)

(Pulse Coding Modulation) přenosová rychlost je 1.544 Mb.s^{-1}.

1. řízení ve společném kanálu
   - kvantifikace analogického signálu do 256 úrovní (8 bitové vzorky)
   - řízení pouze na úrovni rámců (193bitových)
2. řízení po kanálech 526 vzorků je 8bitových a 1 je 7bitový + 1 bit jako řídící (alternuje)

17.4.3 CCITT PCM (2.048)

32 kanálů, na každém 8bitové vzorky, 125 μs na rámec, tj. rychlost 2.048 Mb.s^{-1}. 30 informačních kanálů, a 2 řídící kanály, větší šířka pásma

17.4.4 Přenosové rychlosti s větší šířkou pásma

Po 125 μs se přenáší delší pakety a proto je nutné je vysílat rychleji: 8.848, 34.304, 139.264 nebo 565.148 Mb.s^{-1}

17.5 Bezdrátové spoje

Používají se infračervené světlo, laser, μ-vlny, radiové vysílání. Má tu výhodu, že není nutné klást kabel, ale může mít na to vliv atmosferické prostředí. Používají se frekvence 2 – 40 GHz.

17.6 Komunikační družice

Komunikační družice musí být stacionární („viset" nad stejným místem na Zemi). Šířky pásem jsou až 500 MHz.
Pásmo:

- 4 – 6 GHz: již dnes plno
- 12 – 14 GHz: volno, projevuje se počasí
- 20 – 30 GHz: rezerva pro budoucnost

V jednom pásmu se vysílá k družici a ve druhém k Zemi. Časová prodleva činí asi 200 - 300 ms, ale maximální rychlost přenosu na družici je 50 Mb.s^{-1} a po zemi 56 Kb.s^{-1}. Družicový přenos je všesměrový. Pokud nebudou optická vlákna všude (vždy je nějaká překážka), je přenos přes družici rychlejší. V éteru je najednou více rámců, takže okénko pro potvrzování je 8b místo 3b.

17.7 DTE

To jsme pohovořili o možnostech DCE, nyní trošku o DTE. Řízení vstupu a výstupu na DTE (např. terminál) vyžaduje synchronizaci:

- bitová synchronizace:
  - ⋆ hodinami (synchronní přenos, globální taktování)
  - ⋆ informacemi (asynchronní přenos, časovací informace vložená do zprávy)
- znaková synchronizace:
  - ⋆ bitová vždy pro každý znak znovu (asynchronní přenos)
  - ⋆ bitová trvalá (synchronní přenos)

Zde není dán žádný protokol, rámce, pakety, nic!
Synchronní přenos vyžaduje:

- způsob identifikace znaku zprávy v proudu bitů
- pevný časový rastr v každé stanici
- každá stanice má své hodiny
- vysílající stanice udržuje synchronnost (vysílá dohodnutý vzorek, např. `... syn, syn, znak, znak, syn, syn, znak, znak, ...`

Asynchronní přenos: Synchronizace se provádí pomocí dvou signálů: `startbit znak ... znak stopbit` a předpokládá se, že dokud jdou znaky, je synchronnost zaručena. Klesá však efektivní přenosová rychlost.

18 Linková vrstva

- jak organizovat rámce
- funkce: řešení chyb při přenosu dat fyzickou vrstvou a regulace toku dat
- služby:
  - ⋆ nepotvrzované, nespojované (datagram)
  - ⋆ potvrzované, nespojované (data + potvrzení – rámce nebo signály ve zvláštním spoji)
  - ⋆ spojované

Spojovaná služba

- navázání spojení: `request` ⟶ *indication* ⟶ *response* ⟶ `confirm`
- přenos dat (data + potvrzení)
- zrušení spojení (ukončuje ten, kdo začínal)

Problémy při návrhu linkové vrstvy

- vytváření rámců (framing) – znakově / bitově orientované protokoly
- řízení reakcí na chyby přenosu
- řízení toku dat (relativní rychlost vysílače / přijímače)
- zřizování spojení

18.1 Vytváření rámců

Přenos po částech, každá z nich je spravována zvlášť

1. znakové závorky: DLE (data link escape), STX (start of text), ETX (end of text, přenos probíhá například takto:

   ```
   DLE STX A DLE B DLE ETX
   ```

   - vkládání znaků: zadaný znak (`DLE`), vyslaný znak (`DLE DLE`), předáno přijímači (`DLE`).
   - přenášenou informaci nutno pasovat do slabik
2. bitové závorky: křídelní značka (01111110), takže přenos probíhá třeba takto:

   ```
   01111110 10110011010011001010001110101l0 01111110
   ```

   - vkládání bitů: zadaný text `11111 11111 11...`, vyslaný text `111110 111110 11...` přijatý text `11111 11111 11...`
   - po každé páté jedničce za sebou se natvrdo vyslala nula
   - bývá definována určitá struktura rámce (na začátku) pro řídící informaci, pak libovolná data

18.2 Řízení toku dat

Je-li rychlost producenta větší než rychlost konzumenta, musí být nějaká zpětnovazebná brzda.

Stop-and-wait rotocol

- každý rámec je povrzený, teprve pak se vysílá další rámec
- problém ztráty `msg / ack`: alternující potvrzení (`ack0` a `ack1`). Nedojde-li `ack`, vysílač pošle `elq`, přijímač na to, pošle `ack` a podle čísla potvrzení, které došlo a mělo dojít si odvodím, co se ztratilo (tento postup byl navržen pro half-duplex).

XON/XOFF protocol Pokud přijímač nestíhá, pošle signál `počkej`, vysílač se zastaví ve vysílání a až od přijímače přijde signál `můžeš`, začne znovu sypat další data (navrženo pro simplex)

18.3 Obousměrné potvrzování

- zpět pošlu data + potvrzení příjatého nebo jen pouze potvrzení
- vysílací okno – čísla rámců, které můžu vyslat
- přijímací okno – čísla rámců, které můžu přijmout
- Stop-and-wait protocol je vlastně okno s kapacitou 1

19 X.25

X.25 je protokol definující přístup do veřejné datové sítě (PDN).

- protokol – formát, význam, časování
- veřejná datová síť (public data network)
- DTE, DCE
- uzel sítě (vnitřní PDN) – PSE (packet switch exchange)
- konvertor na/z paketový protokol – PAD (packet assembler / disassembler)
  - ⋆ ITI (interactive terminal interface) asynchronní interaktivní terminál
  - ⋆ BMTI (block mode terminal interface) synchronní terminál
- existuje více sítí, propojení pomocí PSE (který funguje jako směrovač) nebo STE (half router)

Pomocná doporučení X.25 Jsou určená tomu, kdo má zařízení, které neumí pracovat s pakety.

- X.3: předpis vzhledu tabulky, která popisuje periferní zařízení (rozměr obrazovky, terminálu,...)
- X.29: popis vzdáleného ovládání zařízení (PADu) – popis služebních paketů
- X.28: popis vzdáleného ovládání terminálu (popis vzhledu signálů a časových posloupností)
- X.75: protokol o směrování mezi sítěmi
- X.121: popis způsobu směrování (3B číslo země, 1B číslo PDN v zemi, 14B číslo terminálu sítě).

Literatura

[1] Jan Staudek, Lenka Motyčková *2x15 kapitol z distribuovaných systémů, Část I, Distribuované algoritmy a počítačové sítě*, skriptum VUT Brno, 1993, ISBN 80-214-0495-7

[2] Martin Kuba *Principy programovacích jazyků, záznamy z přednášky Tomáše Havláta*, příspěvek do projektu ZKUSTO, 1994