Distribuované systémy

Jan Janeček

Listopad 2000

Předmluva

Tento text je učební pomůckou pro předmět *Distribuované systémy* pro studenty oboru Výpočetní technika a informatika. Pokrývá problematiku dvou úzce souvisejících oblastí: komunikačního systému a distribuovaného výpočtu.

Komunikační systém, podporující distribuovaný výpočet, je popisován s ohledem na obecně přijímanou vrstvovou architekturu a tomu odpovídá i členění kapitol prvé části. Pozornost je věnována zvláště problematice sítí s přepojováním, problematika sdíleného média je podstatně podrobněji pokryta učebním textem pro předmět Lokální sítě [1]. Vhodným doplňkem tohoto textu je skriptum Distribuované systémy – cvičení [2] zaměřené na technologie Internetu a programování v tomto prostředí (BSD sockety, RPC kommunikace, Java RMI).

Druhá část textu je věnována prostředkům zajišťujícím synchronizaci a komunikaci procesů v distribuovaném prostředí a základním vlastnostem distribuovaného výpočtu. Zahrnuje přehled vybraných algoritmů zajišťujících výlučný přístup, detekujících zablokování a chránících před ním, detekujících ukončení výpočtu, a zajišťujících funkčnost a korektnost při výpadcích.

Myslím si, že na toto místo patří i jazyková poznámka. Text se zabývá oblastí, ve které se objevuje a běžně používá řada termínů jazyka současné techniky - angličtiny. Při psaní tohoto textu jsem se snažil respektovat pravidla a duch češtiny. Tam, kde existuje zavedený, nebo dokonce standardizovaný český odborný termín, užívám ten a vyhýbám se oborovému slangu (např. standardizovaný termín slabika nebo oktet je použit tam, kde dnes v řadě publikací najdeme, v českém textu nehezký, termín *"bajt"*). Tam, kde alespoň částečně akceptovaný český termín neexistuje, a kde doslovný překlad anglického termínu není dostatečně výstižný a/nebo přetížení českého termínu odlišným významem není vhodné nebo by vedlo ke dvojznačnosti, jsem raději zůstal u původních termínů anglických (pochopitelně bez pokusů o problematický fonetický přepis, české skloňování nebo dokonce časování) a u zkratek. Z čistě praktických důvodů (využitelnost pro výuku v angličtině) jsou anglické termíny použity v obrázcích.

Text vychází v této formě poprvé a autor uvítá poznámky pečlivého čtenáře k jeho formě a obsahu.

Praha, listopad 2000 autor

Obsah

1. Distribuované systémy a počítačové sítě

Distribuovaný výpočet, u kterého se na cílovém řešení zadané úlohy podílí více procesorů schopných provádět výpočet na rozdělených datech souběžně, se stává velice častým řešením pro řadu aplikací. K rozložení výpočtu na více procesorů může vést řada důvodů:

Historicky nejstarším důvodem pro distribuci výpočtu je přirozená *geografická distribuce* dat a jejich zpracování. Je běžná v bankovních systémech, výpočetních systémech velkých firem a státní správy, v rezervačních systémech dopravních a hotelových společností.

V řadě systémů, které jsou navrhovány pro reálné aplikace (technologické řídící systémy, bankovní systémy), se setkáváme se situací, kdy je nutné jednotlivé procesory distribuovaného systému speciálně vybavit, aby byly schopné interakce s reálným prostředím. Procesory neplní jednotnou výpočetní funkci. Mluvíme o *distribuci funkcí*, specifické funkci vyhrazujeme specifický výpočet realizovaný na specifickém procesoru.

Dalším důvodem pro rozložení výpočtu na více procesorů může být jeho výpočetní složitost a potřeba sdílet výpočetní kapacitu procesorů. Mluvíme o *distribuci výkonu* a opíráme se buď o systém navržený speciálně pro daný výpočet, nebo o univerzální systém s *dynamickým přidělováním* procesorů.

Konečně, jako poslední si uvedeme požadavky na spolehlivost, které lze často splnit pouze zálohováním, schopností systému *rekonfigurovat* svou strukturu a přidělit výpočtu, kterému selhal procesor, procesor jiný.

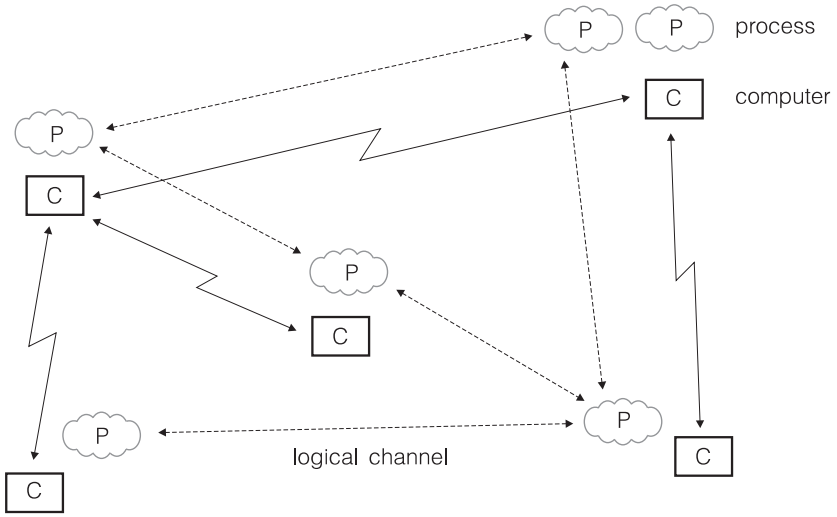
1.1 Základní pojmy

Jako *distribuovaný systém* budeme označovat takový výpočetní systém (obr. 1.1) , který zahrnuje

-více než jeden procesor/počítač,

-má svůj program rozdělený na části, které si vzájemně předávají data, a

-které jsou počítány na různých spolupracujících procesorech/počítačích systému.



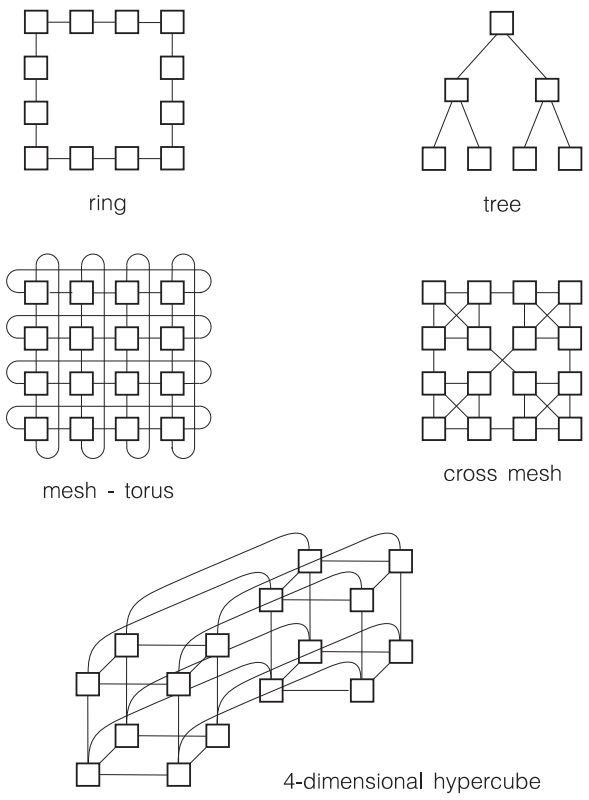
Obrázek 1.1: Architektura distribuovaného systému

Obrázek 1.1 uvádí příklad rozložení programu na jednotlivé počítače vzájemně propojené dvoubodovými spoji. Komunikační spoje mezi procesy zde nekopírují propojení počítačů, některé z počítačů zde kromě vlastního výpočtu musí i zprostředkovat komunikaci.

1.1.1 Distribuovaná architektura

Pod pojmem *distribuovaná architektura* budeme rozumět skupinu procesorů, jejichž technické vybavení jim dovolí vzájemnou spolupráci při výpočtu distribuovaného programu. Klíčovým prvkem distribuované architektury je způsob, jakým si procesory distribuovaného systému vyměňují data.

Na jednom konci spektra stojí architektury opírající se o sdílení paměti. Takové systémy označujeme také jako *systémy s velmi těsnou vazbou* nebo multipočítače (případně multiprocesory). Jsou výhodné při potřebě soustředění vysokého výkonu na zpracování velice úzce provázaných dat, příklady aplikací se najdou zvláště v oblasti výpočetně náročných problémů (řešení diferenciálních rovnic při modelování nebo analýze reálného světa). Systémy s tímto typem distribuce výkonu obvykle označujeme jako paralelní počítače.



Obrázek 1.2: Statické propojovací struktury

Odlišný přístup ke spolupráci procesorů nalezneme u architektur, které se opírají o *výměnu zpráv*. Jednotlivé procesory systému jsou vybaveny perifériemi — komunikačními řadiči, a jejich prostřednictvím jsou připojeny na *dvoubodové* nebo *vícebodové komunikační kanály*.

Dvoubodové kanály spojují dvojice procesorů a jsou většinou základem polygonálních propojení procesorů. Systémy opírající se o propojení tohoto typu označujeme jako systémy s *volnou vazbou* (loosely coupled).

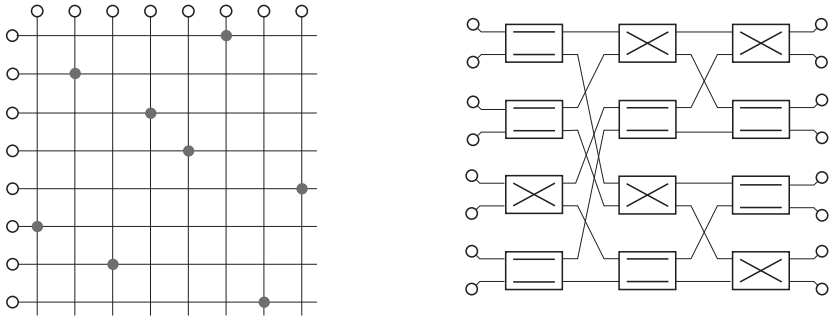
U polygonálně propojených systémů se setkáváme s problémem zprostředkování komunikace procesorů, které nejsou přímými sousedy, důvodem jsou převážně praktická omezení (cena propojení, velký počet rozhraní). Výsledkem může být buď vyčlenění určitých procesů na všech procesorech pouze pro komunikační funkce, mluvíme o vytvoření komunikační — *transportní služby*, nebo vyčlenění určité skupiny procesorů, která se na předávání zpráv specializuje, mluvíme o *komunikačním podsystému*.

U geograficky malých systémů, které obvykle označujeme jako *paralelní systémy* a u kterých nehraje podstatnou roli celkový počet spojů mezi procesory, často saháme po *regulárních strukturách* propojení. Propojení procesorů jsou navrhována tak, abychom s daným počtem rozhraní jednoho procesoru dosáhli co nejvýhodnějších parametrů sítě - co nejmenší *střední* nebo *maximální vzdálenosti* mezi procesory (měříme je v počtu spojů na cestě mezi procesory), co nejvyššího *stupně souvislosti* sítě a s ní související *průchodnosti*. Příklady některých teoreticky zajímavých regulárních struktur a struktur využívaných při konstrukci reálných systémů uvádí obr. 1.2.

Kruh je topologií výhodnou vzhledem k nízkému počtu propojení a relativní efektivitě algoritmů pracujících na kruhu. Praktické aplikace, které pro distribuci problému využívají hierarchického rozkladu, preferují *stromovou strukturu*. Vysoký stupeň spojitosti mají při pevném počtu rozhraní plošné a prostorové *mříže* (meshes), obr. 1.2 ukazuje často využívaný toroid a hierarchicky strukturovanou mříž s překříženími. Z teoretického hlediska je velice zajímavá *hyperkrychle*, která je považována za univerzální regulární propojovací strukturu. Její nevýhodou je s logaritmem počtu propojených procesorů rostoucí počet rozhraní.

Statická propojení procesorů jsou vhodná v případech, kdy buď struktura propojení procesů struktuře propojení procesorů přímo odpovídá (např. výpočet na kruhu na kruhové síti procesorů), nebo kdy ji lze na strukturu procesorů snadno mapovat. Často se dá mluvit o potřebě přizpůsobit strukturu propojení procesorů struktuře distribuovaného výpočtu.

Možností, jak propojit skupinu procesorů do konfigurací odpovídajících konkrétnímu výpočtu je více. Většinou se opírají o speciální propojovací prvky *křížové přepínače* nebo *přepojovací sítě*, které dovolí propojit procesory systému podle konkrétních požadavků daného programu.

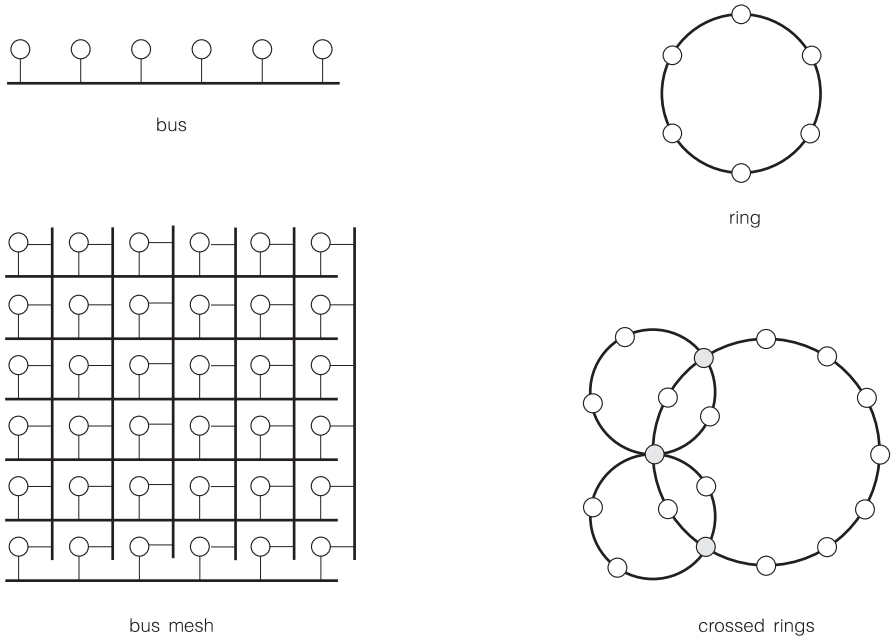


Obrázek 1.3: Dynamické propojovací struktury

Další, univerzálně použitelnou cestou, jak vyřešit problém konkrétního propojení procesů prostřednictvím existující statické struktury je, vytvořit komunikační podsystém poskytující *virtuální kanály* mezi procesory. Tato cesta nemá teoretická omezení běžná u mapování a může být při vhodné technologické podpoře (specializované komunikační procesory) i dostatečně efektivní.

Vícebodové spoje dovolují vzájemné propojení celé skupiny procesorů, snižují vzdálenosti mezi procesory a dovolují využít efektivních metod *skupinové komunikace — multicast a broadcast*. Většinou se opírají o lokální komunikační systémy — sběrnicové a kruhové sítě s vysokou přenosovou rychlostí. Systémy tohoto typu často označujeme jako systémy s *těsnou vazbou* (tightly coupled). Jejich omezením je, ve srovnání s přepojovacími architekturami, omezená kapacita komunikačního kanálu a závislost komunikačního podsystému na jediném prvku.

Podobně, jako jsme u dvoubodových spojů dosahovali větší pružnosti zprostředkovaným přenosem — přepojováním, lze i u vícebodových spojů překonat řadu omezení jejich propojením do složitějších celků. Příklady mohou být *mříž sběrnic* (*bus mesh*), o kterou se opírá multipočítačový systém Linda, nebo přepojování nad kruhovými sítěmi.



Obrázek 1.4: Vícebodové spoje a struktury

1.1.2 Distribuovaný program

Distribuovaný program lze pro výpočet na distribuované architektuře rozdělit na relativně velké komponenty, které realizují logicky uzavřené části výpočtu, a které si vzájemně vyměňují minimum informace. Mluvíme o *hrubé granularitě výpočtu* (coarse grain granularity). Takový způsob distribuce je typický pro systémy s málo výkonným komunikačním systémem, pro systémy s volnou vazbou.

Opačným extrémem vhodným pro systémy s velmi těsnou vazbou je rozklad programu na co nejmenší komponenty dovolující optimálně využít výpočetní kapacitu procesorů, komunikační náklady zanedbáváme. Mluvíme o *jemné granularitě výpočtu* (fine grain granularity).

Spolupráce jednotlivých komponent je dána způsobem, jakým si jednotlivé komponenty předávají výsledky svého výpočtu. Pro předávání dat mohou využívat *sdílené proměnné* (které lze velice snadno realizovat v architekturách s velmi těsnou vazbou), *výměnu zpráv* mezi jedním odesílatelem a jedním příjemcem (kterou lze velmi snadno realizovat v architekturách s volnou vazbou), nebo výměnu zpráv mezi jedním odesílatelem a více příjemci — *broadcast* (kterou lze velmi snadno realizovat v architekturách s těsnou vazbou — lokálních sítích). Rozdělením programu na komponenty a definováním jejich vzájemných vazeb (a to včetně metody) definujeme strukturu distribuovaného programu.

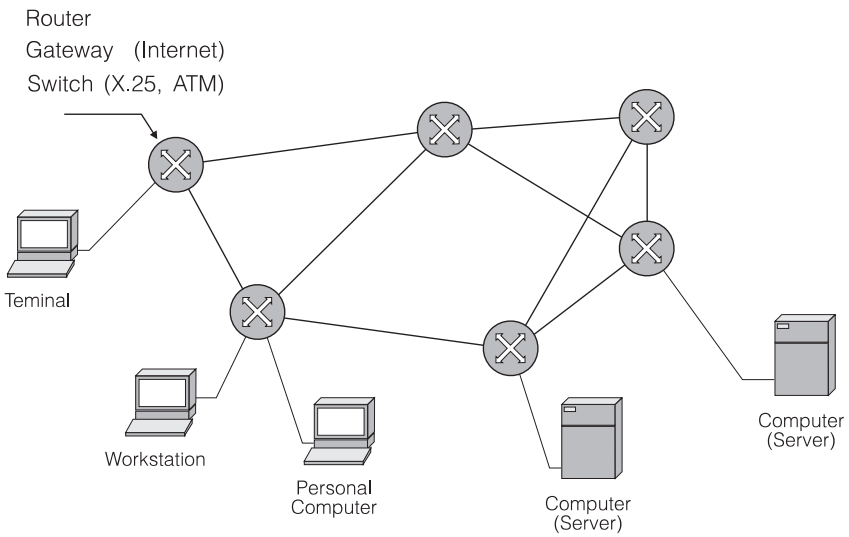
Důležitým prvkem návrhu distribuovaného systému je rozdělení jednotlivých komponent distribuovaného programu na jednotlivé procesory distribuované architektury. Mluvíme o *mapování* procesů na procesory, o mapování meziprocesových vazeb na strukturu propojení procesorů. Problém se může redukovat na stále velice komplikovaný problém optimálního mapování grafu meziprocesových komunikací (dvoubodových kanálů) na graf propojení procesorů (dvoubodovými spoji). Často se však setkáme s potřebou realizovat i složitější mapování,

jakými jsou mapování paměti sdílené více procesy na architekturu bez sdílené fyzické paměti (mluvíme o *distribuované sdílené paměti*) nebo mapování broadcast komunikace mezi procesy na polygonální architekturu. Posledním dvěma problémům se budeme věnovat v kapitolách pojednávajících o systémové podpoře distribuovaného výpočtu a o distribuovaných algoritmech.

Mapování procesů na procesory a komunikačních prostředků programu na komunikační prostředky architektury může být definováno *staticky* v programu, určeno *automaticky* před výpočtem nebo dokonce měněno dynamicky během výpočtu.

1.2 Komunikační architektura

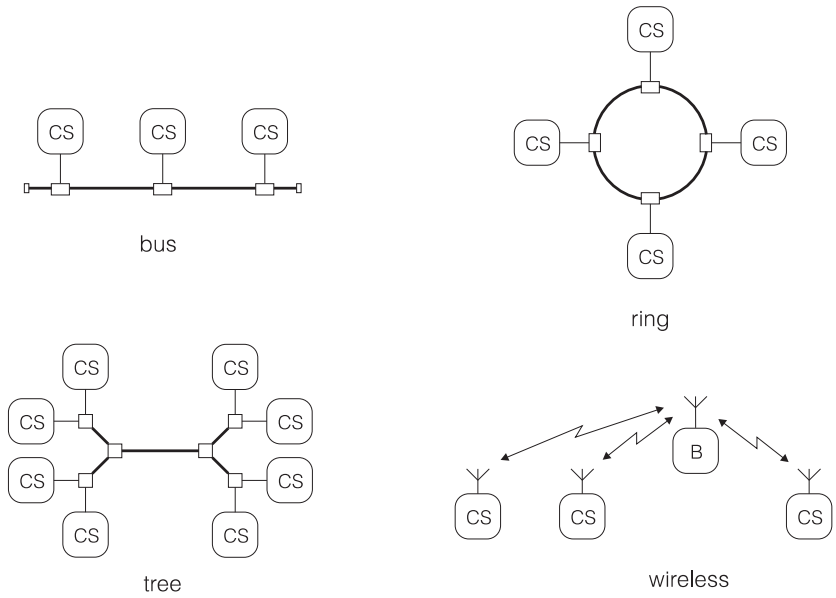
Distribuované aplikace se obvykle opírají o *komunikační podsystém — počítačovou síť*. Ta bývá vystavěna jako univerzální systém pro přenos dat. *Rozlehlé sítě WAN* (Wide Area Network) využívají pro spojení na velké vzdálenosti dvoubodových spojů. Tyto spoje propojujeme specializovanými počítači, které obvykle označujeme jako *směrovače* (router), do polygonálních sítí. Ty jsou schopné zajistit jak dostatečnou přenosovou kapacitu, tak i odolnost proti výpadkům jednotlivých prvků (dvoubodových spojů i směrovačů). Polygonální architektura přepojovacích sítí (obr. 1.5) je typická pro Internet, pro veřejné datové sítě (X.25), ale i pro moderní přenosové sítě ATM.



Obrázek 1.5: Rozlehlé přepojovací sítě — WAN

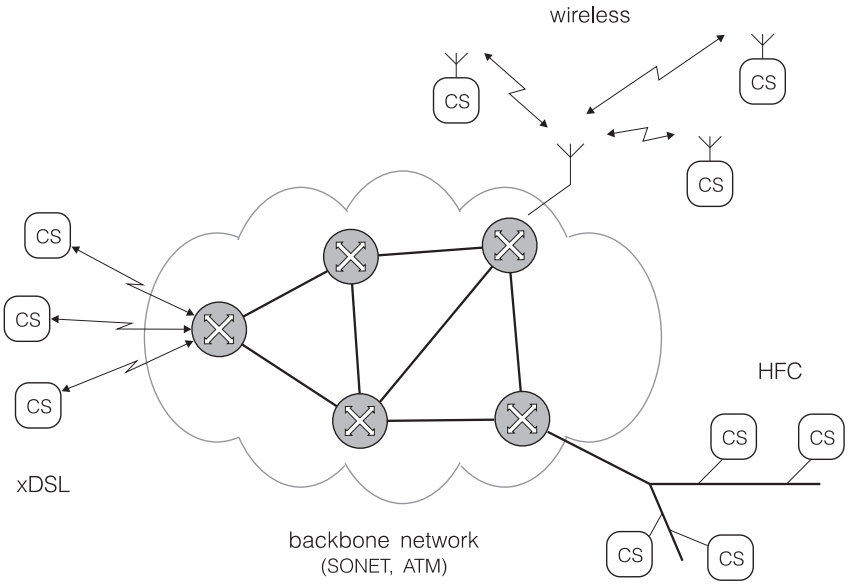
(Výjimečně se setkáme i s jinými označeními prvků s funkcí směrovače, např. veřejná datová síť své uzly označuje jako *přepínače X.25*, v terminologii Internetu se uzly sítě označují jako *gateway*.)

Alternativou k polygonálním přepojovacím sítím pro skupiny počítačů v geograficky omezené oblasti jsou *lokální sítě LAN* (Local Area Network). Sítě LAN se opírají o sdílený komunikační kanál, tím může být sběrnicový spoj, hvězdice tvořená dvoubodovými spoji a opakovači (repeater), nebo všesměrový rádiový kanál (obr. 1.6). Charakteristickou vlastností sdíleného kanálu je, že datový signál vyslaný jednou stanicí lze přijmout libovolnou jinou stanicí (u hvězdicových sítí lze opakování datového signálu z bezpečnostních důvodů omezit). Schopnost doručit datový signál všem stanicím má i kruhová síť opírající se o dvoubodové spoje.



Obrázek 1.6: Lokální sítě — LAN

Architektura polygonálních sítí WAN odpovídá představě přenosu dat mezi omezeným počtem koncových počítačů. S rostoucím počtem připojovaných účastníků a rostoucími požadavky na kvalitu přenosových služeb (doručování dat v reálném čase pro multimediální distribuci a hovorové služby) se typická struktura komunikačního systému podstatně mění (obr. 1.7).



Obrázek 1.7: Přenosové a přístupové sítě

Jádrem se stávají rychlé *přenosové sítě* využívající optických vláken, případně doplněné o směrové mikrovlnné spoje. Ty jsou schopné zajistit současnou podporu nejrůznějších služeb: přenos digitalizovaných telefonních hovorů, přenos digitalizovaného a případně komprimovaného videosignálu, klasický přenos dat, přenos dat s vyššími požadavky na kvalitu (paketová telefonie, distribuce zvukového a obrazového vysílání).

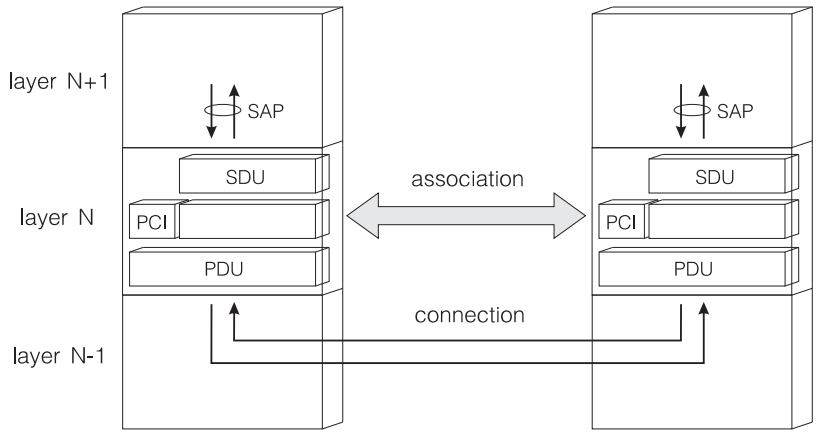
Připojení velkého množství koncových účastníků s vysokými požadavky na kvalitu doručování datového signálu (kapacita spojů, zpoždění datového signálu) si vyžádalo nová řešení, označovaná jako *přístupové sítě*. Ty se dnes, s ohledem na cenu řešení, opírají o efektivní využití existujících účastnických přípojek (telefonních linek, rozvodů kabelové televize). Přípojky ADSL (Asymmetric Digital Subscriber Link) dovolují doručovat datový signál rychlostí až 8 Mb/s, na kabelových rozvodech CATV lze vytvářet sdílené kanály s celkovou kapacitou až 40 Mb/s. Alternativou metalických spojů jsou směrové a všesměrové spoje v mikrovlnných pásmech (ISM 2.4 GHz, FWA 26 GHz), v budoucnu lze počítat s postupným přibližováním optických spojů ke koncovým účastníkům.

1.2.1 Model OSI

Složitost problémů, se kterými se setkáváme při propojování počítačů do počítačových sítí, vyžaduje použití vhodných modelů, které umožní vytvoření standardizovaných prvků a usnadní jejich použití v praktických implementacích.

Na síťové vybavení, technické a programové, jsme zvyklí se dívat jako na systém funkčních vrstev, ve kterém každá vyšší vrstva rozšiřuje možnosti vrstvy nižší. Pro přepojovací počítačové sítě, ze kterých se na počátku osmdesátých let vyvinuly dnes provozované veřejné datové sítě, byl vytvořen standardní model síťové architektury označovaný jako ISO/OSI (ISO Open Systems Interconnection).

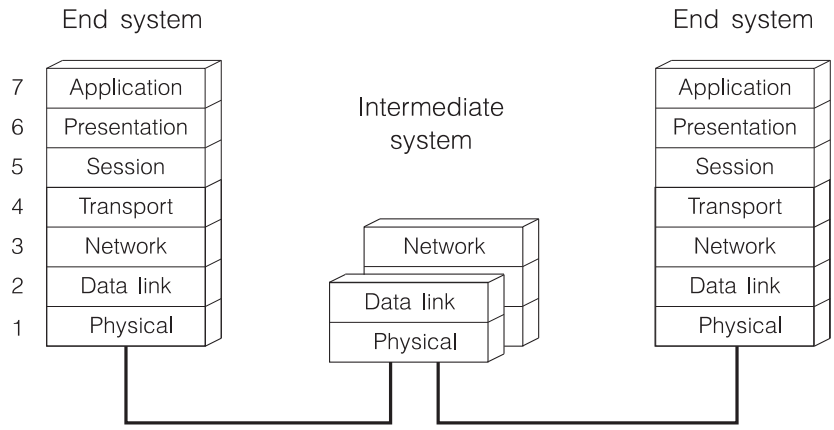
Model OSI popisuje komunikaci zajišťovanou počítači, jako hierarchii sedmi vrstev technických a programových prostředků, kde každá z vrstev zajišťuje funkce potřebné pro vrstvu vyšší a využívá služeb vrstvy nižší. Mezi jednotlivými vrstvami jsou (formou standardů a doporučení) definována rozhraní (*mezivrstvové protokoly*), mezi prvky stejné vrstvy jsou definována pravidla komunikace (*vrstvové protokoly*). Architekturu vrstev ilustruje obr. 1.8.



Obrázek 1.8: Architektura vrstev

Datové bloky předávané vyšší vrstvou SDU (Service Data Unit) jsou doplněny o řídící informace PCI (Protocol Control Information) a předávány ve formě datových bloků PDU (Protocol Data Unit) nižší vrstvě. Podobně v opačném směru, z doručovaných bloků PDU je jsou vybírána data SDU, řídící informace PCI je využita procedurou řízení, která implementuje vrstvový protokol.

Rozdělením funkcí zajišťovaných v síti (ochrana proti chybám přenosu, řízení toku dat, směrování) do jednotlivých vrstev má za cíl oddělit jejich řešení. Standard ISO OSI definuje sedm vrstev architektury (obr. 1.9).



Obrázek 1.9: Vrstvy síťové architektury ISO OSI

Fyzická vrstva definuje fyzické propojení mezi prvky sítě, jeho mechanické vlastnosti (konektory, typ média), elektrické vlastnosti (napěťové úrovně, způsob kódování a modulace) a u lokálních sítí i způsob propojení jednotlivých počítačů a metodu přístupu k přenosovému médiu.

Linková vrstva zajišťuje data proti chybám při přenosu. Zprávy jsou sítí přenášeny v pevně definovaných rámcích, rámce dovolují chránit předávaná data proti chybám. Struktura rámce často limituje délku bloků dat síťové vrstvy — mluvíme o tzv. paketech.

Síťová vrstva definuje způsob, jakým se pakety pohybují sítí, jak si je jednotlivé prvky sítě předávají na jejich cestě od odesílatele k adresátovi. Mechanismy vrstvy se konečně starají i o ochranu sítě proti nadměrné zátěži.

Transportní vrstva umožňuje komunikaci aplikačních programů v síti, zajišťuje vytváření dočasných komunikačních spojení mezi nimi a konečně i rozklad zpráv do paketů a skládání paketů do zpráv.

Relační vrstva (vrstva sezení) doplňuje logické rozhraní pro aplikační programy o funkce jakými jsou podpora poloduplexu a vkládání synchronizačních bodů. Zjednodušuje komunikaci programů i uživatelský pohled na komunikační kanál.

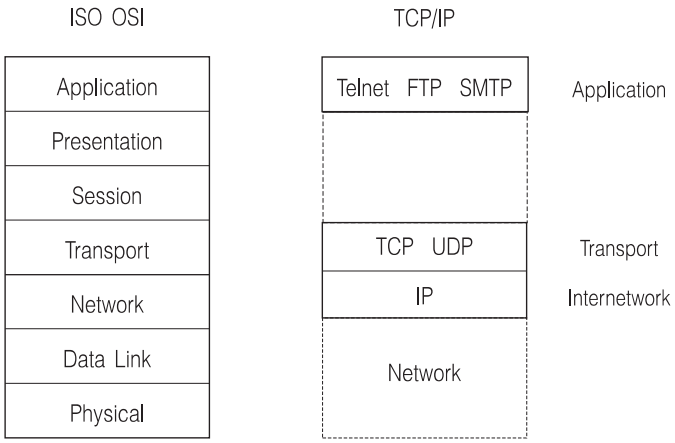
Presentační vrstva transformuje přenášená data – zajišťuje převody kódů a formátů dat pro nekompatibilní počítače, kompresi přenášených dat a konečně i utajování přenášených dat.

Aplikační vrstva je konečně vrstvou standardních aplikací a vrstvou rozhraní, která zjednodušují programování jednoúčelových aplikací.

Model OSI se stal základem pro standardy v oblasti rozlehlých přepojovacích sítí. V oblasti lokálních sítí jsou nejdůležitější doporučení vytvářená pracovními skupinami IEEE (Institute of Electrical and Electronics Engineers). Tyto normy (IEEE 802.x) pokrývají nižší vrstvy (fyzickou a linkovou) architektury lokálních sítí (Ethernet, Token Ring, HFC, Wireless LANs).

1.2.2 Model TCP/IP

Sloučení různorodých síťových technologií do jednotného komunikačního systému se stalo cílem návrhu technologie, označované jako *internetworking*. Technologie *TCP/IP* (Transmission Control Protocol/Internet Protocol) se stala základem současného Internetu. Návrhy, experimentální verze a konečné standardy TCP/IP jsou označovány jako *RFC* (Request for Comments) a pokrývají hlavně síťovou a transportní vrstvu (obr. 1.10).



Obrázek 1.10: Vrstvy síťové architektury TCP/IP

Základem architektury TCP/IP je síťový protokol *IP* (*Internet Protocol*). Jeho pakety, případně rozložené do fragmentů, lze pro přenos mezi směrovači IP vkládat do rámců libovolné linkové technologie, nebo do paketů jiné síťové architektury (např. veřejné datové sítě X.25). Jednotlivé pakety, případně jejich fragmenty jsou k cíli směrovány nezávisle.

Vlastnosti protokolu IP nedovolují zabránit ztrátám, duplikaci a záměně pořadí paketů, resp. jejich fragmentů. Zvládnout tyto situace korektně přímo v aplikaci je obtížné, proto jsou běžně využívány transportní protokoly *UDP* (User Datagram Protocol) a *TCP* (Transmission Control Protocol), které potřebné podpůrné mechanismy zahrnují. Protokol UDP zajisťuje skládání fragmentů do paketů, multiplex (možnost využívat síťovou komunikaci více aplikacemi současně), a případně detekci chyb. Podstatně komplikovanější protokol TCP navíc podporuje koncové potvrzování (dovoluje eliminovat ztráty a duplikace paketů), rozklad aplikačních toků dat (stream) na pakety a poměrně komplikované mechanismy řízení toku.

Důležitou součástí modelu TCP/IP jsou protokoly standardních služeb (např. vzdálený interaktivní přístup Telnet, přenos souborů FTP, elektronická pošta SMTP, správa síťových prvků SNMP). Protokolům relační a presentační vrstvy jsou materiály RFC věnovány výjimečně (např. protokoly SunRPC a XDR), nebo tehdy, jsou-li nutnou podporou standardních služeb (např. definice MIB pro systém správy SNMP).

2. Přepojovací komunikační systémy

Použití přepojovací sítě jako komunikačního systému je velmi výhodné, přepojovací síť zvyšuje využitelnost kapacity přenosových linek jejich sdílením, a současně může zvýšit spolehlivost komunikace díky topologické redundanci.

Existence více vzájemně disjunktních cest mezi koncovými uzly dovolí zajistit, že se výpadky jednotlivých prvků sítě, uzlů nebo linek, neprojeví jako ztráta možnosti komunikovat. Chceme, aby se síť z pohledu koncového účastníka jevila jako spolehlivé komunikační médium. To ovšem souvisí s *topologií sítě* a právě topologii se budeme věnovat v prvé části této kapitoly.

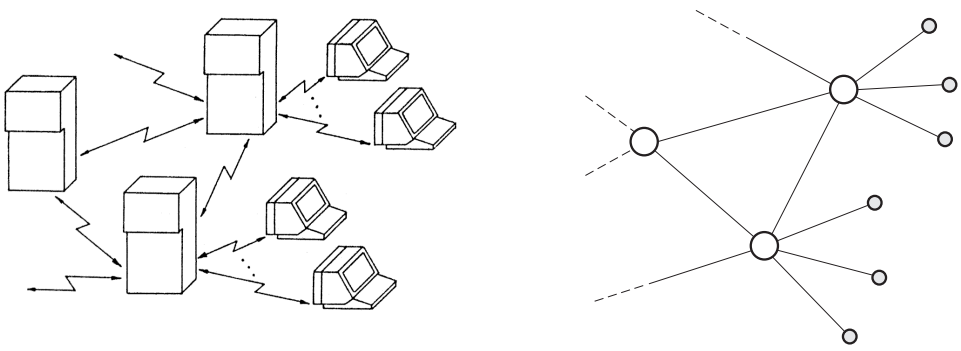
Vysoké propustnosti přepojovací sítě dosahujeme časovým multiplexem na přenosových linkách — přepojováním zpráv nebo paketů. Přenášené zprávy jsou přitom v uzlech dočasně ukládány do paměti, kde vytvářejí fronty, v nichž čekají na odeslání. Na přepojovací uzel se lze dívat jako na *systém hromadné obsluhy*, takový přístup dovolí studovat závislost zpoždění dat při přenosu na zátěži sítě.

Datové toky mezi koncovými uzly je třeba vést po cestách s co nejmenším počtem linek a vybírat přitom linky, které zajistí co nejnižší zpoždění pro přenášená data. Pouze tak lze dosáhnout vysoké propustnosti sítě a malého středního zpoždění přenášených dat. Výběr nejvýhodnějších cest se opírá o *směrovací algoritmy*.

Konečně, na návrh topologie přepojovací sítě a na rozložení toků do jednotlivých cest se můžeme dívat jako na složitou optimalizační úlohu. Chceme vybudovat co nejspolehlivější přepojovací síť, s co nejlepší časovou odezvou, a to za limitovanou cenu. Některé principy *optimalizace sítí* si uvedeme v závěru této kapitoly.

2.1 Topologie

Zajištění komunikace v síti i při poruchách jednotlivých prvků (linek a uzlů) a výběr nejvhodnější cesty pro přenos dat jsou dvě základní úlohy, které při návrhu a provozu sítě řešíme. Vhodným modelem je graf — neorientovaný i orientovaný, neohodnocený i ohodnocený. Obrázek 2.1 uvádí korespondenci mezi strukturou přepojovací sítě a jejím grafovým modelem. každému přepojovacímu uzlu je přiřazen uzel grafu, každé lince je přiřazena hrana grafu. V grafovém modelu respektujeme pouze ty vlastnosti, které nás zajímají, například zpoždění linek, jejich kapacitu, a podobně.



Obrázek 2.1: Grafový model přepojovací sítě

2.1.1 Grafový model

Neorientovaný graf

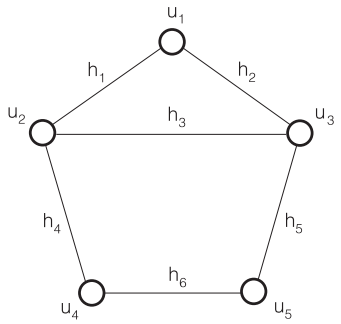
Neorientovaným grafem rozumíme dvojici

$$\mathbf{G} = (U, H) \ ,$$

kde U je neprázdná konečná množina a H je množina neuspořádaných dvojic $h_k = \{u_i, u_j\}$, $u_i, u_j \in U$ a $u_i \neq u_j$. Prvky množiny U nazýváme *uzly* grafu $\mathbf{G}$, o prvcích množiny H mluvíme jako o *hranách* grafu $\mathbf{G}$. Uzly u_i a u_j nazýváme *koncovými uzly* hrany $h_k = \{u_i, u_j\}$.

Poznámka
V teorii grafů se setkáváme i s dalšími grafovými strukturami. *Pseudografy* mohou obsahovat hrany $h_k = \{u_i, u_i\}$ a jsou z hlediska přepojovacích sítí nezajímavé. *Multigrafy* mohou obsahovat více hran pro jednu dvojici uzlů a jejich použití pro popis přepojovacích sítí je celkem přirozené (multigraf jako model sítě používala např. síťová architektura SNA firmy IBM).

Neorientovaný graf obvykle znázorňujeme tak, že uzel zobrazíme jako kroužek a hranu jako čáru spojující obrazy koncových uzlů. Příklad neorientovaného grafu uvádí obrázek 2.2.



$$U = \{u_1, u_2, u_3, u_4, u_5\}$$

$$H = \{\{u_1, u_2\}, \{u_1, u_3\}, \{u_2, u_3\},$$

$$\{u_2, u_4\}, \{u_3, u_5\}, \{u_4, u_5\}\}$$

Obrázek 2.2: Neorientovaný graf

Maticí neorientovaného grafu $\mathbf{G}$ rozumíme čtvercovou matici $\mathbf{A} = [a_{ij}]$, pro jejíž prvky a_{ij} platí

$$a_{ij} = \begin{cases} 1 & \text{pokud } \{u_i, u_j\} \in H \ , \\ 0 & \text{pokud } \{u_i, u_j\} \notin H \ . \end{cases}$$

Matice grafu z obrázku 2.2 má tvar

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix} \ .$$

Matice grafu A vypovídá o bezprostředním sousedství uzlů grafu a často ji nazýváme maticí přímých spojů (adjacency matrix).

Incidenční maticí neorientovaného grafu $\mathbf{G}$ rozumíme matici $\mathbf{I} = [i_{ik}]$, ve které

$$i_{ik} = \begin{cases} 1 & \text{pokud uzel } u_i \text{ je koncovm uzlem hrany } h_k \ , \\ 0 & \text{pokud uzel } u_i \text{ nen koncovm uzlem hrany } h_k \ . \end{cases}$$

Pokud $i_{ik} = 1$, pak o hraně h_k říkáme, že inciduje s uzlem u_i. Incidenční matice našeho grafu z obrázku 2.2 má tvar

$$\mathbf{I} = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} .$$

Stupněm uzlu u_i nazýváme počet hran, které s tímto uzlem incidují, a označujeme ho $deg(u_i)$. Číslo

$$\delta(\mathbf{G}) = \min_{(i)}[deg(u_i)]$$

nazýváme *stupněm grafu*. Grafy, jejichž všechny uzly mají týž stupeň, označujeme jako *grafy regulární*.

Posloupnost hran $\{u_{i0}, u_{i1}\}$,$\{u_{i1}, u_{i2}\}$,...,$\{u_{i(l-1)}, u_{il}\}$ nazýváme cestou délky l. Uzly u_{i0} a u_{il} nazýváme *koncovými uzly* cesty, a ze všech cest, kterými můžeme tyto uzly spojit, nás bude nejvíce zajímat *nejkratší cesta*, jejíž délku názýváme *vzdáleností* mezi uzly u_{i0} a u_{il}, a označujeme ji $d(u_{i0}, u_{il})$. Číslo

$$d(\mathbf{G}) = \max_{(i,j)}[d(u_i, u_j)]$$

nazýváme *průměrem grafu* $\mathbf{G}$.

Graf, ve kterém lze spojit cestou libovolné dva uzly, nazýváme *souvislým*. Postupným odebíráním hran grafu $\mathbf{G}$ lze získat podgraf $\mathbf{G}^* \subset \mathbf{G}$, který je ještě souvislý, ale po odebrání libovolné hrany se rozpadne na dvě komponenty. Takový podgraf grafu $\mathbf{G}$ nazýváme *kostrou* grafu; kostra grafu je stromem a mezi každými jejími dvěma uzly existuje jediná cesta. Kostra grafu $\mathbf{G}$, pro kterou platí, že

$$\sum_{(i,j)} d(u_i, u_j)$$

dosahuje minima (mezi všemi kostrami), se nazývá *minimální kostrou grafu*.

Kružnicí nazýváme cestu, pro jejíž koncové uzly u_i a u_j platí $u_i = u_j$. Kružnici, která prochází všemi uzly grafu nazýváme *Hamiltonovou kružnicí*.

Cesty mezi dvěma koncovými uzly, které neobsahují společnou hranu, nazýváme *hranově disjunktní*. Počet hranově disjunktních cest odpovídá kapacitě *minimálního hranového řezu* oddělujícího zadané koncové uzly. Kapacita minimálního hranového řezu udává počet hran, které musíme odebrat, aby přestala existovat třeba i jediná cesta mezi danými koncovými uzly.

Vyhodnotíme-li počet hranově disjunktních cest pro všechny dvojice koncových uzlů grafu $\mathbf{G}$, pak minimum těchto čísel nazýváme *stupněm hranové souvislosti* grafu a označujeme ho jako $\lambda(\mathbf{G})$. Stupeň hranové souvislosti udává kapacitu *minimálního hranového řezu* grafu, počet hran, které musíme odebrat, aby se graf rozpadl na dvě komponenty.

Cesty mezi dvěma koncovými uzly, které neprocházejí společným uzlem, nazýváme *uzlově disjunktní*. Počet uzlově disjunktních cest odpovídá kapacitě *minimálního uzlového řezu* oddělujícího zadané koncové uzly. Kapacita minimálního uzlového řezu udává počet uzlů, které musíme z grafu odebrat, aby přestala existovat třeba i jediná cesta mezi danými koncovými uzly.

Vyhodnotíme-li počet uzlově disjunktních cest pro všechny dvojice koncových uzlů grafu $\mathbf{G}$, pak minimum těchto čísel nazýváme *stupněm uzlové souvislosti* grafu a označujeme ho jako $\kappa(\mathbf{G})$. Stupeň uzlové souvislosti udává kapacitu *minimálního uzlového řezu* grafu, počet uzlů, které musíme odebrat, aby se graf rozpadl na dvě komponenty.

Pro stupeň grafu $\delta(\mathbf{G})$, stupeň hranové souvislosti $\lambda(\mathbf{G})$ a stupeň uzlové souvislosti $\kappa(\mathbf{G})$ platí Whitneyho teorém

$$\kappa(\mathbf{G}) \leq \lambda(\mathbf{G}) \leq \delta(\mathbf{G}) \ .$$

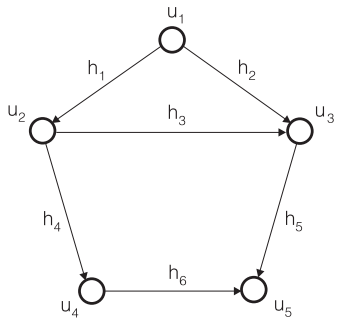
Orientovaný graf

Orientovaným grafem rozumíme dvojici

$$\mathbf{G} = (U, H) \ ,$$

kde U je neprázdná konečná množina a H je množina uspořádaných dvojic $h_k = (u_i, u_j)$, kde $u_i, u_j \in U$ a $u_i \neq u_j$. Stejně jako u neorientovaného grafu, mluvíme o prvcích množinu U jako o *uzlech* grafu $\mathbf{G}$, prvky množiny H nazýváme *orientovanými hranami* grafu $\mathbf{G}$. Uzel u_i nazýváme *počátečním uzlem* a uzel u_j *koncovým uzlem* hrany $h_k = (u_i, u_j)$.

Orientaci hran vyjadřujeme graficky šipkou na hraně orientovanou ve směru od počátečního ke koncovému uzlu. Obrázek 2.3. uvádí příklad orientovaného grafu a jeho znázornění.



$$U = \{u_1, u_2, u_3, u_4, u_5\}$$

$$H = \{(u_1, u_2), (u_1, u_3), (u_2, u_3),$$

$$(u_2, u_4), (u_3, u_5), (u_4, u_5)\}$$

Obrázek 2.3: Orientovaný graf

Maticí orientovaného grafu $\mathbf{G}$ rozumíme čtvercovou matici $\mathbf{A} = [a_{ij}]$, pro jejíž prvky a_{ij} platí

$$a_{ij} = \begin{cases} 1 & \text{pokud } (u_i, u_j) \in H \ , \\ -1 & \text{pokud } (u_j, u_i) \in H \ , \\ 0 & \text{pokud } (u_i, u_j) \notin H \ . \end{cases}$$

Matice grafu z našeho obrázku 2.3 má tvar

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ -1 & 0 & -1 & 1 & 0 \\ -1 & 1 & 0 & 0 & -1 \\ 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 1 & -1 & 0 \end{bmatrix} \ .$$

U orientovaých grafů mluvíme o *výstupním stupni uzlu* u, jako o počtu hran, které z uzlu u vycházejí, a o *vstupním stupni uzlu* u, jako o počtu hran, které v uzlu u končí.

Podobně jako u neorientovaných grafů zavádíme pojmy *orientovaná cesta*, *nejkratší orientovaná cesta*, *vzdálenost* a *průměr grafu*.

Graf, ve kterém lze spojit orientovanou cestou každé dva uzly, nazýváme *silně souvislým*. U orientovaných grafů má význam i *stupeň hranové souvislosti* a *stupeň uzlové souvislosti*.

Přestože je orientovaný graf korektnějším modelem přepojovací sítě než graf neorientovaný, většinou se pokoušíme vystačit s neorientovaným grafem.

Ohodnocený graf

Ohodnoceným grafem (orientovaným nebo neorientovaným) nazýváme trojici

$$\mathbf{G} = (U, H, C) \ ,$$

kde U je množina uzlů, H je množina hran (orientovaných nebo neorientovaných) a C je zobrazení, které každé hraně přiřadí číselnou hodnotu. Při analýze přpojovacích sítí obvykle ohodnocujeme hrany grafů dvěma způsoby: jejich *délkou* (zpožděním) nebo jejich *kapacitou*.

Matice grafu $\mathbf{G}$ ohodnoceného *délkami* $\mathbf{A} = [a_{ij}]$ je tvořena prvky a_{ij}, pro které platí

$$a_{ij} = \begin{cases} 0 & \text{pokud } i = j \ , \\ l(u_i, u_j) & \text{pokud } \{u_i, u_j\} \in H \ , \\ \infty & \text{pokud } \{u_i, u_j\} \notin H \ . \end{cases}$$

Pro graf ohodnocený délkou hran má smysl hledat *nejkratší cestu* mezi dvěma koncovými uzly, tedy cestu, pro kterou je součet délek hran cesty minimální. Nejkratší cesta reprezentuje například nejmenší zpoždění zprávy při přenosu. Pro síť ohodnocenou cenami linek má smysl hledat *minimální kostru*, která reprezentuje nejlevnější variantu sítě.

Matice grafu $\mathbf{G}$ ohodnoceného *kapacitami* $\mathbf{A} = [a_{ij}]$ je tvořena prvky a_{ij}, pro které platí

$$a_{ij} = \begin{cases} \infty & \text{pokud } i = j \ , \\ C(i, j) & \text{pokud } \{u_i, u_j\} \in H \ , \\ 0 & \text{pokud } \{u_i, u_j\} \notin H \ . \end{cases}$$

U grafu ohodnoceného kapacitami má smysl mluvit o *kapacitě hranového řezu*, o *kapacitě minimálního hranového řezu*, která odpovídá velikosti maximálního toku mezi dvěma danými koncovými uzly, a o *kapacitě minimálního hranového řezu grafu*.

2.2 Algoritmy nejkratší cesty

Většina směrovacích metod v přepojovacích sítích se snaží o minimalizaci střední doby zpoždění paketu. Opírají se přitom o znalost (alespoň částečnou) topologie sítě a o změřená nebo odhadnutá zpoždění paketů na jednotlivých linkách. Pro graf ohodnocený těmito údaji lze určit nejkratší cestu k zadanému cílovému uzlu. Uvedeme si zde tři, v oblasti počítačové komunikace nejpoužívanější, algoritmy.

Nejznámější *Dijkstrův* algoritmus se opírá o znalost topologie sítě a jejího ohodnocení. Jeho použití je výhodné pro lokální výpočet směrovacích tabulek a pro návrh tras virtuálních kanálů (např. u technologie ATM).

Ford-Fulkersonův algoritmus se opírá o výměnu údajů o vzdálenostech k cílovým uzlům mezi sousedy. Jeho výhodou je jednoduchá implementace, bez výraznějších úprav má však nepříjemné dynamické chování při výraznějších změnách ohodnocení linek a při výpadcích.

Poslední, *Floyd-Washallův* algoritmus vyžaduje kompletní informaci o topologii a ohodnocení sítě. Jedním spuštěním lze získat kompletní směrovací tabulky pro všechny uzly sítě, je tedy vhodný pro centralizovaný výpočet směrovacích tabulek.

2.2.1 Dijkstra

Nejznámějším algoritmem nejkratší cesty je asi algoritmus známý jako *Dijkstrův* algoritmus. Opírá o ohodnocení hran grafu nezápornou funkcí $l(u, v)$ a nalezne pro zvolený uzel s grafu **G** strom nejkratších cest vedoucích ze všech ostatních uzlů grafu **G** do uzlu s:

1. Každý z uzlů grafu označíme trojicí hodnot (L, P, D). Hodnota L udává dosud zjištěnou vzdálenost k uzlu s, hodnota D udává index (označení) uzlu, který je sousedem na nejkratší dosud zjištěné cestě k uzlu s. Hodnota P reprezentuje fakt, že získané údaje L a D se už v dalším výpočtu nebudou měnit. Výpočet zahájíme s hodnotami:

$$L(s) = 0,\ D(s) = s,\ P(s) = 1 \text{ pro uzel } s$$
$$L(v) = \infty,\ D(v) = v,\ P(v) = 0 \text{ pro uzly } v \neq s.$$

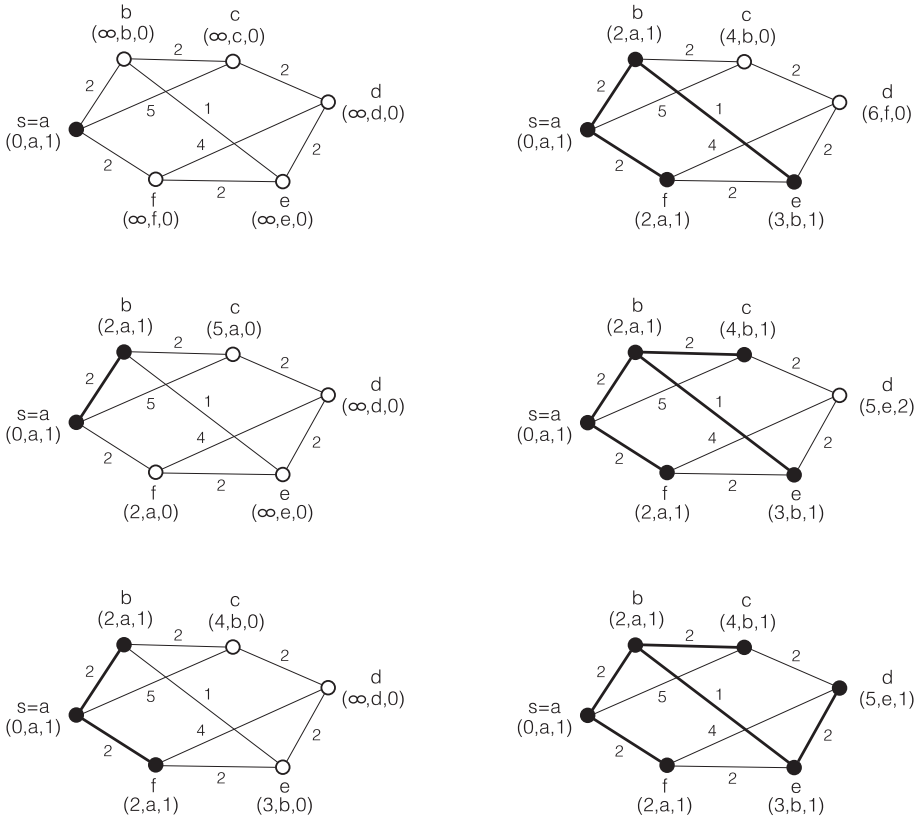
   Uzel s si pro další krok označíme jako uzel u.

2. Pro každý uzel v, pro který je $P(v) = 0$ vypočteme hodnotu $M = min\{L(v), L(u)+l(u, v)\}$ Pokud platí $M < L(v)$ změníme označení uzlu v na $L(v) = M$, $D(v) = u$.

3. Mezi uzly v, pro které $P(v) = 0$ vybereme uzel s minimální hodnotou $L(v)$. (Je-li takových uzlů více, zvolíme ten, pro který je cesta do uzlu s nejkratší i z hlediska počtu hran.) Pro tento uzel, který pro příští krok bude naším uzlem u, změníme hodnotu $P(u)$ na $P(u) = 1$.

4. Pokud v grafu zbývá alespoň jeden uzel s hodnotou $P(v) = 0$, vrátíme se k bodu 2.

Použití algoritmu si budeme ilustrovat na obrázku 2.4, silně vytažené jsou hrany postupně vytvářeného stromu nejkratších cest s kořenem s.



Obrázek 2.4: Aplikace Dijkstrova algoritmu nejkratší cesty

Algoritmus poskytuje jeden řádek směrovací tabulky pro každý uzel grafu **G**. Abychom zkonstruovali úplné tabulky, musíme Dijkstrův algoritmus spustit pro každý z uzlů grafu jako jeho kořen.

2.2.2 Bellman-Ford / Ford-Fulkerson

Malou modifikací Dijkstrova algoritmu získáme symetrický distribuovaný algoritmus vytvářející strom nejkratších cest vedoucích do uzlu s v grafu **G**.

1. Pro každý z uzlů grafu udržujeme dvojici hodnot (L, D). Hodnota L udává dosud zjištěnou vzdálenost k uzlu s, hodnota D udává index (označení) uzlu, který je sousedem na nejkratší dosud zjištěné cestě k uzlu s. Výpočet zahájíme s hodnotami:
$$L(s) = 0,\ D(s) = s \text{ pro uzel } s$$
$$L(v) = \infty,\ D(v) = v \text{ pro uzly } v \neq s.$$
2. Pro každý uzel grafu $v \in \mathbf{G}$ vypočteme hodnotu $M(v) = min\{L(v), L(u)+l(u,v)\}, u \in \mathbf{G}$.
3. Pro každý uzel, pro který platí $M(v) < L(v)$ změníme označení na $L(v) = M(v), D(v) = u$, kde u je označení sousedního uzlu, pro který bylo zjištěno minimum.
4. Pokud bylo změněno označení u alespoň jednoho uzlu, vrátíme se k bodu 2.

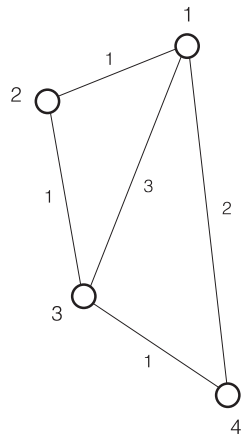
Algoritmus lze přímočaře implementovat v síti propojených počítačů. Krok 2 algoritmu odpovídá výměně dosud zjištěné hodnoty L se sousedy. V praxi budeme pracovat s vektorem vzdáleností ke všem uzlům sítě, algoritmus pak vytváří přímo směrovací tabulky.

2.2.3 Floyd-Warshall

Pro výpočet na počítači jsou výhodné maticové algoritmy (snadno se paralelizují). Takovým algoritmem je algoritmus popsaný Floydem. Opírá se o matici ohodnoceného grafu $\mathbf{A} = [a_{ij}]$ a počítá matici vzdáleností uzlů v grafu a směrovací tabulky pro jednotlivé uzly.

1. Zavedeme matici vzdáleností $\mathbf{W} = [w_{ij}]$ a matici směrů $\mathbf{Z} = [z_{ij}]$. Maticím přiřadíme počáteční hodnoty: $w_{ij} = a_{ij}$ pro všechna $i, j \in 1..n$, $z_{ij} = j$, pokud existuje hrana z uzlu u_i do uzlu u_j, $z_{ij} = \infty$, pokud neexistuje hrana z uzlu u_i do uzlu u_i.
2. Vezmeme $k = 0$.
3. Zvýšíme k o jedničku, $k = k + 1$. Pro každou dvojici uzlů u_i a u_j takových, že $i \neq k$ a $j \neq k$ nalezneme $M = min\{w_{ij}\ , w_{ik} + w_{kj}\}$. Pokud $M < w_{ij}$, pak změníme hodnoty w_{ij} a z_{ij} tak, že $w_{ij} = M$ a $z_{ij} = z_{ik}$.
4. Pokud je $k < n$, vrátíme se k bodu 3.
5. Matice **W** udává vzdálenosti na grafu, matice **Z** obsahuje směrovací tabulky.

Použití algoritmu ilustruje obrázek 2.5. Uvádí postupně úpravy matic **W** a **Z**, změny v jednotlivých krocích výpočtu jsou zvýrazněny podtržením.



$$\begin{bmatrix} 0 & 1 & 3 & 2 \\ 1 & 0 & 1 & \\ 3 & 1 & 0 & 1 \\ 2 & & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & \\ 1 & 2 & 3 & 4 \\ 1 & & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 3 & 2 \\ 1 & 0 & 1 & \underline{3} \\ 3 & 1 & 0 & 1 \\ 2 & \underline{3} & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & \underline{1} \\ 1 & 2 & 3 & 4 \\ 1 & \underline{1} & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & \underline{2} & 2 \\ 1 & 0 & 1 & 3 \\ \underline{2} & 1 & 0 & 1 \\ 2 & 3 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & \underline{2} & 4 \\ 1 & 2 & 3 & 1 \\ \underline{2} & 2 & 3 & 4 \\ 1 & 1 & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 & 2 \\ 1 & 0 & 1 & \underline{2} \\ 2 & 1 & 0 & 1 \\ 2 & \underline{2} & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 2 & 4 \\ 1 & 2 & 3 & \underline{3} \\ 2 & 2 & 3 & 4 \\ 1 & \underline{3} & 3 & 4 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & 2 & 2 \\ 1 & 0 & 1 & 2 \\ 2 & 1 & 0 & 1 \\ 2 & 2 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 2 & 2 & 4 \\ 1 & 2 & 3 & 3 \\ 2 & 2 & 3 & 4 \\ 1 & 3 & 3 & 4 \end{bmatrix}$$

Obrázek 2.5: Použití Floydova algoritmu

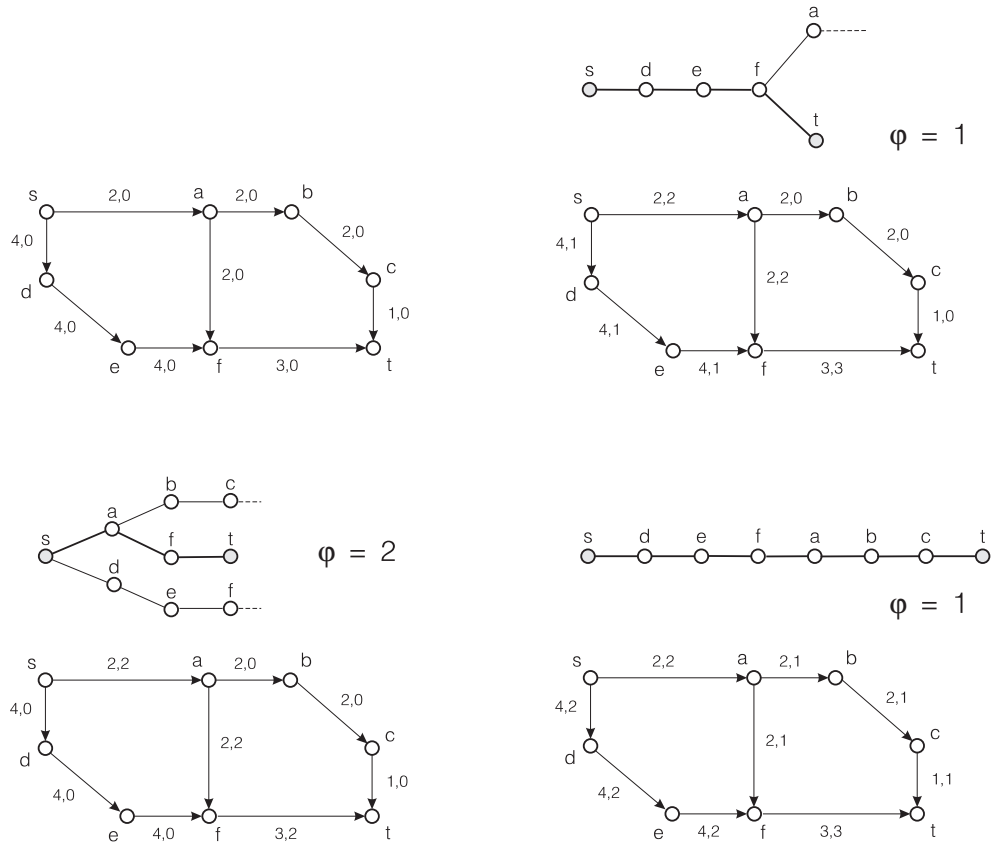
2.3 Topologické vlastnosti sítě

2.3.1 Určení maximálního toku

Pro přepojovací síť s konečnou kapacitou linek může být zajímavé znát, jak velké informační toky lze sítí přenášet. Odpověď na takovou otázku dává algoritmus *maximálního toku*. Algoritmus je formulován pro orientovaný graf, jehož hrany jsou ohodnoceny kapacitami. Výsledkem je maximální tok, který lze přenést mezi dvěma zvolenými uzly.

1. Máme orientovaný graf, jehož hrany jsou ohodnocené kapacitou. Každé hraně připíšeme další ohodnocení, velikost toku, který hranou prochází. Koncové uzly, pro které budeme vyhodnocovat maximální tok si označíme jako s (vstup toku sítě) a t (výstup toku ze sítě). Algoritmus startujeme s nulovým tokem z uzlu s do uzlu t.
2. Nalezneme nejkratší cestu z uzlu s do uzlu t, přičemž bereme v úvahu pouze hrany, které
   a) jsou orientovány ve směru cesty a jejich kapacita je větší než dosud procházející tok,
   b) jsou orientovány proti směru cesty a prochází jimi nenulový tok.
3. Pokud taková cesta neexistuje, nelze již k toku, který dosud mezi uzly s a t teče, přidat žádný tok. Dosavadní tok je hledaným tokem maximálním.
4. Pokud vhodnou cestu nalezneme, najdeme minimum z
   a) rozdílů kapacit a dosavadního toku pro hrany orientované ve směru cesty,
   b) dosavadního toku pro hrany orientované proti směru cesty.
5. Proložený tok přičteme k dosavadnímu toku mezi uzly s a t a vrátíme se k bodu 2.

Postupné přidávání toku nejkratším cestám nalezeným v bodě 2 algoritmu ilustruje obrázek 2.6.

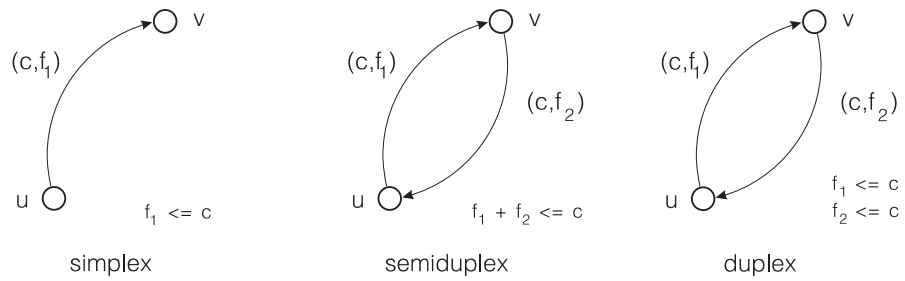


Obrázek 2.6: Určení maximálního toku

Hrany jsou ohodnoceny dvojicemi (kapacita, dosavadní tok). Schémata u jednotlivých kroků popisují postup vyhledávání nejkratší cesty, které lze ještě nějaký tok přidat. Poslední krok našeho příkladu odpovídá využití opačně orientované hrany na cestě mezi koncovými uzly.

Poznámka

Při použití uvedeného algoritmu si musíme uvědomit, že orientovaná hrana reprezentuje simplexní spoj. Duplexní spoj musíme reprezentovat dvojicí opačně orientovaných hran. U poloduplexního spoje musíme navíc zajistit, aby součet toků v obou směrech nepřesáhl kapacitu linky (režii přepínání směru přenosu zanedbáváme).



Obrázek 2.7: Reprezentace simplexních, poloduplexních a duplexních spojů

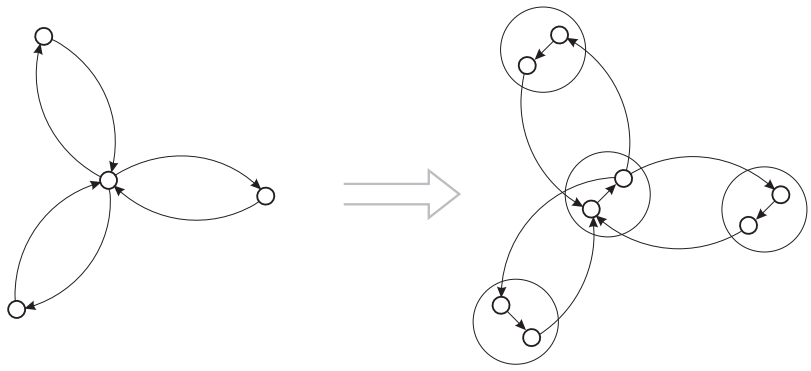
V případě poloduplexních a duplexních spojů lze respektovat zbývající kapacitu v dopředném směru a tok ve zpětném směru současně, a zrychlit tak práci algoritmu.

2.3.2 Souvislost sítě

Algoritmus maximálního toku lze využít pro zjištění stupně hranové a uzlové souvislosti sítě. Pokud ohodnotíme každou hranu grafu **G** jednotkovou kapacitou, je výsledkem algoritmu maximálního toku kapacita minimálního hranového řezu oddělujícího zvolené uzly s a t. Spuštěním algoritmu pro všechny dvojice uzlů grafu a nalezením minima vypočtených hodnot dostaneme stupeň hranové souvislosti $\lambda(\mathbf{G})$.

Stupeň hranové souvislosti nám udává, kolik linek musí vypadnout, aby mohlo dojít k rozpadu sítě na oddělené části. Často nás však zajímá nejen odolnost sítě proti výpadkům linek, ale také odolnost sítě proti výpadkům uzlů. Zajímá nás tedy určení stupně uzlové souvislosti $\kappa(\mathbf{G})$.

Zjištění stupně uzlové souvislosti lze naštěstí poměrně snadno převést na zjištění stupně souvislosti hranové: Každý uzel sítě rozdělíme na dvojici uzlů (u_{in}, u_{out}). Hrany vedoucí k původnímu uzlu rozdělíme na vstupující hrany, ty vedeme k prvnímu z uzlů dvojice u_{in}, a na vystupující hrany, ty budou vycházet z druhého uzlu dvojice u_{out}. Oba nové uzly u_{in} a u_{out} konečně propojíme orientovanou hranou (u_{in},u_{out}) s jednotkovou kapacitou (obr. 2.8).



Obrázek 2.8: Rozklad uzlu pro určení uzlové souvislosti

Výpadek uzlu tak převedeme na výpadek hrany (u_{in},u_{out}) a minimální hranový řez upraveného grafu má kapacitu rovnou stupni uzlové souvislosti grafu původního (opíráme se zde o fakt, že pro každý graf platí platí $\kappa(\mathbf{G}) \leq \lambda(\mathbf{G})$).

Použití algoritmu minimálního toku na modifikovaný graf (ve kterém se zdvojnásobil počet uzlů) je neefektivní a byly vypracovány efektivnější algoritmy pro zjištění stupně uzlové souvislosti. Uvedeme si nejjednodušší z těchto algoritmů — *Kleitmannův test*.

Kleitmanův test k-souvislosti grafu **G** spočívá v následujícím rekurzivním postupu:

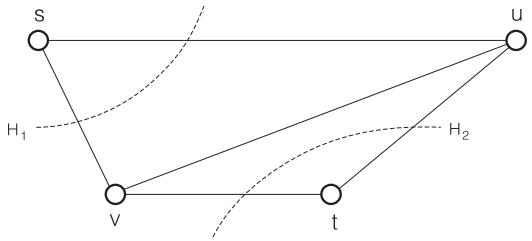
1. Vybereme libovolný uzel grafu **G** (označíme ho například u) a prověříme, že mezi tímto uzlem a každým jiným uzlem grafu existuje k uzlově disjunkních cest.
2. Pokud je $k = 1$ je testovaný graf 1-souvislý. Pokud jde $k > 1$, odebereme uzel u a jeho vstupní i výstupní hrany, a zbylý graf otestujeme na (k-1)-souvislost. Je-li takto redukovaný graf (k-1)-souvislý, pak je původní graf **G** k-souvislý.

Kleitmannův test redukuje složitost výpočtu uzlové souvislosti. Zatímco při testování k-souvislosti grafu o n uzlech algoritmem maximálního toku potřebujeme n^2 výpočtů kapacity minimálního hranového řezu, Kleitmannův algoritmus vyžaduje nejvýše kn výpočtů.

Stupeň hranové nebo uzlové souvislosti je pouze hrubým měřítkem pro posouzení provozuschopnosti přepojovací sítě. V konkrétních případech můžeme provozuschopnost jednotlivých linek a uzlů popsat jako pravděpodobnost chybového stavu (výpadku) a očekávat, že budeme schopni vyčíslit provozuschopnost sítě jako celku.

Problém vyčíslení provozuschopnosti sítě si budeme ilustrovat na jednoduchém příkladě (obr. 2.9). Máme síť se čtyřmi uzly, které budou zcela spolehlivé, linky mezi uzly mají shodnou pravděpodobnost poruchy — označíme si ji p. Pravděpodobnost, že dojde k rozpojení uzlů s a t je pro tuto síť dána výrazem

$$f = 2p^2(1-p)^3 + 10p^3(1-p)^2 + 5p^4(1-p) + p^5 \quad .$$



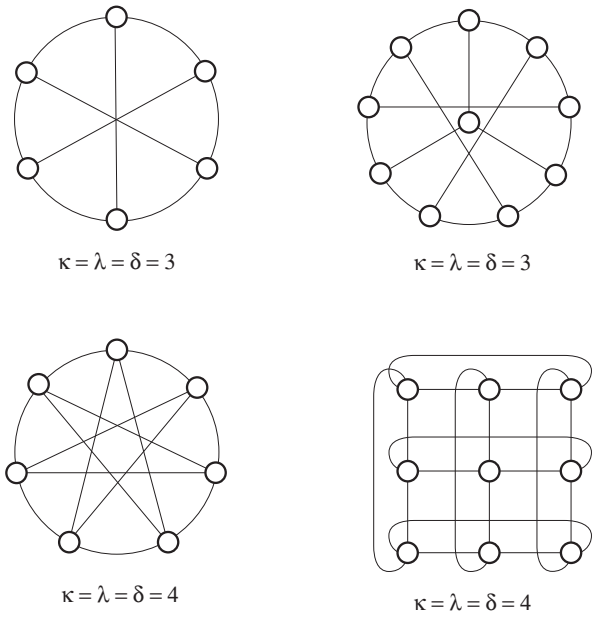
Obrázek 2.9: Vyhodnocení provozuschopnosti sítě

První člen výrazu vyjadřuje pravděpodobnost, že k rozpojení dojde při poruše dvou linek. V úvahu přicházejí pouze dvě kombinace porouchaných linek: hranové řezy vyznačené na obrázku 2.9. Další členy výrazu udávají, že k rozpojení uzlů s a t dojde při všech deseti kombinacích současných poruch tří linek, při všech pěti kombinacích současných poruch čtyř linek a při současné poruše všech pěti linek. Uvedený postup lze využít i v případě, kdy vyhodnocujeme samotné výpadky uzlů, nebo současně výpadky uzlů i linek, stačí rozložit uzly původního grafu na dvojice $\{u_{in}, u_{out}\}$.

Z našich posledních úvah je vidět, že v případech, kdy bude provozuschopnost sítě hlavním kritériem jejího hodnocení, se budeme zajímat o topologie, které maximalizují souvislost. Z Whitneyho teorému vyplývá, že ani stupeň hranové souvislosti, ani stupeň uzlové souvislosti nemůže překročit stupeň grafu, zajímavé však budou grafy, pro které je dosaženo rovnosti

$$\kappa(\mathbf{G}) = \lambda(\mathbf{G}) = \delta(\mathbf{G}) \quad .$$

Následující obrázek 2.10 uvádí příklady regulárních grafů s maximalizovanou hranovou a uzlovou souvislostí.



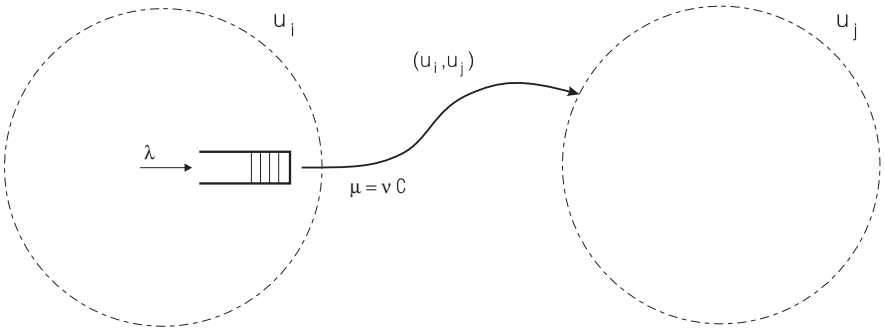
Obrázek 2.10: Příklady regulárních grafů s maximalizovanou souvislostí

2.4 Zpoždění v přepojovací síti

Přínosem přepojovacích technologií na polygonální síti je efektivní využití kapacit kanálů a zvýšení spolehlivosti spojení. Nevýhodou je nutnost postupného předávání zpráv (nebo paketů, buněk) mezi uzly, které zvyšuje dobu přenosu zprávy. Vážným problémem je vliv zátěže komunikační sítě na dobu přenosu.

2.4.1 Zpoždění zpráv v komunikačním kanále

Obsluhu komunikačního kanálu v síti s přepojováním paketů můžeme znázornit schématem (obr. 2.11).



Obrázek 2.11: Obsluha komunikačního kanálu

Požadavky na přenos zpráv kanálem **C** přicházejí s *intenzitou požadavků* λ [zpráv/sec]. Předpokládejme, že intervaly mezi příchody jednotlivých zpráv mají pravděpodobnost

$$a(t) = \lambda e^{-\lambda t} \quad .$$

Volba exponenciálního rozložení hustoty pravděpodobnosti $a(t)$ není jedinou možností, popisuje však náhodný charakter zdroje zpráv, který produkuje nové požadavky bez ohledu na historii.

Dále předpokládejme, že kapacita kanálu je C [bit/sec] a střední délka zprávy je $1/\nu$ [bit]. Délku zprávy l budeme považovat za náhodnou veličinu s hustotou pravděpodobnosti

$$b(l) = \nu e^{-\nu l} \ .$$

Pro *intenzitu obsluhy* zpráv kanálem (počet přenesených zpráv za jednotku času) pak platí

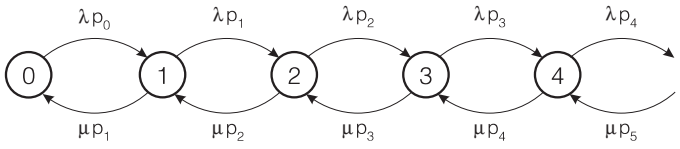
$$\mu = \nu C \ ,$$

a doba obsluhy má tedy také exponenciální rozložení

$$c(t) = \mu e^{-\mu t} \ .$$

Volba exponenciálního rozložení hustoty pravděpodobnosti *b(l)* zjednodušuje analytické řešení obsluhy kanálu. Skutečnosti by více odpovídalo rozložení deterministické, nebo obecné, získané měřením skutečné zátěže.

Náš popis obsluhy komunikačního kanálu odpovídá *systému hromadné obsluhy typu M/M/1* (exponenciální rozložení intenzity požadavků λ i intenzity obsluhy μ, jediné obslužné místo). Jeho chování můžeme modelovat automatem (obr. 2.12)



Obrázek 2.12: Markovův automat

Označení stavů Markovova automatu vyjadřuje délku fronty, přechody automatu odpovídají příchodům požadavků (prodloužení fronty) a ukončení obsluhy (zkrácení fronty). Pravděpodobnost, že automat je ve stavu k (má na vstupu frontu o délce k), udává hodnota p_k. V ustáleném stavu musí pro hodnoty p_k platit

$$\lambda p_k = \mu p_{k+1} \ \text{, a tedy} \ \ p_k = (\lambda/\mu)^k . p_0 = \rho^k . p_0 \ .$$

Poměr $\rho = \lambda/\mu$ nazýváme *intenzitou provozu*. Pro stabilitu systému M/M/1 musí být splněna podmínka $\rho < 1$. Pokud je intenzita požadavků λ větší nebo rovna intenzitě obsluhy μ, je systém nestabilní, a délka fronty na jeho vstupu roste nad všechny meze.

S využitím faktu, že systém musí být v některém ze svých stavů, tj.

$$\sum_{k=0}^{\infty} p_k = \sum_{k=0}^{\infty} \rho^k . p_0 = 1 \ ,$$

lze pro střední délku fronty odvodit

$$N = \frac{\rho}{1-\rho} = \frac{\lambda}{\mu - \lambda} = \frac{\lambda}{\nu C - \lambda} \ .$$

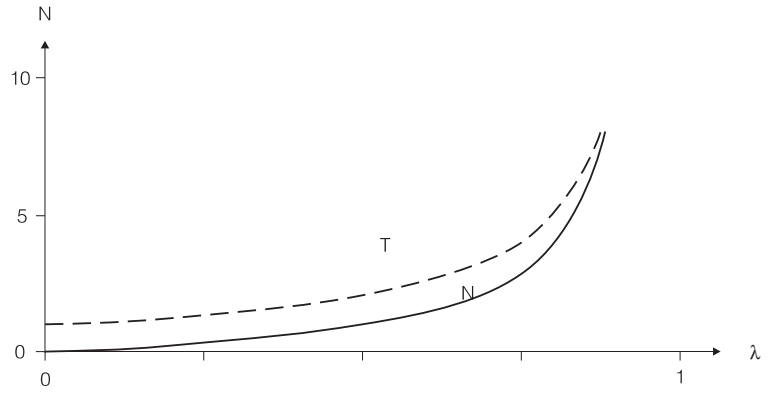
Střední dobu, za kterou bude zpráva odeslána určíme z Littleova důsledku

$$N = \lambda . T \ .$$

(Požadavek opouští po době T systém ve stavu, který odpovídá příchodu N nových požadavků.) Platí tedy

$$T = \frac{N}{T} = \frac{1}{\mu - \lambda} = \frac{1}{\nu C - \lambda} \ .$$

zpoždění kanálu Závislost střední délky fronty N středního zpoždění kanálu T na intenzitě vstupního toku λ si znázorníme graficky (obr. 2.13).



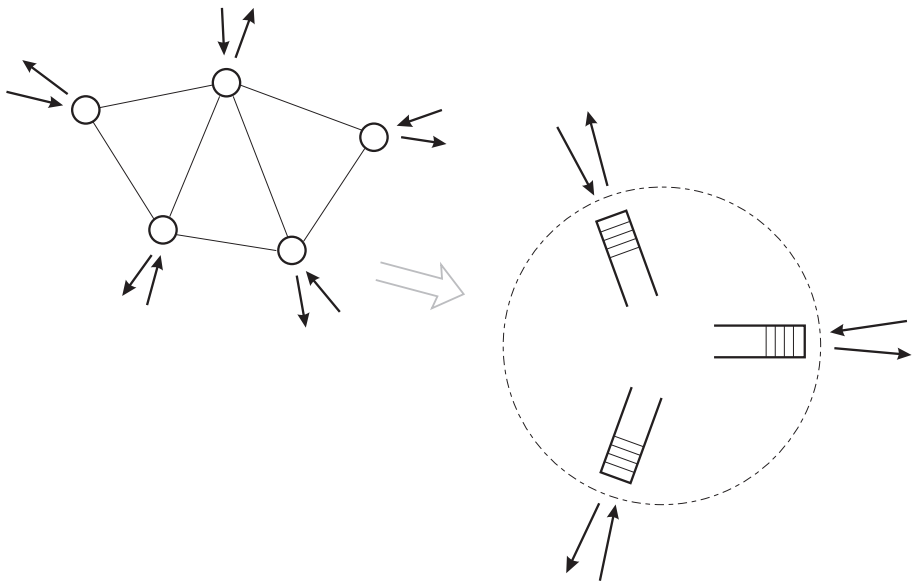
Obrázek 2.13: Závislost zpoždění kanálu na intenzitě požadavků

2.4.2 Zpoždění zpráv v přepojovací síti

V polygonální síti s přepojováním zpráv se na každou linku můžeme dívat jako na samostatný obslužný systém (obr. 2.14). Síť zprostředkuje přenos datových toků γ_{ij} mezi koncovými uzly i a j. Sečtením všech toků γ_{ij} dostaneme celkový tok přenášený sítí

$$\gamma = \sum_{i=1}^{n} \sum_{j=1}^{n} \gamma_{ij} \ ,$$

kde n je počet koncových uzlů sítě. Dílčí toky γ_{ij} i celkový tok γ budeme měřit stejně jako toky λ_k linek, počtem zpráv za sekundu.



Obrázek 2.14: Přepojovací síť jako obslužný systém

Budeme předpokládat, že všechny toky v síti mají náhodný charakter s exponenciálním rozložením hustoty pravděpodobnosti, a že takový náhodný charakter mají i délky zpráv. Tento předpoklad, tzv. *předpoklad nezávislosti* použil Kleinrock [13] a ukázal, že nevede k podstatným chybám.

Předpoklad nezávislosti dovoluje dívat se na síť obslužných systémů jako na jediný obslužný systém. Pro ten platí již uvedený Littleův důsledek

$$N = \gamma.T \ ,$$

kde N je celkový počet zpráv ve frontách všech uzlů a T je střední doba průchodu zprávy sítí.

Díky předpokladu nezávislosti můžeme považovat toky linek za exponenciální a platí pro ně

$$N_i = \frac{\lambda_i}{\nu C - \lambda i} \ ,$$

kde N_i, λ_i a C_i jsou střední délka fronty, intenzita požadavků (tok linkou) a kapacita linky. Střední dobu průchodu zprávy sítí určit jako

$$T = \frac{\sum_{i=1}^{m} N_i}{\gamma} = \sum_{i=1}^{m} \frac{\lambda_i}{\gamma}(\frac{1}{\nu C_i - \lambda_i}) \ ,$$

kde m je počet linek a hodnoty toků λ_i potřebné pro vyhodnocení výrazu získáme rozložením koncových toků γ_{jk} po cestách určených směrováním.

Poznámka
Budeme-li uvažovat konečnou rychlost šíření signálu c v linkách o délce l_i, a dobu potřebnou ke zpracování zprávy v uzlech (směrování) K_i, bude střední zpoždění zprávy

$$T = \sum_{i=1}^{m} \frac{\lambda_i}{\gamma}(\frac{1}{\nu C_i - \lambda_i} + \frac{l_i}{c} + K_i) \ .$$

Pro hodnocení vlastností konkrétní přepojovací sítě není ani zdaleka tak podstatná znalost detailního průběhu *"zatěžovací charakteristiky"* jako znalost limitních hodnot — středního zpoždění v nezatížené síti T_0 a mezního zatížení sítě γ^*.

2.5 Optimalizace přepojovacích sítí

Topologie sítě, volba kapacit linek a metoda směrování mají podstatný vliv na zpoždění sítě a její celkovou propustnost. Je proto přirozené, že má smysl se snažit pro dané zatížení sítě hledat optimální topologii, optimální kapacity spojů a optimální směrování. Nebude nás zajímat triviální řešení — přímé propojení koncových uzlů spoji s potřebnou kapacitou, ale vybudování přepojovací sítě s omezenými náklady.

2.5.1 Optimální přidělování kapacit

Máme-li zadanou topologii, toky mezi koncovými uzly a směrování (rozložení toků do linek), a pro cenu P_i linek s kapacitou C_i platí (lineární závislost ceny na kapacitě)

$$P_i = d_i C_i + e_i \ ,$$

můžeme se pokusit optimalizovat výběr kapacit jednotlivých spojů tak, aby bylo minimalizováno střední zpoždění zpráv při dodržení celkové ceny sítě $P = \sum_i P_i$.

Optimální přidělení kapacit linkám lze stanovit metodou Lagrangeových koeficientů, určením extrému funkce

$$F = \sum_{i=1}^{m} \frac{\lambda_i}{\gamma}(\frac{1}{\nu C_i - \lambda_i}) + \beta[\sum_{i=1}^{m}(d_i C_i + e_i) - P] \ .$$

S využitím faktu, že pro extrém musí platit

$$\frac{\delta F}{\delta C_i} = 0 \ ,$$

dostáváme pro optimální kapacity linek

$$C_{iopt} = \frac{\lambda_i}{\nu} + k\,\sqrt{\lambda_i/d_i}\ ,\ kde \quad k\ = \frac{P - \sum_{i-1}^{m}(\frac{\lambda_i}{\nu}d_i + e_i)}{\sum_{i=1}^{m}\sqrt{\lambda_i d_i}}\ .$$

Výraz λ_i udává minimální potřebnou kapacitu linky, výraz $k\sqrt{\lambda_i/d_i}$ udává jakou kapacitu máme přidat jednotlivým linkám na účet zbývajících finančních prostředků.

Dodržení optimálního přidělení kapacit, získaného uvedeným postupem (*CA – Capacity Assignment*), výrazně snižuje střední zpoždění zpráv, preferuje velké toky ve spojích s velkou kapacitou před menšími toky ve spojích s malou kapacitou.

2.5.2 Optimální rozložení toků

Optimální přidělení kapacit vede na volbu hodnot, které nemusí odpovídat technickým parametrům použitelných spojů. V praxi tedy musíme volit reálné kapacity, pokud možno blízké optimálním, a pokusit se o úpravu rozložení toků dat, které by takové rozložení kapacit respektovalo.

Algoritmus, který nám dovolí nalézt optimální rozložení toků v síti se zadanou topologií a kapacitami spojů (*FA - Flow Assignement*) využívá následujícího faktu: máme-li dvě rozložení vstupních toků f_1 a f_2 do spojů sítě, pak i rozložení toků, které dostaneme jejich lineární kombinací $f = \alpha f_1 + (1-\alpha)f_2$ je rozložením vstupních toků do spojů sítě.

Algoritmus odklánění toků pracuje následovně:

1. Máme rozložení toků f_1, které každé lince přiřazuje tok λ_i. Pro každou linku sítě určíme závislost celkového středního zpoždění na toku touto linkou

$$\frac{\delta T}{\delta \lambda_i} = \frac{\nu C_i}{\gamma}[\frac{1}{\nu C_i - \lambda i}]^2$$

2. Pro síť ohodnocenou závislostmi $\delta T/\delta\lambda_i$ nalezneme algoritmem nejkratší cesty rozložení toků f_2.

3. Určíme nové rozložení toků jako $f_1 = \alpha f_1 + (1-\alpha)f_2$, které přidává zátěž málo zatíženým spojům a ubírá zátěž spojům více zatíženým.

4. Dojde-li k podstatnější změně v rozložení toku, vrátíme se k bodu 1.

Určitým problémem může být, potřebujeme-li odstartovat algoritmus s některou kapacitou spoje nižší, než pro něj určuje počáteční rozložení toků f_1. V takovém případě snížíme proporcionálně všechny vstupní toky, určíme optimální rozložení toků, a vrátíme se k původním hodnotám toků.

Algoritmus se opírá o nelineární (hyperbolickou) závislost zpoždění (délky fronty) na zátěži. Přeložení toku z linky s velkou zátěží, a tedy i vyšší citlivostí $\delta T/\delta\lambda$ na zátěž, na linku s malou zátěží, a nižší citlivostí na zátěž, snižuje celkovou střední hodnotu zpoždění T sítě.

Algoritmus rozkládá jednotlivé toky do více cest. Pro praxi je jistě mnohem zajímavější optimalizace rozložení celistvých vstupních toků; problém je však podstatně složitější a výsledky FA algoritmu jako základ pro volbu směrování mohou stačit. Pokud má minimalizovaná funkce více extrémů, musíme si uvědomit, že na jedno spuštění lze zjistit jediný z nich (lokální extrém).

Rozložení toků získané FA algoritmem můžeme vzít za základ pro nové přidělení kapacit algoritmem CA. Opakováním takového postupu můžeme optimalizovat přidělení kapacit a rozložení toků pro zadanou topologii a zátěž.

2.5.3 Optimální topologie

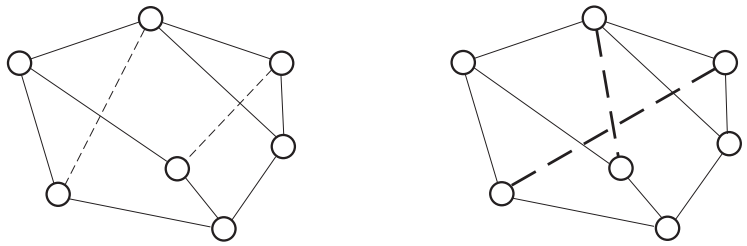
Optimalizační postupy, kterým jsme se dosud věnovali, vycházejí z předpokladu, že známe topologii sítě. V praktických situacích je to jistě často pravda, vycházíme z geografického rozložení uzlů a existujících spojů mezi nimi. Zajímavé však může být i nalezení optimální topologie pro známé toky v síti a cenové funkce spojů, pochopitelně při respektování požadavků na spolehlivost sítě (její k-souvislost).

Extrémní cesty k nalezení optimální topologie jsou: vypouštění hran úplného grafu, a záměna hran v grafu respektujícím požadavek na k-souvislost.

Prvá z metod: vypouštění hran, spočívá v postupném vypouštění hran s malou kapacitou z grafu sítě. Postup odpovídá lze popsat následujícími kroky:

1. Zvolíme úplný graf jako počáteční topologii sítě.
2. Na síť opakovaně aplikujeme optimalizační postupy CA a FA. Výsledkem je určité rozložení kapacit spojů.
3. Spoj s nejnižší kapacitou z grafu vyjmeme (pokud tomu nebrání ohled na k-souvislost sítě) a vrátíme se k bodu 2. Nelze-li žádný spoj odebrat, postup výběru topologie ukončíme.

Alternativou tohoto postupu je metoda *záměny větví*, kdy startujeme výběr topologie s grafem, který splňuje počáteční požadavky na souvislost. V takovém grafu pak nahrazujeme dvojice hran a zkoušíme, zda opakovaná aplikace optimalizačních postupů CA a FA nedá lepší výsledek (obr. 2.15).



Obrázek 2.15: Záměna větví

Konečně, i v běžné praxi lze využít informací o reálné zátěži jako podkladu pro úpravu její topologie. Postup se může opřít o nalezení hranového řezu, zahrnujícího silně zatížené linky:

1. Z grafu sítě postupně odebíráme hrany, počínaje těmi s vyšší zátěží $\lambda/\nu C$ až do rozpadu sítě na dvě komponenty.
2. Odebrané hrany se pokoušíme do grafu vrátit. Pokud vrácená hrana nepropojí obě komponenty, v grafu ji ponecháme.
3. Hrany, jejichž přidání do grafu vede na propojení obou komponent jsou prvky silně zatíženého hranového řezu.

Zvýšení průchodnosti sítě lze dosáhnout přidáním spoje, který propojí komponenty, oddělené nalezeným hranovým řezem.

3. Fyzická vrstva

3.1 Přenosové médium

Jedním z důležitých prvků, který charakterizuje konkrétní komunikační technologii, je použité přenosové médium. Kromě malého počtu sítí, které používaly paralelní přenos po vícevodičových kabelech (například sběrnice měřícího systému IEEE 488, nebo historické lokální sítě), jde u naprosté většiny sítí o přenos sériový. Nejčastěji se setkáváme s nesymetrickým (koaxiální kabel) a symetrickým (kroucený dvoudrát – twist) metalickým vedením, moderní technologie se opírají o optická vlákna a rádiové (mikrovlnné) kanály.

Symetrické vedení

Symetrické vedení ve formě krouceného dvoudrátu (twisted pair), jak ho známe z telefonních kabelů, je nejlevnějším přenosovým médiem. Pro datové přenosy s vyšší rychlostí přenosu jsou vyžívány speciální kabely, obsahující více párů. Ve většině případů jde o stíněný (STP - Shielded Twisted Pair) nebo nestíněný (UTP - Unshielded Twisted Pair) kabel, po jehož párech lze přenášet datový signál na vzdálenost 100 m (limit norem pro strukturovanou kabeláž v budovách) přenosovou rychlostí do 100 Mb/s (Ethernet 100BASE-Tx). U dostatečně kvalitních kabelů lze dosáhnout i vyšších přenosových rychlostí (ATM 155 Mb/s, Ethernet 1000BASE-T 1 Gb/s).

Symetrické vedení je používáno pro přenos kódovaných signálů v základním pásmu. Vedle využití v oblasti lokálních sítí (Ethernet, IBM Token Ring, ale i FDDI a ATM) se často setkáme s použitím v oblasti nižších přenosových rychlostí se signály odpovídajícími standardním rozhraním RS-422 EIA a RS-423 (modemová rozhraní), případně RS-485 EIA (průmyslové sběrnicové sítě).

Víceméně raritou jsou sítě, které pracují s nižší přenosovou rychlostí, v oblasti 9.6 – 115.2 kb/s. Takové sítě jsou však velice snadno realizovatelné bez speciálních komunikačních řadičů; opírají se o použití běžného sériového rozhraní podle RS-232C EIA (V.24 CCITT). Dovolují propojit osobní počítače na vzdálenost jednotek metrů (např. pro synchronizaci souborových systémů).

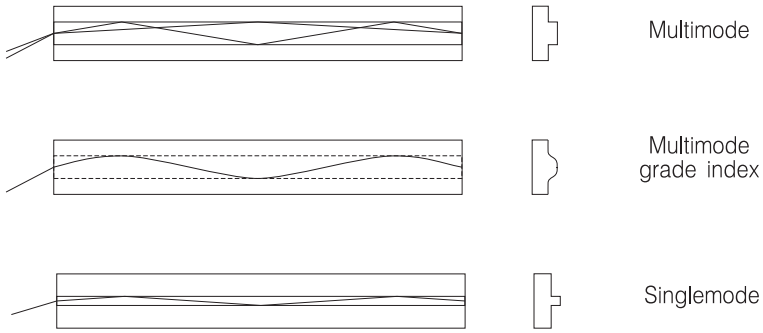
Nesymetrické vedení

Nesymetrická vedení (koaxiální kabely) dovolují využití pásma 0 – 50 MHz v základním pásmu (kódovaný datový signál) a pásma 5 – 800 MHz v přeloženém pásmu (modulovaný signál). V základním pásmu je dosahováno přenosové rychlosti v rozmezí 1 – 20 Mb/s, v přeloženém pásmu lze vytvořit skupinu přenosových kanálů s přenosovou rychlostí až 40 Mb/s. Při přenosu v základním pásmu omezují elektrické vlastnosti vedení překlenutou vzdálenost na stovky metrů, proto byly často používány drahé speciální kabely (jako tomu bylo např. u sítě Ethernet 10BASE-5). Koaxiální kabel v základním pásmu byl typickým médiem lokálních sítí, má relativně dobrou odolnost proti rušení.

Přeložené pásmo lze využít pro přenos na kilometrové vzdálenosti, podstatnou výhodou je možnost použít kabely a další prvky určené pro kabelovou televizi. O využití existujících rozvodů kabelové televize se opírají technologie HFC (Hybrid Fibre-Coax) a MCNS (Multimedia Communication Network System), efektivně zpřístupňující Internet individuálním účastníkům.

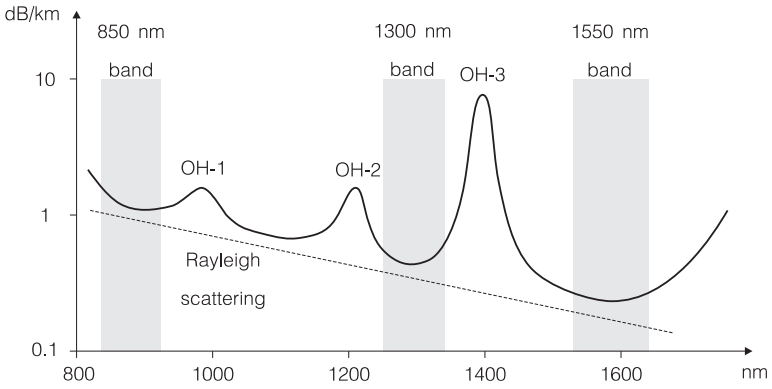
Optická vlákna

Optická (světlovodná) vlákna využívají infračervené oblasti světelného spektra (800–1550 nm) pro přenos dat rychlostmi do 100 Mb/s (vícevidová vlákna), případně do 10 Gb/s (jednovidová vlákna) na kilometrové vzdálenosti (obr. 3.1).



Obrázek 3.1: Optická vlákna

Na rozdíl od metalických spojů nelze u optických vláken využít souvislé spektrum vlnových délek. Pro přenos na větší vzdálenosti přicházejí v úvahu pásma v okolí 850 nm, 1300 nm a 1550 nm, oddělená intervaly vlnových délek se zvýšeným útlumem (obr. 3.2).



Obrázek 3.2: Útlum optických vláken

Výhodou světelných vláken je vysoká přenosová kapacita při nízké ceně média a velká odolnost proti rušení, nevýhodou je dosud vyšší cena spojování vláken a připojování konektorů. Se světlovodnými vlákny se setkáváme v lokálních sítích s kruhovou nebo stromovou topologií (FDDI, Ethernet, ATM), ale hlavně u dvoubodových spojů v telekomunikačních systémech (SDH, ATM).

Bezdrátové spoje

Alternativou k propojení prvků komunikačního systému metalickými spoji nebo optickými vlákny jsou spoje bezdrátové.

Směrové mikrovlnné spoje (v pásmech 2.4GHz – 26 GHz) dovolují propojit zařízení sítě při přímé viditelnosti na vzdálenost v rozsahu jednotek kilometrů. Jsou využívány jako rychle instalovatelná a levná varianta spojů s optickými vlákny, jejich rychlost je však podstatně nižší (v oblasti 1 – 155 Mb/s). V současnosti jsou široce využívány pro zpřístupnění Internetu koncovým uživatelům: technologie IEEE 802.11 (v pásmech ISM 2.4 GHz, 3.5 a 3.8 GHz) dnes poskytuje přenosovou rychlost v rozmezí 1 – 11 Mb/s, technologie FWA - Fixed Wireless Access (v pásmu 26 GHz) rychlost 155 Mb/s .

Pro vytváření lokálních bezdrátových sítí je široce využíváno pásmo ISM 2.4 GHz, které umožňuje vytvářet sítě pokrývající přednáškové sály, nebo skupiny kanceláří. Technologie bezdrátových lokálních sítí IEEE 802.11/802.11b pak poskytuje přenosným zařízením (notebook, PDA) sdílený přenosový kanál s kapacitou v rozmezí 1 – 11 Mb/s.

Konečně, pro omezené vzdálenosti (stovky metrů) a přímou viditelnost lze využít směrové optické spoje. Přenosové rychlosti jsou běžně do 4 Mb/s, výkonná zařízení však dovolují přenosy rychlostí až 155 Mb/s. Na velice krátké spoje: připojování zařízení – tiskáren, PDA, GSM telefonů k počítačovým systémům a mezi sebou, lze využít technologie IRDA poskytující přenosovou rychlost do 4 Mb/s.

Kapacita přenosového kanálu

Základním parametrem, který omezuje přenosovou rychlost, je šířka pásma použitého kanálu. Spojitý signál, který neobsahuje složky s vyšším kmitočtem než W, lze plně charakterizovat $2W$ vzorky za sekundu a z těchto vzorků signál opět rekonstruovat. Jinak řečeno, spojitým signálem s kmitočtovým spektrem omezeným kmitočtem W nemůžeme přenést více než $2W$ vzorků za sekundu. Může–li každý vzorek nabývat V diskrétních hodnot, pak pro přenosovou rychlost C platí *Nyquistova věta*

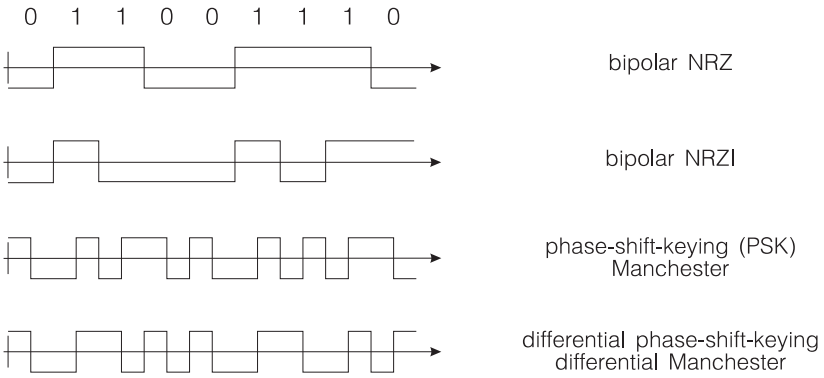
$$C = 2W.log_2(V) \qquad [b/s, Hz] \ .$$

Počet úrovní signálu v nelze s ohledem na poškození spojitého signálu při přenosu (obvykle toto poškození charakterizujeme přídavným signálem – šumem) libovolně zvyšovat; teoretický limit přenosové rychlosti C kanálu s pásmem o šířce W a odstupem signálu od šumu S/N udává *Shannonova věta*

$$C = W.log_2(1 + S/N) \qquad [b/s, Hz] \ .$$

3.2 Kódování a modulace

Neupravený datový signál (např. signál s úrovněmi TTL) není vhodný pro přímý přenos datovým kanálem. Obsahuje stejnosměrnou složku, jejíž přenos je obtížné zajistit, ať už pro elektrické vlastnosti kanálu nebo pro nutnost galvanického oddělení. Další nepříjemnou vlastností původního signálu je nezaručený výskyt napěťových změn (hran signálu), o které se lze opřít při vzorkování na straně přijímače.

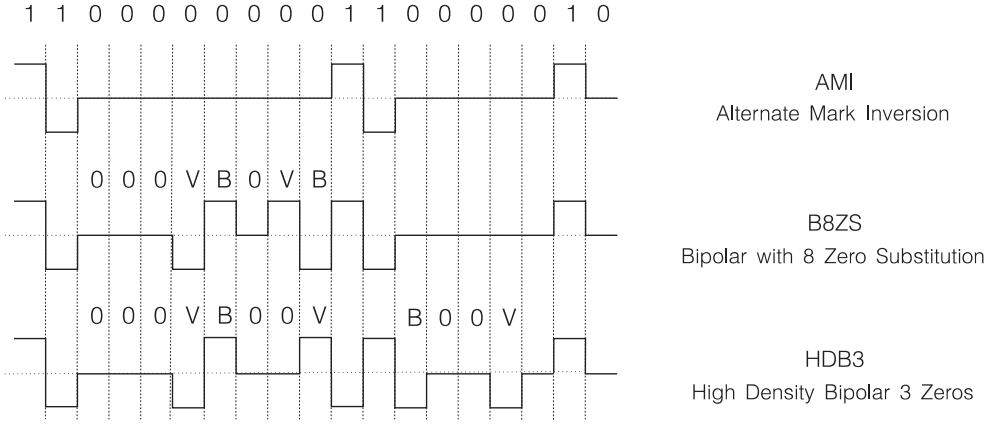


Obrázek 3.3: Jednoduchá kódování datového signálu

Datový signál můžeme zbavit stejnosměrné složky a doplnit o změny usnadňující jeho příjem vhodným kódováním. Starší metody kódování přitom berou při vytváření přenášeného signálu v úvahu jednotlivé bity (obr. 3.3).

Nejjednodušší kódování: bipolární NRZ, u kterého jsou bity zobrazeny jako dvě napěťové úrovně s opačnou polaritou, odstraňuje problémy spojené s přenosem stejnosměrné složky pouze částečně. Kódování NRZI, u kterého jsou jedničky (nebo naopak nuly) zobrazeny jako změny napěťové úrovně, odstraňuje závislost stejnosměrné složky na poměru počtu nul a jedniček v datovém signálu; množství synchronizační informace ale závisí na počtu jedniček (nebo naopak nul).

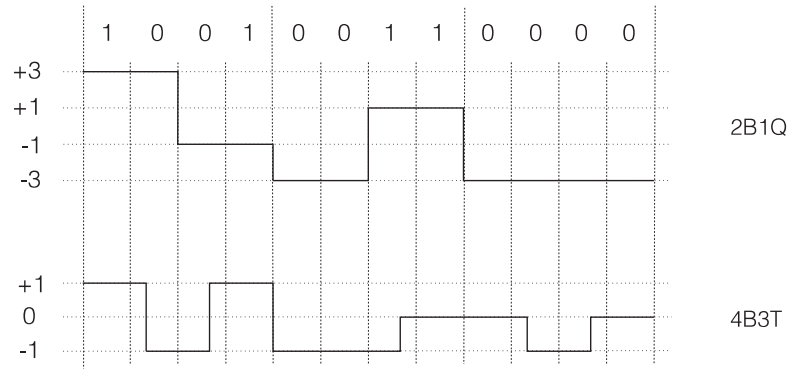
Fázovou modulaci NRZ (označovanou jako PSK nebo kód Manchester) používá lokální síť Ethernet. Diferenciální fázová modulace NRZ (označovaná také jako DPSK nebo diferenciální kód Manchester) je použita v lokální síti IBM Token Ring. Tato kódování obsahují dostatek synchronizační informace (bity je kódovány jako hrany), vyžadují však velkou šířku přenosového pásma.



Obrázek 3.4: AMI kódování

Kódování AMI (psedoternární NZR), u kterého jsou jedničky (nebo naopak nuly) zobrazeny jako pulsy se střídající se polaritou, dovoluje vložit do dlouhých intervalů bez napěťových změn pulsy, které pravidlo střídání polarity porušují. Výsledkem je zajištění dostatečného množství hran, potřebných pro synchronizaci hodin přijímače. Modifikované kódování AMI (kódy B8ZS, HDB3) je používáno v systémech ISDN a v rozhraních systémů PDH (přenosové rychlosti od 2.048 Mb/s do 34 Mb/s).

Modernější metody kódování využívají víceúrovňových signálů rozhraní pro současný přenos více bitů (obr. 3.5).



Obrázek 3.5: Víceúrovňové signály

Jako příklad víceúrovňových kódování nám slouží kódování 2B1Q a 4B3T používané v systémech ISDN a HDSL.

Konečně, v případech, kdy chceme co nejlépe využít dostupnou šířku přenosového pásma (např. u rychlých lokálních sítí), saháme po kódováních posloupností přenášených bitů. Bitové posloupnosti o dané délce převádíme na delší bitové posloupnosti, případně na posloupnosti symbolů vícehodnotových (např. ternárních), tak, aby signál získaný jednoduchým kódováním NRZ nebo NRZI těchto posloupností měl nulovou stejnosměrnou složku a obsahoval dostatek hran (obr. 3.6).

0000	11110	1000	10010
0001	01001	1001	10011
0010	10100	1010	10110
0011	10101	1011	10111
0100	01010	1100	11010
0101	01011	1101	11011
0110	01110	1110	11100
0111	01111	1111	11101

11111 - idle
11110 - start delimiter 1

. . . . . .

Obrázek 3.6: Kódování bitových posloupností

Náš příklad uvádí kódování 4B5B používané v lokálních sítích FDDI a u rychlého Ethernetu (100BASE-Tx), u spojů gigabitového Ethernetu najdeme kódování 8B10B. Víceúrovňové kódování 8B6T používá varianta rychlého Ethernetu pro méně kvalitní kabely (100BASE-T4 pro kabely UTP Cat.3).

Přenos kódovaného datového signálu označujeme jako přenos v *základním pásmu*. Pokud chceme pro přenos využít kmitočtového pásma, které neobsahuje základní harmonické přenášeného datového signálu, musíme sáhnout k modulaci. Je-li nosným signálem harmonický signál

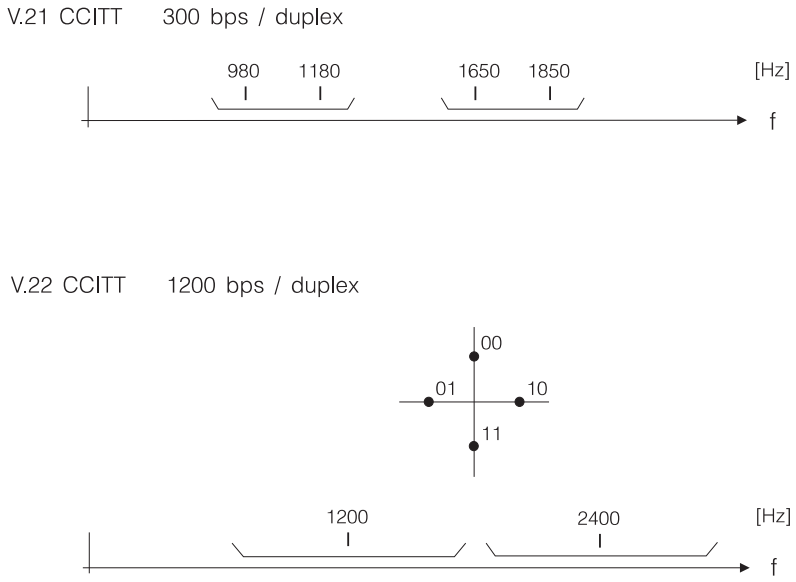
$$u(t) = U.\sin(\omega.t + \phi),$$

můžeme modulací ovlivnit jeho amplitudu U, kmitočet ω, nebo fázi ϕ. Vzhledem k malé efektivitě kmitočtové modulace se s ní setkáváme u systémů, kde nám nezáleží na efektivním využití šířky pásma. Samotná amplitudová modulace je používána téměř výlučně v optických systémech. Tam, kde se snažíme o maximální využití kapacity přenosového kanálu používáme kombinaci fázové a amplitudové modulace (obvykle ji označujeme jako QAM) Příklady jednoduchých způsobů modulace použitých u modemových spojů (kmitočtová modulace, fázová modulace) uvádí obrázek 3.7, příklady kombinované modulace amplitudové a fázové uvádí obrázek 3.7.

Kmitočtové spektrum modulovaného harmonického signálu leží v jiné kmitočtové oblasti než spektrum signálu modulačního – mluvíme o přenosu v *přeloženém pásmu*.

3.2.1 Sdílení přenosového média

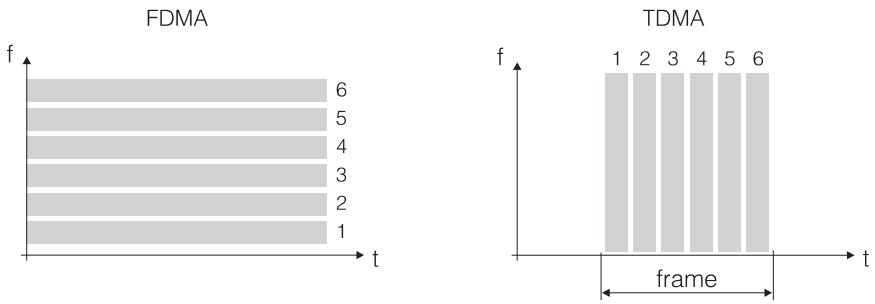
Pokud přenosové médium poskytuje větší šíři pásma (větší přenosovou rychlost) než je potřebné pro realizaci jediného přenosového kanálu, lze médium sdílet více kanály (obr. 3.8).



Obrázek 3.7: Modulace standardizovaných modemů

Kmitočtový multiplex je založen na rozdělení pásma přenášeného médiem na oddělené kmitočtové intervaly, které dále využíváme pro vytvoření samostatných přenosových kanálů. Pro převod datového signálu do daného frekvenčního pásma a zpátky používáme modemů doplněných selektivními filtry. Kmitočtový multiplex je využíván v širokopásmových lokálních sítích (přístup k Internetu po kabelových rozvodech televize CATV).

Časový multiplex využívá faktu, že spojitý signál plně využívající pásma přenášeného médiem je schopen pracovat s vyšší přenosovou rychlostí než potřebujeme pro jeden datový kanál. V takovém případě lze rychlý přenosový kanál rozdělit na více kanálů pomalých.



Obrázek 3.8: Kmitočtový a časový multiplex

Časový mutliplex je dnes snadněji realizovatelný než multiplex kmitočtový, a jeho adaptivní formy (sdílení datového kanálu takovým způsobem, aby bylo maximálně využito jeho kapacity) jsou principem převážné většiny dnešních lokálních sítí a sítí integrovaných služeb (ISDN). Moderní systémy často obě techniky kombinují, například v sítích CATV jsou využívány kanály o šířce 6 MHz pro vytvoření kanálů multiplexu časového o sumární rychlosti 30, resp. 40 Mb/s.

3.2.2 Synchronizace

Využití kanálů schopných přenášet spojité signály (elektrická vedení, světlovody) pro přenos dat předpokládá, že přijímač bude přijímaný signál vzorkovat právě v těch místech, která použil pro uložení přenášených dat vysílač. Zajistit vzájemnou synchronizaci vysílače a přijímače mají za

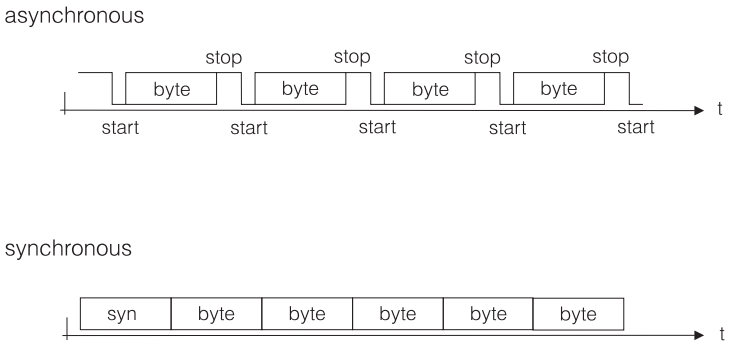
úkol metody bitové synchronizace. *Bitovou synchronizaci* lze zajistit několika způsoby. Můžeme například vedle vlastního datového signálu přenášet signál hodinový, který nám označuje místa, ve kterých máme vzorkovat.

Nebo můžeme přijímač vybavit samostatným generátorem hodin a tento generátor fázově synchronizovat na začátku například každého přenášeného znaku. Oscilátory vysílače a přijímače musí mít tak blízké kmitočty, aby se nemohla projevit odchylka ve fázi jejich signálů během příjmu jednoho znaku. Uvedenou metodu známe pod názvem arytmický nebo asynchronní přenos u terminálů a její výhodou je současné zajištění znakové synchronizace.

Nejčastějším a nejefektivnějším způsobem synchronizace vysílače a přijímače je využití změn v samotném datovém signálu, případně v signálu, který získáme vhodnou metodou kódování. Tyto změny většinou nevyužíváme přímo, ale řídíme jimi kmitočet časové základny přijímače metodou fázového závěsu. Podmínkou správné funkce fázového závěsu je dostatečný výskyt změn v přenášeném signálu, což je nutné zajistit vhodným kódováním přenášených dat.

Dalším úkolem, který musí obvody přijímače řešit, je určení začátku jednotlivých znaků a začátku zprávy v přenášené bitové posloupnosti. Mluvíme o *znakové a rámcové synchronizaci* a obvykle ji zajišťujeme porovnáváním úseku přijímané bitové posloupnosti se synchronizačním znakem nebo rámcovou značkou (křídlová značka, flag).

Na závěr si uvedeme malou praktickou poznámku: u sériového rozhraní se setkáváme se dvěma způsoby znakové synchronizace (obr. 3.9).



Obrázek 3.9: Asynchronní (arytmický) a synchronní přenos

U *asynchronního přenosu* (správněji arytmického) je každý přenášený znak doplněn o synchronizační informaci vyjádřenou počátečním prvkem (start-bit) a koncovým prvkem (stop-bit). *Synchronní přenos* se opírá pouze o hrany v přenášeném signálu. Nutností je synchronizační sekvence (preambule) na začátku bloku dovolující synchronizovat hodiny příjímače, a synchronizační znak (znak *SYN*, křídlová značka) dovolující dělit posloupnost bitů na znaky.

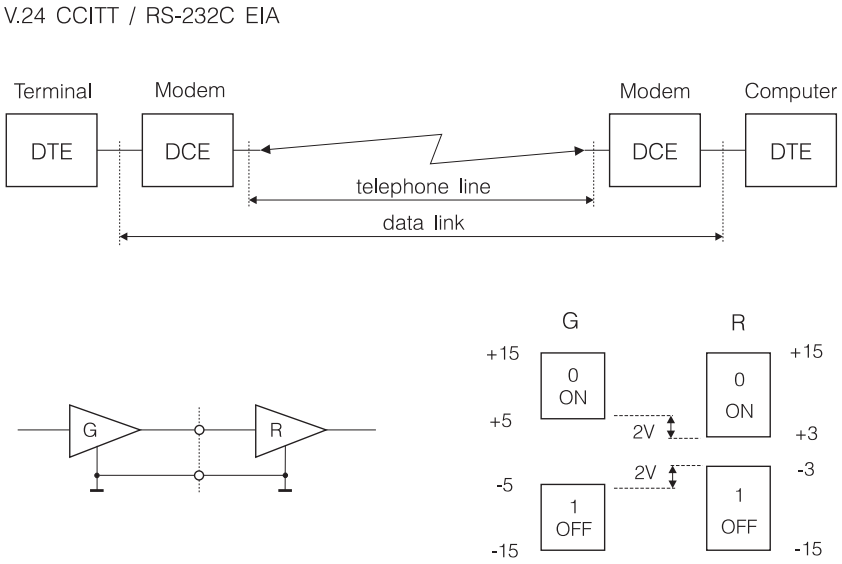
3.3 Sériová rozhraní

Sériová (modemová) rozhraní jsou využívána pro připojení pomalejších dvoubodových přenosových kanálů k zařízením sítě. Patří sem řada starších, převážně modemových rozhraní (RS-232/V.24, V.35, RS-423/V.10, RS-422/V.11), ale i jejich modernější a levnější varianty (X.21, RS-530).

Rozhraní definují mechanické, elektrické a funkční vlastnosti zařízení, která ukončují datový spoj (modemy - DCE) a zařízení, která datový spoj využívají (počítače, terminály - DTE). Rozhraní současně oddělují kompetenci a zodpovědnost provozovatele datové sítě a kompetenci a zodpovědnost koncového účastníka.

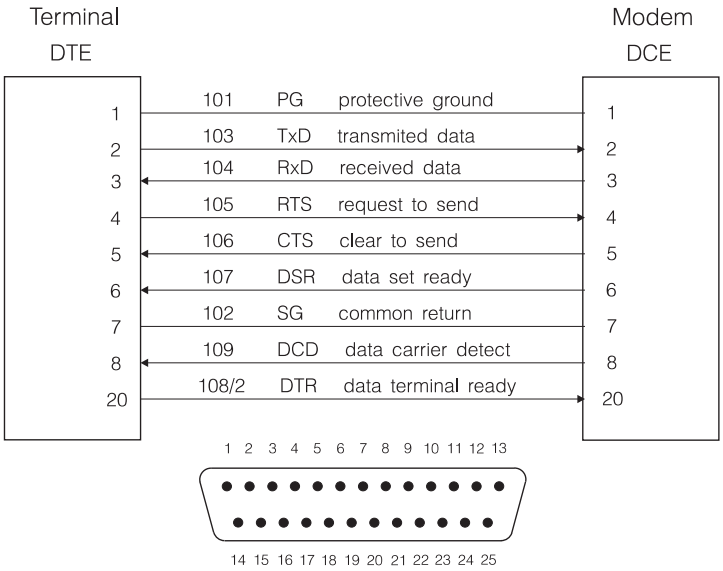
Rozhraní RS-232C EIA / V.24 CCITT

Linkové rozhraní odpovídající doporučením RS-232C EIA nebo V.24 CCITT (známé i pod alternativním označením X.21bis CCITT) je typickým modemovým rozhraním. Dovoluje přenos rychlostí do 19.2 kb/s na vzdálenost 15 m.



Obrázek 3.10: Rozhraní RS-232 / V.24

Využívá konektoru Cannon DB-25 (ISO 2110, obr. 3.11), signálové úrovně (definované doporučením V.28 CCITT) jsou vztaženy ke společnému zemnímu potenciálu (signál 102 - SG).



Obrázek 3.11: Rozhraní RS-232 / V.24

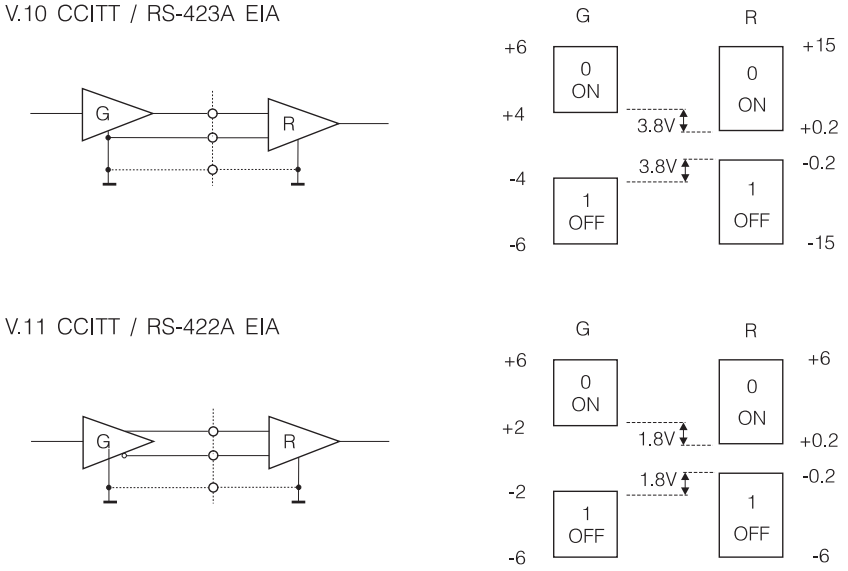
Jednotlivé obvody rozhraní (definované vlastním doporučením V.24 CCITT) přenášejí data (signály 103 - TxD a 104 - RxD), indikují připravenost propojovaných zařízení k přenosu (signály 107 - DSR a 108/2 - DTR) a správnou úroveň signálu na analogovém spoji (signál 109 - DCD). Další, na našem obrázku neuvedené obvody přenášejí hodinové signály (při synchronním přenosu) a indikují vyzvánění. Signály 104 - RTS a 105 - CTS podporují činnost obvodů sloužících synchronnímu přenosu. (Signálem 104 - RTS žádá koncové zařízení modem o vysílání synchronizačního signálu, modem po určité prodlevě dovolí signálem 105 - CTS koncovému zařízení vysílat data.)

Rozhraní RS-232C/V.24 lze využít i pro přímé propojení koncových zařízení (DTE) bez modemů, v takovém případě musí být výstupy jednoho zařízení (např. TxD) připojeny na odpovídající vstupy zařízení druhého (tedy RxD), a naopak. Takové propojení označujeme jako *nullmodem*.

Rozhraní RS-423 EIA a RS-422 EIA

Pokud potřebujeme dosáhnout vyšší přenosové rychlosti, větší překlenuté vzdálenosti a vyšší odolnosti proti rušivým signálům než poskytuje V.24 CCITT, musíme sáhnout k rozhraním, která pracují se symetrickými přijímači (resp. i vysílači) signálu, a mají podstatně nižší výstupní impedanci vysílačů.

Takovými rozhraními jsou RS-423 EIA (V.10/X.26 CCITT) a RS-422 EIA (V.11/X.27 CCITT) — (obr. 3.12).



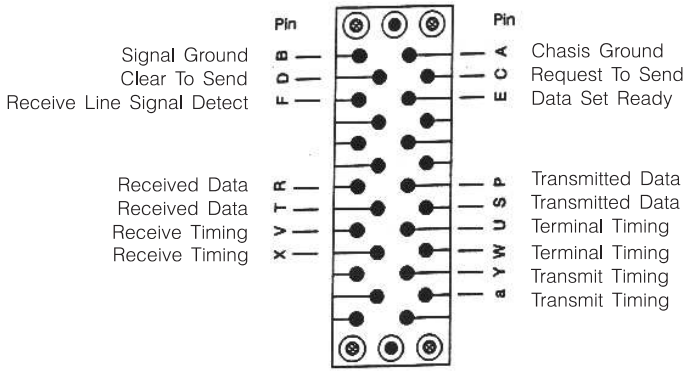
Obrázek 3.12: Rozhraní RS-423 a RS-422

Nesymetrické rozhraní RS-423 dovoluje dosáhnout přenosové rychlosti 3 kb/s na vzdálenost 1 km, resp. 300 kb/s na vzdálenost 10 m. Symetrické rozhraní RS-422 dosahuje přenosové rychlosti 100 kb/s na vzdálenost 1 km, resp. 10 Mb/s na vzdálenost 10 m.

Mechanické vlastnosti rozhraní (konektor Canon DB-37) a vlastnosti funkční (signály a jejich přiřazení špičkám konektoru) definuje doporučení RS-449 EIA.

Rozhraní V.35 CCITT

Modemové rozhraní V.35 CCITT bylo původně určené pro modemy využívající primární skupiny kanálů frekvenčního multiplexu (pásmo 60-108 kHz) a dovolující pracovat s přenosovými rychlostmi 64 resp. 56 kb/s. Používá kvalitní ale rozměrný a drahý konektor (obr. 3.13).

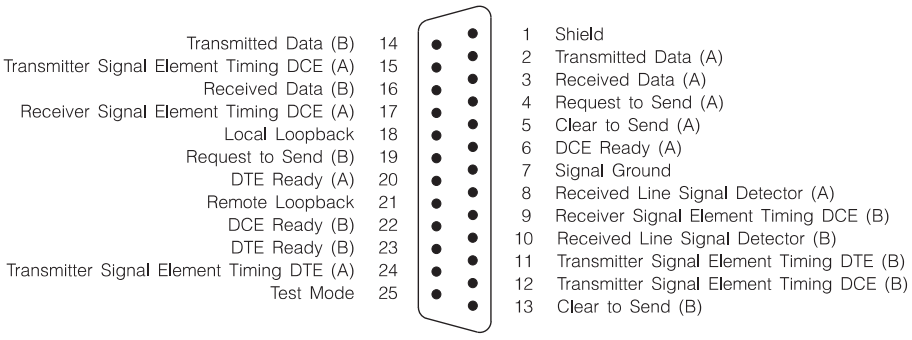


Obrázek 3.13: Rozhraní V.35

Symetrické obvody originálního rozhraní pracují se signály o amplitudě 0.55 V. Rozhraní se dnes využívá pro rychlosti až 2.048 resp. 1.544 Mb/s.

Rozhraní RS-530 EIA

Rozhraní RS-530 je moderní náhradou RS-449 (které používá konektor DB-37). Využívá, stejně jako RS-232C/V.24, konektor Canon DB-25 (obr. 3.14), a je s rozhraním RS-232C/V.24 částečně kompatibilní.



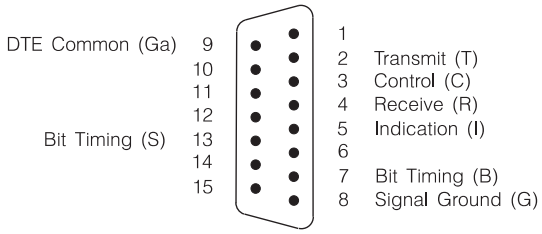
Obrázek 3.14: Rozhraní RS-530

Rozhraní dovoluje pracovat s nesymetrickými signály RS-423A/V.10 i se symetrickými signály RS-423A/V.11. Používá se pro sériový přenos ryclostí do 2.048 Mb/s.

Rozhraní X.21 CCITT

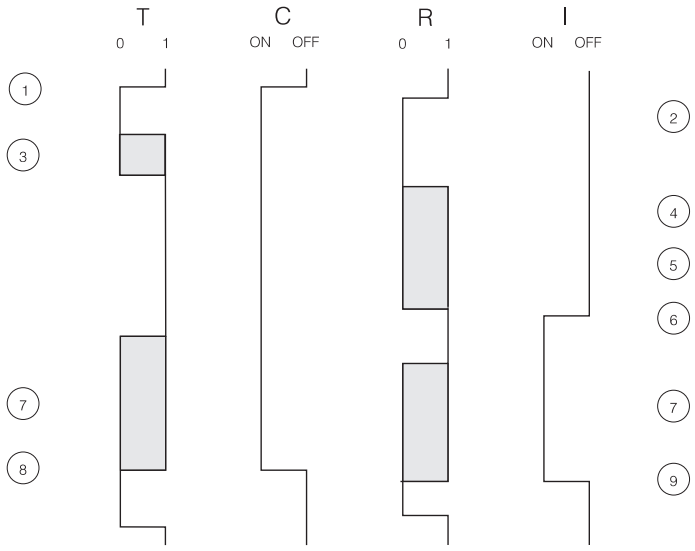
Linkové rozhraní X.21 CCITT bylo navrženo pro digitální datové sítě s přepojováním kanálů a rychlostmi nad 9600 b/s. Minimalizuje počet signálů, funkcí, pro které má rozhraní RS-232C/V.24 zvláštní signály, je dosaženo posloupnostmi na signálových obvodech C a T (ve směru od DTE) a I a R (ve směru od DCE).

Rozhraní využívá konektoru Canon DB-15 (ISO 4903) – obr. 3.15, signálové vlastnosti definují doporučení V.10/X.26 (nesymetrický vysílač) resp. V.11/X.27 (symetrický vysílač).



Obrázek 3.15: Rozhraní X.21

O navázání spojení žádá koncové zařízení "zvednutím sluchátka" - převedením signálů T a C do stavu 0 a OFF (1). — obr. 3.16. Modem odpoví "vyzváněcím tónem" - posloupností znaků "+" na obvodu R (2). To je pro koncové zařízení signál pro odeslání čísla protistanice (3). Modem odpoví informací o výsledku pokusu navázat spojení (4), a v případě úspěchu předá identifikaci protistanice (5). Přechod signálu I do stavu ON ukončuje navazování spojení (6).



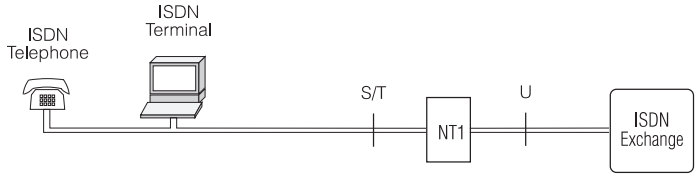
Obrázek 3.16: Navazování spojení na rozhraní X.21

Pro duplexní přenos dat využívají komunikující stanice obvody T a R (7). Spojení může ukončit kterákoliv ze stanic "zavěšením sluchátka", převedením signálu C do stavu OFF (8). Ukončení spojení je protistanici indikováno přechodem signálu I do stavu OFF.

Rozhraní ISDN

Současné přenosové sítě telekomunikačních systémů jsou dnes prakticky digitalizovány a připojení klasických koncových zařízení (analogové telefony, modemy) je zajištěno A/D převodem na rozhraní účastnického okruhu v ústředně. Zpřístupnění digitálních spojů koncovému účastníkovi, které je základní myšlenkou technologie *ISDN* (Integrated Services Digital Networks), dovoluje vedle využití celého spektra přídavných služeb digitálních telefonů využití plné kapacity digitálního hovorového signálu (64 kb/s) pro přenos dat.

Účastnické rozhraní ISDN, označované jako *BRI* (Basic Rate Interface) vytváří sběrnici (označovanou jako S/T), tvořenou dvěma symetrickými vedeními. Ta dovoluje připojit několik zařízení na vzdálenost jednotek metrů od zakončení účastnického okruhu. Zařízení jsou připojována konektory RJ45, které jsou používány i v kabeláži lokálních sítí Ethernet.



Obrázek 3.17: účastnické rozhraní BRI

Koncová zařízení ISDN mají k dispozici dva plně duplexní kanály o rychlosti 64 kb/s (lze je využít pro současný přenos dat a telefonní hovor, případně s použitím vhodného linkového protokolu, např. MPPP i pro přenos dat rychlostí 128 kb/s) a signalizační kanál o rychlosti 16 kb/s (po něm je navazováno spojení a zajišťovány přídavné služby).

Rozhraní USB

Požadavky na zjednodušené připojování nejrůznějších přídavných zařízení k osobním počítačům (včetně modemů, ale i dalších síťových zařízení) vedly k návrhu univerzálního sériového rozhraní *USB* (Universal Serial Bus).

Rozhraní dovoluje připojovat zařízení na vzdálenost do jednoho metru při přenosové rychlosti 1.5 Mb/s (USB 1.1). Čtyřvodičový kabel (s případným přídavným stíněním) přenáší signály

1Vcc - napájení +5V

2Data -

3Data +

4Gnd - zemní vodič

jedná se tedy o symetrickou sběrnici umožňující napájení připojených zařízení (se spotřebou do 100 mA). Rozhraní USB dovoluje teoreticky připojit až 127 zařízení na jeden konektor. Praktickým omezením je nejvýše 500 mA odběru. Více zařízení lze k jednomu konektoru připojit pomocí rozbočovače, fyzickou topologií rozhraní USB je tedy běžně hvězda.

Podstatnou součástí definice rozhraní USB je komunikační protokol dovolující registraci zařízení a identifikaci jednotlivého zařízení ve skupině zařízení připojených na jeden konektor.

Přenosová rychlost základního rozhraní USB (USB 1.1) nepostačuje pro připojování zařízení s většími požadavky na přenosovou rychlost. Modernizovaná verze rozhraní USB 2.0 zvyšuje přenosovou rychlost na až 480 Mb/s, při současném rozšíření funkcí a zachování slučitelnosti se zařízeními USB 1.1.

Rozhraní Ethernet

Možnost připojení osobních počítačů k lokálním sítím se stala téměř standardem, běžným vybavením osobních počítačů je rozhraní Ethernetu. Vedle jeho základní funkce, připojení osobního počítače do lokální sítě lze rozhraní využít jako univerzálního rozhraní osobního počítače s přídavnými zařízeními. V této funkci je rozhraní běžně využíváno při připojování počítačů k moderním přístupovým sítím, jako jsou sítě HFC, využívající koaxiální kabely rozvodů kabelové televize.

Rozhraní má vysokou přenosovou rychlost (10/100 Mb/s) a dovoluje propojení na vzdálenost do 100 m kvalitním UTP kabelem. Konektorem je RJ-45, pro komunikaci jsou využívány dva dvoudráty kabelu UTP připojené na vývody:

1Rx +
2Rx -
3Tx +
4Tx -

Při potřebě připojit na jeden konektor rozhraní více zařízení můžeme použít běžný opakovač pro lokální síť a standardní propojovací kabely. Přímé propojení dvou koncových zařízení vyžaduje překřížený propojovací kabel (obdoba nullmodemu).

Protokoly vyšších vrstev využívané rozhraním jsou plně standardizovány doporučeními pro lokální sítě IEEE 802.

4. Mnohonásobný přístup

Alternativou k dvoubodovým kanálům jsou kanály vícebodové. Lze je využít pro propojení přepojovacích prvků sítě, podstatně častěji jsou však využívány k vytváření sítí lokálních, které pak obvykle jako celé připojujeme k rozlehlým sítím přepojovacím. Lokální sítě se svou topologií od sítí přepojovacích podstatně liší. Opírají se o propojení komunikačních stanic sdíleným kanálem, signál vyslaný jednou ze stanic je přijímán ostatními stanicemi sítě, i bez zprostředkování dalším prvkem. Sítě jsou někdy označovány jako *broadcast* sítě. Volba topologie má vliv na řadu jejich vlastností :

-rozšiřitelnost — možnost a snadnost doplňování stanic do existující sítě,

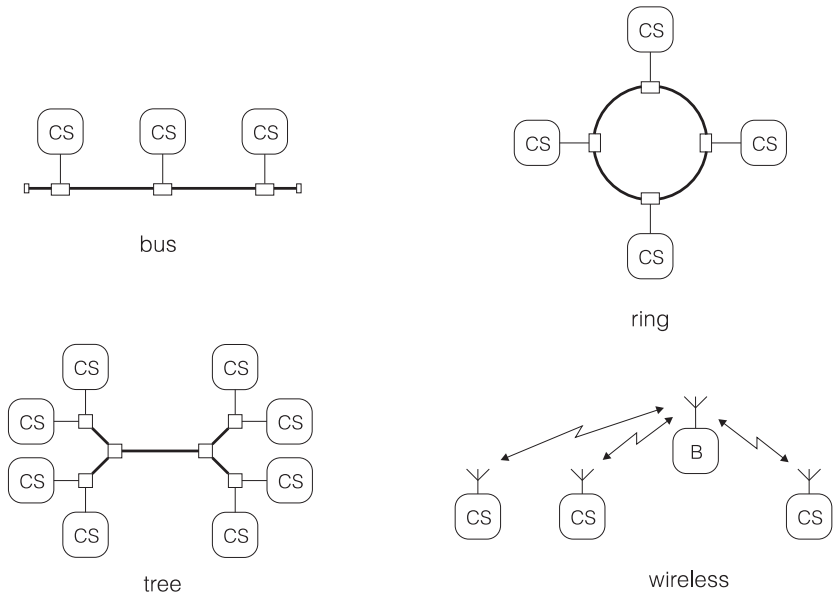
-rekonfigurovatelnost — možnost modifikovat strukturu sítě při závadě některé komponenty,

-spolehlivost — odolnost sítě proti výpadku jednotlivých komponent,

-složitost obsluhy, a

-výkonnost — využití přenosové kapacity média, zpoždění zprávy.

V praxi se setkáváme s topologií sběrnicovou, hvězdicovou, stromovou a kruhovou (obr. 4.1), některé sítě jednotlivé topologie kombinují (např. dnešní Ethernet).



Obrázek 4.1: Topologie lokálních sítí

Sběrnice

Základním prvkem sběrnicové sítě je úsek přenosového média — sběrnice, ke které jsou připojeny stanice sítě. Přenosovým médiem je nejčastěji koaxiální kabel nebo symetrické vedení (kroucený dvoudrát), realizace odboček u světlovodného kabelu je obtížná. Vlastnosti sběrnicové sítě lze shrnout do těchto bodů:

-pasivní médium,

-snadné připojování stanic k médiu, a

-odolnost proti výpadkům stanic (ale citlivost na poškození média).

Pro řízení sběrnicových sítí je využívána řada deterministických i nedeterministických metod, které využívají faktu, že signál vysílaný jednou stanicí je přijímán ostatními stanicemi jen s malým zpožděním

Hvězda a strom

Stanice sítě jsou připojeny k centrálnímu uzlu samostatnými linkami. Centrální uzel označovaný jako *hub* (v překladu "střed loukoťového kola") signál přicházející z jedné linky rozvádí do ostatních linek hvězdy. Rozlišujeme *pasivní hub* (rozbočovač), ve kterém je signál pouze dělen (odporovým děličem, optickým směšovačem), a *aktivní hub* (opakovač), ve kterém je přijatý signál zesilován tak, aby měl na výstupech požadovanou úroveň. Vlastnosti topologie "hvězda" lze shrnout takto:

-dvoubodové spoje mezi stanicemi a centrálním uzlem lze snadno realizovat,

-síť je odolná proti výpadku jednotlivých stanic a linek, a

-síť je citlivá na poruchu centrálního uzlu.

Sítě s topologií *hvězda*, jak jsme si ji právě popsali, se tím, že signál jedné stanice mohou přijímat současně stanice ostatní, blíží sítím sběrnicovým a lze u nich použít i obdobné metody řízení. Topologii *hvězda* s pasivním centrálním uzlem často nacházíme u optických sítí. Stromové (hvězdicové) sítě jsou přímým rozšířením sítí hvězdicových (dovolují více uzlů mezi stanicemi). Používají týchž metod řízení jako sítě sběrnicové a sítě *hvězda*, navíc jsou snadno rozšiřitelné. Stromovou topologii využíváme v současném Ethernetu (technologie 10BASE-T).

Kruh

U kruhových sítí jsou komunikační stanice propojeny spoji, které jsou využívány pouze jednosměrně. Signál vyslaný jednou stanicí je postupně předáván ostatními stanicemi kruhu (základní prvkem stanice je krátký posuvný registr) a po oběhu sítí se vrací ke stanici, která jej odeslala. Vlastnosti kruhových sítí lze shrnout do těchto bodů:

-dvoubodové jednosměrné spoje lze snadno realizovat i na světlovodech,

-v síti lze kombinovat různá média (pro krátké spoje elektrická vedeni, pro dlouhé světlovody), a

-síť je citlivá na výpadek libovolného prvku (stanice nebo spoje).

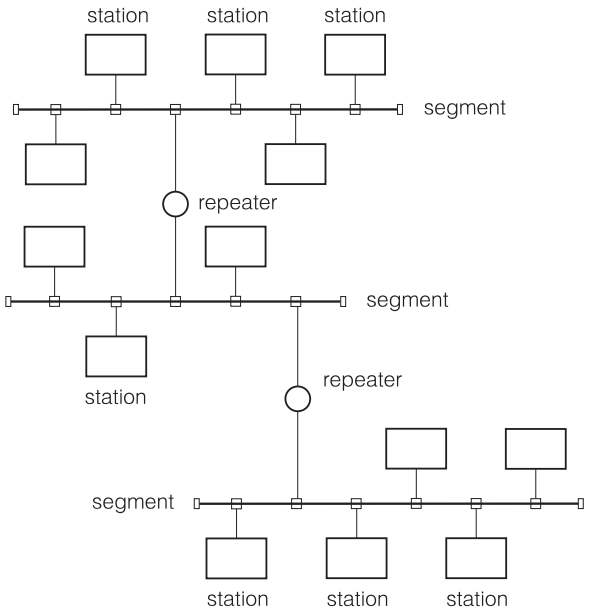
U kruhových sítí jsou pravidelně používány deterministické metody řízení.

Uvedené dělení sítí na sítě sběrnicové, stromové a kruhové se opírá o *elektrickou topologii*, tedy o způsob vzájemného propojení stanic. Z hlediska vlastností sítě má velký vliv i *topologie fyzická* (způsob vedení kabelů ovlivňuje schopnost rekonfigurace a tím celkovou spolehlivost) a *topologie logická* (metoda spolupráce stanic u deterministických metod).

4.1 Sběrnicové sítě

Sítě se sběrnicovou strukturou patří k nejčastěji používaným. Jejich výhodou je pasivní médium, ke kterému je poměrně snadné připojit řadu stanic. Určitou nevýhodou je značný vliv kvality přenosového média na parametry sítě — rychlost přenosu, délku segmentu média, počet stanic připojitelných na segment. U většiny sítí však existuje možnost segmenty sítě vzájemně propojovat opakovači; struktura sítě pak může odpovídat obr. 4.2.

Stejné nebo podobné metody řízení jako sběrnicové sítě používají i sítě s topologií hvězda a sítě stromové, jejichž určitou výhodou je použití dvoubodových, dobře přizpůsobených spojů. Bývá obvyklé, že obě řešení (sběrnice, strom) lze v jedné síti vzájemně kombinovat.

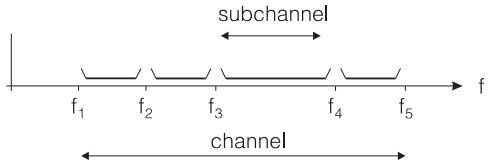


Obrázek 4.2: Topologie sběrnicových sítí

4.1.1 Statické rozdělení kapacity kanálu

Pevné rozdělení kapacity sdíleného kanálu označujeme jako statické přidělování. Zahrnuje známé metody rozdělení kapacity — frekvenční a časový multiplex.

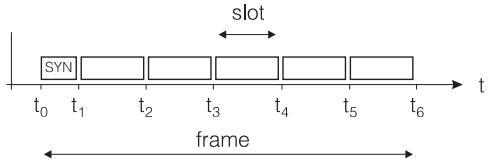
Frekvenční multiplex



Obrázek 4.3: Princip frekvenčního multiplexu

Frekvenční multiplex (FDMA — Frequency Division Multiple Access) využívá skutečnosti, že pro přenos dat s danou přenosovou rychlostí vystačíme s určitou šíří frekvenčního pásma. Je-li šíře pásma, kterou nám poskytuje přenosový kanál, větší, lze kanál rozdělit na více podkanálů a každý z nich použít nezávisle. Frekvenční multiplex je základem širokopásmových sítí.

Časový multiplex



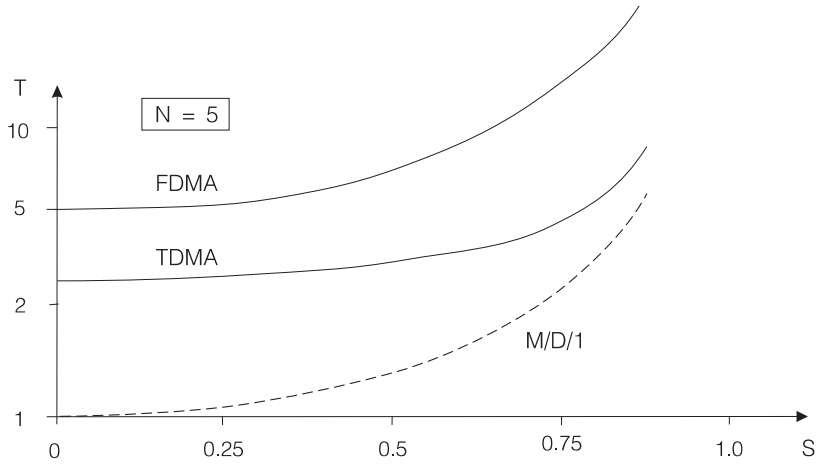
Obrázek 4.4: Princip časového multiplexu

Při časovém multiplexu (TDMA — Time Division Multiple Access) přidělujeme postupně přenosový kanál jednotlivým stanicím. Každé stanici je vyhrazen časový úsek (slot), ve kterém

může vyslat paket určité délky. Časové úseky jednotlivých stanic se pravidelně střídají s periodou, kterou obvykle označujeme jako časový rámec (frame).

Pro přenos dat zřejmě nelze plně využít kapacitu kanálu, v každém časovém slotu je nutné věnovat čas na sfázování přijímače a rámec je nutné doplnit synchronizačním slotem SYN. Metoda je použitelná pro lokální sítě s malým rozsahem ($a = t/\tau < 0.1$, kde t je doba vysílání stanice a τ doba šíření signálu přenosovým médiem).

Nevýhodou pevného rozdělení kapacity sdíleného kanálu je neschopnost přizpůsobit jeho využití nárazovému charakteru požadavků jednotlivých stanic. Optimálního využití kapacity bychom dosáhli v případě, že bychom měli k dispozici algoritmus, který by evidoval požadavky jednotlivých stanic a přiděloval podle nich stanicím médium. Takovému algoritmu se můžeme vhodnými metodami řízení pouze do určité míry přiblížit. Porovnání středního zpoždění, ke kterému dojde při přenosu sítí s frekvenčním multiplexem, sítí s časovým multiplexem a sítí s ideálním přidělováním typu M/M/1 uvádí obr. 4.5.



Obrázek 4.5: Závislost zpoždění paketu na zátěži

O metodě TDMA mluvíme jako o synchronním časovém multiplexu. Asynchronní časový multiplex (ATDMA), u kterého dokážeme ideálně přidělovat přístup ke kanálu přicházejícím požadavkům, odpovídá přidělování M/M/1.

4.1.2 Centralizované řízení

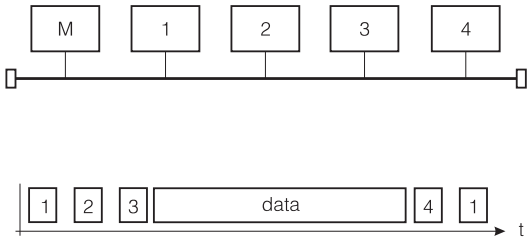
Nejjednodušší cestou, jak přizpůsobit řízení přístupu jednotlivých stanic ke kanálu náhodnému charakteru jejich požadavků při zajištění co nejmenšího zpoždění, je vyhradit jednu ze stanic jako stanici řídící. Řídící stanice přiděluje kapacitu kanálu ostatním — podřízeným stanicím. Výhoda jednoduchého algoritmu je porušena potřebou obětovat část kapacity kanálu (nebo speciální podkanál) pro žádosti podřízených stanic. Další nevýhodou je závislost sítě na spolehlivosti řídící stanice.

Přidělování na výzvu

Přidělování na výzvu je nejstarší metodou adaptivního přidělování kapacity přenosového kanálu (používají ji například linkové protokoly podle ISO 1745). Nejjednodušší modifikací metody je *cyklická výzva*.

Centrální stanice postupně vyzývá stanice podřízené. Pokud má podřízená stanice připravená data k odeslání, pak je odešle, jinak pouze potvrdí výzvu nebo neodpoví. Cyklická

výzva je výhodná pro malý počet stanic na spoji a malé zpoždění signálu (a < 0.1). Příklad přenosu dat po kanále řízeném cyklickou výzvou a vliv počtu stanic a zátěže na střední zpoždění paketu uvádí obr. 4.6.

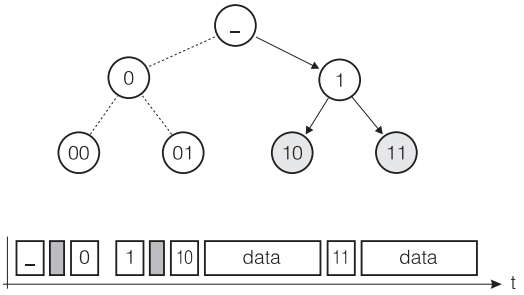


Obrázek 4.6: Řízení kanálu cyklickou výzvou

Metoda cyklické výzvy má rozumné chování při vysokém rovnoměrném využití kapacity kanálu, pro malé zatížení kanálu a velký počet stanic je zpoždění paketu zbytečně dlouhé.

Zlepšení přináší modifikace metody použitelná u speciálního typu kanálu, který dovolí stanici rozeznat, zda v daném okamžiku vysílá jedna nebo více stanic. Metoda je založena na faktu, že při malém zatížení a velkém počtu stanic lze aktivní stanici podstatně rychleji nalézt *binárním vyhledáváním.*

Pro binární vyhledávání stanice rozdělíme do dvou přibližně stejně velkých skupin, a každou skupinu dále rozdělíme do dvou přibližně stejně velkých skupin, atd., až máme v každé skupině jedinou stanici.Příklad rozdělení stanic do skupin uvádí obr. 4.7.



Obrázek 4.7: Metoda binárního vyhledávání

Řídící stanice při vyhledávání aktivní stanice postupně vyzývá skupiny stanic počínaje od kořene binárního stromu, aktivní stanice odpovídají signálem po sdíleném kanále. Pokud je ve vyzývané skupině jediná aktivní stanice, pak může zahájit přenos paketu. Pokud je aktivních stanic více, řídící stanice sestoupí ve stromu o jednu úroveň a výzvu opakuje.

Algoritmus binárního vyhledávání je rychlejší pro malé zátěže, algoritmus cyklické výzvy pro zátěže velké. Přizpůsobíme-li úroveň, od které procházíme binární strom, změřené zátěži, lze dosáhnout optimálních výsledků; metodu označujeme jako metodu *adaptivní výzvy.*

4.1.3 Distribuované řízení

Nevýhodou centralizovaného přidělování je závislost na funkci centrální stanice, výhodou (proti dále popisovaným metodám náhodného přístupu) je limitovaná doba předání paketu adresátovi. Tuto vlastnost zachovávají i deterministické metody distribuovaného řízení, které odstraňují závislost na jediné řídící stanici. Patří sem řada metod, které mají spíše teoretický charakter, praktické použití má rezervační metoda, metoda binárního vyhledávání (prioritního přístupu) a metoda logického kruhu (token passing bus).

Rezervace kanálu

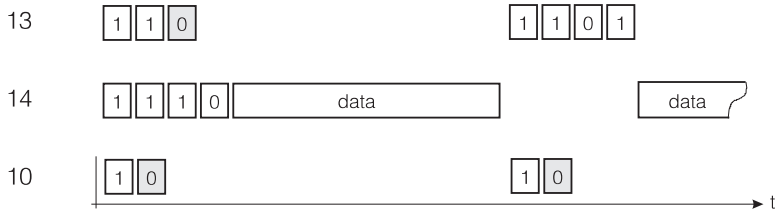
Rezervační metody jsou distribuovanou variantou přidělování kanálu na žádost. Vyčleňují z přenosového kanálu rezervační rámec, ve kterém si jednotlivé stanice rezervují přidělení kanálu datového. Rezervační rámec má charakter *bitové mapy* — každé stanici je přidělen minislot o délce odpovídající době šíření signálu médiem, ve kterém může stanice požádat o přidělení datového slotu. Datové sloty jsou přidělovány v pořadí minislotů v rezervačním rámci, algoritmus přidělování běží synchronně na všech stanicích.

Obr. 4.8 uvádí dvě modifikace rezervační metody, popsanou metodu bitové mapy a modifikaci "round-robin".

Obrázek 4.8: Rezervační metody

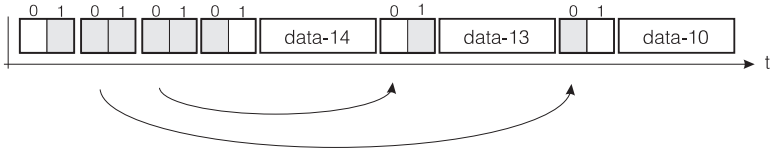
Binární vyhledávání (prioritní přístup)

Přidělíme-li jednotlivým stanicím jednoznačně binární adresy, můžeme je využít pro bezkolizní přidělování kanálu. Předpokládejme, že zprávu má připravenou k vyslání několik stanic. Stanice zahajuje svou činnost vysíláním adresy počínajíc od nejvyššího bitu a vyhodnocuje situaci na médiu. Pokud stanice zjistí na médiu bit shodný s vyslaným bitem adresy, může ve vysílání pokračovat, pokud tomu tak není musí vysílání zastavit. Po odvysílání adresy může právě jedna ze stanic pokračovat odesláním připravené zprávy a celý postup se cyklicky opakuje (obr. 4.9).



Obrázek 4.9: Prioritní přístup

Adresace stanic definuje jejich prioritu, a metoda je proto označována jako *prioritní přístup*. Pokud po odvysílání vyhledané stanice umožníme prohledat i další část binárního stromu (namísto návratu ke kořeni) dosáhneme rovnoprávnosti stanic (obr. 4.10).



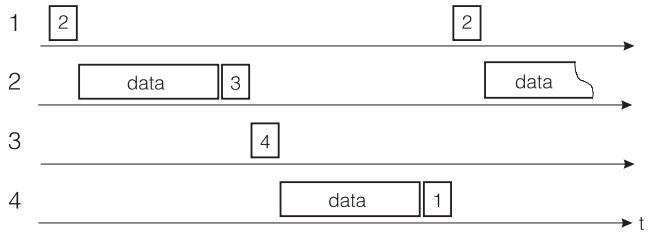
Obrázek 4.10: Distribuovaná verze binárního vyhledávání

Metoda binárního vyhledávání aktivní stanice využívá speciální schopnosti některých přenosových kanálů — realizovat funkci logického součtu nebo součinu signálů více stanic

(takovým kanálem je například sběrnice s otevřenými kolektory). V praxi se často jako signál sloužící k vyhledávání používá šum (přesněji signál s náhodně generovaným průběhem).

Logický kruh (Token Passing Bus)

Prakticky univerzálně využívanou metodou distribuovaného přidělování je metoda *logického kruhu* nebo některá její modifikace (obr. 4.11).

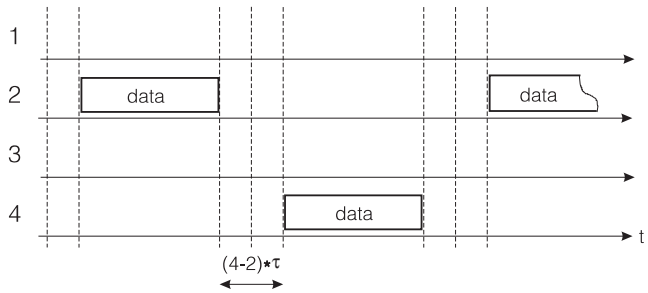


Obrázek 4.11: Logický kruh

Stanice sdílející přenosový kanál jsou označeny adresou a tyto adresy tvoří cyklickou posloupnost. Každá ze stanic zná svou vlastní adresu a adresu stanice, která smí vysílat po ní. Jedna ze stanic je vždy aktivní, v tomto stavu smí odvysílat datový paket, a/nebo předat řízení následující stanici speciálním paketem — *pověřením* (označovaným jako *token* — pešek). Určitým problémem metody je její startování a změna posloupnosti stanic pro stanice, které během provozu sítě z logického kruhu odstupují nebo se do něj naopak chtějí zapojit. Metody pro modifikaci posloupnosti stanic v těchto případech jsou označovány jako metody *rekonfigurace*.

Virtuální logický kruh

Výhodou logického kruhu je zaručení časového limitu pro doručení datového paketu, nevýhodou je však podobně jako u předchozích metod distribuovaného řízení zbytečně velké zpoždění při malé zátěži. Snížení režie způsobené předáváním pověření řadou neaktivních stanic dosahuje metoda řízení označovaná jako *virtuální logický kruh* (obr. 4.12).



Obrázek 4.12: Virtuální logický kruh

Činnost stanice ve virtuálním logickém kruhu lze popsat takto: Stanice (nechť má adresu m) sleduje provoz na médiu a dojde-li po ukončení vysílání stanice s adresou n k uvolnění média na dobu $((m-n)_{\mathbf{mod}N}).\tau$, kde N je počet stanic a τ doba šíření signálu médiem, pak stanice, pokud má zprávu k vyslání, může začít vysílat. Metoda má v oblasti malých zátěží lepší chování než metoda logického kruhu.

Metoda vyžaduje dobrou vzájemnou synchronizaci stanic, kterou je obtížné spolehlivě zajistit. Pokud uvolníme pravidla pro převzetí kanálu tak, že stanice smí zahájit vysílání do kanálu, který byl po dobu $N.\tau$ neobsazený, dostáváme kanál s možností kolize — obdobu metody

CSMA/CA (Carrier Sense Multiple Access / Collision Avoidance).

4.1.4 Náhodný přístup

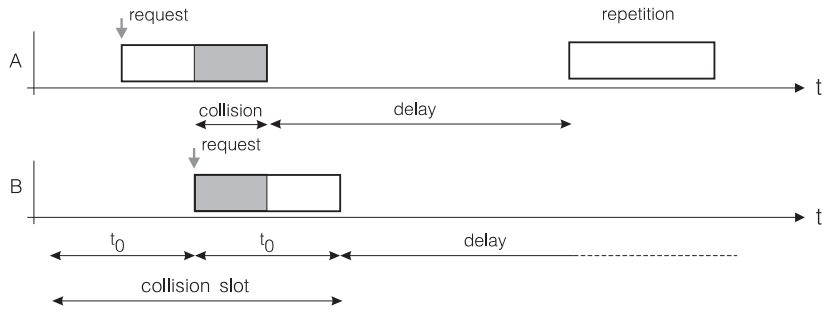
Náhodný přístup ke sdílenému přenosovému kanálu můžeme považovat za protipól deterministických metod. Jednotlivé stanice podřizují přístup na kanál pouze svému odhadu (případně pozorování).

Metody Aloha

Logickým předchůdcem metod řízení, které používají dnešní lokální sítě nasazované v administrativě, jsou metody náhodného přístupu, které byly vyvinuty pro komunikaci na sdíleném rádiovém kanále — metody označované jako metody Aloha. Nejrůznější modifikace těchto metod jsou základem přidělování komunikačního kanálu i v moderních rádiových sítích.

Prostá Aloha

Nejjednodušší metodou *náhodného přístupu* je prostá Aloha, která byla poprvé použita v roce 1970 pro rádiovou síť na Havajské universitě. Stanice, která má paket připravený k odeslání, začne vysílat bez ohledu na případné obsazení kanálu jiným přenosem. Důsledkem jsou pochopitelně kolize; situaci, ve které dochází ke kolizi ilustruje obr. 4.13.



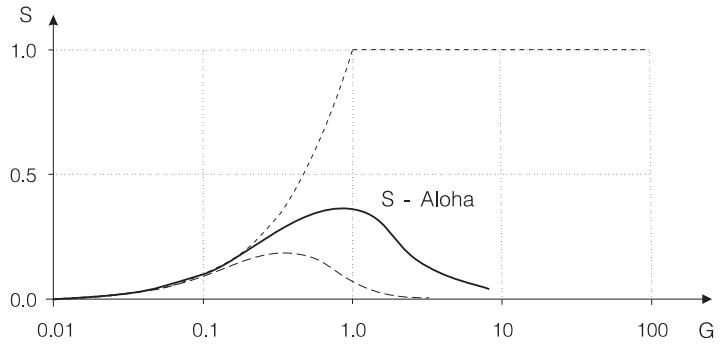
Obrázek 4.13: Prostá Aloha — kolize stanic

Pakety poškozené při kolizi je nutné po uplynutí časového limitu, do kterého měly být potvrzeny opakovat, prodleva před zahájením dalšího vysílání je volena náhodně, aby nedošlo k opakování kolize.

Budeme-li měřit vstupní tok sítě počtem paketů, které mají být přeneseny, a tento tok označíme S, je zřejmé, že v ustáleném stavu je tento tok roven toku výstupnímu (pakety přenesené sítí). V důsledku kolizí a z toho vyplývající nutnosti opakovat poškozené pakety je celkový tok vnucovaný stanicemi kanálu vyšší, označujeme ho G. Vztah obou toků, průchozího S a celkového G lze (za předpokladu, že opakující stanice nesmí generovat nový paket) vyjádřit analyticky jako

$$S = G.e^{-2G} \quad .$$

Průběh této závislosti znázorňuje obr. 4.14 a z grafu vyplývá, že i u metody prostá Aloha lze dosáhnout využití kapacity kanálu až 18.4 %, při dosažení odpovídající zátěže je každý paket v průměru třikrát vysílán. Za povšimnutí stojí pokles průchodnosti pro rostoucí celkový tok, této oblasti je nutné se vyhýbat vhodným řízením.



Obrázek 4.14: Charakteristiky metod Aloha

Taktovaná Aloha

Podstatného zvýšení průchodnosti sítě lze dosáhnout jednoduchou modifikací metody Aloha, stanicím dovolíme zahájit vysílání pouze v okamžicích, které definují začátky časových úseků postačujících pro odeslání jednoho paketu.

Důvodem zlepšení, které je patrné v grafu na obr. 4.14, je zkrácení tzv. *kolizního slotu*, jehož délka odpovídala v případě prosté Alohy dvojnásobku doby potřebné pro odeslání jednoho paketu, na polovinu.

Výhodou metod Aloha je okamžité odvysílání paketu. Překročí-li však zátěž určitou mez, zvýší se počet opakovaných paketů a silně poklesne pravděpodobnost přenosu nepoškozeného kolizí. Síť přechází do tzv. zablokovaného stavu, ze kterého se nelze bez modifikace parametrů sítě dostat. Metody, které nastavují parametr opakování v závislosti na zátěži, označujeme jako metody řízené.

Řízená Aloha

Pakety, které kolidovaly jsou u metod Aloha opakovány po náhodně volené době. Označíme-li střední hodnotu této doby $1/\alpha$ (o parametru α mluvíme jako o intenzitě opakování), lze volbou parametru α dosáhnout toho, že metoda Aloha pracuje ve výhodnější oblasti charakteristiky.

Pro volbu parametru α existuje řada heuristik, například sledování průměrného počtu opakování paketu nebo sledování podílu doby, po kterou je kanál obsazen. Nejjednodušší a velice účinnou metodou je řada postupně klesajících hodnot parametru α, které stanice postupně používá pro výpočet odkladu dalšího opakování, mluvíme o *exponenciální ustupování*.

Řízené opakování kolizí poškozených paketů má podstatný význam nejen pro metody Aloha, ale i pro metody, které uvádíme dále (metody CSMA a jejich modifikace); bez řízení nelze ani u těchto metod zajistit trvalou efektivní činnost (exponenciální ustupování je využívané např. i u Ethernetu).

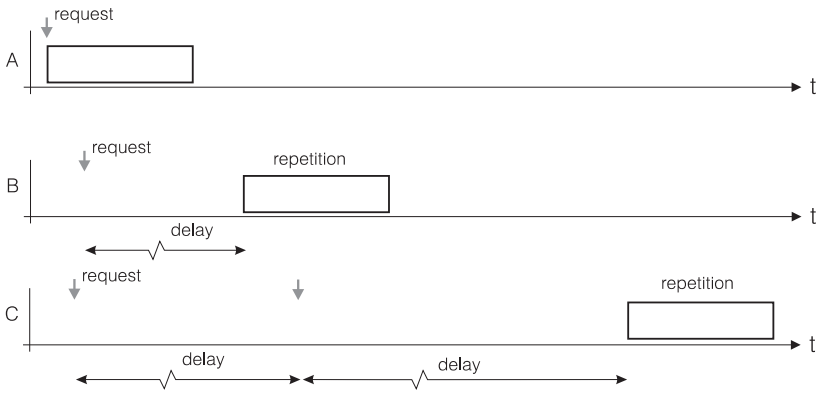
Metody CSMA

Metody Aloha byly navrženy pro rádiové sítě a nevyužívaly možnosti zjistit obsazenost přenosového kanálu před zahájením vlastního vysílání. U lokálních sítí, které se vyznačují malým zpožděním signálu a dokonalou slyšitelností stanic, je však taková informace užitečná a dovolí podstatně omezit pravděpodobnost kolize. Metody, které znalost obsazení kanálu využívají, nazýváme metodami *náhodného přístupu s příposlechem nosné*, zkráceně metodami CSMA (Carier Sense Multiple Access).

Nenaléhající CSMA

Stanice, která používá metodu *nenaléhající CSMA* (non-persistent CSMA), před odesláním paketu testuje stav kanálu. Je-li kanál volný, stanice zahájí vysílání. Pokud je kanál obsazen, stanice počká náhodně zvolený časový okamžik a znovu testuje stav kanálu. Postup opakuje do odeslání paketu.

Volbu náhodného časového okamžiku obvykle převádíme na volbu náhodného násobku taktu, který obvykle vybíráme tak, že odpovídá době průchodu signálu sběrnicí. (obr. 4.15).

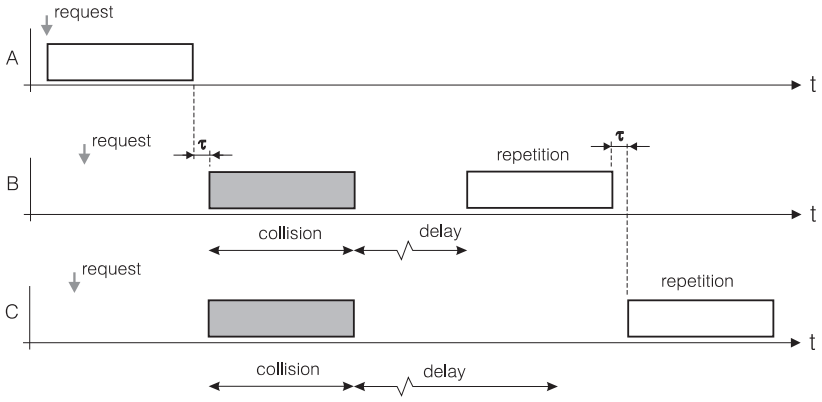


Obrázek 4.15: Nenaléhající CSMA

Naléhající CSMA

Stanice, která používá metodu *naléhající CSMA* (1-persistent CSMA), před odesláním paketu testuje stav kanálu. Je-li kanál obsazen, stanice odloží vysílání na okamžik jeho uvolnění.

Zjevnou nevýhodou této jednoduché metody je riziko kolize stanic, které čekají na uvolnění kanálu. Toto poměrně vysoké riziko se projeví nižší průchodnosti kanálu (zhruba 53 %, obr. 4.16).

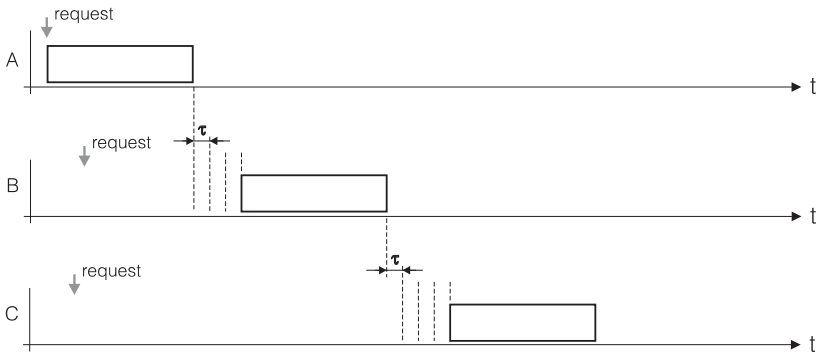


Obrázek 4.16: Naléhající CSMA

p-naléhající CSMA

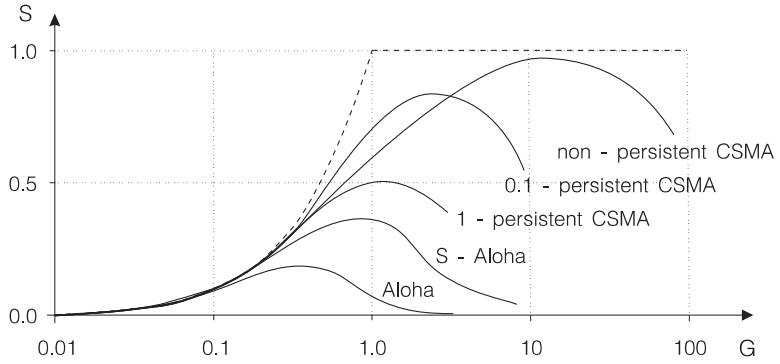
Stanice, která používá metodu *p-naléhající CSMA* (p-persistent CSMA), před odesláním paketu testuje stav kanálu. Je-li kanál volný, stanice zahájí vysílání. Pokud je kanál obsazen, stanice počká na uvolnění kanálu. Byl-li kanál volný nebo se právě uvolnil, začne stanice s pravděpodobností p vysílat a s pravděpodobností $q = 1 - p$ odloží další činnost o krátký

časový interval (může odpovídat délce šíření signálu médiem). Po uplynutí této doby celou činnost opakuje až do úspěšného odeslání paketu.



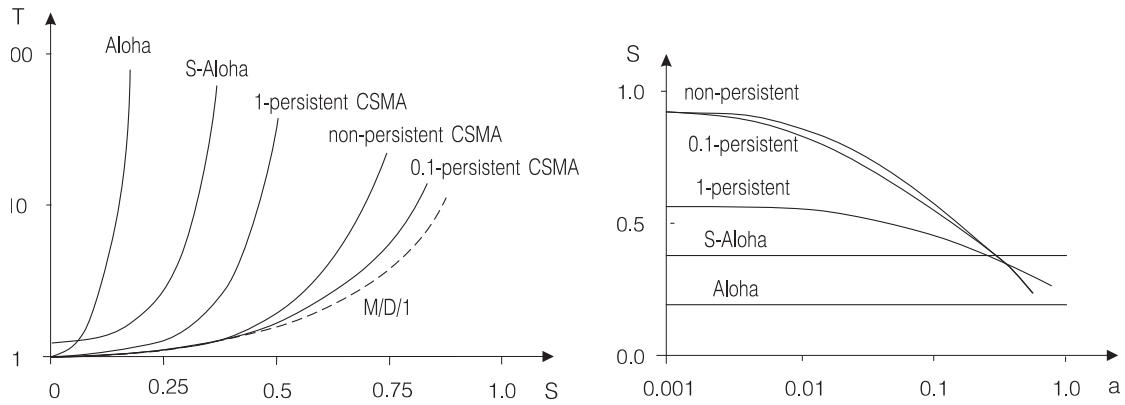
Obrázek 4.17: p-naléhající CSMA

Volba parametru p dovolí optimálně nastavit využití kanálu a střední zpoždění paketu vzhledem k zátěži. Pro $p = 1$ metoda přechází v naléhající CSMA pro $p \to 0$ se sice průchodnost kanálu blíží hodnotě $S = 1$, ale střední doba přenosu paketu roste nade všechny meze.



Obrázek 4.18: Propustnost metod CSMA

Metody CSMA samy o sobě nezajišťují stabilitu řízení. Pro udržení kanálu v pracovním stavu je stejně jako v případě metod Aloha nutné použít vhodnou metodu řízení (například měnit intenzitu opakování nebo hodnotu parametru p u p-naléhající CSMA).



Obrázek 4.19: Zpoždění u metod CSMA

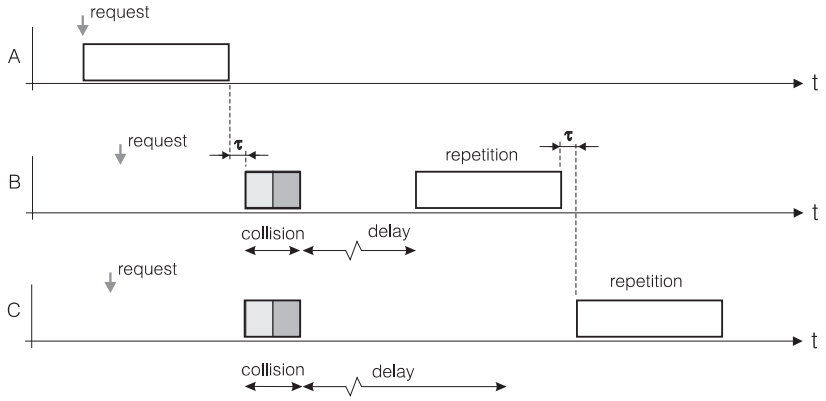
Metody CSMA jsou použitelné pouze v sítích s malým rozsahem, ve kterých se koeficient rozlehlost a pohybuje v mezích $0 < a < 0.1$. Pro rozsáhlé lokální sítě efektivita metod klesá a pro hodnoty $a \to 1$ je dokonce horší než pro metody Aloha.

U dosud popisovaných metod jsme neuvažovali potřebu potvrzování přijatých paketů (přesněji řečeno, neuvažovali jsme, že potvrzení budou muset soupeřit o přidělení kanálu). Na potvrzení se konečně můžeme dívat jako na nutnou přídavnou zátěž, která pouze v určitém poměru sníží čistou průchodnost sítě. Chceme-li tuto přídavnou zátěž eliminovat, můžeme pro potvrzení rezervovat časový interval bezprostředně navazující na vysílání paketu a zajistit, že žádná ze stanic nesmí v tomto intervalu zahájit vysílání nového datového paketu. Taková modifikace bývá označována jako *CSMA/CA* (Collission Avoidance).

Obdobou uvedené metody je modifikace metody CSMA, u které povolíme stanici s adresou m zahájit vysílání paketu nejdříve po době $((m-n)_{modN}).\tau$ po uvolnění média stanicí s adresou n (N je celkový počet stanic sítě a τ je doba šíření signálu médiem). Tato metoda je také označována jako CSMA/CA, my jsme ji poznali již dříve jako *virtuální logický kruh* (virtual token passing bus).

Metody CSMA/CD

Metody CSMA nejsou schopné zabránit kolizi, pokud je časový interval mezi zahájením vysílání dvou stanic menší než jistá mez, daná konečnou rychlostí šíření signálu v kanále, vzdáleností stanic a rychlostí reakce detekčních obvodů. Důsledkem nenulové pravděpodobnosti kolize a velké délky paketů je snížení průchodnosti kanálu (obr. 4.20).



Obrázek 4.20: Řešení kolize v metodě CSMA/CD

Některé způsoby realizace sdíleného kanálu dovolují rozpoznat, že došlo ke kolizi, ještě v době vysílání paketu, a vysílání paketu, který je kolizí poškozen, přerušit. Nejjednodušším kanálem, který detekci kolize umožňuje je sběrnice typu otevřený kolektor, v praxi však obvykle kolizi detekujeme jinak (např. sledováním napětí na médiu, které je buzeno proudovými zdroji vysílačů). Abychom zajistili, že kolizi rozpoznají všechny kolidující stanice, vyšle stanice po detekci kolize tzv. kolizní posloupnost (jam). Detekce kolize zvyšuje využití kanálu u všech metod CSMA a můžeme mluvit o naléhající, p-naléhající a nenaléhající metodě CSMA/CD.

Poznamenejme, že u metod CSMA/CD stejně jako u všech metod CSMA musíme zajistit stabilitu režimu práce.

Deterministické řešení kolize

Metoda CSMA/CD není posledním krokem v oblasti metod náhodného řízení. Dalšího zlepšení vlastností (zvýšení průchodnosti a snížení doby doručení zprávy) dosahují metody, které po zjištění kolize nejdříve zajistí přenos zpráv pro stanice, které se kolize zúčastnily, a teprve potom dovolí přistup stanic ostatních.

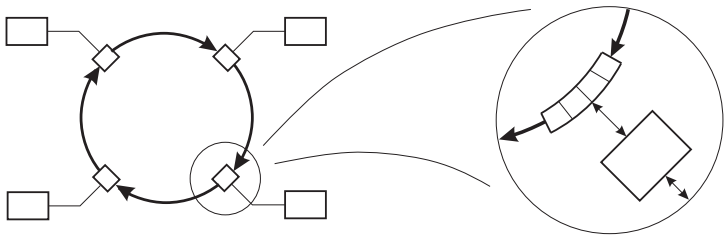
Nejjednodušší metodou řešení kolize je metoda, která se opírá o vyhledávání aktivních stanic metodou binárního stromu. Dojde-li ke kolizi v režimu CSMA/CD, stanice, které se kolize účastnily se rozdělí do dvou skupin (například podle nejvýznamnějšího bitu adresy). Stanice z prvé skupiny se pokusí o vyslání zprávy, stanice druhé skupiny počkají na ukončení přenosů stanic v prvé skupině. Dojde-li v prvé skupině opět ke kolizi, celý postup dělení skupiny se opakuje (po konečném počtu kroků je ve skupině jediná stanice).

Modifikací metody, při které rozdělíme v každém kroku soupeřící stanice na větší počet skupin, můžeme dosáhnout rychlejšího řešení kolize; krajním případem je rozdělení stanic na skupiny o jediné stanici, který připomíná deterministickou rezervaci kanálu metodou *round-robin*.

Prvý uvedený postup (binární vyhledávání aktivních stanic) je výhodnější pro malé zátěže, druhý (postupné vyhledávání) pro zátěže velké. Řada modifikací metody řešení kolize se pokouší o nalezení kompromisu mezi těmito extrémy na základě informací o okamžitém zatížení sítě.

4.2 Kruhové sítě

Kruhové sítě tvoří zvláštní kategorii lokálních sítí; kruhová síť je tvořena stanicemi, které jsou vzájemně propojené jednosměrnými dvoubodovými spoji (obr. 4.21).



Obrázek 4.21: Struktura kruhové sítě

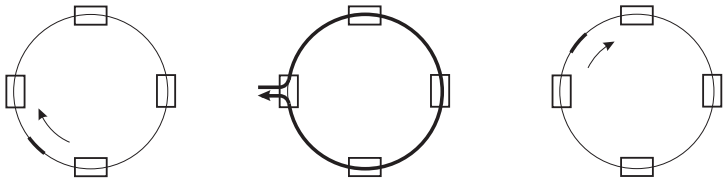
Kruhové stanice obsahují posuvný registr (o délce alespoň jednoho bitu); celou síť si lze představit jako kanál, ve kterém se signály šíří od vysílající stanice jedním směrem a po průchodu sítí se k ní opět vracejí. Doba, kterou signál k průchodu sítí potřebuje je dána počtem komunikačních stanic a délkou jejich registrů a u sítí s vysokou přenosovou rychlostí a dlouhými spoji také dobou přenosu signálu vlastním médiem.

Kruhové sítě mají mnoho výhodných vlastností. Dovolují nasazení metod distribuovaného řízení přístupu i v případech, kdy stanice jsou velmi vzdálené (desítky kilometrů). Takové metody zajišťují ohraničenou dobu zpoždění paketu a vysoké využití kapacity kanálu. Spoje lze snadno realizovat jako světlovody, síť je potom velmi odolná proti vnějšímu rušení. Jedinou vážnou nevýhodou kruhových sítí je jejich závislost na správné činnosti všech komponent, výpadek kteréhokoliv uzlu nebo spoje přeruší komunikační kanál.

Pro kruhové sítě se v praxi využívá tří základních metod řízení; podle použité metody mluvíme o sítích Newhallova typu, sítích Pierceova typu a o sítích s vkládáním rámců.

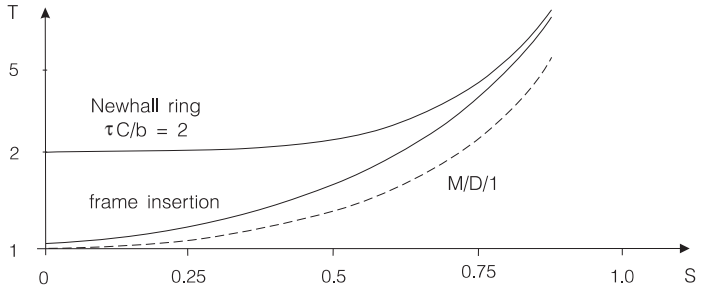
4.2.1 Newhallův kruh (Token Ring)

V síti Newhallova typu obíhá v klidovém stavu, kdy žádná stanice nepřenáší zprávu, pouze tzv. pověření (token, pešek). Stanice, která má zprávu k vyslání a získá pověření, může zprávu do kruhu vyslat. Vyslání zprávy spočívá ve změně pověření na znak (posloupnost znaků) identifikující počátek paketu a v odeslání vlastního paketu. Po odvysílání zprávy odevzdá stanice řízení vysláním pověření svému sousedu. Přenos jednoho paketu kruhovým spojem ilustruje obr. 4.22.



Obrázek 4.22: Přenos paketu Newhallovým kruhem

Zpoždění paketu v síti je pro malou zátěž dáno dobou, kterou musí paket čekat na obíhající pověření, a dobou vlastního přenosu. Doba oběhu pověření závisí na rychlosti přenosu, počtu stanic a délce jejich registrů a konečně také na délce propojovacích spojů. Pro velké zátěže se zpoždění v síti blíží ideálnímu přidělování kanálu, závislost zpoždění na počtu stanic a délce registrů uvádí obr. 4.23.



Obrázek 4.23: Zpoždění v kruhové síti

Správná funkce Newhallovy sítě je závislá na obíhání jediného pověření. Nechceme-li odstartováním sítě nebo jejím znovuspuštěním po ztrátě pověření pověřit jedinou stanici, můžeme poměrně snadno realizovat distribuovaný algoritmus.

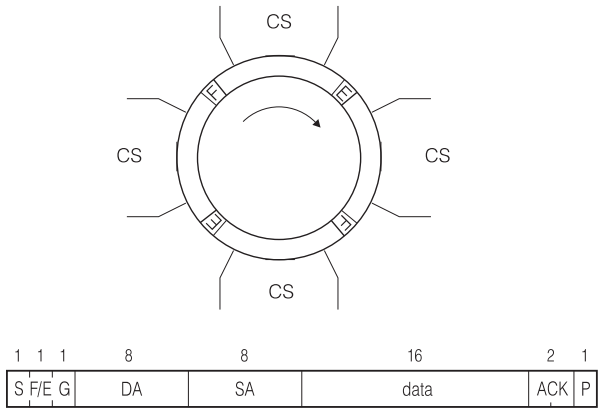
Síť Newhalova typu je pro své chování často používána jako komunikační prostředek distribuovaných řídících systémů. Na principu sítě Newhallova typu je založena metoda přístupu sítí IBM Token Ring a FDDI.

4.2.2 Pierceův kruh

Rozdělením paměťové kapacity kruhové sítě na krátké segmenty — minipakety dostáváme síť Pierceova typu (obr. 4.24).

Minipakety přenášejí jediné šestnáctibitové slovo. Obsazení minipaketu je indikováno nastavením bitu E/F, stanice, která má data k vysílání a volný minipaket ve svém registru, minipaket obsadí a po jeho oběhu sítí (a potvrzení adresátem) jej opět uvolní.

Problémem sítě je likvidace minipaketů s poškozenou adresou odesílatele, počáteční naformátování segmentů a kontrola správnosti formátu. Tyto činnosti jsou pravidelně realizovány jednou ze stanic (centrálně), tuto stanici označujeme jako řídící stanici spoje.

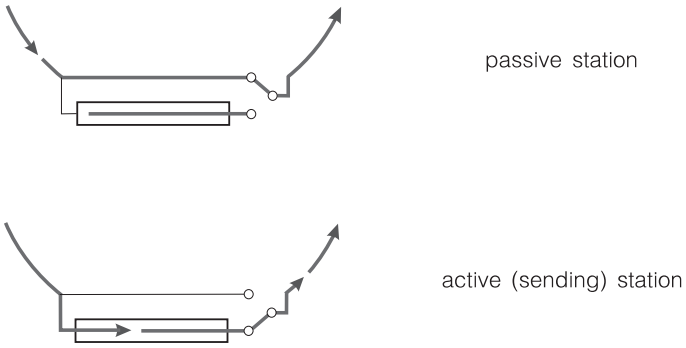


Obrázek 4.24: Kruhová síť Pierceova typu

Sítě Pierceova typu patří k nejdéle využívaným lokálním sítím a přes malé využití kapacity kanálu a špatné chování při malých zátěžích byly často používány.

4.2.3 Vkládání rámců

Poslední metodou, kterou si uvedeme, je metoda vkládání rámců. Má dobré chování v oblasti malých i velkých zátěží, její nevýhodou je však složitější technická realizace. Přenos paketu sítí ilustruje obr. 4.25.



Obrázek 4.25: Kruhová síť s vkládáním rámců

Stanice sítě, která chce vyslat paket, uloží paket do registru. Počká na konec paketu, který stanicí právě prochází a přepnutím přepínače prodlouží síť o registr. Odeslaný paket projde sítí, je převzat adresátem a vrací se do registru stanice, která registr ze sítě opět odepne.

5. Linková vrstva

Linková vrstva je nejnižší vrstvou síťových prostředků postavenou nad prostředky fyzickými (tedy nad datovým spojem a komunikačními řadiči). Jejím úkolem je chránit přenášená data před chybami v kanále a zajistit přitom dostatečnou efektivitu. Opírá se o přenos dat v rámcích zabezpečených vhodným detekčním nebo korekčním kódováním mezi dvojicí *protokolových stanic*. Úkolem protokolových stanic je podle pravidel definovaných *linkovým protokolem* reagovat na situace, které při přenosu dat nastávají (správně i s chybami přijaté rámce, ztracené rámce). Chování protokolových stanic odpovídající linkovému protokolu obvykle popisujeme procedurami řízení.

Prvým problémem, který je nutné v linkové vrstvě řešit, je rozlišení jednotlivých rámců (případně znaků) v bitové posloupnosti - mluvíme o *rámcové* (případně bitové) *synchronizaci*. Začátek rámce indikuje vysílající strana specializovanou bitovou posloupnosti - *křídlovou značkou*, případně *synchronizačním znakem*. Podobným způsobem může být ošetřen i konec rámce (tedy křídlovou značkou, nebo speciálním znakem, označujícím konec rámce). Oddělovací prvky často plní obě funkce současně - oddělují od sebe datové rámce.

Výskyt synchronizačních posloupností bitů v datech, která přenášíme, nesmí funkci rámcové synchronizace narušit. Dosáhnout tohoto stavu, *transparence dat*, lze úpravou rámce na straně vysílače a následnou inverzní úpravou na straně přijímače. Transparence dat může být zajištěna technicky nebo programově.

Dalším problémem, který je důsledkem chybovosti datového spoje, je zajištění bezchybovosti přenášených dat. Doplnění přenášených dat v rámci dostatečně silným *samoopravným kódem* vede na vysokou redundanci a tu si většinou s ohledem na omezenou kapacitu přenosového kanálu nemůžeme dovolit. (Výjimkou jsou kanály s vysokou chybovostí a kanály s velkým přenosovým zpožděním - většinou kanály rádiové.) Zbývá tedy zajištění bezpečného přenosu *detekčním kódem a zpětnou vazbou*, zde však musíme přenášená data kromě bezpečnostního kódu doplnit o informaci dovolující vzájemnou spolupráci vysílající a přijímající stanice. Schémata spolupráce, která dovolí detekované chyby opravit opakováním přenosu označujeme jako *potvrzovací schémata*.

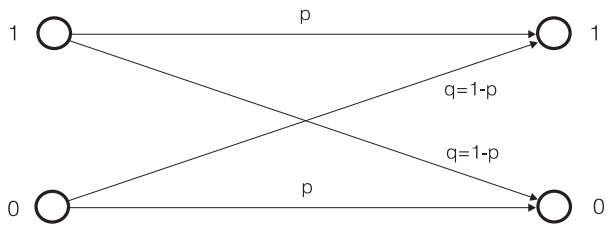
Potvrzovací schémata se opírají o vzájemnou synchronizaci komunikujících stran, využití pro potřeby vyšších vrstev síťové architektury (například abychom zabránili vyčerpání paměti na straně přijímače) je označováno jako *řízení toku* (*flow control*), a je typické zvláště pro protokoly transportní vrstvy (příkladem je TCP).

Potřeba oddělit přidané řídící informace od přenášených dat vede na definici *formátu rámců*. Formáty rámců jsou jednou částí definice *protokolu linkové vrstvy*, druhou částí definice je postup, kterým si vzájemně komunikující stanice rámce vyměňují. Tento postup obvykle nazýváme *linkovou procedurou*, nebo *procedurou řízení linkové vrstvy*.

5.1 Chyby v přenosovém kanále

Chybovost přenosového kanálu je dána chybami ukončujících zařízení, ale hlavně chybovostí vlastního přenosu. Chybovost přenosu je důsledkem poškození signálu na médiu a má řadu příčin: kmitočtovou a fázovou charakteristiku přenosového kanálu, nelineární zkreslení, vliv tepelného šumu, impulsní rušení, přeslechy z jiných přenosových kanálů, odrazy na nepřizpůsobeném vedení, úniky signálu na rádiových spojích, atd.. Vliv jednotlivých zdrojů chyb se značně liší, a při vážnější práci je musíme respektovat.

Vliv bílého šumu na signál přenášený kanálem lze dostatečně realisticky popsat jednoduchým modelem - *symetrickým binárním kanálem bez paměti* (obr. 5.1).



Obrázek 5.1: Symetrický binární kanál bez paměti

Symetrický binární kanál přenáší vstupující bit signálu (nulu nebo jedničku) bezchybně s pravděpodobností p. Doplňková pravděpodobnost $q = 1 - p$ je pravděpodobností chyby.

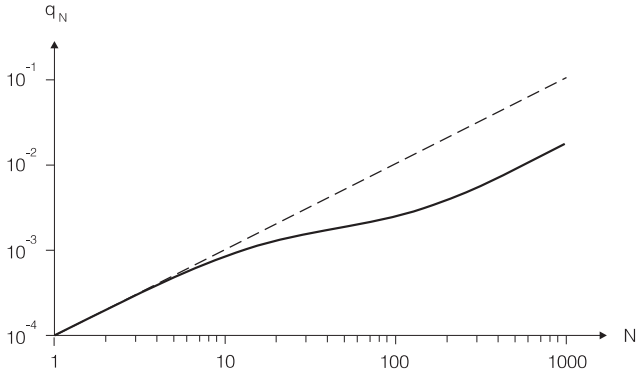
Předpokládejme, že bílý šum je hlavním zdrojem chybovosti. Ta se např. u telefonních linek použitých pro přenos dat pohybuje v mezích 10^{-5} (jakostní pevné spoje) až 10^{-3} (velmi nekvalitní komutovaná linka). Za předpokladu statistické nezávislostosti jednotlivých bitových chyb bude pravděpodobnost správného přenosu rámce o délce N bitů dána výrazem

$$p_N \;=\; p^N \;=\; (1-q)^N \;\;.$$

Použijeme-li například nekvalitní telefonní linku ($q = 10^{-3}$) pro přenos rámce o délce 32 znaků (256 bitů), bude pravděpodobnost jeho přenosu bez chyby pouze

$$p_{256} \;=\; (1-10^{-3})^{256} \;=\; 0.774 \;\;.$$

Pro reálné přenosové kanály naštěstí náš předpoklad, že chybovost má na svědomí bílý šum neplatí. Převažující vliv mají přeslechy, impulsní rušení a přenosové charakteristiky, vliv těchto faktorů je podstatně odlišný.



Obrázek 5.2: Závislost četnosti chyb na délce rámce

Zatímco bílý šum poškozuje jednotlivé bity, např. impulsní rušení poškodí celé posloupnosti bitů. Díky tomu, že se chyby vyskytují ve shlucích, je pravděpodobnost bezchybového přenosu rámců vyšší. Např., obrázek 5.2 uvádí závislost šetnosti chyb na délce bloku pro komutovaný telefonní kanál a přenosovou rychlost 1200 b/s.

Ochrana proti chybám se opírá o detekční (a výjimečně samoopravné) kódy. Byly a jsou využívány paritní kódy, iterační kódy a kódy cyklické. Volba kódu by měla odpovídat charakteru chyb převažujících v daném kanále, např. zajištění pouhou paritou je u přenosů málo účinné. Pro přenosový kanál z obr. 5.2 je navíc vysoká pravděpodobnost toho, že se po jedné chybě objeví další chyba v následujících bitech

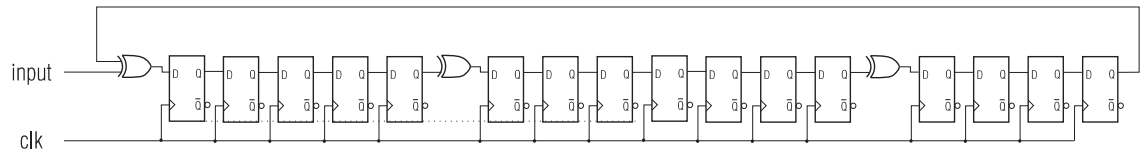
Použijeme-li proto pro zajištění osmibitových znaků přenášených takovým kanálem paritní kontrolu (neindikuje sudý počet chybných bitů) zjistíme, že paritní kontrola nezjistí chybu u 38 % poškozených znaků. Použití samotné parity je pro přenosové kanály nepostačující, samotná parita není schopna detekovat shluky chyb, pro datové kanály typické. Lepší vlastnosti mají *iterační kódy* (např. kombinace podélné a příčné parity) a zvláště kódy cyklické (CRC - Cyclic Redundancy Code), které jsou schopné zjistit

chyby v jednom bitu	s pravděpodobností	1.0
chyby ve dvou bitech	- " -	1.0
chyby v lichém počtu bitů	- " -	1.0
shluk chyb délky menší než $(r+1)$ bitů	- " -	1.0
shluk chyb délky $(r+1)$ bitů	- " -	$1.0 - 0.5^{r-1}$
shluk chyb délky větší než $(r+1)$ bitů	- " -	$1.0 - 0.5^{r}$

Parametr r udává stupeň generujícího polynomu a délku zajišťující posloupnosti.

Výhodou cyklických kódů je jejich snadná technická implementace, obr. 5.3 uvádí schéma generátoru pro polynom CRC-CCITT definovaný generujícím polynomem

$$g(x) = x^{16} + x^{12} + x^5 + 1 \ .$$



Obrázek 5.3: Generátor CRC-CCITT

Ani sebelepší zabezpečovací kódy nejsou schopné neindikovanou chybu zcela vyloučit. Pro rozumné přenosové kanály (s chybovostí $q < 10^{-4}$) je však pravděpodobnost neindikované chyby prakticky zanedbatelná ($10^{-8} a 10^{-9}$).

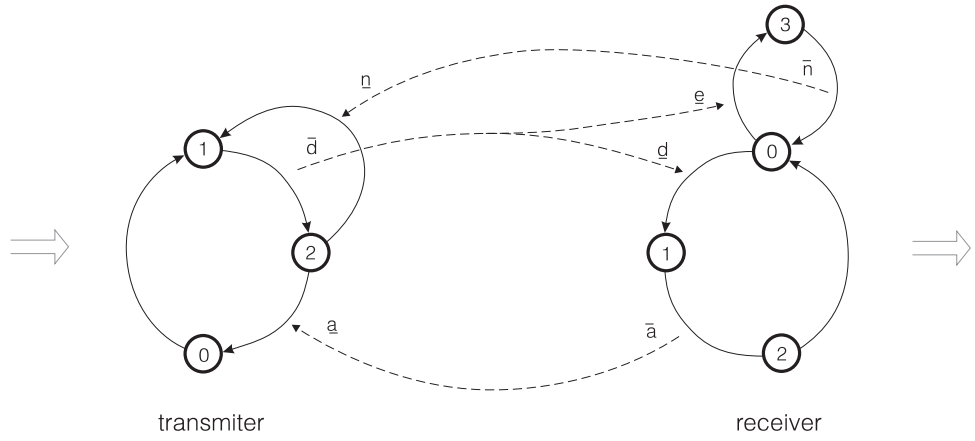
5.2 Modely potvrzovacích mechanismů

Úkolem procedury řízení (datového spoje) zajistit bezpečný přenos datových rámců při maximálním využití kapacity přenosového kanálu; kanál má navíc dopravní zpoždění. Tohoto cíle obvykle nelze dosáhnout nejjednoduššími prostředky. Popis komplikovaných procedur řízení volným textem není příliš přehledný a dostatečně přesný. Přehledněji a přesněji lze chování protokolových stanic popsat automatovými modely. Mezi nejčastěji používané patří *komunikující* (nebo spolupracující) *automaty* a *Petriho sítě*.

5.2.1 Komunikující automaty

Protokolová stanice linkové vrstvy přebírá od nadřazené síťové vrstvy bloky dat - *pakety*. Doplňuje je o zabezpečovací informace a služební údaje a takto vytvořené *rámce* odesílá protistanici. Ta při jejich příjmu zjišťuje, zda nebyla detekována chyba, potvrzuje je v rámcích odesílaných opačným směrem a obsažená data (*pakety*) předává nadřazené síťové vrstvě.

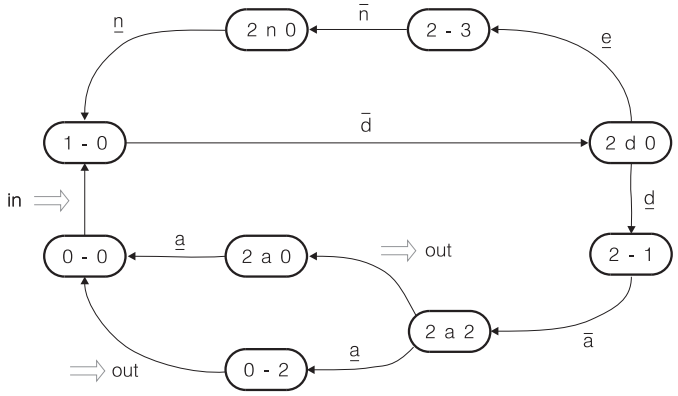
Chování protokolové stanice můžeme popsat automatem, který akceptuje podněty (vstupy) jednak z nadřazené vrstvy (pakety k odeslání) a jednak od protistanice (datové a služební rámce). Při jednotlivých přechodech pak generuje výstupy, kterými jsou rámce určené protistanici a pakety odevzdávané nadřízené vrstvě. Procedura řízení jako celek může být popsána *komunikujícími automaty* propojenými navzájem tak, že některé výstupy jednoho z nich jsou vstupy druhého a opačně. Popis jednoduché procedury řízení dvojicí komunikujících automatů si budeme ilustrovat na následujícím obrázku (obr. 5.4).



Obrázek 5.4: Komunikující automaty

Pro jednoduchost předpokládáme jednosměrný přenos dat a jednoduché potvrzování (bez ztrát rámců, ale s detekcí chyb). Vysílající stanice na převzetí bloku dat z nadřízené vrstvy reaguje odesláním datového rámce $\overline{d}$, který je převzat přijímající stanicí jako $\underline{d}$ (bez chyby) nebo $\underline{e}$ (s chybou). (Vodorovná čára nad symbolem vyznačuje odeslání rámce, podtržení jeho příjem.) Přijímající stanice na příjem rámce reaguje vysláním potvrzení $\overline{a}$ (kladného) nebo $\overline{a}$ (záporného) a je schopna předat data nadřízené vrstvě. Vysílající stanice podle přijatého potvrzení buď datový rámec zopakuje (přijato $\underline{n}$) nebo převezme další blok dat k vyslání.

Pro vyšetření chování spolupracujících automatů (zvláště z hlediska jejich odolnosti proti takovým situacím jakou je zablokování) je obvykle potřeba analyzovat celý *stavový prostor*. Pokud do něj zahrneme i přenosový kanál (které může obsahovat rámec $\underline{d}$ a jeho potvrzení $\underline{a}$ nebo $\underline{n}$) bude tento stavový prostor odpovídat obr. 5.5.

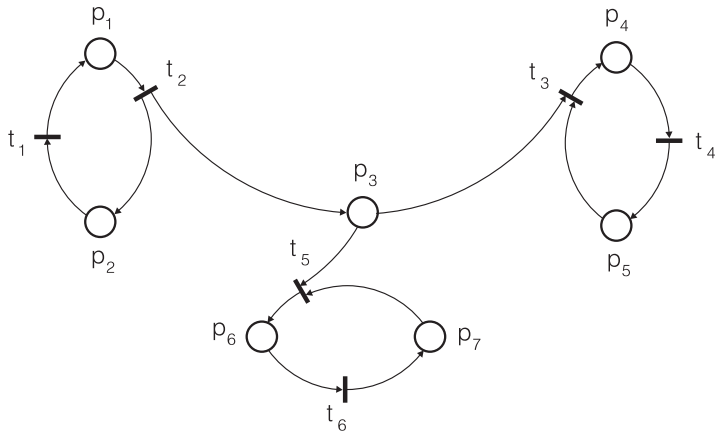


Obrázek 5.5: Stavový prostor komunikujících automatů

Analýzou stavového prostoru jsme schopni zjistit reakci na nejrůznější očekávané situace a prověřit, zda během spolupráce protokolových stanic nedochází k nepřijatelným situacím (zablokování, duplikace nebo ztrácení dat).

5.2.2 Petriho sítě

Dalším prostředkem používaným pro popis linkových protokolů jsou Petriho sítě, které dovolí popsat spolupráci protokolových stanic přehledněji než spolupracující automaty. Pro naši potřebu si uvedeme přehled základních pojmů a příklad popisu jednoduchého komunikačního protokolu.

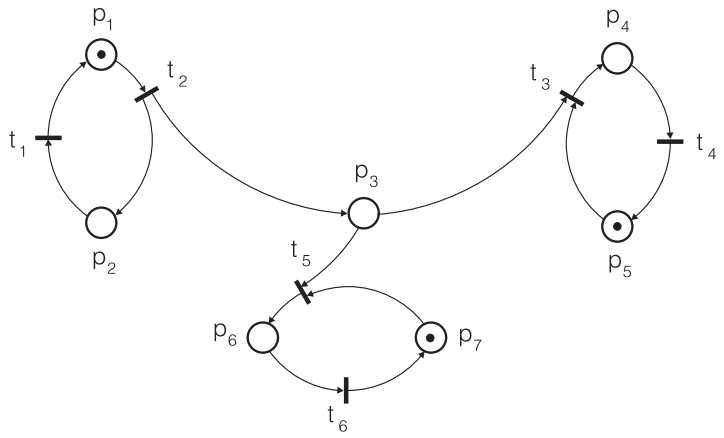


Obrázek 5.6: Znázornění grafu N

Petriho síť se opírá o bipartitní orientovaný graf $N = (P, T, A)$, kde P je neprázdná množina uzlů označovaných jako místa, T je neprázdná množina uzlů označovaných jako přechody a A je množina orientovaných hran. Pro každou hranu $a_i \in A$ platí $a_i = (p_j, t_k)$ nebo $a_i = (t_j, p_k)$, hrana tedy spojuje buď místo p_j s přechodem t_k (místo p_j nazýváme vstupním místem přechodu t_k) nebo přechod t_j s místem p_k (místo p_k nazýváme výstupním místem přechodu t_j).

Graf N znázorňujeme tak, že místa zobrazujeme jako kroužky, přechody jako tlusté čáry a hrany jako čáry, které spojují místa a přechody a mají orientaci vyjádřenou šipkou. Příklad si uvedeme na obr. 5.6.

Doplněním grafu N o *značení* (marking) μ, které každému místu $p \in P$ přiřadí přirozené číslo $\mu(p)$ dostaneme *Petriho síť* $M = (P, T, A, \mu)$. Konkrétní značení vyjádříme graficky počtem teček v každém místě (obr. 5.7).



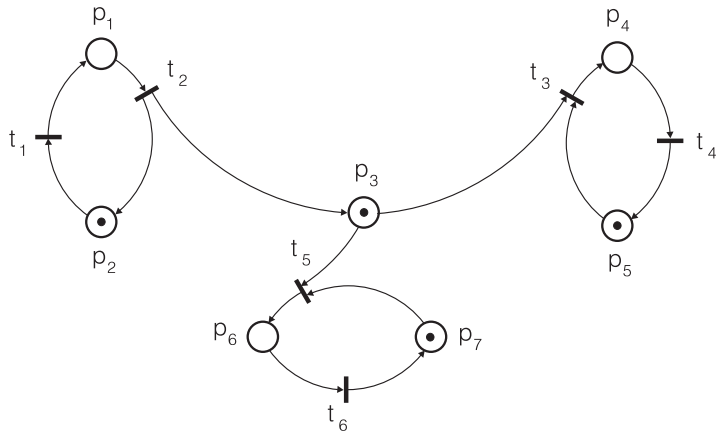
Obrázek 5.7: Petriho síť

Petriho síť lze použít jako popis dynamického chování nedeterministických systémů podobně, jako používáme konečný automat pro popis systémů deterministických.

Stavem sítě rozumíme konkrétní značení u Petriho sítě. Přechod, pro jehož všechny vstupy platí $\mu(p) > 0$, je *dovolený* (enabled). V daném stavu Petriho sítě může být počet dovolených přechodů větší. Změna stavu Petriho sítě označovaná jako *spuštění* (firing) spočívá v náhodné volbě jednoho z aktivovaných přechodů t a ve změně značení μ takové, že

$$\mu(p) = \begin{cases} \mu(p) - 1 & \text{pro vstupy pechodu t,} \\ \mu(p) + 1 & \text{pro vstupy pechodu t, a} \\ \mu(p) & \text{pro ostatn msta.} \end{cases}$$

Možnou změnu stavu Petriho sítě z předchozího obrázku uvádí obr. 5.8.



Obrázek 5.8: Stav po spuštění přechodu

Změnami stavu se Petriho síť pohybuje ve svém stavovém prostoru. Stavy, do kterých se síť může dostat z počátečního stavu μ_0, nazýváme *dosažitelnými stavy*.

Lze-li z každého dosažitelného stavu sítě přejít do nějakého stavu jiného, označujeme síť jako *živou* (live). Dosažitelný stav sítě, ve kterém není aktivován žádný přechod, indikuje *zablokování* (deadlock).

Stavový prostor Petriho sítě je obecně nekonečný (každé z míst označujeme přirozeným číslem). V praxi nás však zajímají zvláště sítě s konečným počtem stavů — sítě omezené, k-omezené, bezpečné a konzervativní.

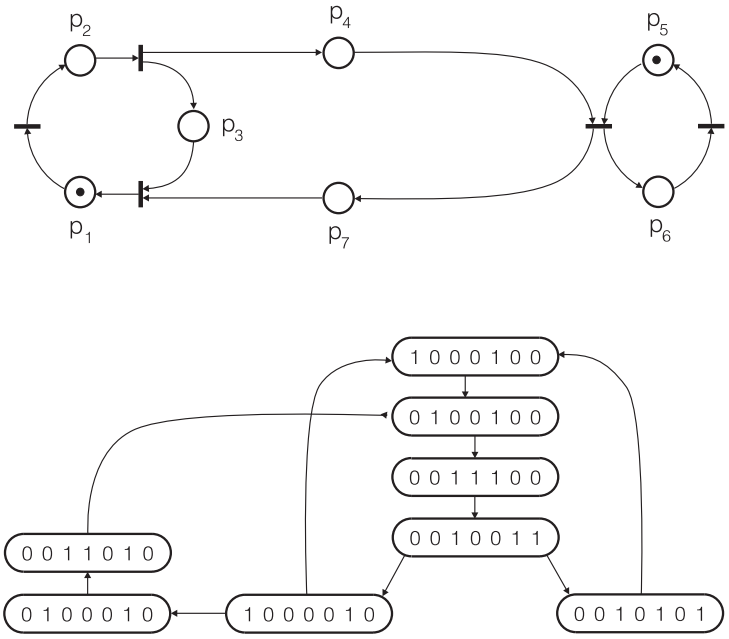
Petriho síť nazveme *bezpečnou*, jestliže pro každý její stav platí: $\mu(p_i) <= 1$ pro všechna místa $p \in P$ (v každém místě je nejvýše jedna tečka).

Petriho síť nazveme *k-omezenou*, jestliže pro každý její stav platí: $\mu(p_i) <= k$ pro všechna místa $p \in P$ (v každém místě je nejvýše k teček).

Petriho síť nazveme *omezenou*, je-li k-omezená pro nějaké konečné k.

Petriho síť nazveme *konzervativní*, jestliže pro každý její stav platí $\sum_{(i)} \mu(p_i) = const$ (počet teček v síti se nemění).

Při analýze stavového prostoru Petriho sítě se opíráme o *graf dosažitelnosti*, který konstruujeme takto: Uzly grafu dosažitelnosti přiřadíme stavům značené Petriho sítě. Počáteční stav Petriho sítě tedy reprezentujeme uzlem grafu dosažitelnosti, každý dovolený přechod hranou vedoucí do uzlu, který reprezentuje následující stav. Obrázek 5.9 uvádí příklad Petriho sítě a odpovídajícího grafu dosažitelnosti.

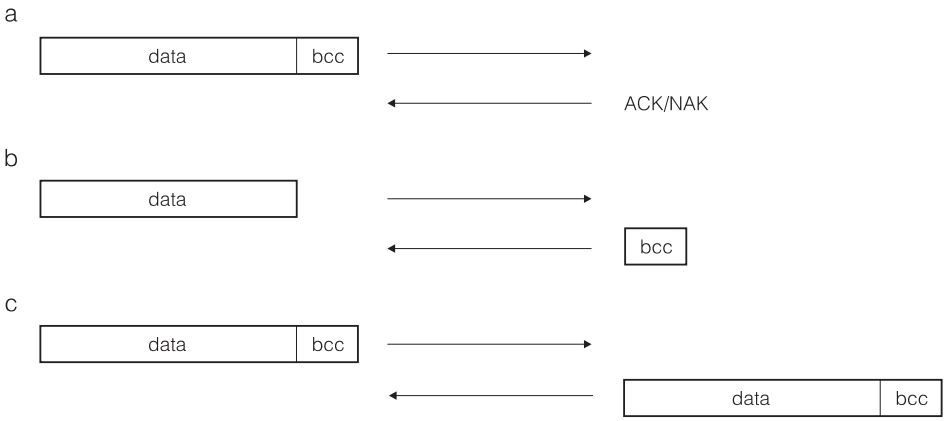


Obrázek 5.9: Značená Petriho síť a její graf dosažitelnosti

Graf dosažitelnosti není nic jiného než znázornění automatu, který má chování shodné s chováním analyzované Petriho sítě. Protože může mít až $[2^{(cardP)}]^k$ uzlů pro k-omezenou síť, může být i značně rozsáhlý. Není-li analyzovaná síť omezená, je její graf dosažitelnosti nekonečný.

5.3 Metody potvrzování

Pokud možno bezchybný přenos informací po reálných komunikačních kanálech (s chybovostí v rozsahu $10^{-9}a10^{-4}$) lze zajistit doplněním rámce o detekční kód, a využít ho pro rozhodnutí, zda má být rámec převzat bez detekované chyby, nebo, zda se máme pokusit o jeho opakované vyslání. Využití detekčních kódů (obvykle cyklických) takovýmto způsobem (*ARQ - Automatic Retransmission Request*) je označováno jako potvrzování a lze ho realizovat několika způsoby (obr. 5.10):



Obrázek 5.10: Potvrzovací metody

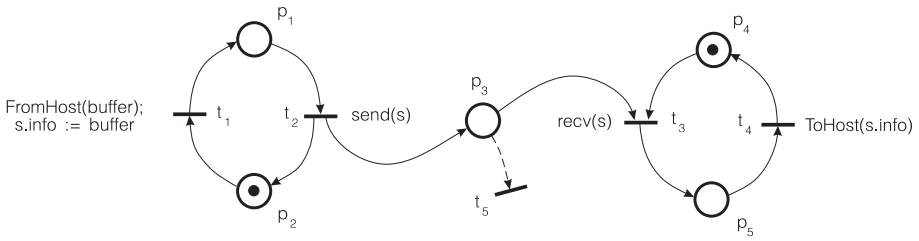
Potvrzovací zpětná vazba (a) je používána na dvoubodových spojích prakticky výlučně, s detekční zpětnou vazbou (b) se setkáme pouze výjimečně. U kruhových lokálních sítí a u družicových sítí lze využít informační zpětnou vazbu (c).

Alternativou potvrzovacích schémat je použití samoopravných kódů. Metody označované jako *FEC* (*Forward Error Recovery*) jsou pro svou větší složitost používány tam, kde by dlouhé přenosové zpoždění, vysoká chybovost, nebo složité přepínání směru přenosu, vedlo ke zpožděnému doručení dat a snížení propustnosti kanálu. Jsou využívány u družicových přenosů a v rádiových sítích. Nejčastěji se používají BCH a Fireovy kódy, někdy i kódy konvoluční.

V dalším textu se budeme věnovat příkladům typických potvrzovacích schémat a uvedeme si jejich podstatné vlastnosti.

Synchronní simplexní protokol

Nejjednodušší formou linkové komunikace je synchronní simplexní protokol, který si můžeme znázornit Petriho sítí na obrázku 5.11. Synchronní mu budeme říkat proto, že nelze pozastavit vysílač (přijímač musí pracovat synchronně s vysílačem), a simplexní proto, že přenos je jednosměrný.



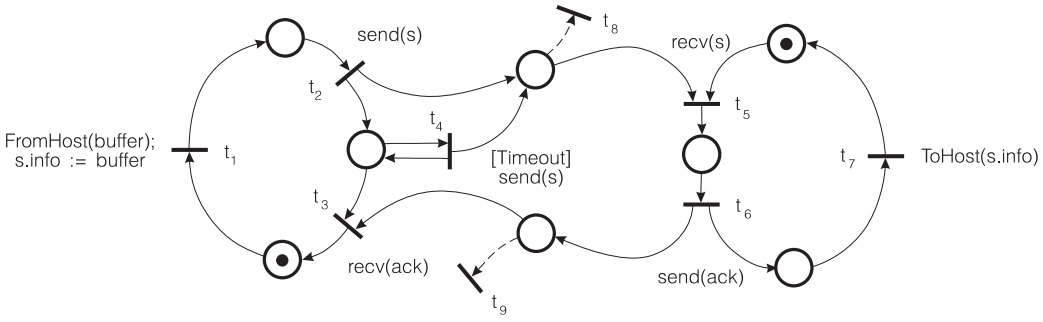
Obrázek 5.11: Synchronní simplexní protokol

Místa p_1 a p_2 reprezentují stranu vysílače, místo p_3 komunikační kanál, a místa p_4 a p_5 stranu přijímače. Přechod t_1 odpovídá převzetí zprávy od účastníka linkového spojení (uložení výstupu funkce $FromHost$ do datové části rámce $s.info$, přechod t_2 odpovídá odeslání rámce s. Přechod t_3 odpovídá příjmu rámce s a přechod t_4 předání datové části rámce $s.info$ adresátovi.

Praktické použití uvedeného protokolu je omezené. Protokol není schopen reagovat na detekované chyby kanálu (reprezentujeme je přechodem t_5). Navíc, Petriho síť nevylučuje hromadění značek v místě p_3; to odpovídá možné potřebě nekonečné vyrovnávací paměti na straně přijímače nebo možnému ztrácení rámců. Protokol je použitelný pro FEC metody, vyžaduje použití samoopravných kódů a musíme zajistit dostatečně rychlé zpracování na straně příjemce.

Simplexní protokol s pozitivním potvrzováním

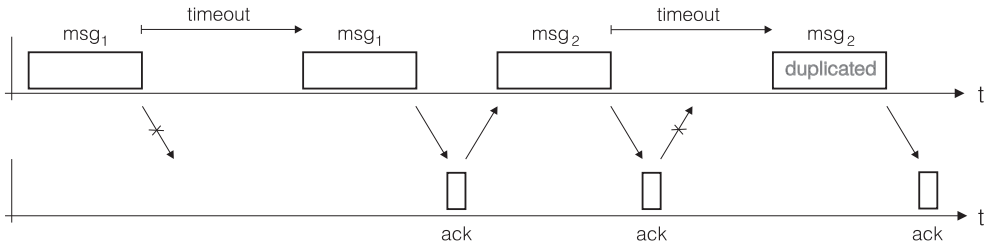
Činnost nejednoduššího potvrzovacího schématu - simplexního protokolu s pozitivním potvrzováním, lze popsat Petriho sítí na obrázku 5.12.



Obrázek 5.12: Simplexní protokol s pozitivním potvrzováním

Vysílající strana odešle při přechodu t_2 rámec a očekává potvrzení — povolení přechodu t_3. Pokud potvrzení nepřijde do doby, určené limitem $TimeOut$, přechod t_4 rámec zopakuje. Přechod t_8 odpovídá ztrátě/poškození rámce, přechod t_9 ztrátě/poškození potvrzení.

Protokol je schopen vyrovnat se se ztrátou rámce, ztráta potvrzení vede na duplikaci rámce. Obě situace ilustruje obrázek 5.13.

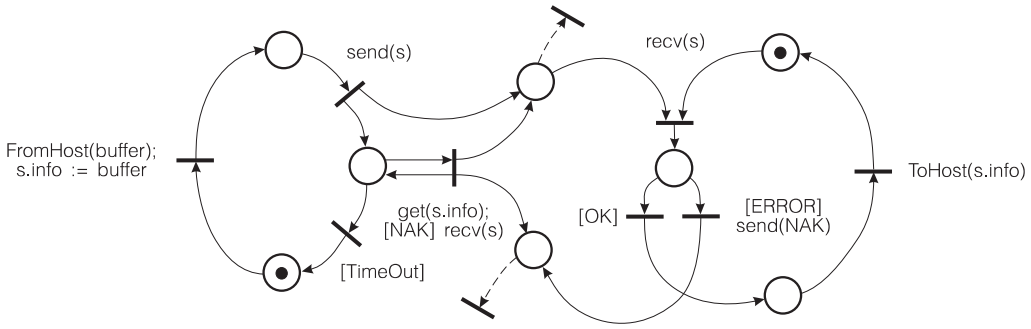


Obrázek 5.13: Pozitivní potvrzování

Pozitivní potvrzování je efektivní pro kanály s malou chybovostí. Při zvýšené chybovosti se negativně projeví vliv časového limitu. Poznamenejme, že simplexní přenos s potvrzováním vyžaduje použít alespoň poloduplexního fyzického kanálu.

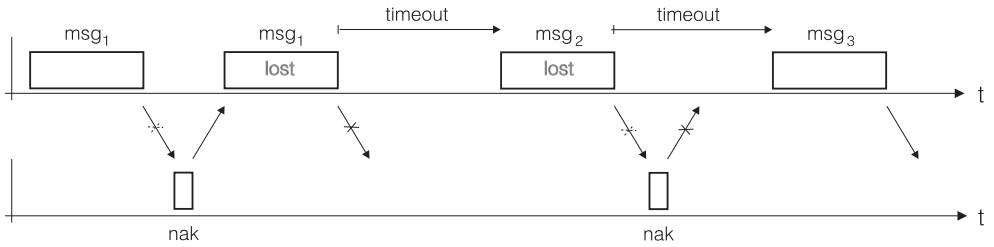
Simplexní protokol s čistě negativním potvrzováním

Zajímavým řešením ochrany proti chybám je simplexní protokol s čistě negativním potvrzováním, jeho funkce odpovídá Petriho síti na obrázku 5.14.



Obrázek 5.14: Simplexní protokol s čistě negativním potvrzováním

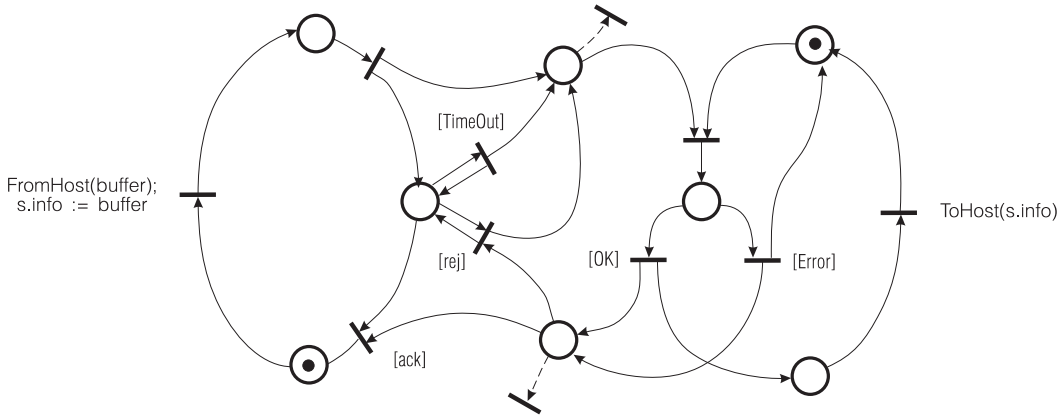
Vysílající strana odešle při přechodu t_2 rámec a očekává negativní potvrzení (odmítnutí - NAK, REJ). Nedojde-li odmítnutí do časového limitu $TimeOut$ je rámec považován za přijatý. Po přijatém odmítnutí je rámec přechodem t_4 zopakován. Odmítnutí vysílá přechod t_6 při zjištění chyby. Negativní potvrzování samotné není schopné rámce zopakovat při jeho ztrátě, ani při ztrátě odmítnutí — dochází ke ztrátám (obr. 5.15).



Obrázek 5.15: Čistě negativní potvrzování

Výhodou čistě negativního potvrzování je ale rychlá reakce na detekovanou chybu. Toho lze využít v protokolu, který kombinuje pozitivní potvrzování s potvrzováním čistě negativním.

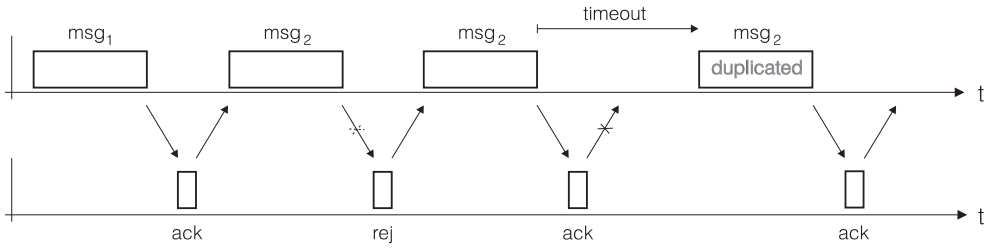
Simplexní protokol s negativním potvrzováním



Obrázek 5.16: Simplexní protokol s negativním potvrzováním

Negativním potvrzováním obvykle rozumíme doplnění schématu s pozitivním potvrzováním o možnost odmítnutí poškozených rámců. Jeho funkci uvádí obrázek 5.16.

Doplnění negativního potvrzování zrychluje reakci na příjem poškozeného rámce, časový limit brání zablokování vysílače po ztrátě rámce nebo potvrzení.

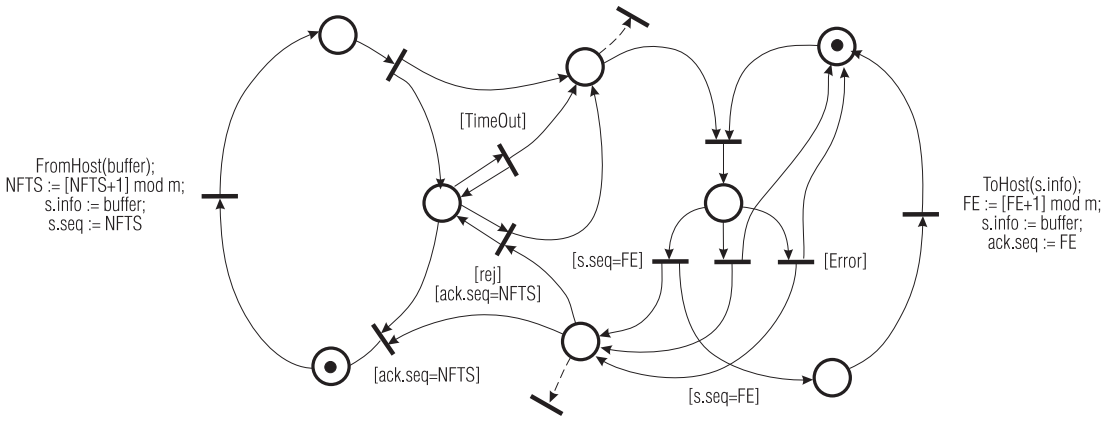


Obrázek 5.17: Negativní potvrzování

Ani kombinace pozitivního a negativního potvrzování není schopna zabránit duplikaci rámců (obr. 5.17). Dalším zdrojem nebezpečí je nedetekovaná záměna potvrzení: příjem ACK namísto REJ způsobí ztrátu rámce, příjem REJ místo ACK duplikaci.

Číslování rámců

Metody potvrzování, jak jsme si je dosud uvedli, nejsou schopné zabránit ztrátám nebo duplikaci rámců. Obojímu lze zabránit až jednoznačným číslováním rámců vysílačem a kontrolou posloupnosti rámců přijímačem. Jednoznačné číslování však není praktické — vyžaduje neomezenou délku příslušného pole v rámci. V praxi běžně vystačíme s *číslováním modulárním* — v protokolech linkové vrstvy se setkáváme s číslováním modulo 2 (*střídavé číslování*), modulo 8 a modulo 128. Číslovat lze rámce/příkazy, ale také potvrzení/odpovědi, příklad protokolu s číslováním příkazů i odpovědí uvádí obrázek 5.18.

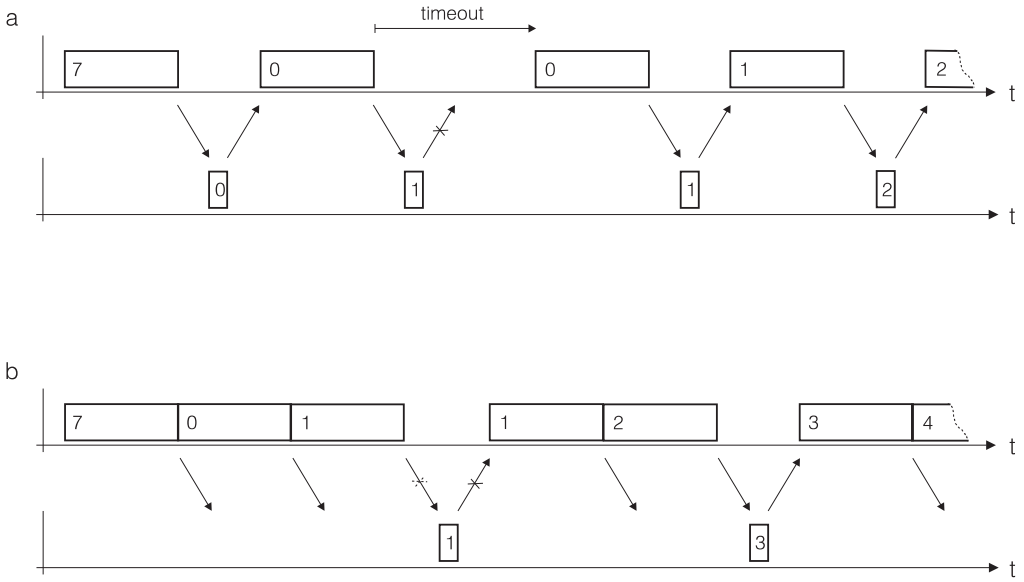


Obrázek 5.18: Simplexní protokol s číslováním rámců

Petriho síť na obrázku 5.18 je doplněna o proměnné $NFTS$ a FE, které jsou využity pro predikci přechodů t_8 a t_9. O takto doplněné Petriho síti mluvíme jako o *rozšířené Petriho síti.*

Skupinové potvrzování

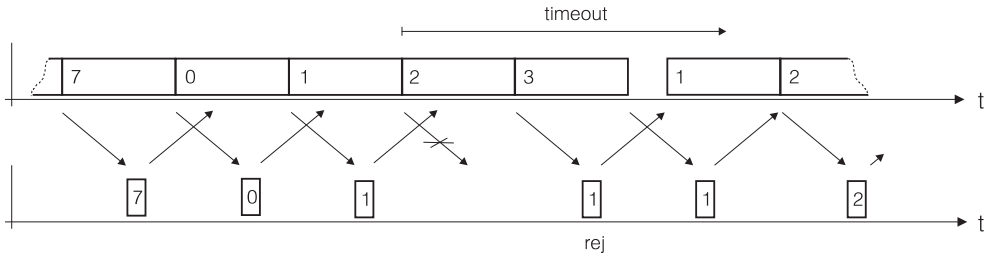
Při větších zpožděních mohou být časové ztráty dosud popisovaného *jednotlivého potvrzování* neúnosné (obr. 5.19a). Zvýšení propustnosti lze dosáhnout prodloužením rámců (pokud to dovolí chybovost kanálu), nebo potvrzováním posloupnosti více rámců společným potvrzením (obr. 5.19b).



Obrázek 5.19: Skupinové potvrzování

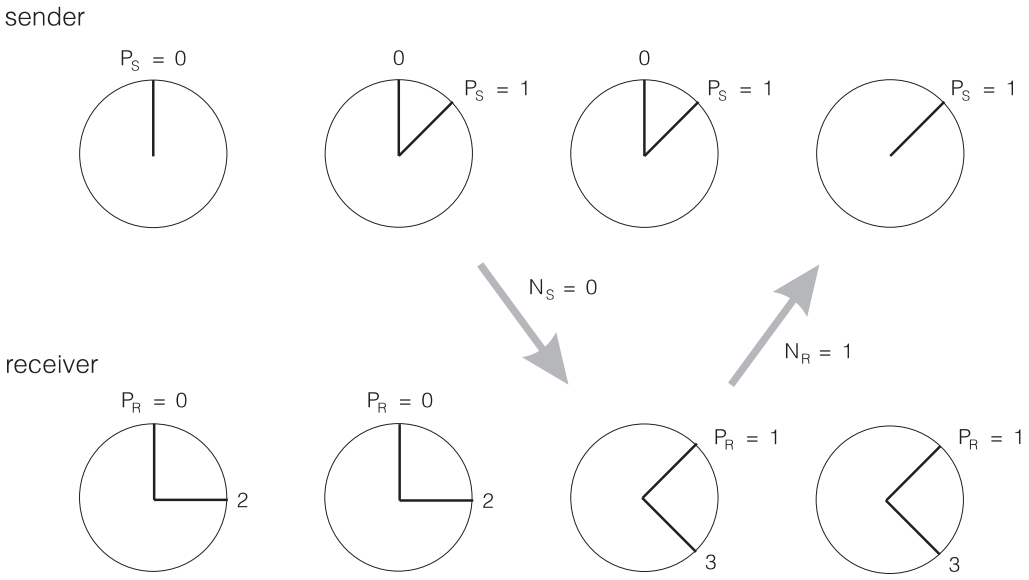
Vysílající strana po odvysílání skupiny rámců očekává potvrzení a po jeho obdržení pokračuje ve vysílání. Přijímací strana může potvrzovat číslem dalšího očekávaného rámce. Taková volba je obvyklá a bude využita ve všech následujících příkladech.

Malou úpravou skupinového potvrzování lze dosáhnout nepřetržitého toku rámců (obr. 5.20). Metodu označujeme jako *kontinuální potvrzování*.



Obrázek 5.20: Skupinové potvrzování - okénková metoda

Kontinuální potvrzování vyžaduje duplexní kanál a dovoluje vysílači odeslat omezený počet rámců bez potvrzení. Počet rámců N, které může vysílač odeslat nazýváme okénkem — metodu proto obvykle označujeme jako *okénkové schéma* (*Sliding Window*). Princip metody ilustruje obrázek 5.21.



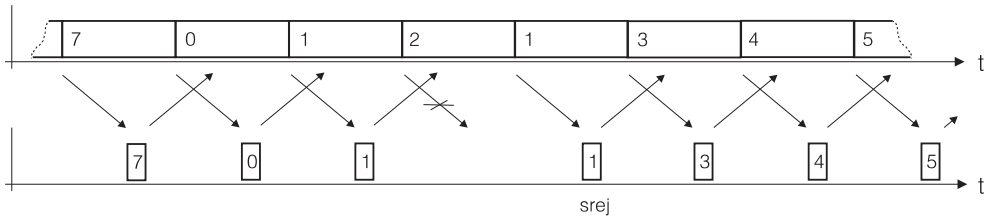
Obrázek 5.21: Skupinové potvrzování - metoda okénka

Vysílač i přijímač jsou inicializovány s hodnotami vnitřních proměnných $P_S = 0$ a $P_R = 0$. Hodnota P_S je při odeslání datového rámce použita jako číslo rámce N_S, a proměnná P_S je inkrementována. Je-li číslo přijatého rámce rovno hodnotě proměnné P_R, je tato proměnná inkrementována; hodnota proměnné P_R je vložena jako N_R do odeslaného potvrzení.

Cenou, kterou zaplatíme za efektivitu skupinového potvrzování, je vyrovnávací paměť pro N rámců na straně vysílače i přijímače. Musí platit $N < M$ (kde M je modul číslování), větší hodnota N zvyšuje využití kanálu s velkým zpožděním a malou chybovostí.

Selektivní opakování

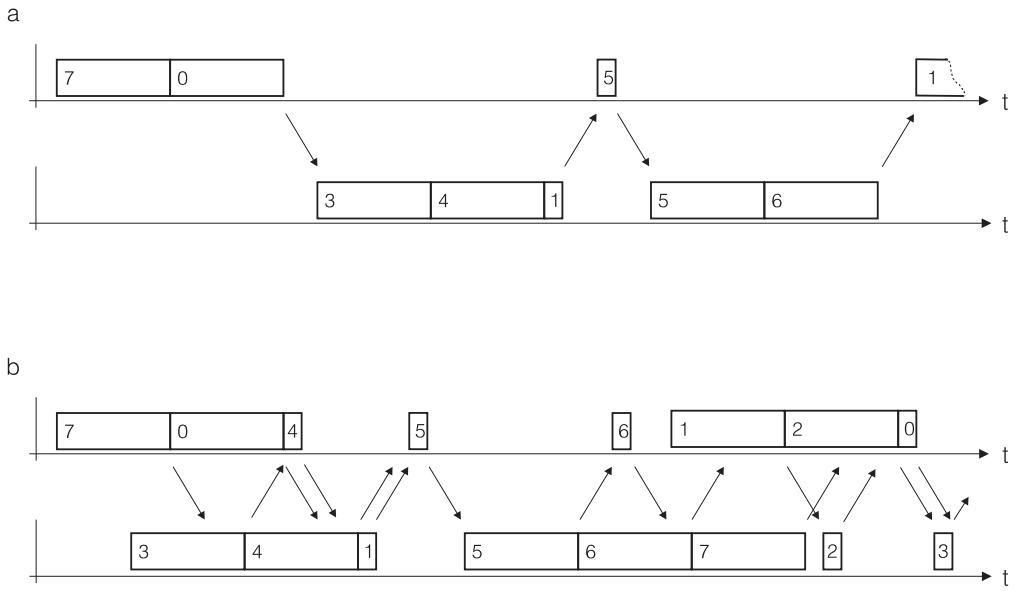
Detekce chyby vede u skupinového potvrzování (a tedy i u okénkového schématu) na opakování rámců počínaje poškozeným, tento postup označujeme jako *Go-Back-N*. Zavedením selektivního odmítnutí SREJ umožníme přijímající stanici požádat o zopakování jediného rámce — o *selektivní opakování*. Selektivní opakování zvyšuje využití kanálu, velikost okénka je ale omezena na $N < M/2$.



Obrázek 5.22: Selektivní opakování

Poloduplexní a duplexní přenos

Poloduplexního nebo duplexního kanálu lze s výhodou použít pro obousměrný přenos rámců. Obě protokolové stanice pak zahrnují vysílač i přijímač. Příklady použití poloduplexního a duplexního přenosu pro skupinové potvrzování uvádí obrázek 5.23.

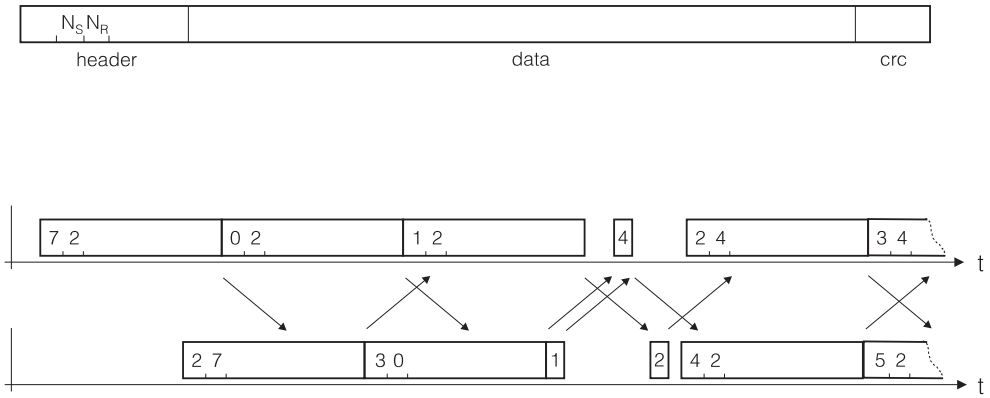


Obrázek 5.23: Poloduplexní (a) a duplexní (b) přenos

Pečlivý čtenář jistě zjistí, že u plně duplexního schématu obě stanice pozdržují potvrzení; takový postup může díky schopnosti potvrdit více rámců jedním potvrzením zvýšit efektivitu.

Nesamostatné potvrzování

Vysílání *samostatných potvrzení* (ACK, REJ) u poloduplexního a duplexního provozu zbytečně zvyšuje časovou režii. Proto obvykle rozšiřujeme datový rámec o možnost potvrdit rámec v opačném směru, mluvíme pak o *nesamostatném potvrzování* (*piggy-backing*). Rámec pak může mít formát, který spolu s příkladem chování stanic uvádí obrázek 5.24.

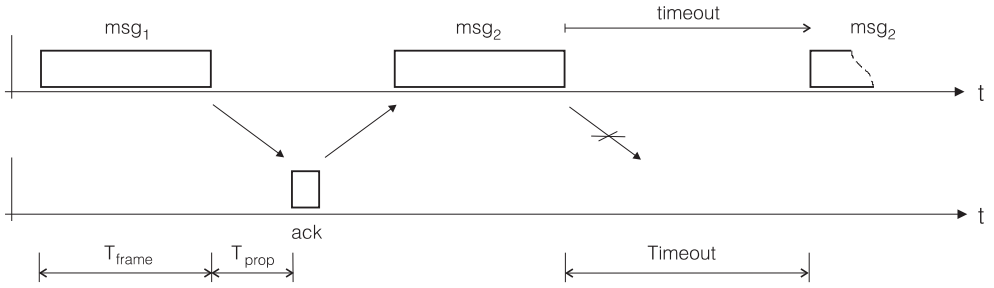


Obrázek 5.24: Nesamostatné potvrzování

5.4 Efektivita potvrzovacích metod

Při výběru vhodné potvrzovací metody je třeba vážit složitost metody a zvýšení propustnosti kanálu, které metoda zajistí. V této části se budeme věnovat efektivitě potvrzovacích schémat, jejich schopnosti potlačit vliv přenosových zpoždění v komunikačních kanálech.

Jednoduché potvrzovací metody jsou na dobu přenosu poměrně citlivé. Situaci pro pozitivní potvrzování ilustruje obrázek 5.25.



Obrázek 5.25: Efektivita jednoduchého potvrzování

kde T_{frame} reprezentuje dobu vysílání rámce o délce lpři dané přenosové rychlosti C, $T_{frame} = l/C$, a T_{prop} reprezentuje dobu přenosu na komunikačním kanále T_{prop}.

Efektivitu přenosu lze pro naši zjednodušenou situaci,

-kdy si nevšímáme režie hlaviček rámců, délek potvrzení a dob potřebných pro obsluhu operačním systémem: laskavý čtenář si jistě následující analýzu snadno doplní,

-kdy prozatím nepředpokládáme chyby na kanále,

vyjádřit vztahem:

$$S = \frac{T_{frame}}{T_{frame} + 2T_{prop}} ,$$

nebo po substituci $a = T_{prop}/T_{frame}$

$$S = \frac{1}{1 + 2a} .$$

Budeme-li uvažovat chybovost kanálu, můžeme efektivitu kanálu vyjádřit jako

$$S = \frac{1}{N(1 + 2a)} ,$$

kde hodnota N_t reprezentuje průměrné prodloužení doby přenosu způsobené opakováním po chybě a vypršením časového limitu $TimeOut$. Označíme-li si pravděpodobnost chyby při přenosu rámce jako P, bude pravděpodobnost, že pro přenos jednoho rámce budeme potřebovat k pokusů,

$$(1 - P)P^{k-1} .$$

Průměrný počet potřebných pokusů N pak bude

$$N = \sum_{i=1}^{\infty} i(1 - P)P^{i-1} = \frac{1}{1 - P} .$$

Za předpokladu, že časový limit bude mít minimální možnou hodnotu $Timeout = 2T_{prop}$, a tedy $N_x = N$, můžeme efektivitu přenosu vujádřit jako

$$S = \frac{1 - P}{1 + 2a} .$$

V moderních linkových procedurách převládají *okénkové metody*, tedy *skupinové nesamostatné provrzování*; selektivní opakování je spíše výjimkou.

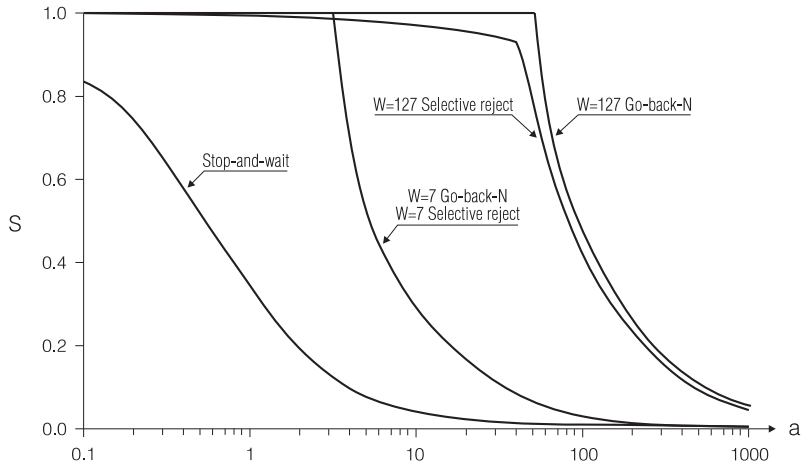
Analýzu si zjednodušíme normalizováním, budeme předpokládat jednotkový čas potřebný pro vyslání paketu, $T_{frame} = 1$. Pak, pokud bude W okénko potvrzovacího schématu, pro efektivitu bezchybového přenosu platí

$$S = \begin{cases} 1 & \text{pokud } W \geq 2a+1 \ , \\ \frac{W}{2a+1} & \text{pokud } W < 2a+1 \ , \end{cases}$$

kde a vyjadřuje přímo normalizovanou dobu přenosu.

Snížení efektivity způsobené chybami lze pro selektivní opakování (*Selective Reject*) opřít, podobně jako u jednoduchého potvrzování, o průměrný počet potřebných opakování N a z něj, pro minimální hodnotu $Timeout$ odvozenou hodnotu $N_x = N$. Výsledný vztah pro efektivitu přenosu při pravděpodobnosti P chyby v přenášeném rámci pak je

$$S = \begin{cases} 1-P & \text{pokud } W \geq 2a+1 \ , \\ \frac{W(1-P)}{2a+1} & \text{pokud } W < 2a+1 \ , \end{cases}$$



Obrázek 5.26: Efektivita potvrzovacích schémat

U skupinového opakování *Go-Back-N* musíme respektovat skutečnost, že chyba vyvolá potřebu opakovat skupinu K rámců. Střední počet pokusů o vyslání rámce pak bude

$$N = \sum_{i=1}^{\infty} f(i)(1-P)P^{i-1} \quad \text{, kde} \quad f(i) = 1 + (i-1)K = (1-K) + iK$$

Po dosazení dostaneme

$$N = \frac{1-P+KP}{1-P} \ .$$

Za předpokladu, že můžeme aproximovat

$$K = \begin{cases} 2a+1 & \text{pokud } W \geq 2a+1 \ , \\ W & \text{pokud } W < 2a+1 \ , \end{cases}$$

lze výslednou efektivitu vyjádřit jako

$$S = \begin{cases} 1-P & \text{pokud } W \geq 2a+1 \ , \\ \frac{W(1-P)}{(2a+1)(1-P+2WP)} & \text{pokud } W < 2a+1 \ . \end{cases}$$

Na závěr si uvedeme ve formě grafu chování uváděných metod potvrzování pro pravděpodobnost chyby rámce $P = 10^{-3}$ (obr. 5.26).

5.5 Znakově orientované protokoly

Starší protokoly linkové vrstvy jsou obvykle realizovány programem. Zpracovávají rámce po znacích a jedinou technickou podporou, kterou vyžadují, je dělení bitových posloupností na znaky.

Ochrana proti chybám při přenosu je u starších realizací protokolů zajišťována kontrolními součty, které mohou chránit i proti shlukům chyb (iterační kód). Novější řešení se opírají o vhodnější cyklické kódy.

Transparence přenášených dat je zajištěna vhodným výběrem znaků, které jsou využity jako oddělovače rámců a jejich polí; případný výskyt těchto znaků v přenášených datech prefixují pomocným znakem (s tím, že výskyt prefixu je sám prefixován). Alternativou prefixování je uložení údaje o délce přenášeného bloku dat do hlavičky rámce, znaky bloku dat pak mohou být z hlediska analýzy rámce ignorovány. Tohoto přístupu využíval například protokol DDCMP v sítích DECNet.

Funkce protokolů bývá doplněna o řízení toku na linkové úrovni, přijímající strana může přenos pozastavit opakovaným odmítáním rámce. Speciální řídící rámce dovolující pozastavit přenos (RNR) najdeme až u modernějších bitově orientovaných protokolů.

5.5.1 Protokol SLIP

Protokol *SLIP* (*Serial Link Internet Protocol — RFC 1055*) je nejjednodušším příkladem (znakově orientovaného) linkového protokolu. Jeho úkol je jediný, rozdělit tok dat na rámce, do kterých jsou ukládány IP pakety; nezajišťuje ani detekci chyb, ani potvrzování. Moderní modemy ale již potřebnou ochranu proti chybám zajišťují (jsou pro ně definovány zvláštní linkové protokoly — MNP.4 a V.42 CCITT (LAPM)).

Protokol SLIP postačuje pro připojení malého počtu koncových účastníků modemovými spoji k Internetu. Důvodem, proč byl protokol SLIP v této funkci nahrazen protokolem PPP je neschopnost protokolu SLIP rozlišit protokoly vyšších vrstev, což je pro dané použití, kdy potřebujeme koncovému účastníku sdělit IP adresu, domluvit se na parametrech komunikace (např. délka IP paketů) a zajistit jeho autentizaci, důležité.

Protokol předpokládá zpracování posloupnosti znaků předávaných komunikačním kanálem (tím je obvykle modemový spoj s asynchronním přenosem na sériovém rozhraní modemu). Odděluje jednotlivé IP pakety řídícím znakem *END* (je definován jako $C0_H$), případný výskyt tohoto znaku v přenášených datech je nahrazen dvojicí DB_H a DC_H. Pokud se první znak této dvojice (*ESC* — DB_H) objeví v přenášených datech, je sám nahrazen dvojicí DB_H a DD_H.

Pozdější verze protokolu (RFC 1144) redukuje režii hlaviček IP protokolu, na základní funkci protokolu však nic nemění.

5.5.2 Protokol BSC

Protokol *BSC* (*Binary Synchronous Protocol*) firmy IBM se stal vzorem pro řadu dalších firemních řešení a základem pro doporučení ISO1745, které shrnuje podstatné vlastnosti znakově orientovaných protokolů.

Protokol BSC byl navržen pro poloduplexní přenos dat po poloduplexním mnohabodovém spoji s jednou řídící a jednou nebo více stanicemi podřízenými (obr. 5.28). (V době návrhu protokolu BSC takové řešení dovolilo sdílet jeden modemový spoj více zařízeními vzdálenými od centrálního počítače.)

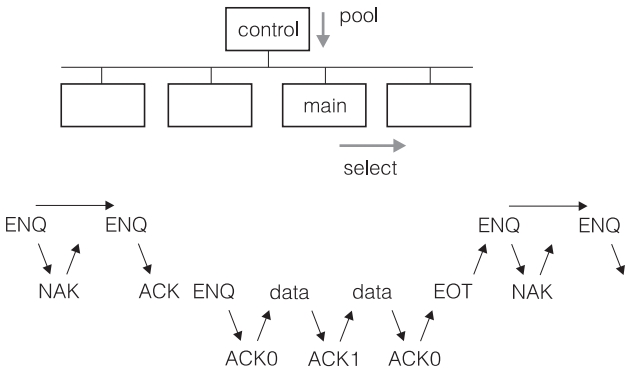
Data jsou přenášena v rámcích, jejichž oddělovači jsou řídící znaky ASCII. Dlouhé zprávy lze při přenosu dělit do více bloků, protokol BSC tedy zajišťuje i segmentaci zpráv, která je dnes obvykle funkcí transportní vrstvy.

Datový rámec je uveden znakem *SYN* (16_H) kódu ASCII (ale obvykle je jich vysíláno více). Výběr synchronizačního znaku dovoluje jednoznačnou identifikaci řadičem sériového rozhraní, který již pak dále dělí posloupnost bitů na znaky. Řídící znaky *SOH* (01_H), *STX* (02_H) a *ETX* (03_H) oddělují hlavičku (nepovinná), přenášený blok dat, a kód zabezpečující proti chybám (cyklický kód nebo kontrolní součet) — obr. 5.27.

Obrázek 5.27: BSC - formát datového rámce

Transparence dat je zajištěna prefixací řídících znaků kódu ASCII v přenášených datech řídícím znakem *DLE* (10_H), prefixován je i výskyt řídícího znaku *DLE*.

Provoz na vícebodovém spoji je řízen řídící stanicí. Ta, obvykle v cyklu, *vyzývá* (*polling*) jednotlivé podřízené stanice. Stanice, která nemá data k odeslání, výzvu odmítne; stanice, která chce vyslat datový blok libovolné jiné stanici, výzvu přijme. Stává se pak hlavní stanicí a zahajuje přenos dat *výběrem* příjemce (*select*). Přenos je podpořen střídavým potvrzováním, číslována jsou pouze potvrzení (předávají číslo očekávaného rámce). Přenos je ukončen vysláním znaku *EOT*, jím hlavní stanice vrací řízení spoje stanici řídící (obr. 5.28).



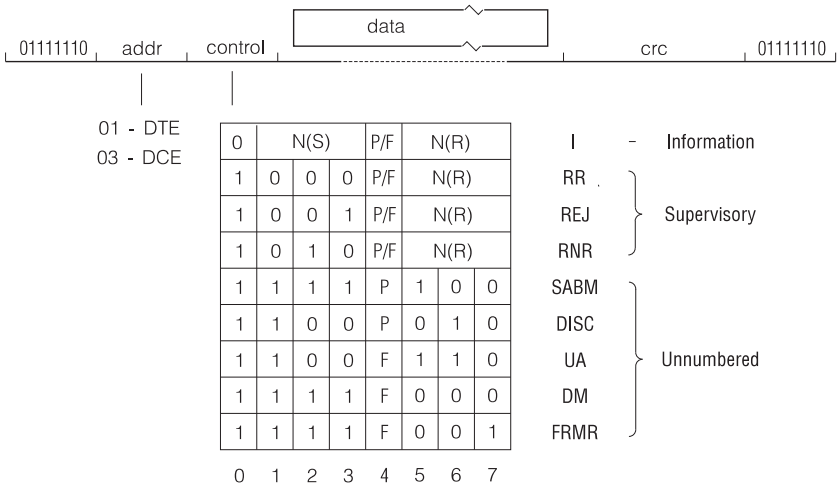
Obrázek 5.28: BSC - řízení přenosu

Protokol BSC chrání proti chybám přenášená data, řídící *"rámce"* zajištěné nejsou. Adresace stanic na vícebodovém spoji se objevuje pouze ve výzvách, výběrech a odpovědích na ně.

5.6 Bitově orientované protokoly

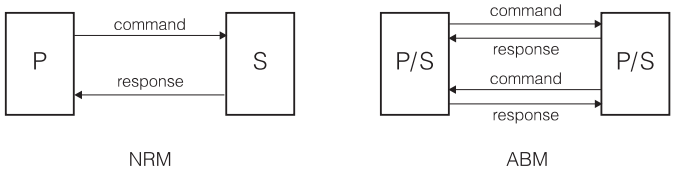
Bitově orientované protokoly odstraňují nedostatky protokolů znakově orientovaných. Nejstarší z nich je *protokol SDLC* (Synchronous Data Link Control) určený pro vícebodové spoje a navržený firmou IBM. Protokol se stal základem pro doporučení *HDLC*, ze kterého vychází řada v současnosti používaných protokolů — *LAPB* pro veřejné datové sítě, *LAPM* (*V.42 CCITT*) pro telefonní modemy, *LAPD* pro signalizaci ISDN a *LAPF* pro spoje *Frame Relay*. Bitově orientované protokoly zajišťují transparenci dat technickými prostředky, opírají se o obvodovou podporu pro výpočet cyklického kódu, využívají okénkového potvrzování a dovolují plně duplexní přenos a vysoké využití kapacity kanálu.

Formát rámce odpovídá obrázku 5.29. Začátek a konec rámce je určen křídlovo značkou, bitovou sekvencí 01111110. Transparenci přenášených dat podporují komunikační řadiče mechanismem, označovaným jako *vkládání bitů*. Vysílač vkládá za každou pětici jedniček vysílaných dat nulu, přijímač tuto nulu opět odebere. Výskyt posloupnosti šesti jedniček následovaných nulou indikuje řadič přijímače jako křídlovou značku.



Obrázek 5.29: HDLC - formát rámců

Osmibitové pole *addr* identifikuje stanice na na vícebodovém spoji (pro který byl protokol) SDLC navržen. Současně odlišuje *příkazy*, vysílané s adresou příjemce od *odpovědí* vysílaných s vlastní adresou.



Obrázek 5.30: HDLC - třídy provozu

Osmibitové pole *control* rozlišuje informační a řídící rámce (nejméně významným bitem), obsahuje čísla rámců N_S a potvrzení N_R. Bit P/F podporuje pomocné pozitivní potvrzování, to je využíváno při otevírání a uzavírání spojení, a jako ochrana proti rozsynchronizování okénkového schématu na spojích s velkým zpožděním. Pro zajištění přenosu proti chybám je využíván cyklický kód s generačním polynomem $x^{16} + x^{12} + x^5 + 1$.

Bitově orientované protokoly byly navrženy pro vícebodové spoje s jednou řídící stanicí a více stanicemi podřízenými (*NRM* - Normal Response Mode). Pro efektivní činnost na

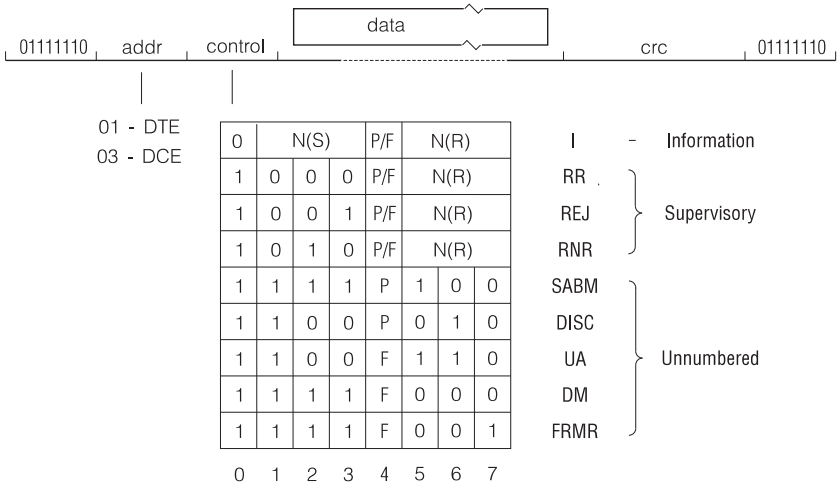
dvoubodových spojích byly později doplněny režimy *ARM* (Asynchronous Response Mode) a *ABM* (Asynchronous Balance Mode) (obr. 5.30). Režim *ABM* je základem moderních protokolů, patří sem linkový protokol LAPB X.25 CCITT, který si jako příklad popíšeme podrobněji.

5.6.1 Protokol LAPB X.25

Linková procedura LAPB (Link Access Procedure – Balanced) definuje linkové rozhraní koncového a ukončujícího zařízení ve veřejných datových sítích podle doporučení X.25 CCITT.

Formát rámce

Formát rámce a zajištění transparence dat odpovídá ostatním bitově–orientovaným protokolům (obr. **??**). Protože procedura popisuje komunikaci pouze dvou stanic, jsou adresy voleny pevně. Koncové zařízení (DTE - Data Terminal Equipment) má adresu $01H$ a ukončující zařízení (DCE - Data Circuit Equipment) má adresu $03H$.



Obrázek 5.31: LAPB - formát rámců

Řídící pole definuje typ rámce. Procedura LAPB používá příkazy I, RR, RNR, REJ, SABM a DISC a odpovědi RR, REJ, RNR, UA a DM. Bit P je nastaven na hodnotu $P = 1$ ve všech příkazech, výjimkou jsou pouze I–rámce, kde je na hodnotu $P = 1$ nastavován pouze jako reakce na vypršení časového limitu. Hodnota bitu F v odpovědi musí souhlasit s hodnotou bitu P v příkaze. Informační rámce jsou číslovány modulo 8, pořadové číslo rámce obsahuje pole $N(S)$. Pole $N(R)$ slouží pro potvrzování, stanice v poli $N(R)$ uvádí číslo příštího očekávaného rámce.

Typy rámců

I -Informační rámec (I–rámec) jako jediný přenáší data. Používá se pouze jako příkaz.

RR -Řídící rámec, slouží jako odpověď v případě, že stanice nemá data k vysílání, ale má dostatečnou kapacitu vyrovnávacích pamětí. Příkaz RR je použitelný jako dotaz na stav protistanice.

RNR -Řídící rámec, slouží jako odpověď v případě, že stanice nemá dostatek vyrovnávací paměti a žádá tak protistanici o pozastavení vysílání dalších I–rámců.

REJ -Řídící rámec, odpověď, kterou stanice reaguje na chybu v číslování rámců (po ztrátě rámce). Po příjmu rámce REJ přechází stanice do stavu, ve kterém opakuje vysílání I-rámců počínaje rámcem, jehož číslo je uvedeno v poli N(R) rámce REJ. Příkaz

REJ lze použít jako dotaz na stav protistanice.

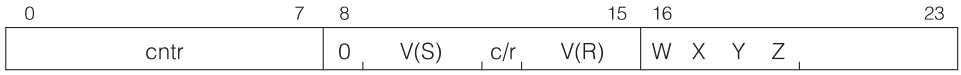
SABM -Nečíslovaný rámec, příkaz, kterým stanice žádá o navázání spojení.

DISC -Nečíslovaný rámec, příkaz, kterým stanice žádá o ukončení spojení.

UA -Nečíslovaný rámec, odpověď na příkazy SABM a DISC.

DM -Nečíslovaný rámec, odpověď odpojované a odpojené stanice na příkazy s nastaveným bitem P.

FRMR -Nečíslovaný rámec, odpověď stanice na příjem rámce s poškozeným řídícím polem. Rámcem FRMR reaguje stanice na příjem:
-rámce s neznámým řídícím polem,
-rámce, který má datové pole, ale nejde ani o I–rámec, ani o rámec FRMR (X=1,W=1),
-rámce, jehož délka přesahuje povolené maximum (Y=1),
-rámce s chybnou hodnotou N(R) (R=1).
Rámec FRMR má datové pole o délce tří slabik (obr. 5.32).



Obrázek 5.32: LAPB - rámec FRMR

Pole CNTR obsahuje řídící pole přijatého rámce, pole V(S) a V(R) stavy čítačů stanice, pole C/R informaci, zda šlo o příkaz (C/R=1) nebo o odpověď (C/R=0) a pole W,X,Y a Z diagnostické informace.

Režimy činnosti

Každá ze staic může pracovat ve dvou režimech, v režimu odpojení a v režimu přenosu dat. Odpojená stanice sama nevysílá žádné příkazy a odpovídá rámcem DM pouze na příkazy s nastaveným P–bitem. Na rámec SABM odpovídá rámcem UA, nastaví své proměnné $V(R)$ a $V(S)$ na ulu a přechází do režimu přenosu dat. Odpojená stanice může na žádost z vyšší vrstvy vyslat příkaz SABM; po příjmu odpovědi UA přechází do režimu přenosu dat.

V režimu přenosu dat stanice vysílá a přijímá datové rámce, potvrzuje je a žádá o opakování rámců poškozených. Rámce s poškozeným řídícím polem může odmítnout (FRMR). Stavy, ve kterých se stanice nachází po odeslání a příjmu rámců RNR, REJ a FRMR nebo po vypršení časového limitu jsou obslouženy speciálně.

Z režimu přenosu dat přechází stanice do režimu odpojení pokud došlo k některé z následujích situací:

1 -byla o odpojení požádána příkazem DISC,

2 -přijala odpověď UA na svůj příkaz DISC,

3 -po N2 pokusech neobdržela odpověď na příkaz DISC do vypršení časového limitu,

4 -přijala rámec DM s nulovým F–bitem,

5 -po N2 pokusech neobdržela opověď na rámec FRMR do vypršení časového limitu,

6 -po N2 pokusech neobdržela potvrzení I–rámce do vypršení časového limitu,

7 -přijala rámec FRMR,

8 -přijala rámec SABM po odeslání rámce DISC,

9 -přijala odpověď s nevyžádaným, ale nastaveným F–bitem.

Situace 4–9 jsou chybové situace při přenosu dat a stanice se po odpojení automaticky pokusí o nové navázání spojení.

Navázání spojení

O navázání spojení žádá odpojená stanice vysláním příkazu SABM s nastaveným P–bitem a spuštěním časovače. Protistanice na příkaz SABM odpoví rámcem UA s nastaveným F–bitem a přchází do režimu přenosu dat; své čítače $V(S)$ a $V(R)$ nastaví na nulu. Pokud stanice obdrží odpověď UA do časového limitu, nastaví své čítače $V(S)$ a $V(R)$ na nulu a spojení je navázáno. Na případný příchod příkazu SABM od protistanice (při současném pokusu obou stran o navázání spojení) odpoví stanice rámcem UA a přechází do režimu přenosu dat okamžitě. Pokud protistanice nemůže přejít do režimu přenosu dat, odpoví na příkaz SABM rámcem DM.

Po vypršení časového limitu stanice pokus o navázání spojení automaticky zopakuje. Počet pokusů o navázání spojení je omezen parametrem N2, při jeho překročení požádá stanice o zásah programy vyšší vrstvy nebo operátora.

Ukončení spojení

O ukončení spojení žádá stanice vysláním rámce DISC s nastaveným P–bitem a spuštěním časovače. Protistanice na příkaz DISC odpoví rámcem UA s nastaveným F–bitem a přechází do režimu odpojení. Prvá stanice přechází do režimu odpojení po příchodu odpovědi UA. Pokud odpověď UA nepřijde do vypršení časového limitu, stanice příkaz DISC opakuje. Počet pokusů o ukončení spojení je omezen parametrem $N2$.

Po odeslání rámce DISC stanice ignoruje všechny odpovědi, kromě rámce UA s nastaveným F–bitem a všechny příkazy s nulovým P–bitem. Na příkazy s nastaveným P–bitem odpovídá rámcem DM s nastaveným F–bitem. Výjimkou je reakce na příkaz SABM, kdy stanice odpoví rámcem UA (znovunavázání spojení).

O ukončení spojení žádá stanice automaticky v chybových stavech 4–9 odesláním rámce DM a pokouší se o nové navázání spojení.

Vyslání I–rámce

Stanice, která má připravený I–rámec k odeslání a není ve stavu po přijetí rámců REJ, RNR a FRMR rámec odešle s hodnotami $N(S) = V(S)$ a $N(R) = V(R)$. Po odeslání rámce zvyšuje hodnotu $V(S)$ o jedničku. Pokud neběží časovač hlídající časové limity, pak je spuštěn. Stanice smí vyslat I–rámec pouze tehdy, jestliže pro posledně přijaté potvrzení $N(R)$ platí $N(R) < V(S) \leq N(R) + k$. Jakmile je $V(S) = N(R) + k$ musí stanice další vysílání I–rámců pozastavit. Hodnota parametru k určuje okénko potvrzovacího schématu a musí platit $k \leq 7$. (Všechny výpočty jsou v aritmetice modulo 8.)

Příjem I–rámce

Stanice, která přijala I–rámec, reaguje některým z těchto způsobů:

1 -Pokud má přijatý rámec správné číslo N(S)=V(R) a stnice má připravený rámec k odeslání a dostatek vyrovnávací paměti, pak odpoví zvýšením čítače V(R) o jedničku (modulo 8) a odesláním rámce s čísly N(R)=V(R) a N(S)=V(S). Po odeslání I-I–rámce zvýší stanice čítač V(S) o jedničku (modulo 8). Tímto způsobem stanice reaguje pouze na I–rámec s nulovým P–bitem. Na I–rámec s nastaveným P–bitem musí stanice odpovědět rámcem RR.

2 -Pokud má přijatý rámec správné číslo N(S)=V(R) a stanice má dostatek vyrovnávací paměti, ale nemá připravený I–rámec, nebo přijatý rámec má nastavený P–bit, pak stanice odpoví zvýšením čítače V(R) o jedničku (modulo 8) a odesláním rámce RR s číslem N(R)=V(R).

3 -Pokud má přijatý rámec chybné číslo N(S)=V(R) pak stanice odpoví rámcem REJ s číslem N(R)=V(R). Přijatý rámec edy není potvrzen.

4 -Pokud má přijatý rámec správné číslo N(S)=V(R) a stnice nemá dostatek vyrovnávací paměti pro uložení dalších rámců, pak odpoví zvýšením čítače V(R) o jedničku (modulo 8) a odesláním rámce RNR s číslem N(R)=V(R).

5 -Pokud stanice nemá dostatek vyrovnávací paměti pro uložení přijatého rámce, pak odpoví rámcem RNR s číslem N(R)=V(R). Přijatý rámec tedy není potvrzen.

Příjem potvrzení

Čísla $N(R)$ v přijatých informačních nebo řídících rámcích slouží jako potvrzení. Pokud je přijaté číslo $N(R)$ větší než bylo $N(R)$ posledně přijaté, stanice zastaví časovač. Pokud již byl odeslán rámec s dosud nepotvrzeným číslem, pak stanice časovač znovu spustí.

Reakce na rámec REJ

Po příjmu rámce REJ stanice nastaví čítač $V(R)$ na přijatou hodnotu $N(R)$ a zopakuje poškozený I–rámec. Tento rámec je odeslán okamžitě (případné vysílání I–rámce je přerušeno) nebo po odvysílání právě vysílaného řídícího rámce.

Reakce na rámec RNR

Po příjmu rámce RNR lze vyslat I–rámec s číslem, které odpovídá hodnotě $N(R)$ v rámci RNR. Ve vysílání dalších rámců lze pokračovat až po přijetí rámce RR nebo REJ.

Vypršení časového limitu

Pokud dojde k vypršení časového limitu po odeslání I–rámce, je opakovaně odeslán nejstarší nepotvrzený I–rámec s nastaveným P–bitem a je očekávána odpověď RR, REJ nebo RNR s nastaveným F-bitem. Příchod odpovědi do časového limitu dovoluje přejít k normálnímu přenosu dat. Pokud časový limit opět vyprší, pokus o odeslání se opakuje. Počet opakování je omezen parametrem $N2$.

Reakce na poškozené rámce

Rámce, u kterých je zabezpečovacím kódem detekována chyba jsou likvidovány. Žádná informace obsažená v těchto rámcích není brána v úvahu.

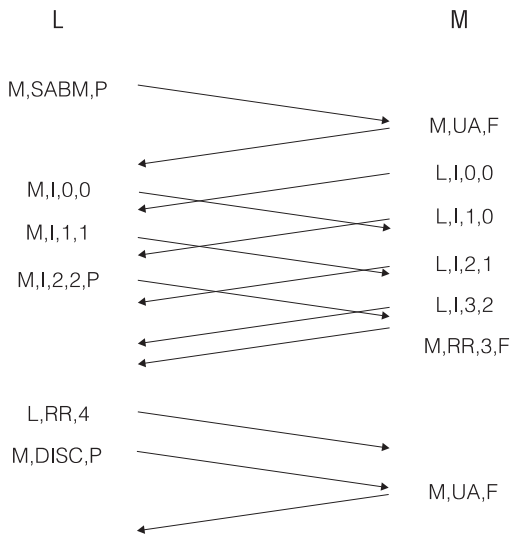
Reakce na rámce s poškozeným řídícím polem

Stanice odpovídá rámcem FRMR, spouští časovač a očekává znovunavázání spojení příkazem SABM. Do té doby likviduje přijaté rámce a na příkazy s nastaveným P–bitem odpovídá opět rámcem FRMR. Pokud vyprší časovač, stanice se pokusí o nové vyslání rámce FRMR. Počet opakování je omezen a po jeho překročení se stanice odpojuje a vysláním rámce SABM žádá o znovunavázání spojení.

Systémové parametry

Procedura LAPB používá tři nasavitelné parametry – časový limit $T1$, maximální počet vypršení časovače $N2$ (maximální počet pokusů o přenos rámce) a okénko potvrzovacího schématu k. Vhodná volba těchto parametrů dovolí přizpůsobit linkovou proceduru charakteru linkového spoje.

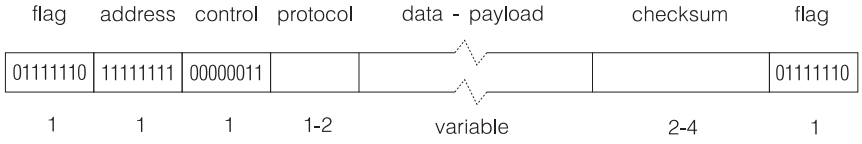
Na závěr popisu linkové procedury LAPB si uvedeme příklad činnosti stanic při duplexním přenosu dat (obr. 5.33).



Obrázek 5.33: LAPB - duplexní přenos dat

5.6.2 Protokol PPP

Protokol *PPP* (*Point-to-Point Protocol — RFC 1661, 1662 a 1663*) byl navržen na základě požadavků na připojení koncových účastníků k Internetu. Formát jeho rámců vychází z formátu zavedeného protokolem HDLC (obr. 5.34).



Obrázek 5.34: PPP - formát rámců

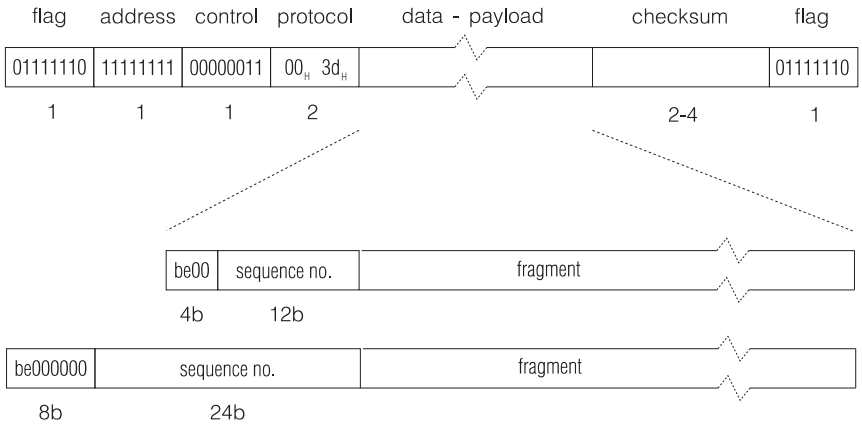
Jedná se o protokol pro dvoubodové spoje, který dovoluje detekovat chyby (zabezpečení se opírá o cyklický kód a rámce s chybou jsou zahazovány), ale případnou opravu nechává na protokolech vyšší vrstvy (např. transportní protokol TCP). Adresní pole není využito, řídící pole přenáší pouze indikaci faktu, že obsahem datového rámce jsou neidentifikovaná data (UI - 03_H).

Zajímavostí je pole *protocol*, které dovoluje odlišit komunikaci pod různými protokoly síťové vrstvy (např. IP a IPX). Zároveň podporuje řízení linkového spoje (LCP - Link Control Protocol), autentifikaci koncového účastníka a nastavení některých parametrů síťové vrstvy (IP adresa, délka paketu).

5.6.3 Protokol MPPP

Protokol *MPPP* (*Multilink Point-to-Point Protocol — RFC 1717*) dovoluje využít souběžných fyzických spojů (např. dvojice kanálů B základní přípojky ISDN, ale i linek s rozdílnou přenosovou rychlostí) pro vytvoření jednoho kanálu rychlejšího.

Umí dělit dlouhé pakety do fragmentů, které čísluje a na straně přijímače opět skládá. Číslování (čítač má délku 12, případně 24 bitů) dovoluje složit fragmenty ve správném pořadí. Formát rámce protokolu MPPP (pro obě varianty číslování znaků ve fragmentech) uvádí následující obrázek 5.35.



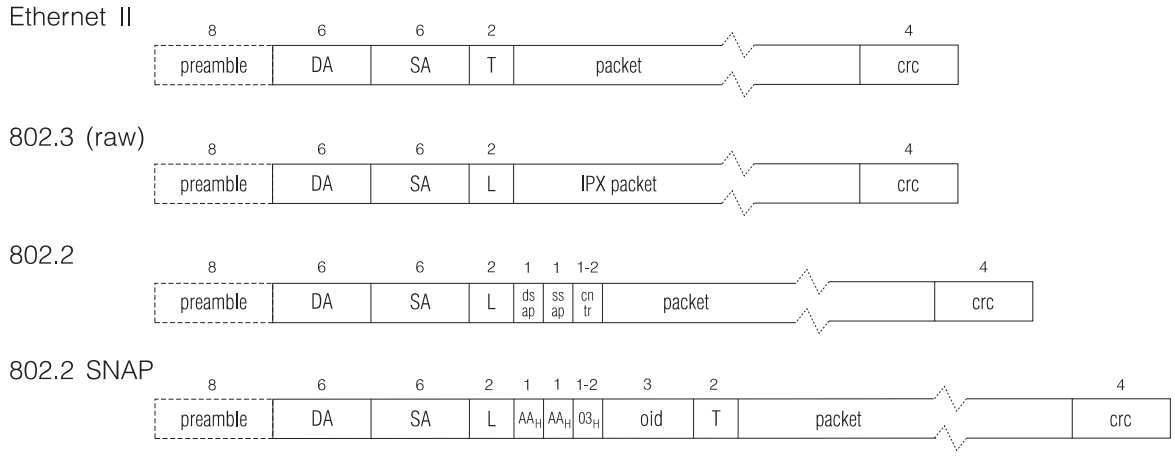
Obrázek 5.35: MPPP - formát rámců

5.7 Linkové protokoly lokálních sítí

Nejdůležitější funkcí protokolů linkové vrstvy u lokálních sítí je adresace stanic na sdíleném médiu (MAC adresa) a možná podpora více současně využívaných protokolů síťové vrstvy. Jako příklad si uvedeme linkové protokoly, se kterými se běžně setkáváme u technologie Ethernet (obr. 5.36), protokoly IEEE 802.2 a IEEE 802.2 SNAP jsou přitom definovány společně pro všechny lokální sítě definované řadou standardů IEEE 802.

Rámce první specifikace Ethernetu, označované jako *DIX* nebo *Ethernet II*, dovolují identifikovat příjemce rámce (48-bitovou adresou *DA*) a odesílatele rámce (49-bitovou adresou *SA*), a indikovat poškození rámce 32-bitovým cyklickým kódem *crc*. Šestnáctibitové pole *T* dovoluje specifikovat formát paketu, který je rámcem přenášen, lze tak odlišit současný provoz pod více síťovými protokoly (např. IP, IPX, AppleTalk, DECNet).

U linkových protokolů *LLC* (*Logical Link Control*), specifikovaných standardem IEEE 802.2 je pole *T* udávající typ přenášeného paketu nahrazeno polem *L* udávajícím délku přenášeného paketu. Následující osmibitová pole *dsap* a *ssap* identifikují způsob komunikace, tedy protokolovou sadu, na straně odesílatele a příjemce. Konečně, v poli *cntr* jsou přenášeny informace potřebné pro potvrzování.



Obrázek 5.36: Linkové protokoly lokální sítě Ethernet

Současné použití protokolů Ethernet II a IEEE 802.2 na jedné lokální síti umožňuje omezení délky rámce na 1500 oktetů ($1500 = 5DC_H$), vyšší hodnoty v poli L/T jsou využívány jako označení typu. Nejčastěji se setkáme s identifikacemi typu:

800_H - protokol IP,

806_H - protokol ARP a

8137_H - protokol IPX.

Běžnější protokol *LLC1* pouze odlišuje nepotvrzované rámce přenášených dat ($cntr = 03_H$) od rámců služebních (ty dovolují např. testovat provozuschopnost spoje). Protokol *LLC2* je okénkové potvrzovací schéma obdobné tomu, které jsme poznali u protokolu LAPB X.25 s tím, že jediný oktet pole *cntr* postačí pro číslování modulo 8, zatímco číslování modulo 128 vyžaduje oktety dva. Konečně, protokol *LLC3* poskytuje střídavé potvrzování bez nutnosti navazovat spojení.

Protokoly IEEE 802.2 LLC nedovolují rozlišit typy přenášených paketů definované pro Ethernet II. Řešením je protokol IEEE 802.2 SNAP (SubNetwork Access Point), který dovoluje přenést v hlavičce rámce jednak informaci o délce paketu L, jednak informaci o typu T. Opírá se o identifikaci protokolové sady ($dsap = ssap = AA_H$) a nepodporuje potvrzování na linkové úrovni ($cntr = 03_H$).

Konečně, linkový protokol označovaný jako IEEE 802.3 je využíván pro přenos IPX paketů v sítích Novell. Odlišení jeho rámců od rámců IEEE 802.2 se opírá o nevyužívané pole kontrolního součtu na začátku paketu. Ve všech přenášených IPX paketech má toto pole hodnotu $FFFF_H$ a lze ho odlišit od protokolů, které mají $dsap = ssap \neq FF_H$.

Linkový protokol IEEE 802.2 LLC je definován společně pro řadu technologií lokálních sítí, setkáme se s ním tedy i u sítě Token Ring nebo FDDI. V obou případech je však formát rámce doplněn ještě o další pole, která dovolují přenos řídit a zajistit prioritní přístup k médiu pro přenášené rámce.

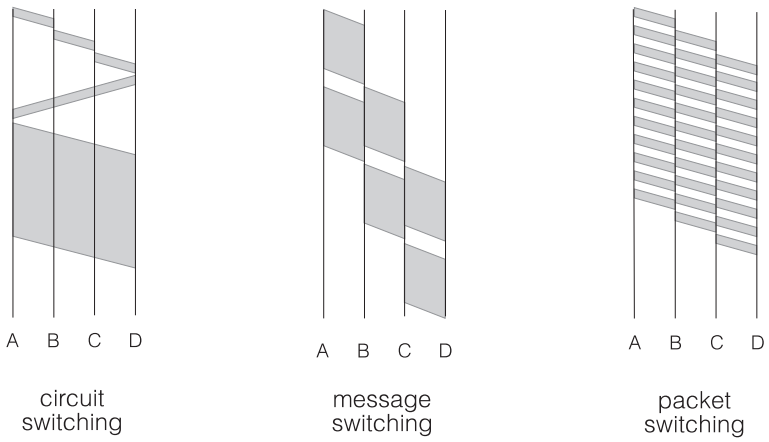
6. Síťová vstva

Dvoubodové komunikační kanály obsluhované procedurami linkové vrstvy lze využít k vybudování polygonální přepojovací sítě. Jednotlivé dvoubodové linky speciální komunikační počítače — *přepojovací uzly* (switching node, router), které dovolí vytvořit z jednotlivých navazujících linek cestu pro data přenášená mezi koncovými účastníky. Těmi jsou jednak výkonné počítače — servery, jednak koncová účastnická zařízení — terminály. Podle způsobu, jakým přepojovací síť zajišťuje přenos dat, mluvíme o přepojování kanálů, přepojování zpráv a přepojování paketů.

Přepojováním kanálů rozumíme vyhrazení linek na cestě mezi koncovými účastníky spojení. Metoda je obdobou spojování v telefonní síti, přepojovací uzel propojuje jednotlivé linky tak, že konečným výsledkem je vybudování dvoubodového spoje mezi koncovými účastníky. Přepojování kanálů je základem přepojovacích sítí podle doporučení X.21 CCITT a sítí ISDN.

Přepojování zpráv a *přepojování paketů* se od přepojování kanálů principiálně liší. Data přenášená mezi koncovými účastníky jsou rozdělena do *zpráv* nebo do délkou omezených fragmentů zpráv — *paketů* a jsou přenášena postupně mezi jednotlivými uzly na cestě mezi koncovými účastníky. Uzly přijaté zprávy, resp. pakety, ukládají do své paměti, určují nejvýhodnější směr jejich další cesty k adresátovi a postupně je po lince vedoucí zvoleným směrem odešlou. Metodu lze přirovnat k funkci listovní pošty, bývá označována jako metoda *store-and-forward*.

Následující obrázek 6.1 porovnává mechanismy přepojování kanálů, zpráv a paketů. (U přepojování zpráv a paketů je předpokládána datagramová služba, u virtuálních kanálů vlastnímu přenosu předchází budování virtuálního spoje).



Obrázek 6.1: Přepojování kanálů, zpráv a paketů

Při své činnosti se jednotlivé uzly sítě snaží nalézt pro komunikaci uživatelů sítě co nejkratší cestu. Cílem je nejen co nejmenší zpoždění zpráv (paketů), ale také minimalizace vlastního zatížení sítě. Uzly se při směrování opírají o informace o nejvýhodnějších cestách k cíli, většinou ve formě směrovacích tabulek. Tyto informace mohou být pro zadanou topologii předdefinovány nebo modifikovány podle okamžitého stavu a zatížení sítě. Modifikace přitom mohou určovat buď specializované počítače v síti (směrovací centra) nebo samotné směrovací uzly.

Směrování a vytváření datových struktur (tabulek) potřebných pro směrování je klíčovou funkcí síťové vrstvy přepojovacích sítí. V praxi je nutné doplnit mechanismy, které zabraňují lokálnímu nebo globálnímu přetížení sítě, mluvíme o mechanismech *řízení toku*.

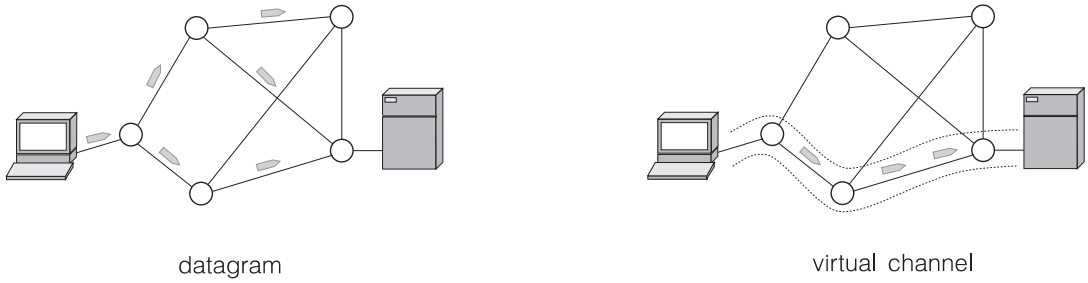
6.1 Datagram a virtuální kanál

Zprávy (resp. pakety), lze v síti s přepojováním zpráv (resp. paketů) doručovat dvěma odlišnými způsoby:

-každou jednotlivou zprávu (paket) označit adresou cílového účastníka a předávat ji polygonální sítí nezávisle na předávání ostatních zpráv (paketů),

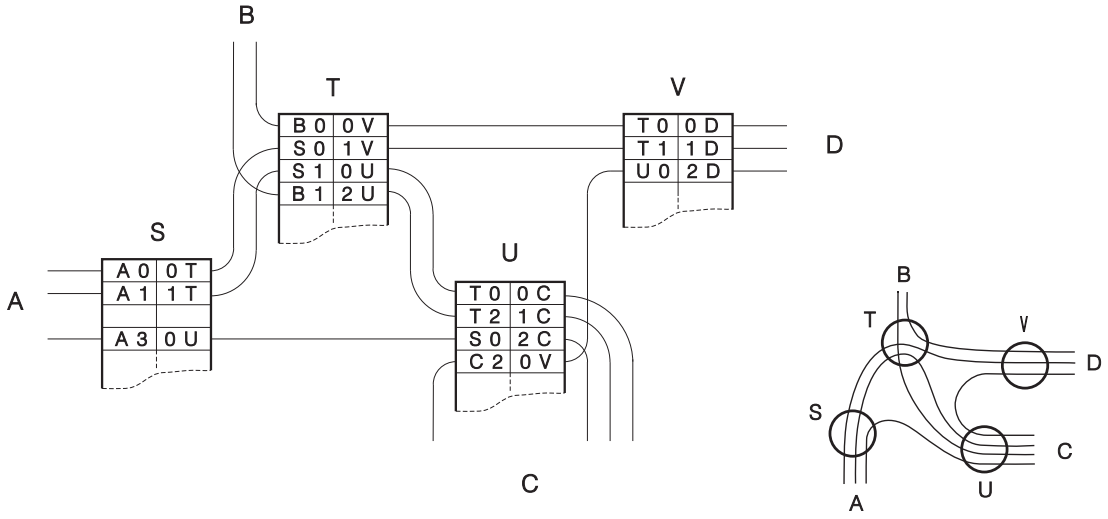
-při otevírání spojení koncových účastníků uložit v paměti přepojovacích uzlů poznámku o zvolené cestě, a pouze tuto cestu cestu pak využívat pro další komunikaci.

V prvém případě mluvíme o *datagramové službě*, ve druhém případě o službě *virtuálních spojů* (kanálů).



Obrázek 6.2: Datagramová služba a virtuální spoj

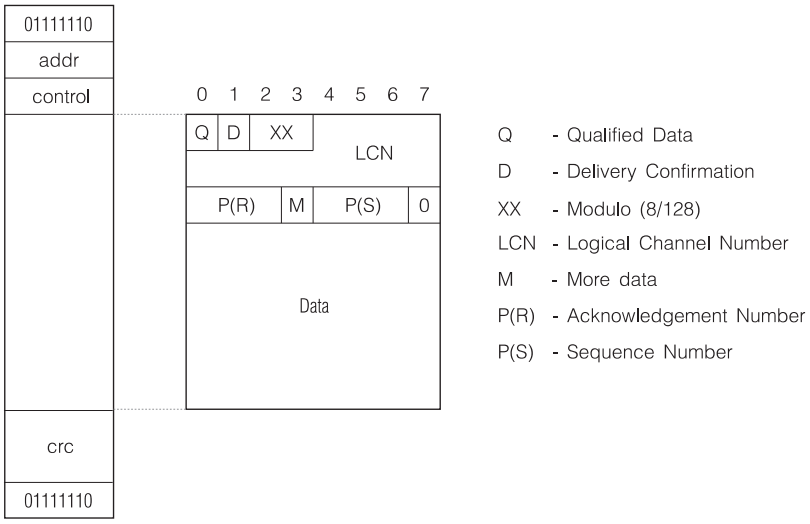
Datagramem nazýváme paket, který obsahuje kompletní adresu adresáta a je směrován nezávisle na ostatních paketech příslušejících komunikaci dvou účastníků (obr. 6.2). Datagramová služba byla navržena s cílem zajistit přenos dat v sítích s předpokládanými výpadky linek a uzlů. Datagramová síť je schopna rychle se přizpůsobovat nejen změnám topologie (při výpadku prvků) ale i změnám zátěže. Její nevýhodou je, že nezachovává pořadí paketů, a že výpadky prvků sítě (případně ochranné mechanismy řízení toku) způsobují ztráty paketů. Je proto nutné ji doplnit o služby vyšší — *transportní vrstvy*, které tyto nepříjemné vlastnosti překryjí, ale které jsou poměrně implementačně složité.

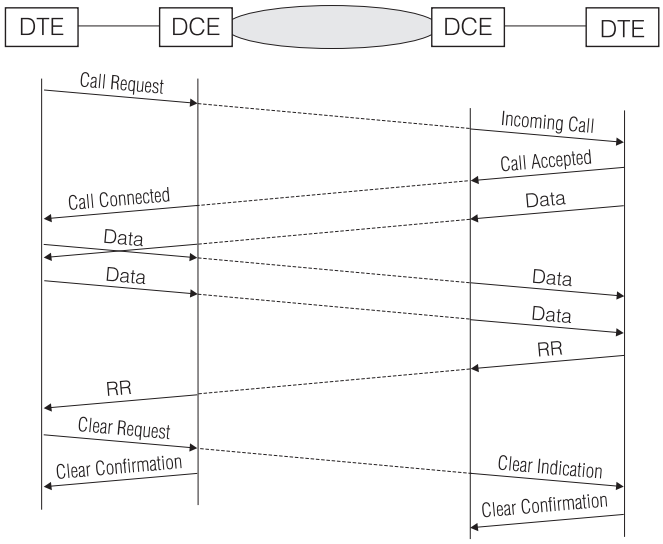


Obrázek 6.3: Virtuální kanál, přepojovací tabulky, struktura paketu

U sítí s virtuálními spoji procházejí všechny pakety příslušející komunikaci dvou účastníků po stejné cestě. Výsledkem je zachování pořadí přenesených paketů, ztráty paketů se omezují na snadno detekovatelné situace (ke ztrátě může dojít při výpadku některého prvku ležícího

na zvolené cestě nebo při vynuceném rozpojení virtuálního kanálu). Z hlediska síťové vrstvy je realizace služby virtuálních kanálů složitější než datagramová služba. Virtuální spoj je třeba navázat, paket otevírající spojení za sebou nechává v přepojovacích uzlech stopu — položku v přepojovacích tabulkách uzlů. Této informace potom uzel využívá při směrování ostatních paketů, které již nemusejí obsahovat adresu adresáta. Virtuální spoj je konečně nutno zrušit při uzavření spojení koncových účastníků — smazat položky v přepojovacích tabulkách uzlů. Možnou strukturu přepojovacích tabulek a jejich využití při vytváření virtuálních spojů ilustruje obrázek 6.3.





Obrázek 6.4: Navazování spojení a přenos dat v síti X.25

Linka zabezpečená procedurou řízení je při výstavbě virtuálních kanálů využita vícenásobně. Uzel identifikuje příslušnost přenášeného paketu ke konkrétnímu virtuálnímu kanálu na lince podle logického čísla kanálu v záhlaví paketu. Logická čísla jsou kanálům přidělována při jejich otevírání. Uzel po příchodu speciálního řídícího paketu, který odpovídá žádosti o otevření kanálu, vybere vhodný neobsazený výstupní kanál ve směru určeném směrovací strategií, a logická čísla příchozího i odchozího virtuálního kanálu si poznamená do přepojovací tabulky. Pro další pakety přicházející s daným logickým číslem kanálu pak jednoduše podle přepojovací

tabulky zamění logické číslo kanálu a paket předá k odeslání do výstupní linky. Strategie FIFO ve frontě této linky zajistí, že celá síť zachovává pořadí paketů. Příklad otevření virtuálního kanálu, přenosu dat a uzavření virtuálního kanálu pro veřejnou datovou síť X.25 uvádí obrázek 6.4.

K datagramovým sítím patřila síť ARPANET, patří k nim firemní síť DECNet (DEC) i celosvětová síť Internet. Služby virtuálních kanálů nalezneme u veřejných datových sítí, které odpovídají doporučením ITU-T (CCITT) X.25 a ISDN Frame Relay , u technologie ATM.

6.2 Metody směrování

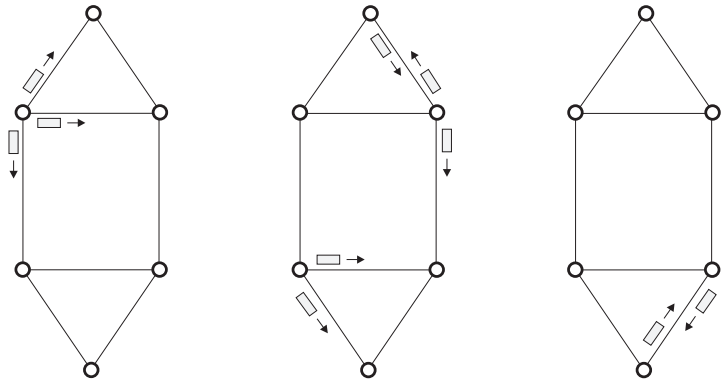
Cesta datagramu nebo paketu otevírajícího virtuální kanál je určována v jednotlivých uzlech přepojovací sítě. Výstupní linku, po které bude paket odeslán, volí uzel na základě informací o topologii sítě a zatížení jednotlivých linek, na základě lokálních stavových informací a/nebo na základě informací přenášených přímo směrovanými pakety.

Úkolem směrování je dosáhnout rychlého průchodu paketu sítí (minimalizovat zpoždění paketu) i při poruchách linek nebo uzlů. Samotná optimalizace směrování při známém rozložení toků v síti je komplikovaná záležitost , požavek rozumné implementace a nutně neúplné informace o okamžitém stavu sítě při proměnlivé zátěži vedou na používání zjednodušených postupů.

6.2.1 Záplavové směrování

Nejjednodušším směrovacím postupem, který má navíc tu příjemno vlastnost, že vždy využije nejkratší cestu, je záplavové směrování. Uzel přijatý paket (pokud není adresován jemu) rozešle do všech linek s výjimkou té, z níž paket přijal. Nebere přitom ohled na jakoukoliv další informaci o adrese adresáta, topologii sítě nebo vlastní stav.

Výhodou metody je vysoká spolehlivost, nevýhodou nadbytečné zatížení sítě kopiemi paketu a nutnost kopie likvidovat. Je vhodná pro sítě s neznámou nebo proměnlivou topologií (vojenské aplikace) a v situacích, kdy je nutné rychle dopravit shodnou informaci všem uzlům (rozesílání zpráv o změnách topologie, časová synchronizace).



Obrázek 6.5: Záplavové směrování

Další metodou, která nebere při směrování v úvahu jinou informaci než informaci o směru, odkud paket přišel, je náhodné směrování. Uzel vybírá výstupní linku náhodně, metoda je odolná proti změnám topologie, nezajišťuje však omezenou dobu doručení paketu. Při částečné znalosti topologie lzed výběr výstupního směru omezit, taková modifikace náhodného směrování je již v praxi užitečná.

6.2.2 Izolované směrování

Nejjednodušší z metod, které se opírají pouze o lokální stavovou informaci (a neberou v úvahu informace, které mají ostatní uzly sítě), a které označujeme jako metody izolované je metoda "horkého bramboru".

Uzel sítě při rozhodování o směru, ve kterém odešle přijatý paket bere v úvahu pouze délky front na výstupních linkách (a jejich kapacitu). Paket zařadí do nejkratší fronty (vylučujíc frontu ve směru, odkud byl paket přijat), tedy frontu, kde bude paket nejdříve odeslán (respektujeme různé délky paketů a kapacity linek). Metoda samotná nezaručujje doručení paketu v konečném čase, v praxi je užitečná jako součást kombinovaných metod (delta směrování, Transpac).

Zajímavou metodou je "metoda zpětného učení", u které se uzel při odhadu směru nejkratší cesty opírá o informace přenášené pakety přicházejícími z jednotllivých uzlů sítě. Pro funkci metody je podstatné, že každý paket přenáší jako služební informaci údaj o době svého dosavadního putování sítí (v podobě globálního časového údaje o čase odeslání, nebo v podobě počtu dosud navštívených uzlů). Uzly, kterými paket prochází, si údaj o trvání cesty poznamenávají do tabulky, indexované odesílatelem uzlu, spolu s údajem o směru, ze kterého paket přišel. Poznámku upravují při zjištění, že přicházející paket prošel cestou kratší, než byla cesta dosud zaznamenaná. Vytvářená tabulka odpovídá směrovací tabulce v dále uváděných metodách.

Nepříjemností metody je, že je "optimistická", nereaguje na zhoršení situace v síti, nebo je její reakce velmi pomalá. Pro praktické použití je nutné ji doplnit o "zapomínání", např. ovlivňováním poznámek v tabulce pakety, které prošly delší cestou. Se zjednodušenou formou metody se setkáme při směrování v lokálních sítích propojených mosty.

6.2.3 Statické směrování

Při statickém směrování se uzel opírá o směrovací tabulku, ve které je pro každý cílový uzel určen směr, ve kterém k němu vede nejkratší cesta. Tabulka je vytvářena při návrhu sítě, na základě předpokládaných údajů o topologii, kapacitách linek a zatížení .

Nevýhodu statického směrování, která spočívá v tom, že v případě výpadku linky nemá uzel možnost rozumně zvolit náhradní cestu, odstraňuje uvedení alternativních směrů ve směrovací tabulce. Alternativní cestu uzel volí pouze při skutečném výpadku (např. DECNET), nebo i při silném zatížení směru nejvýhodnějšího (SNA IBM).

Optimalizace rozdělení toků v síti obvykle vyžaduje rozdělení jednotlivých toků do různých větví. Takovému rozdělení by optimálně vyhovovalo stochastické směrování, u kterého je pro každý cílový uzel a pro každý odchozí směr určena pravděpodobnost toho, že uzel tímto směrem paket odešle. Na tuto spíše teoretickou metodu lze pohlížet jako na přechod mezi náhodným a statickým směrováním.

6.2.4 Adaptivní směrování

Statické směrovací tabulky optimalizují chování sítě pouze pro určité rozložení datových toků. Zátěž sítě se však v praxi rychle mění, výpadek linky nebo uzlu může optimální rozložení toku podtatně ovlivnit. Uvedené vlivy nelze respektovat jinak než opakovaným výpočtem směrovacích tabulek za provozu sítě.

Postup, při kterém sledujeme okamžitý stav sítě (změny topologie, zpoždění na linkách) a počítáme nové směrovací tabulky, nazýváme adaptivním směrováním. Centralizovaná forma adaptivního směrování (se směrovacím centrem) je sice možná, ale s ohledem na výpadky centra (a nutnost jeho zálohování) a s ohledem na značné zatížení sítě v oblasti centra přenosem

stavových informací a distribucí směrovacích tabulek není nejvýhodnější. Centrální výpočet směrovacích údajů používá například počítačová síť TYMNET.

Přídavné zatížení sítě lze minimalizovat, omezíme-li frekvenci přepočtu tabulek. Ty pak ale nerespektují okamžité zatížení sítě. Dobrým kompromisem je kombinovat občasný globální výpočet tabulek s využitím lokální informace o zatížení sítě.

Jako příklad kombinované metody si uvedeme delta směrování: Každý uzel v síti ohodnotí své výstupní linky (např. délkou fronty, zpožděním paketu) a předá tuto informaci směrovacímu centru. To pak vypočte pro každý uzel délku nejkratší cesty přes každého z jeho sousedů. Z této tabulky (mohla by v plné formě sloužit například pro statické směrování s alternativami) pak vybere ty sousedy, pro které je určená délka větší než minimum nejvýše o zadaný parametr δ (počet vybraných sousedů je možné dále omezit zadaným číslem k). Výsledné tabulky směrovací centrum rozešle, směrovací uzly považují směry uvedené v tabulce za rovnocenné a vybírají si konkrétní směr podle stavu linky (odhadované zpoždění).

Metoda pro $\delta \to 0$ přechází ve statické směrování, pro $\delta \to \infty$ přechází v metodu "horkého bramboru".

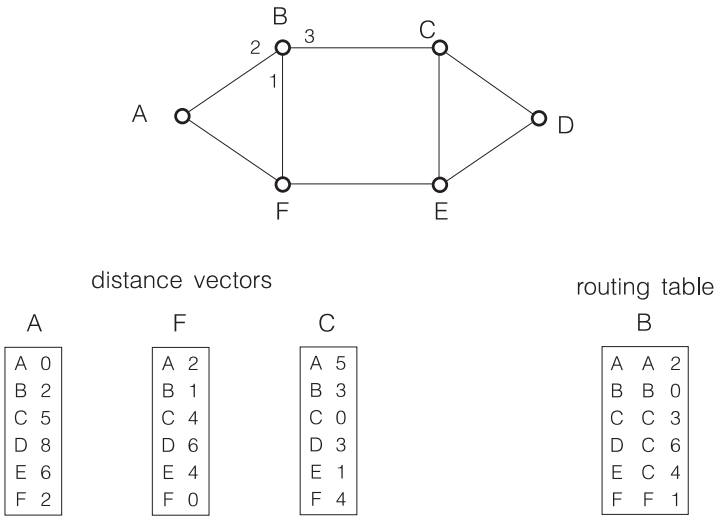
Podobné metody používají některé veřejné datové sítě X.25. Ve směrovací tabulce, která se využívá pouze při otevírání virtuálních kanálů, jsou pro každý cíl výstupy rozdéleny na primární a sekundární. Virtuální kanály jsou nejdříve otevírány na nejméně obsazených primárních výstupech, při jejich naplnění na výstupech sekundárních. Tabulka je téměř statická, přepočítává se jen při změnách topologie, vstupními údaji jsou kapacity linek.

Poněkud jemněji kombinuje informace z centrálně stanovených směrovacích tabulek a lokální informace metoda použitá ve veřejné datové síti TRANSPAC. Na základě vypočtených "vzdáleností" od svých sousedů k cíli a na základě svého odhadu "vzdálenosti" k sousedům určuje minimum součtů těchto dvou složek a odpovídající směr volí jako nejvýhodnější. "Vzdálenost" k sousedům je odvozena z počtu obsazenych virtuálních kanálů a délek front.

Distance-vector algoritmus — RIP

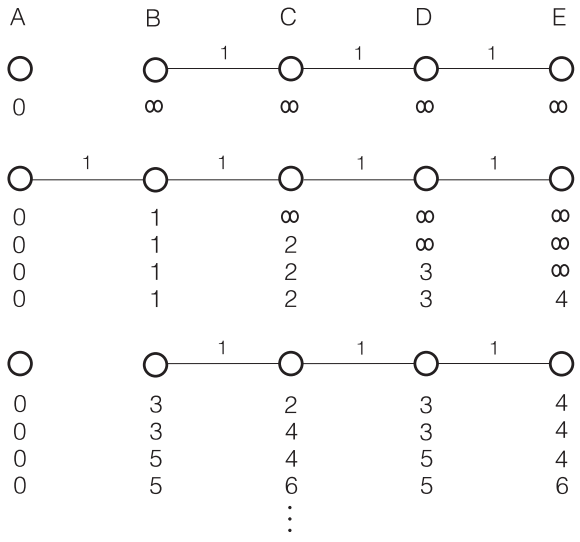
Ford-Fulkersonův algoritmus, který jsme si uvedli na str.21 lze přímočaře implementovat v síti propojených počítačů. Krok 2 algoritmu odpovídá výměně dosud zjištěné hodnoty L se sousedy. V praxi budeme pracovat s vektorem vzdáleností ke všem uzlům sítě, algoritmus vytváří přímo směrovací tabulky. Pod jménem *Distance-Vector* algoritmus je využíván v jednodušších autonomních systémech Internetu (RIP - Routing Information Protocol).

Pro výpočet směrovacích tabulek se s výhodou používá distribuovaného výpočtu. Uzly si v pravidelných intervalech vyměňují informace (odhady) o svých vzdálenostech k ostatním uzlům sítě (obr. 6.6). Kombinací přijatých vektorů vzdáleností a znalosti o vzdálenostech sousedů lze vybudovat směrovací tabulky. Přestože je perioda výměn poměrně dlouhá, může být režie mechanismu pro sítě s velkým počtem uzlů a pomalé linky mezi uzly příliš vysoká.



Obrázek 6.6: Distance-vector algoritmus pro výpočet směrovacích tabulek

Neupravený algoritmus má nepříjemnou vlastnost, reaguje rychle na *"dobré zprávy"* ale pomalu na *"špatné zprávy"*, navíc může vyvolat směrování paketů v cyklech. Problém lze ilustrovat na následujícím příkladě, který uvádí změny vzdáleností (určených tímto algoritmem) k uzlu A od ostatních uzlů sítě po připojení uzlu A k síti a po jeho odpojení od sítě (obr. 6.7).



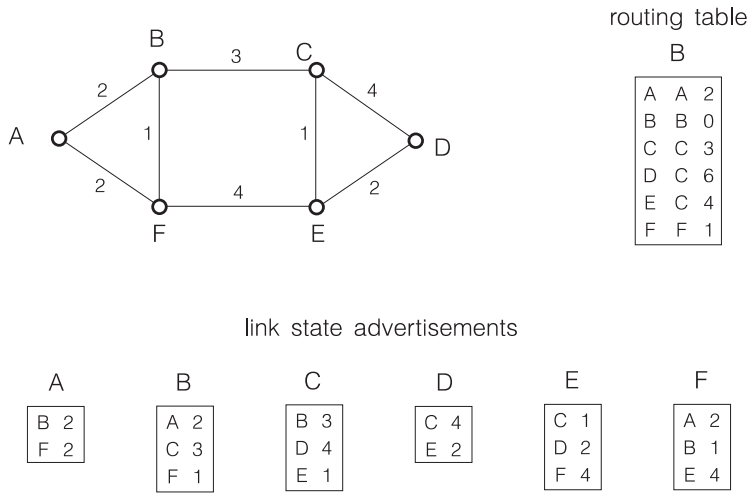
Obrázek 6.7: Reakce na připojení a odpojení uzlu A

Ke zjištění, že uzel A není propojen se zbývajícími uzly sítě lze dospět rychleji při vhodné volbě *"nekonečna"*. Algoritmus RIP (Routing Information Protocol) sítě Internet považuje za *"nekonečno"* hodnotu 15 (hran mezi uzly).

Do okamžiku, než všechny uzly zjistí výpadek linky (dopočítají do patnácti), může docházet k cyklickému směrování paketů (v našem příkladě například mezi uzly B a C). Zjištění výpadku linky lze urychlit, pokud uzel C, který získal informaci o nejkratší cestě k uzlu A od uzlu B, svůj údaj uzlu B zpět nepředá (tato úprava algoritmu je označována jako *split horison*). Dalšího urychlení lze dosáhnout, pokud uzel C předává uzlu B v takovém případě údaj o nekonečné vzdálenosti (tato úprave je označována jako *inverse poisson*). nekonečné vzdálenosti () s uzlem A nebude uzlu C indikovat pokud uzel nepředává informaci o vzdálenosti při ztrátě spojení se lze bránit například *"setrvačností"* směrovacího algoritmu, obrácení smyslu směrování po lince dovolíme až po určitém počtu kroků.

Link-state algoritmus — OSPF

Velice výhodnou metodou výpočtu směrovacích tabulek je postup opírající se o broadcastem šířené informace o stavech linek, s nimiž jednotlivé uzly sítě sousedí. Takové informace (*LSA — Link State Advertisement*), které si uzly ukládají ve směrovací databázi, dovolí jednotlivým uzlům vytvořit si úplný obraz sítě (obr. 6.8). Metoda je základem postupu *OSPF* (*Open Shortest Path First*), který je široce využíván v Internetu; pro výpočet směrovací tabulky lze s výhodou použít Dijkstrova algoritmu.



Obrázek 6.8: Link-state algoritmus pro výpočet směrovacích tabulek

Určitým problémem distribuce LSA paketů je nutnost zaručit jejich příjem ve správném pořadí. Toho lze dosáhnout jejich číslováním, které dovolí starší pakety ignorovat. Ignorování starších LSA paketů přináší jiný problém: tím je restart směrovače, po něm by mohly být jím rozesílané LSA pakety ignorovány. Tomu se sice lze bránit omezením časové platnosti rozesílaných údajů a mechanismem, který dovolí jejich rychlé smazání v celé síti (rozesláním LSA paketů s nulovou časovou platností); dosáhnout však rozumné rovnováhy mezi rychlostí reakce na výpadek uzlu (teprve po vypršení časového limitu smí směrovač znovu zahájit provoz) a periodou opakování LSA paketů může být pro rozsáhlé sítě obtížné.

Restart směrovače lze detekovat, pokud zajistíme, že číslování začíná s hodnotami, ke kterým se již při modulo inkrementaci nevracíme (např. startujeme s nejzápornější hodnotou, ale po překročení největší kladné hodnoty se vracíme k nule. Libovolný směrovač v síti, který takovou situaci detekuje rozešle LSA pakety s nulovou časovou platností a tím současně informuje restartovaný směrovač o posledním použitém čísle LSA paketu.

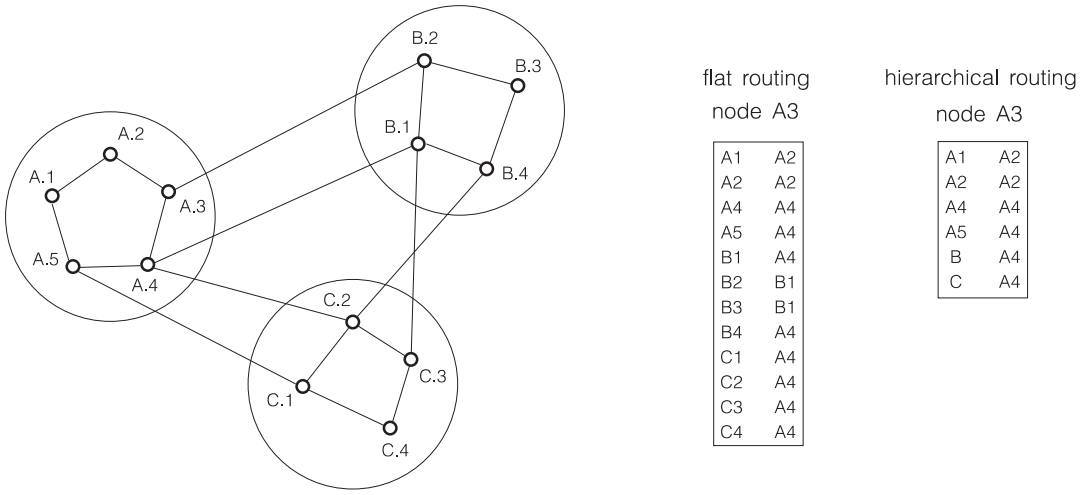
Mechanismus rozesílání LSA paketů dovoluje zrekonstruovat informace o topologii sítě i po jejím rozpadu na izolované části a pozdějším propojení (případně po propojení dosud izolovaných sítí). Pro urychlení je doplněn o výměnu informací ze směrovacích databází po propojení izolovaných částí.

Detekci provozuschopnosti jednotlivých linek, o kterou se opírá rozesílání LSA paketů, podporuje jednoduchý protokol *Hello*. Jím se sousední směrovače vzájemně informují o své vlastní provozuschopnosti a o provozuschopnosti spoje mezi nimi. Případné zvláštní situace, které může způsobit selhání protokolu *Hello* (například vysílání *Hello* paketů s chybnou adresou odesílatele, které může zakrýt výpadek jiného směrovače) lze ošetřit stárnutím údajů ve směrovacích databázích.

Konečně, protokol OSPF podporuje dvouúrovňový hierarchický model sítě; rozsáhlejší sítě lze pak složit z více sítí vzájemně propojených sítí páteřní.

6.2.5 Hierarchické směrování

Předpokládáme-li přepojovací síť s N uzly a E hranami, má výpočet směrovací tabulky pro jeden uzel výpočetní složitost $O(ElogE)$ a směrovací tabulka má rozměr N položek. Takové hodnoty pro opravdu rozsáhlé sítě (jakou je např. Internet) složitost výpočtu a paměťové nároky překračují rozumné meze. Cestou, jak redukovat rozměr směrovacích tabulek a snížit složitost výpočtu je rozdělit síť na oblasti a využít hierarchického směrování (obr. 6.9).



Obrázek 6.9: Hierarchické směrování

Hierarchické směrování se opírá o rozdělení adresy na *prefix* označující oblast, a na adresu uzlu uvnitř oblasti. Hierarchické směrování předpokládá, že rozdělení uzlů do oblastí respektuje topologii (oblasti jsou souvislými podgrafy sítě).

Příkladem sítě, která používá hierarchického směrování je Internet. Adresa je složena z adresy sítě (7 bitů pro třídu A, 14 bitů pro třídu B, nebo 21 bitů pro třídu C) a adresy počítače. Pro rozsáhlejší sítě (všech tří tříd) může být potřebné jemnější členění na podsítě. Rozhraní mezi identifikací počítače v podsíti a prefixem je pak určeno bitovou *maskou*, případně délkou prefixu v bitech.

6.2.6 Jmenná služba

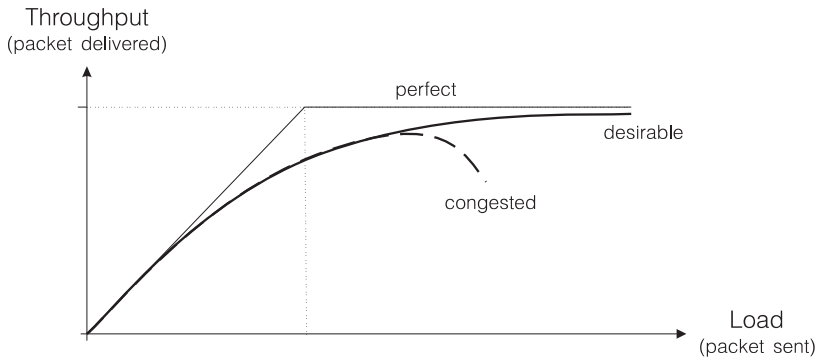
Označení počítačů číselnými adresami a respektování hierarchické adresace komplikuje identifikaci jednotlivých počítačů koncovým uživatelům/aplikacím. Podstatné zjednodušení přináší jmenná služba *DNS* (Domain Name Service).

Služba DNS je založena na jednoduché myšlence: Každý počítač v určité skupině (*domain* – jmenná doména) je jednoznačně pojmenován a korespondence jména s číselnou adresou je zapsána do adresáře dostupného na známé adrese (*primární/autoritativní DNS server*), který je případně zálohován (*sekundární DNS server*). Zjištění adresy počítače ve skupině je převedeno na dotaz v adresáři, jednou zjištěnou korespondenci jména a adresy si lze po určitou dobu lokálně uschovat (*DNS cache*). Poznamenejme, že rozmístění počítačů jmenné domény v síti může zcela ignorovat rozdělení sítě na podsítě.

Služba DNS je hierarchická, jednoznačně pojmenované domény (a jednotlivé počítače) lze sdružovat do domén vyšší úrovně, jména počítačů v doménách jsou pak tvořena posloupností jmen *počítač.doména.doména.....* Domény nejvyšší úrovně jsou spravovány na národní úrovni (např. cz, de, at), vedle toho existují domény oborové (např. edu, com, mil, net).

6.3 Řízení toku

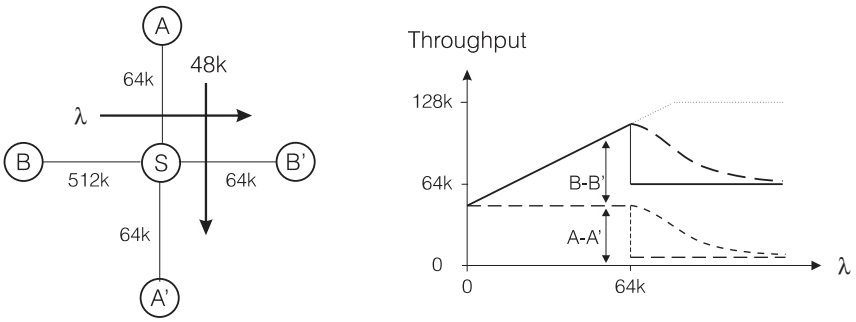
Kapacita přepojovací sítě, jako schopnost přenosu datových toků mezi koncovými uzly, je omezena kapacitami spojů a topologií sítě. Maximální propustnosti je však poměrně obtížné v praxi dosáhnout, závislost propustnosti přepojovací sítě na zátěži bez použití mechanismů řízení toku může odpovídat obrázku obr. 6.10.



Obrázek 6.10: Průchodnost přepojovací sítě

K poklesu propustnosti dochází v důsledku překročení konečných limitů pro délky front v uzlech sítě, vyčerpání kapacity paměti uzlu označujeme jako *zahlcení*.

Důsledky vyčerpání kapacity front na průchodnost sítě si můžeme ilustrovat na jednoduchém příkladě (obr. 6.11).



Obrázek 6.11: Zahlcení sítě a ovlivňování toků

Pakety, které nelze v uzlu uložit, můžeme při vyčerpání paměti odmítat (například tak, že budeme sousedním uzlům posílat negativní potvrzení). Důsledkem této strategie a důsledkem různých kapacit spojů (pravděpodobnost převzetí paketu ze spoje $B - S$ je osmkrát vyšší než pravděpodobnost převzetí paketu ze spoje $A - S$) je skokový pokles průchodnosti sítě při překročení limitního toku $B - B'$ a vzájemné ovlivňování toků (tok $A - A'$, který nepřekračuje možnosti sítě je ovliňován tokem $B - B'$). Jednoduché *odmítání paketů* má v rozsáhlejších sítích, než je náš příklad, nepříjemný důsledek: sousedním uzlům se snižuje kapacita výstupních linek. To může vést k jejich následnému zahlcení, zahlcení tak může rozšířit na celé oblasti sítě.

6.3.1 Zahazování paketů

Lepší výsledky než odmítání přináší *zahazování* paketů. O zopakování zahozeného paketu se musí postarat *koncové potvrzování*, delší časová odezva koncového potvrzování omezí pokles celkové průchodnosti sítě i vzájemné ovlivňování toků. Současně se tak omezí nebezpečí šíření

zahlcení.

Zahazování paketů je rozumné zahájit dříve, než dojde k úplnému vyčerpání paměti. V takovém případě lze využít informaci, vkládanou do paketu koncovým mechanismem řízení toku. Označování paketů, které je možné zahazovat při překročení kritické zátěže, využívají technologie Frame Relay a ATM.

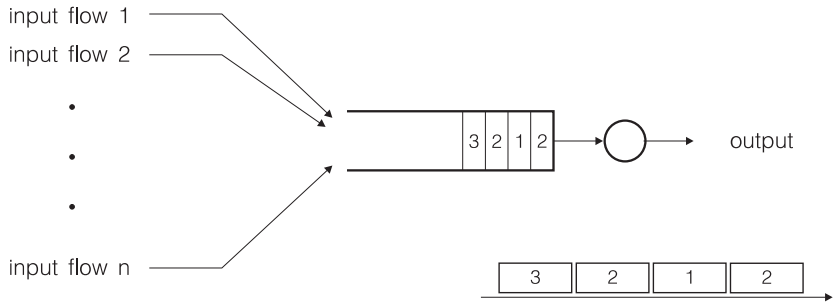
Chování směrovače při vysoké zátěži lze ovlivnit způsobem přidělování paměti jednotlivým výstupním frontám. Náš jednoduchý příklad předpokládal, že neomezujeme délky jednotlivých front (přesněji, na jedinou frontu můžeme věnovat veškerou paměť). Opačný extrém, kdy určíme limity délek front tak, že jejich součet odpovídá kapacitě paměti, vyloučí vzájemné ovlivňování toků v našem jednoduchém případě, jeho nevýhodou však je, že zatíženější směr nemůže přechodně využít volnou kapacitu paměti jiného směru. Za nejvýhodnější je považováno takové omezení délky fronty, kdy každému z n směrů je nastaven limit na $1/\sqrt{n}$ kapacity paměti.

6.3.2 Diferencovaná obsluha

Dosud jsme předpokládali, že směrovače se chovají ke všem datovým tokům shodně. V moderních sítích, které přenášejí vedle běžných dat i hovorové a obrazové kanály, je nutné jednotlivé třídy toků obsluhovat diferencovaně.

Režim FIFO

Běžné směrovače mají pro každý výstupní kanál společnou frontu. Nově přicházející pakety jsou ukládány na konec fronty, z ní jsou postupně předávány k odeslání. Strategie výběru paketů je označována jako FIFO (First-In First-Out) — (obr. 6.12).



Obrázek 6.12: Obsluha FIFO

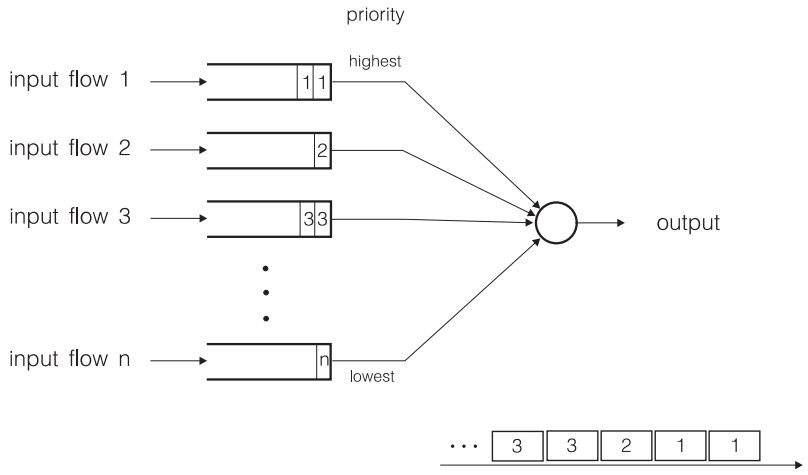
Uvedené řešení je sice jednoduché, ale má jednu podstatnou nevýhodu. Zpoždění způsobené čekáním ve frontě závisí (nelineárně) na čerpání přenosové kapacity výstupního kanálu. Datové toky se vzájemně ovlivňují, zvětšení jednoho toku zvyšuje dobu přenosu u toků ostatních. V krajním případě, při překročení kapacity výstupní linky a s tím souvisejícím vyčerpáním paměti směrovač přicházející pakety zahazuje.

Zahazování paketů má nepříjemné důsledky, a to nejen pro datové toky, které s ohledem na časové limity nemohou využít koncového potvrzování (např. počítačová telefonie). Mechanismy koncového potvrzování (speciálně TCP) na ztracené pakety reagují snížením toku, časové zpoždění a vzájemná synchronizace reakcí na zahlcení mohou celý systém rozkmitat a snížit využitelnost přenosových kapacit.

Problémy spojené se zahazováním paketů při vyčerpání paměti lze zdánlivě řešit zvýšením její kapacity. Důsledkem však mohou být delší fronty a tomu odpovídající větší zpoždění přenosu.

Prioritní obsluha

Zvýhodnění některých datových toků proti ostatním lze dosáhnout rozdělením společné výstupní fronty. Jednotlivé fronty sice mohou příslušet jednotlivým tokům, rozumnější je však přiřadit fronty třídám toků (obr. 6.13). Prioritní režim obsluhy front dovolí zvýhodnit některé třídy toků. Lze tak poměrně jednoduše respektovat požadavek zvýšené kvality služeb pro konkrétní třídu(y) (např. pro přenos hovorových signálů).



Obrázek 6.13: Prioritní obsluha

Výhodou prioritní obsluhy je jednoduchá implementace, nevýhodou je zvýšená citlivost na překročení požadavků u prioritních datových toků (datové toky s nižší prioritou mohou být při zahlcení zcela zastaveny). Nevhodná klasifikace toků (např. nízká priorita protokolů správy sítě) může vést k závažným důsledkům (vyřazení mechanismů správy sítě).

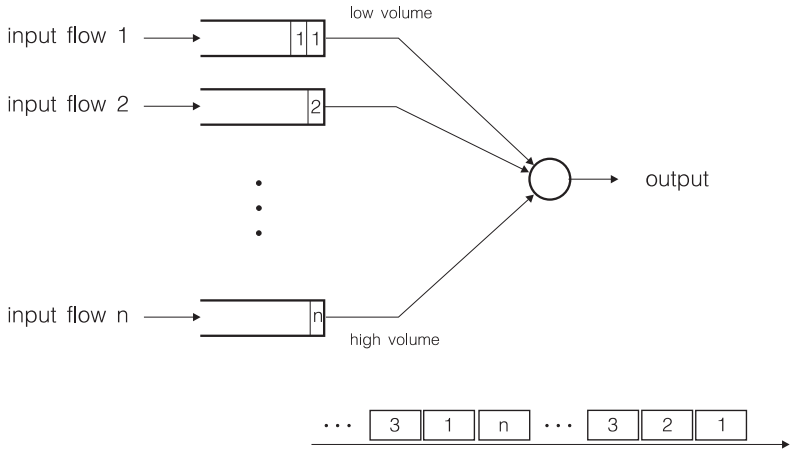
Prioritní obsluha front je pro svou jednoduchost využívána hlavně u lokálních sítí (IBM Token Ring, FDDI, Ethernet - doporučení IEEE 802.1p). Odlišení toků s různou prioritou se opírá o údaj v hlavičce rámce/paketu (doplňkové pole určující prioritu v hlavičce 802.1p, nebo pole TOS v hlavičce IP paketu).

Obsluha WFQ

Nevýhodu absolutního zvýhodnění některé třídy toků lze potlačit mechanismem, který omezí podíl konkrétní třídy toků na celkovém výstupním toku směrovače při vysokém zatížení. Mechanismus WFQ (Weighted-Fair Queueing), se opírá o definovaný podíl třídy na kapacitě výstupního kanálu, při jeho vyčerpání je obsluhována další třída (cyklicky) - obr. 6.14.

Mechanismus byl navržen s cílem omezit ovlivňování malých interaktivních toků (např. Telnet) velkými toky (např. FTP), které lze bez problémů pozdržet. Pro odlišení interaktivních toků s nároky na co nejmenší zpoždění od toků s vysokými nároky na přenosové kapacity lze využít bitů pole TOS v hlavičce IP paketu, případně údaj o typu transportního protokolu a čísle portu.

Principiálně jednoduchý mechanismus je pro práci s pakety o proměnné délce poměrně implementačně náročný.



Obrázek 6.14: Obsluha WFQ (Weighted Fair Queueing)

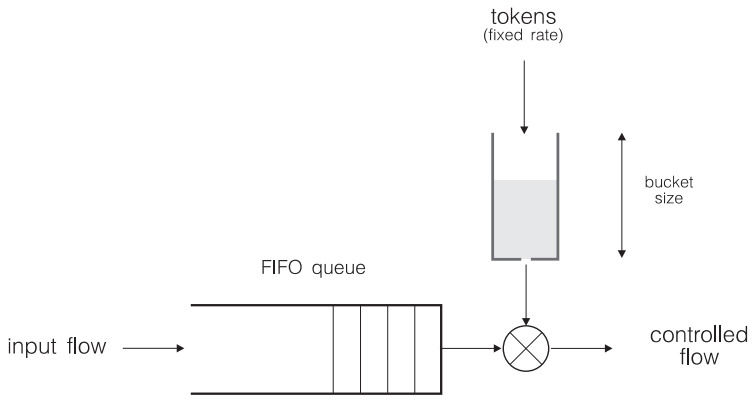
6.3.3 Koncové řízení toků

Výrazně proměnný charakter, který mají mnohé datové toky, může vyvolat přechodné zahlcení v síti. Zlepšení lze dosáhnout vyloučením výrazných špiček v datových tocích, takové vyrovnání požadovaných přenosových kapacit v řadě případů poskytne připojení aplikace k síti kanálem s omezenou kapacitou (např. pevná linka s rychlostí 64 kb/s). Primitivním řešením problému by tedy mohlo být omezení toku do sítě pevně určeným limitem (podobně jako u připojení pomalou linkou), špičky pak musí být ukládány do fronty FIFO.

Nepříjemnou vlastností takového mechanismu je, že zcela vylučuje překročení zvoleného limitu. Přitom uvnitř sítě může, vzhledem k rozptylu zpoždění a interakci mnoha datových toků, dojít k okamžitému zvýšení konkrétního toku nad stanovený limit. Pevné omezení na vstupu je zřejmě příliš přísné, výhodnější by bylo omezit průměrnou rychlost přenosu a dovolit určitou nerovnoměrnost okamžitého toku dat (malé špičky předcházené nevyužitím kapacity kanálu).

Token Bucket mechanismus

Formování vstupního toku, které má požadované vlastnosti, je označováno jako *Token Bucket* (obr. 6.15).



Obrázek 6.15: Mechanismus Token Bucket

Mechanismus je řízen *kredity* (*tokens*), intenzita jejich příchodů odpovídá limitní střední

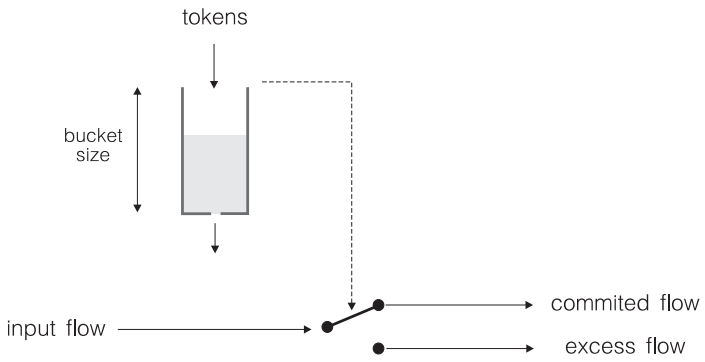
hodnotě vstupního toku. Nevyužité kredity lze ukládat ve *"vědru"*, jeho kapacita určuje velikost povolených špiček. Pakety, pro jejichž přenos nejsou k dispozici volné kredity jsou pozdrženy ve frontě FIFO.

Mechanismem Token Bucket lze realizovat i předchozí primitivní mechanismus. Postačí, má-li *"vědro"* kapacitu jediného kreditu potřebného pro odeslání paketu (pevné délky) a odpovídá-li intenzita příchodu kreditů limitní střední hodnotě vysílání paketů (pevné délky).

Mechanismus formování toku omezuje vstup i v případě, kdy vytížení sítě jinými toky je menší, a kdy by bylo možné přenést i datový tok jdoucí nad limit mechanismu Token Bucket. Dovolují-li uzly sítě bránit se zahlcení zahazováním označených paketů (rámců, buněk), lze do sítě pustit i toky převyšující zadaný limit s tím, že pakety (rámce, buňky) nad tento limit budou v případě kritické zátěže v síti zahazovány.

Leaky Bucket mechanismus

Jako *Leaky Bucket* obvykle je označován mechanismus, u kterého nedochází k formování toku, ale který dovolí jednotlivé pakety vstupního toku klasifikovat jako pakety, které nepřekračují dohodnuté limity (středního toku a akceptovaného chvilkového překročení středního toku) a pakety, které tyto limity překračují (obr. 6.16).



Obrázek 6.16: Mechanismus Leaky Bucket

Obsluha odlišně klasifikovaných paketů závisí na konkrétním systému, zatímco někde slouží informace o překročení limitu k rozhodování o vyhození paketu při vyčerpání kapacit vnitřních uzlů sítě (bit DE u Frame Relay, bit CLP u ATM), jinde mohou být tyto pakety vyloučeny z přenosu již na vstupu sítě.

Mechanismus může být i víceúrovňový, při překročení jednoho limitu jsou pakety odcházející do sítě označovány, při překročení dalšího limitu jsou zahazovány. S takovým postupem se můžeme setkat například u technologie Frame Relay.

Konečně, mechanismus klasifikace (Leaky Bucket) lze kombinovat s mechanismem formování toků (Tokem Bucket). Překročení jednoho limitu vyvolá označování paketů, dalším limitem je omezován tok do sítě.

6.3.4 Zpětnovazební mechanismy

Dosud uváděné mechanismy se opírají o statické limity parametrů vstupních toků. Tyto limity jsou obvykle určeny dlouhodobě kontraktem mezi účastníkem a provozovatelem sítě (např. u sítí Frame Relay), někdy i krátkodobě pro konkrétní dočasné spojení (virtuální kanály SVC sítí ATM). Je na zodpovědnosti provozovatele sítě zajistit, aby i současné požadavky koncových účastníků byly splnitelné, u krátkodobých spojení musí splnitelnost testovat vhodný mechanismus.

Mechanismy klasifikace paketů Leaky Bucket sice dovolují využít okamžitě volnou kapacitu, mohou však vést na výrazné zahazování paketů, které je nepříjemné jak pro koncovou aplikaci, tak pro síť. Je jistě výhodnější omezit tok do sítě, která je silně zatížená, a omezení uvolnit při jejím odlehčení.

Nejstarším zpětnovazebním mechanismem řízení toku je mechanismus *škrtících paketů* (*choke packets*), používaný v sítích X.25. Opírá se o regulaci okénka potvrzovacího schématu koncových účastníků. Hrozící zahlcení uzlu sítě je indikováno koncovým účastníkům škrtícími pakety. Koncová stanice na reaguje na příjem škrtícího paketu přivřením okénka potvrzovacího schématu (na polovinu), pokud po delší dobu škrtící paket nepřijde, je okénko po určitých krocích opět otvíráno.

Indikace zahlcení

Modernější technologie (Frame Relay, ATM) vkládají informaci o průchodu oblastí s kritickou zátěží do hlavičky přenášeného bloku. Například u technologie Frame Relay je přetížení indikováno jednobitovými příznaky FECN (Forward Explicit Congestion Notification) a BECN (Backward Explicit Congestion Notification). První z nich je nastavován v blocích, přenášených kriticky zatíženými linkami uzlů, druhý v blocích jdoucích opačným směrem. Koncové prvky sítě na překročení určitého podílu bloků s nastavenými příznaky reagují snížením toku dat do sítě.

Přesnější řízení toku dovolují mechanismy používané v technologii ATM. U této technologie, která přenáší data v buňkách pevné délky jsou pro řízení toku využívány přídavné *RM buňky* (Rate Management), které dovolují získat aktuální informace o průchodnosti sítě.

Mechanismy zpětnovazebního řízení toku jsou v principu určené pro sítě s virtuálními kanály. V praxi má však jejich použití smysl i u datagramových sítí, u sítí TCP/IP byla specializovanou variantou mechanismu implementována v protokolu TCP s cílem omezit přetěžování sítě. Mechanismus se opírá o předpoklad, že datové toky jsou směrovány téměř staticky a o předpoklad, že za ztráty paketů v síti je zodpovědné primárně zahlcení uzlů. Mechanismus a jeho modifikace si popíšeme v následující kapitole.

7. Transportní vrstva

Služby síťové vrstvy (zvláště u virtuálních kanálů) by nám mohly připadat jako postačující k zabezpečení komunikace mezi počítači. Přesto je nad tyto služby prakticky u všech síťových architektur přidávána vrstva další — *vrstva transportní*. Jejím hlavním úkolem je překrýt nepříjemné vlastnosti některých typů sítí a dát uživateli představu, že má k dispozici opravdu bezchybový kanál odpovídající jeho konkrétním požadavkům, například v tom, že

-data předává ve formátu definovaném aplikací, tedy jako zprávy, a ne ve formátu vyžádaném sítí, tedy jako pakety,

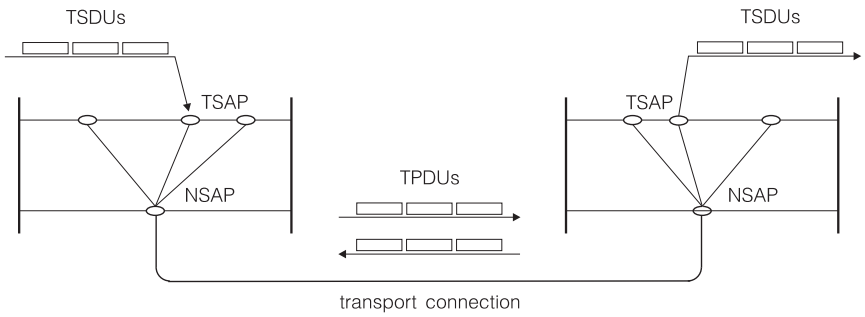
-doručuje aplikační zprávy nejen beze ztrát a duplikací, ale také v nezměněném pořadí.

Navíc transportní vrstva obvykle zajišťuje multiplex — umožňuje nezávislou komunikaci více aplikacím na jednom počítači.

Úroveň síťových služeb v různých sítích je velice rozdílná. Veřejné datové sítě X.25 poskytují virtuální kanály, zachovávají pořadí přenášených paketů a podporují koncové potvrzování. Lokální sítě z principu zachovávají pořadí paketů, ale koncové potvrzování většinou nemají. Chybovost kanálu, která vyvolává ztrátu paketů, je u nich velmi nízká. Naproti tomu datagramové sítě, mezi které můžeme počítat Internet, mohou směrovat jednotlivé pakety různými cestami a narušit jejich pořadí. Pracují obvykle bez koncového potvrzování (strategie *"best effort"*, o vyjímečnosti ztráty paketu se u nich hovořit nedá. Základním úkolem transportní vrstvy je uvedené rozdíly zakrýt a dát uživateli dojem, že má k dispozici ideální komunikační kanál, který přenese všechny pakety bez chyby a v původním pořadí.

Transportní vrstva je obvykle označována jako nejvyšší ze základních síťových vrstev. Její služby jsou standardizovány, její rozhraní je přímo použitelné při psaní aplikací. Strukturu transportní vrstvy odpovídající normě OSI si můžeme popsat na obrázku 7.1. Transportní vrstva poskytuje služby přenosu *zpráv TSDU* (Transport Service Data Unit) prostřednictvím transportních přístupových míst — *socketů TSAP* (Transport Service Access Point). Využívá přitom služeb síťové vrstvy, poskytovaných prostřednictvím síťových přístupových míst *NSAP* (Network Service Access Point). Pakety přenášející data a řídící informace označujeme jako *transportní pakety TPDU* (Transport Protocol Data Unit).

Transportní protokol *TCP* internetu poskytuje službu přenosu *zpráv* na *socketech* (BSD nebo TLI), přičemž data přenáší prostřednictvím síťového protokolu *IP* v *transportních segmentech*.



Obrázek 7.1: Transportní vrstva

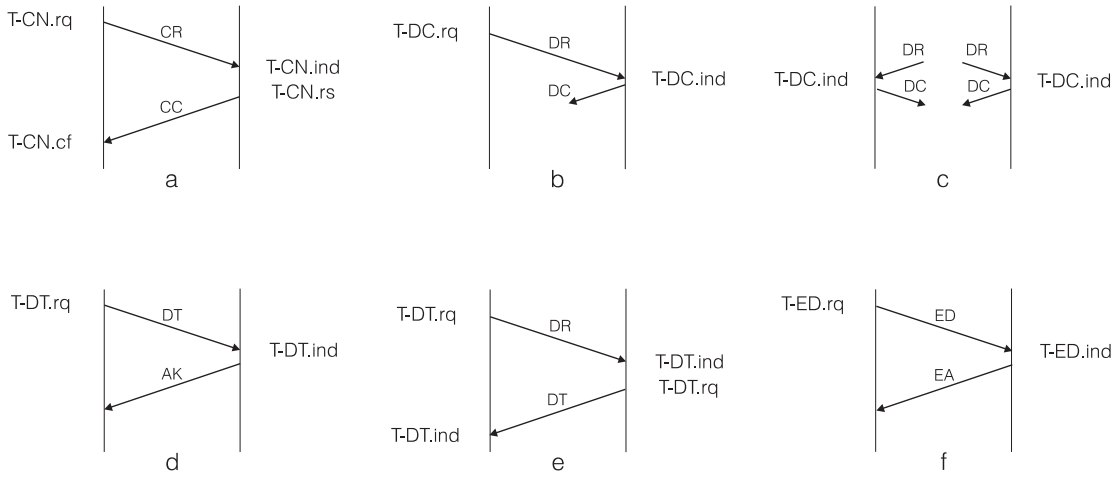
Nadřazené vrstvy, ale i aplikace využívající přímo služeb transportní vrstvy, mají k dispozici primitiva, které dovolují otevřít transportní komunikační kanál s konkrétní úrovní poskytovaných služeb, zajistit předávání dat po otevřeném kanálu a po ukončení datové komunikace kanál uzavřít. Přehled primitiv a postup při otevírání a uzavírání transportního spojení a při

přenosu dat pro transportní protokoly *ISO* uvádí obrázek 7.2.

T-CN.rq T-CONNECT.request(callee,caller,exp_wanted,qos,user_data)
T-CN.in T-CONNECT.indiction(callee,caller,exp_wanted,qos,user_data)
T-CN.rs T-CONNECT.response(qos,responder,exp_wanted,user_data)
T-CN.cf T-CONNECT.confirm(qos,responder,exp_wanted,user_data)

T-DC.rq T-DISCONNECT.request(user_data)
T-DC.in T-DISCONNECT.indication(user_data)

T-DT.rq T-DATA.request(user_data)
T-DT.in T-DATA.indication(user_data)
T-ED.rq T-EXPEDITED_DATA.request(user_data)
T-ED.in T-EXPEDITED_DATA.indication(user_data)



Obrázek 7.2: Primitiva a postupy transportní vrstvy

Zkratky, uvedené před názvy transportních primitiv v jejich seznamu (T-CN.rq, T-DC.in, ...), zavádíme pouze pro obrázky tohoto textu.

Aplikační program může při otevírání komunikačního kanálu požádat o zabezpečení určité kvalitativní úrovně poskytovaných služeb (QoS — Quality of Services). Rozumí se tím splnění zadaných požadavků na parametry jako jsou:

-doba otevírání spojení a pravděpodobnost neúspěchu,

-požadovaná přenosová kapacita (pro každý směr),

-zpoždění při přenosu,

-zbytková chybovost přenosu (teoreticky nulová, ale v praxi ...),

-pravděpodobnost toho, že spojení nedodrží požadavky na kapacitu, zpoždění a zbytkovou chybovost přenosu,

-doba potřebná k ukončení spojení a pravděpodobnost neúspěchu,

-ochrana proti neautorizovanému přístupu k přenášeným datům,

-priorita v rámci otevíraných spojení, a

-pravděpodobnost výpadku spojení.

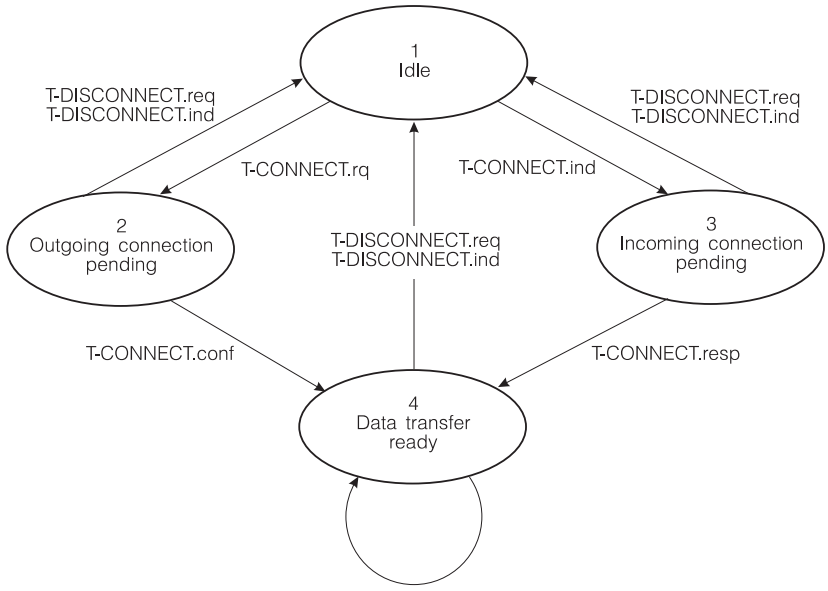
Na dodržení požadované kvality služeb se obě koncové transportní entity musí při otevírání spojení domluvit.

Odmítnout transportní spojení může adresát žádosti T-CONNECT.request nebo transportní vrstva. Ukončení spojení může vyžádat jeden nebo oba z komunikujících partnerů, ukončení spojení z iniciativy transportní vrstvy je důsledkem ztráty spojení mezi transportními entitami (přesněji důsledku nemožnosti přerušené spojení obnovit).

Funkce přednostního přenosu dat (EXPEDITED_DATA) musí být vyžádána při navazování spojení, přednostní data jsou doručována před běžnými daty na otevřeném spojení (např. Ĉ při interaktivní komunikaci může *"předběhnout"* dosud neodebraná data).

Každé primitivě transportního rozhraní odpovídá vyslání nebo příjmu konkrétního transportního paketu TPDU. Jejich seznam najdeme na obrázku **??**.

Koncové stanice musí dodržet přísná pravidla transportního protokolu, programovou realizaci popisu správného chování zahrnuje transportní vrstva. Popisem chování je nejčastěji *automat transportního protokolu*. Grafické znázornění automatu řídícího funkci transportní stanice OSI uvádí obrázek 7.3.



Obrázek 7.3: Automat transportní vrstvy

Stav 1 automatu (IDLE) odpovídá neotevřenému spojení, stav 2 (Outgoing Connection Pending) nedokončené vlastní žádosti o otevření spojení, stav 3 (Incomming Connection Pending) dosud nepotvrzené žádosti druhé strany a stav 4 (Data Transfer Ready) otevřenému transportnímu spojení. Automat transportního protokolu bývá v praxi definován (podobně jako např. síťový protokol X.25 CCITT nebo protokoly vyšších vrstev) tabulkou přechodů.

Postupy otevírání a uzavírání transportního spojení a přenosu dat u protokolu *TCP* jsou obdobné těm, které jsme si popsali pro protokoly *ISO*. Podstatnou odlišností je však pouze jediný formát transportních segmentů, který je vybaven řídícími příznaky a může být použit současně jako řídící i datový.

7.1 Třídy transportních protokolů

Složitost implementace transportního protokolu závisí na vlastnostech síťové vrstvy. Pro potřeby porovnání dělíme síťové technologie z hlediska kvality poskytovaných služeb do tří tříd označovaných A, B a C.

Sítě třídy A zajišťují téměř perfektní komunikaci. Množství poškozených, ztracených nebo duplikovaných paketů je zanedbatelné, stejně jako je zanedbatelná potřeba navazovat přerušená spojení. Do této kategorie lze optimisticky zařadit pouze některé sítě lokální. Implementace transportní vrstvy je pro sítě třídy A velmi snadná a zahrnuje pouze rozklad zpráv na pakety, zpětné skládání paketů do zpráv a případný multiplex.

Přepojovací sítě s virtuálními kanály zajišťují bezchybový sekvenční přenos paketů, ale je občas nutné obnovit přerušené spojení (N-RESET). Při obnovování spojení může dojít ke ztrátě nebo duplikaci transportního paketu a je proto nutné mít vlastní potvrzování. Přepojovací sítě s virtuálními kanály zahrnujeme spolu s realisticky hodnocenými lokálními sítěmi do třídy B.

Nejkomplikovanější je implementace transportní vrstvy nad datagramovými sítěmi, u kterých je nutné zajistit koncové potvrzování a ukládání paketů přicházejících v porušeném pořadí. Tyto sítě označujeme jako sítě třídy C.

Podle třídy sítě, nad kterou transportní vrstva pracuje, a podle toho, zda jedno síťové spojení slouží pouze pro jediné spojení transportní nebo je vyžadován multiplex, zavádí norma OSI pět tříd transportních protokolů. Označuje je jako TP0 až TP4, jejich přehled uvádí následující tabulka:

Protokol	Třída	Zajišťovaná funkce
TP0	A	—
TP1	B	koncové potvrzování
TP2	A	multiplex
TP3	B	koncové potvrzování a multiplex
TP4	C	koncové potvrzování, řazení a multiplex

V praxi se běžně používá protokol *TP4*, a to i tehdy, jestliže vlastnosti sítě tak složitou obsluhu nevyžadují. Zvýšení zabezpečovací schopnost transportního protokolu se ve snížení jeho efektivity příliš neprojeví.

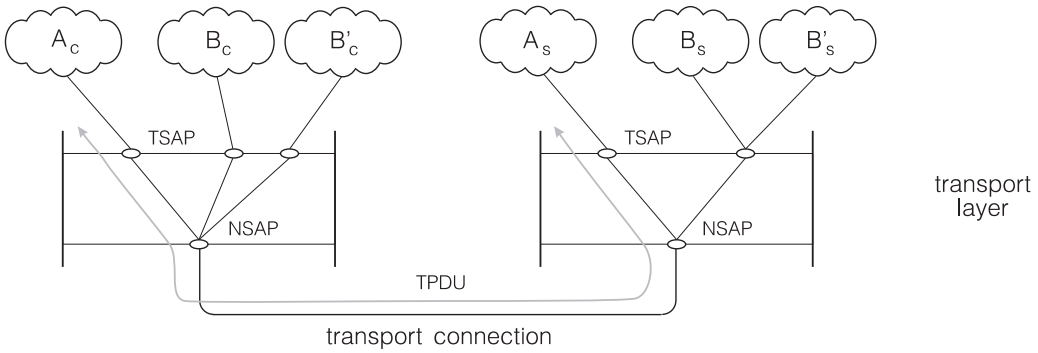
Transportní protokol *TCP* odpovídá svými možnostmi, s ohledem na vlastnosti síťového protokolu IP, transportnímu protokolu *TP4* s tím, že ve své současné formě nepodporuje QoS.

7.2 Multiplex a adresace

Jako transportní multiplex označujeme schopnost transportní vrstvy vytvořit na jediném síťovém spojení mezi dvěma počítači více spojení transportních, tedy spojení mezi procesy, které na těchto počítačích běží. Důvodem pro toto řešení, označované jako multiplex *směrem nahoru* (upward multiplex), jsou u sítí s virtuálními kanály technické limity omezující počet síťových spojení, u sítí datagramových omezení adresace na adresu počítače (procesy nejsou na úrovni sítě rozlišitelné). Dalším důvodem mohou být tarify veřejné datové sítě znevýhodňující větší počet slabě využívaných virtuálních kanálů oproti menšímu množství sdílených kanálů s větší intenzitou provozu.

Pro rozlišení jednotlivých procesů využívajících jednotlivé kanály transportního multiplexu zavádíme adresaci přípojných míst TSAP v rámci počítače. Přípojné místo je jednoznačně

identifikováno dvojicí (NSAP,TSAP), kde NSAP je adresa počítače (například odpovídající normě X.121 CCITT nebo adresaci Internetu).



Obrázek 7.4: Transportní spojení a multiplex

S jednoznačnou adresací procesů jsou spojeny ještě další problémy. Jedním z nich je nutnost znát kompletní adresy (ve tvaru (NSAP,TSAP)) pro všechny služby v síti, které chceme využívat. Částečným řešením je umístění standardních služeb na pevných adresách (např. služby *telnet* a *ftp* v Internetu najdeme vždy na socketu s pevnou adresou), další možností je služba *name serveru* na pevné adrese, která nám číslo socketu pro danou službu na požádání sdělí.

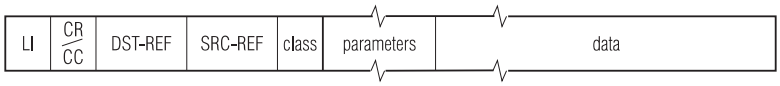
Další problém souvisí s potřebou provozovat souběžně více kopií téže služby na jednom systému. Pevné přidělení socketů kopiím službám by bylo omezující, východiskem je možnost souběžné činnosti více kopií služby (rodičovský proces navazuje komunikaci a synovské procesy zajišťují vlastní službu) na jednom socketu. Každé transportní spojení je pak jednoznacně identifikováno čtveřicí $(NSAP_c, TSAP_c\, NSAP_s, TSAP_s)$. U protokolu *TCP* navíc tento údaj doplňujeme o pátý údaj, o identifikaci protokolu *TCP*.

Výjimečně se můžeme u transportní vrstvy setkat s multiplexem *směrem dolů* (downward multiplex). Toho se využívá, pokud pro jedno transportní spojení vyžadujeme větší přenosovou kapacitu, než je schopné poskytnout jedno spojení síťové. Kapacita jednotlivého virtuálního spoje je například omezena zpožděním při přenosu paketu sítí a okénkem koncového potvrzování. Dalším důvodem pro tento typ multiplexu může být zvýšení spolehlivosti transportního spojení, které multiplexujeme na dvě fyzicky oddělená spojení síťová.

7.3 Formáty transportních paketů

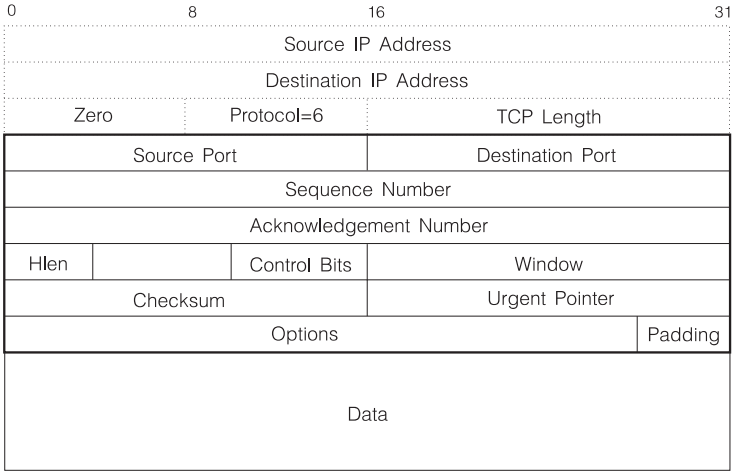
Pro zajištění své funkci si protokolové stanice transportní vrstvy vyměňují služební informace ve formátech datových transportních paketů (například číslování paketů a potvrzení) nebo jako speciální služební pakety. Jako příklad formátu transportního paketu si uvedeme formát paketu *CR*, kterým žádá protokolová stanice OSI o otevření spojení. V jednotlivých polích je uvedena délka *LI*, jednoznačná identifikace spojení *DST-REF* a *SRC-REF*, požadovaná třída a parametry spojení, a případná aplikační data. Spolu se seznamem typů transportních paketů najdeme formát paketu *CR* na obrázku 7.5.

CR	Connection Request	CC	Connection Confirm
DR	Disconnect Request	DC	Disconnect Confirm
DT	Data Transfer	AK	Acknowledgement
ED	Expedited Data	EA	Expedited Acknowledgement
RJ	Reject	ER	Error



Obrázek 7.5: Formát a typy TPDU protokolu TP4

Protokol TCP používá jediný formát transportního segmentu, ten se používá jak pro vlastní spojení, tak pro přenos dat (obr. 7.6).



Obrázek 7.6: Formát transportního segmentu TCP

Řídící funkce se opírají o příznaky *SYN, ACK, URG, PSH, FIN* a *RST*. Jejich kombinace dovolují vytvářet transportní segnenty pro jednotlivé funkce protokolu.

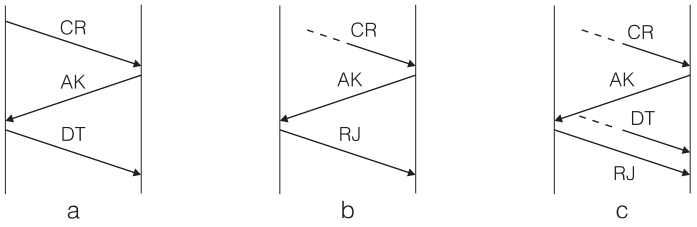
7.4 Navazování spojení

Důležitou a z hlediska implementace poměrně složitou funkcí transportní vrstvy je navazování spojení. Jednoduchá inicializace stavových proměnných, jak jsme ji poznali u vrstvy linkové, zde totiž nepostačuje. Příčin je více, nejpodstatnější je nedeterministické zpoždění transportních paketů při průchodu datagramovými sítěmi. Vliv nedeterministického zpoždění je obvykle ilustrován na následujícím příkladě:

Předpokládejme, že se náš počítač má spojit se vzdáleným bankovním počítačem, požádat o provedení transakce (například o převedení určité částky z našeho účtu) a spojení ukončit. Překročí-li doba přenosu žádosti o otevření spojení časový limit (time-out), náš počítač žádost zopakuje. V síti se pak pohybují dvě žádosti, prvá dorazí k bankovnímu systému a ten potvrdí otevření spojení. Náš počítač požádá o provedení transakce, ale paket opět *"zabloudí"* a bude po vypršení časového limitu zopakován. Bankovní systém po příchodu prvé ze žádostí transakci provede a potvrdí. Po příjmu potvrzení transakce požádáme o ukončení spojení. U něho se může situace s vypršením časového limitu a retransmisí opakovat.

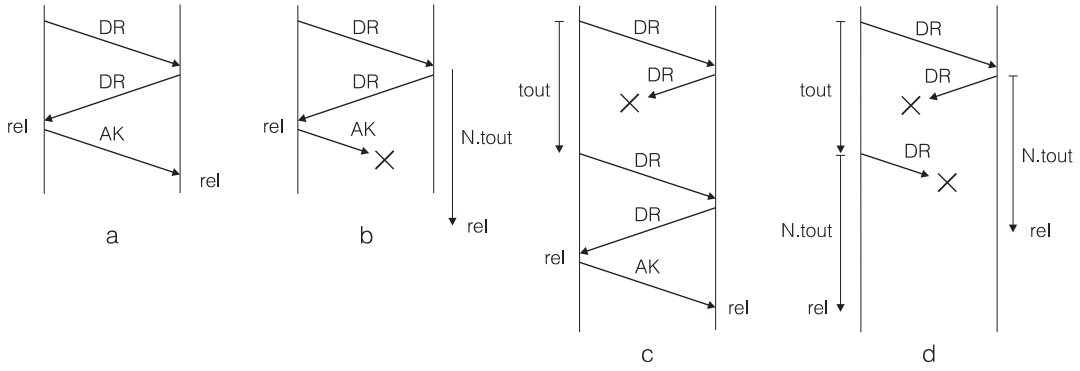
Výsledkem právě popsaného děje bude, že se v síti po uzavření transakce bude pohybovat kopie žádosti o otevření transportního spojení, kopie žádosti o provedení transakce a kopie žádosti o uzavření spojení. Dorazí-li tyto pakety k cílovému systému ve "vhodném" pořadí, je transakce zduplikována.

Chceme-li uvedené situaci zabránit musíme odlišit jednotlivé pokusy o navázání spojení a zabránit tomu, aby cílový systém považoval výše uvedenou posloupnost kopií za korektní posloupnost paketů. Pouhé číslování pokusů nepostačuje s ohledem na možné výpadky počítačů. Dobrou metodou je třífázový protokol (*three-way handshake*) využívaný jak v protokolu TP4, tak v protokolu TCP . Postup při korektním otevření spojení (a) a řešení problému opožděných duplikátů (b,c) uvádí obrázek 7.7.



Obrázek 7.7: Třífázový protokol otevírání spojení

Problémy nejsou spojené pouze s otevíráním spojení. Při uzavírání spojení je třeba zkontrolovat, zda se v síti nepohybují starší datové pakety. Opět zde pomůže třífázový protokol, problémy však může způsobit ztráta paketů. Příklad bezchybného ukončení spojení (a) a řešení problémů (b,c,d) uvádí obrázek 7.8.

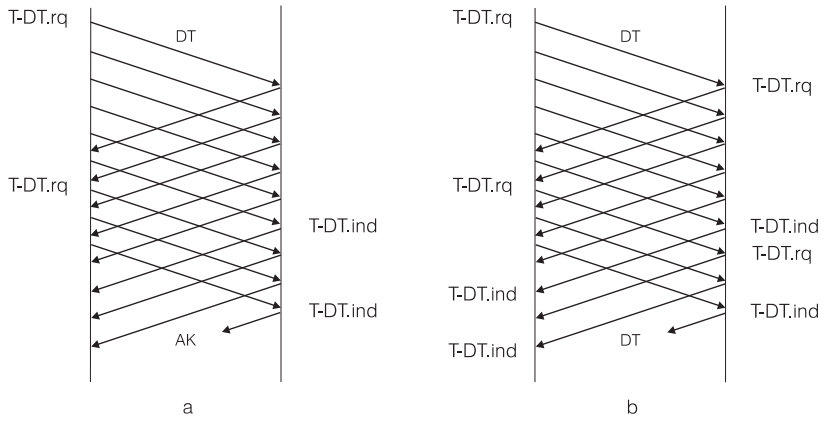


Obrázek 7.8: Uzavírání spojení

7.5 Koncové řízení toku

Úkolem řízení toku na úrovni sítě (ukázali jsme si metodu škrtících paketů) bylo zabránit zahlcení sítě zbytečným množstvím paketů. Úkol řízení toku na transportní úrovni je poněkud jiný. Nejde zde o ochranu sítě, cílem je rozumné využívání paměti v koncových počítačích a dostatečně volná synchronizace obou komunikujících partnerů.

Při otevírání spojení je u protokolu ISO specifikována požadovaná paměť na straně přijímače (v počtu paketů). Přijímající strana v potvrzeních kromě potvrzovacího čísla (u protokolu ISO udává číslo očekávaného paketu) uvádí i okamžitě dostupnou volnou paměť jako tzv. *kredit*. Ten se, na rozdíl od okénka, může i snižovat. Příklad přenosu datových TPDU uvádí obrázek 7.9, ve skutečném protokolu je redukován počet potřebných potvrzení využitím kumulativního (skupinového) potvrzování.



Obrázek 7.9: Přenos dat

7.5.1 Mechanismy koncového řízení toku TCP

Dynamická změna okénka u transportního protokolu může být využita i pro řízení toku do sítě; to je zajímavé zvláště u protokolu TCP pro Internet, kde vnitřní mechanismy řízení toku na úrovni koncových spojení scházejí.

Základní mechanismus řízení toku u protokolu TCP se opírá o pole *Window* hlavičky. V něm příjemce dat sděluje, kolik znaků je schopen přijmout, počínajíc znakem s pozicí uvedenou v poli *Ack*. Mechanismus byl navržen pro přenos souborů (brání vyčerpání vyrovnávací paměti na straně příjemce).

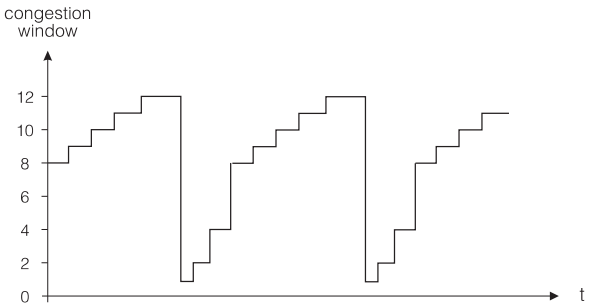
Při nízké rychlosti vstupu u interaktivní práce vede základní algoritmus k nízkému využití kapacity kanálu, jednotlivé znaky vstupu jsou totiž po vypršení časového limitu odesílány jako samostatné TCP segmenty, s příslušnou TCP a IP hlavičkou. Proto je protokol TCP doplňován o *Nagleův* algoritmus. Ten se snaží o maximální využití kanálů s dlouhou dobou přenosu. První znak je odeslán v samostatném TCP segmentu. Další znaky jsou ukládány do vyrovnávací paměti a odeslány až po příjmu potvrzení, a tento postup se dále opakuje. K mimořádnému odeslání TCP segmentu dochází při naplnění poloviny kreditu, nebo při naplnění maximální délky segmentu. V některých případech však může být Nagleův algoritmus na závadu, např. při komunikaci X-terminálu se vzdálenou aplikací.

Podobně nepříjemné důsledky může mít i chování příjemce, který odebírá z vyrovnávací paměti jednotlivé znaky a oznamuje odesílateli jednoznakové kredity ($Window = 1$), řešením je odeslání potvrzení teprve po uvolnění většího bloku vyrovnávací paměti.

Mechanismus kreditu *Window* je primárně určen k omezení toku na velikost, kterou je schopen akceptovat příjemce. Velká hodnota kreditu dovolí vysílající straně odeslat do sítě řadu paketů a může vést k zahlcení sítě s menší průchodností. Tomu lze bránit doplňkovými *mechanismy řízení toku*, které mají za cíl zahlcení sítě zabránit.

Předpokládejme, že známe časový limit, do kterého přijde potvrzení odeslaného segmentu na nepřetížené síti, tuto hodnotu označujeme jako *RTT* (Round Trip Time). Každé překročení tohoto časového limitu pak můžeme považovat za příznak zahlcení sítě (zahození paketu). Vysílající strana zahajuje přenos s hodnotou okénka (*congestion window*) o velikosti jednoho TCP segmentu o maximální délce (samozřejmě, pokud není kredit příjemce nižší, nebo pokud nevypršel časový limit Nagleova algoritmu). S příchodem každého potvrzení je velikost okénka zvětšována o jeden TCP segment o maximální délce. Výsledkem postupu je pomalé, exponenciální, zvětšování okénka *slow-start*, to tak může dosáhnout nejvýše 64 kB (počáteční hodnota limitu *Threshold*). Od tohoto limitu dochází k dalšímu otevírání okénka lineárně.

Výpadek potvrzení je příznakem zahlcení sítě a vede k návratu na velikost okénka o délce jednoho maximálního TCP segmentu a k současnému nastavení limitu *Threshold* na polovinu dosažené hodnoty okénka. Mechanismus, označovaný jako *TCP-Tahoe* chrání síť před zahlcením, jeho činnost z pohledu konkrétního transportního spojení ilustruje obrázek 7.10.



Obrázek 7.10: Chování TCP protokolu při zahlcení

Oscilující koncové toky TCP se mohou snadno zasynchronizovat, výsledkem mohou být oscilace celé sítě. Technologickou ochranou proti takovému chování je náhodné zahazování TCP segmentů ještě před dosažením limitní délky fronty ve směrovači. Mechanismus je označován jako *Random Early Detection*, pravděpodobnost zahození TCP segmentu se od dosažení určité délky fronty lineárně zvyšuje.

Nepříjemnou vlastností mechanismu je skutečnost, že i selektivní výpadek jednoho paketu vyvolá razantní přivření okénka. Modernější mechanismus *TCP-Reno* umí reagovat na příchod tří potvrzení se stejným potvrzovacím číslem, tedy na skutečnost, že došlo ke ztrátě segmentu, ale časová odezva sítě zůstala rozumná. Vysílač sníží limit *Threshold* na polovinu dosažené hodnoty okénka, a okénko na tutéž velikost. Navíc může odeslat segment jako reakci na každé z opakovaných potvrzení, i pokud by došlo k překročení limitu *Threshold*.

Konečně, nejnovější technologie řízení toku TCP – *TCP-Vegas* se snaží odstranit skutečnost, že dosd popisované mechanismy reagují až na příznak zahlcení sítě. TCP-Vegas odhaduje schopnost sítě přenášet určitý tok dat na základě počtu potvrzení, která jsou přijata za dobu odpovídající dosud nejkratšímu RTT (tento údaj považujeme za odezvu na nezatížené síti) a tomu přizpůsobuje frekvenci odesílání TCP segmentů (a nastavení okénka).

Určení RTT

Dosud jsme předpokládali, že hodnota časového limitu je známa. V praxi hodnota časového limitu závisí na zpožděních v síti, je silně závislá na použité technologii a topologii sítě, a je nutné mít k dispozici mechanismus, který dovolí vhodný časový limit nastavit automaticky.

Základem pro nastavení časového limitu je určení hodnoty RTT. Jelikož se jedná o náhodnou veličinu, je rozumné vyfiltrovat střední hodnotu. Je používán *exponenciální filtr* (klouzavý průměr)

$$s_k = as_{k-1} + (1-a)s_k \ ,$$

kde pro koeficient a je v TCP doporučena hodnota $a = 0.9$. Časový limit je pak nastavován na hodnotu

$$Timeout = bs_k \ ,$$

kde pro koeficient b je v TCP doporučena hodnota $b = 2$.

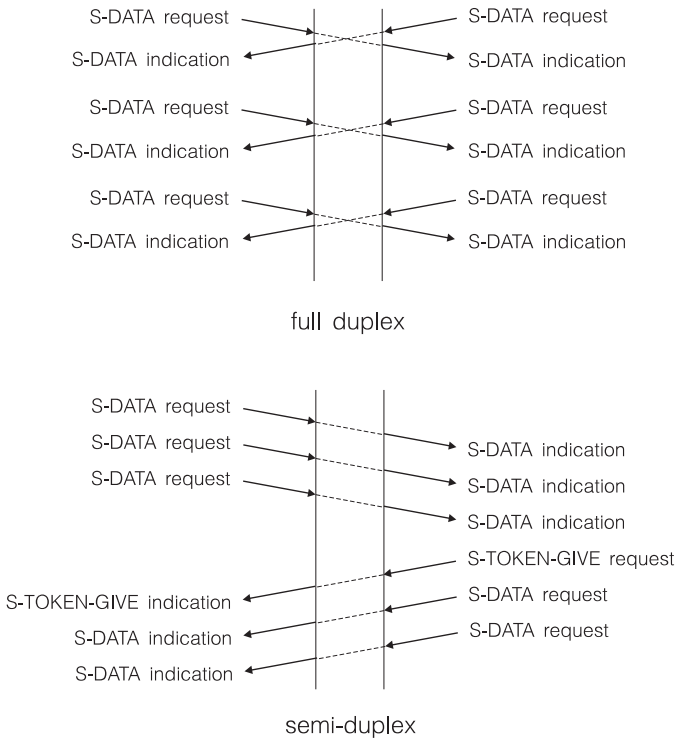
8. Relační vrstva

Relační vrstva je první z vyšších vrstev síťové architektury. Úkolem nižších vrstev bylo zabezpečit spolehlivý přenos dat, úkolem vyšších vrstev je zlepšit funkční vlastnosti přenosu a realizovat přídavné služby.

Relační vrstva, jak ji definuje norma ISO 8327, zabezpečuje řízení dialogu, synchronizaci a řízení aktivit. Instance relační vrstvy na různých počítačích spolu komunikují výměnou zpráv označovaných jako SPDU (Session Protocol Data Unit). Některé přenášejí data pro vyšší vrstvy, jiné slouží pouze řízení relace. Spojení mezi dvěma entitami relační vrstvy má stejně jako spojení transportní tři fáze – navázání spojení (S–CONNECT), vlastní komunikaci a uvolnění spojení (S-RELEASE). Na rozdíl od transportní vrstvy je uvolnění spojení čisté, nemůže při něm dojít ke ztrátě zpráv.

8.1 Řízení dialogu

Plně duplexní přenos dat, který umožňuje transportní vrstva, dává aplikačním programům velkou volnost forem vzájemné spolupráce. Velmi často však tato volnost může být na závadu, a je výhodnější nebo dokonce nutné, aby aplikace spolu komunikovaly poloduplexně. V takovém případě s výhodou využijeme funkci řízení dialogu, kterou ilustruje obrázek 8.1.

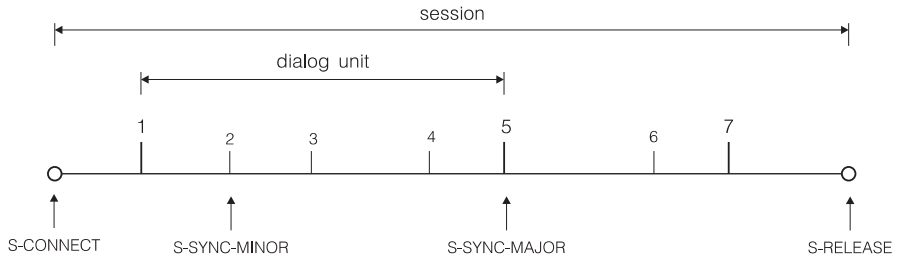


Obrázek 8.1: Řízení dialogu (poloduplex)

8.2 Synchronizace

Další službou relační vrstvy je vkládání synchronizačních bodů do posloupnosti přenášených dat. Tyto synchronizační body dovolí v případě havárie na přijímací straně (např. přetržený

papír v tiskárně, na kterou jsou přenášená data přímo předávána) požádat vysílající stranu o návrat k vhodnému synchronizačnímu bodu. To samozřejmě vyžaduje, aby vysílající strana měla potřebná data zapamatována. Příklad vkládání synchronizačních bodů uvádí obrázek 8.2.

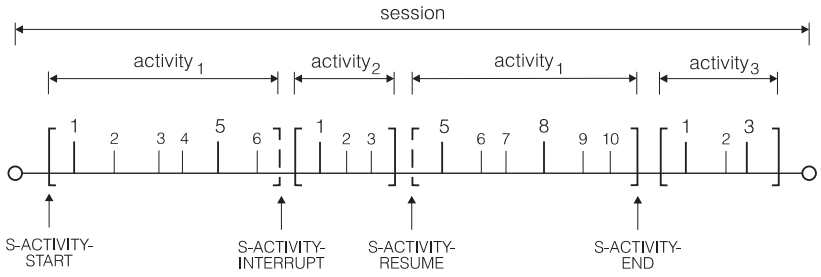


Obrázek 8.2: Vkládání synchronizačních bodů

Velké synchronizační body (mayor sync points) dělí spojení na samostatné dialogové jednotky, jsou přijímací stranou explicitně potvrzovany a nelze se před ně vracet. Malé synchronizační body (minor sync points) nejsou potvrzovány a uvnitř jedné dialogové jednotky se lze vrátit k libovolnému z nich.

8.3 Řízení aktivit

Relační spojení lze rozdělit na kratší úseky, pro které používáme název aktivita. Aktivitou může být například přenos souboru nebo transakce ve vzdálené databance. Aktivity jsou oddělovány zprávami S–ACTITY–START a S–ACTIVITY–END, o jejich vkládání žádá aplikace. To například dovolí uzamknout příslušná data právě na dobu provádění transakce (obr. 8.3).



Obrázek 8.3: Aktivity a možnost jejich přerušení

Déle trvající aktivity, jako je například přenos souborů, je možné přerušit, vložit krátkou interaktivní výměnu a po ní opět v přenosu pokračovat.

Výjimky

Relační vrstva odpovídající normě ISO zahrnuje mechanismus hlášení vyjímečných situací, které vznikly uvnitř relační vrstvy (S–U–EXCEPTION–REPORT) nebo, které mají svůj původ ve vrstvě transportní a relační vrstvou jsou pouze předávány (S–P-EXCEPTION–REPORT).

9. Presentační vrstva

Dosud popisované vrstvy síťové architektury zajišťovaly transparentní přenos dat ve formě zpráv vyměňovaných mezi koncovými aplikacemi. Presentační vrstva transformuje vyměňovaná data před jejich přenosem. Cílem je:

-transformovat data při přenosu mezi počítači s různým zobrazením,

-komprimovat data a tak minimalizovat tok dat sítí, a

-kryptograficky chránit přenášená data před odposlechem a narušením.

V textu této kapitoly si všimneme základních principů těchto funkcí.

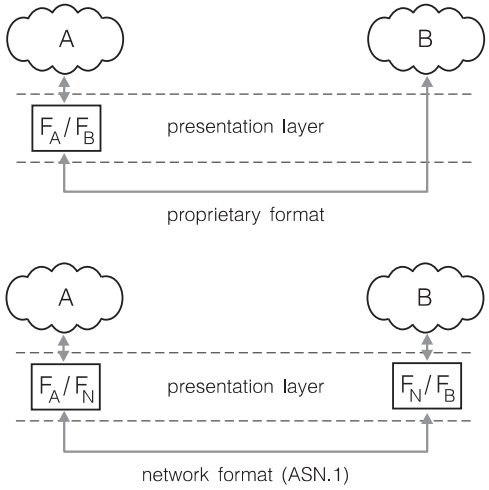
9.1 Síťová reprezentace dat

Počítače různých typů se liší i v tak podstatných vlastnostech jako je zobrazení textových znaků a čísel. I pokud jde o zobrazení znaků anglické abecedy, lze se setkat se dvěma kódy: ASCII a EBCDIC. Podstatně složitější může být situace v oblasti národních abeced (např. řada různých kódů pro písmena s háčky a čárkami v češtině). I další primitivní datový údaj — celé číslo o délce šestnácti bitů lze na počítačích s oktetovou adresací paměti zobrazit dvěma způsoby:

-big endian, kdy jsou vyšší řády čísla ukládány na oktet s nižší adresou: toto přirozené zobrazení je používáno pro ukládání informace využívané sítí (adresy, čísla portů),

-little endian, kdy jsou nižší řády ukládány na oktet s nižší adresou: s tímto zobrazením se setkáváme u počítačů na bázi procesorů Intel.

Překlad zobrazení lze zajistit buď pouze na jedné straně, a pro přenos použít nativní reprezentaci druhé strany, nebo na obou stranách a pro přenos použít standardní síťovou reprezentaci (9.1).

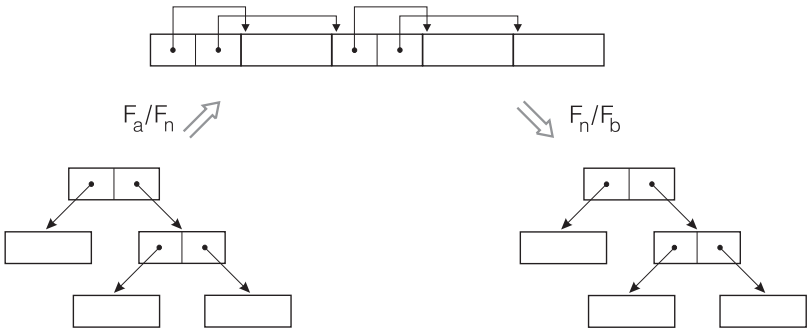


Obrázek 9.1: Varianty transformace dat v síti

Prvá metoda je efektivnější, ale je použitelná v homogenních systémech (postavených na počítačích se shodným zobrazením dat) a v systémech převážně homogenních. V systémech heterogenních vyžaduje větší počet různých překladačů dat (celkem $n.(n-1)$ pro n různých reprezentací). Příkladem použití je technologie NDR (Network Data Representation) využívaná v systému DCE. Druhá metoda je nutná pro data využívaná síťovými prvky (adresy, čísla portů, informace systému správy), její nevýhodou je nižší efektivita (dvojí překlad), výhodou je menší počet nutných překladačů (celkem n pro n různých reprezentací). Příkladem použití je formát

ASN.1 (Abstract Syntax Notation), o který se opírá např. systém správy sítě SNMP (Siple Network Management Protocol).

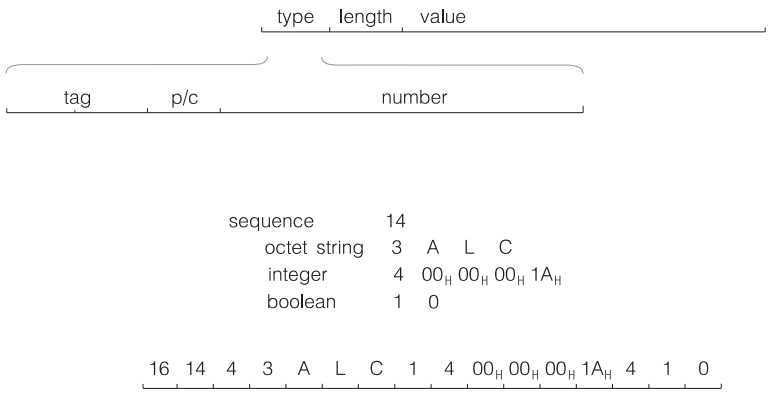
Ještě složitější je přenos čísel s plovoucí řádovou čárkou a datových struktur. U čísel s plovoucí řádovou čárkou je často využívána jejich textová reprezentace (tedy převod do dekadického tvaru se všemi nevýhodami — pomalost a nepřesnost, takového postupu). Přenos lineárních datových struktur je třeba doplnit přenosem informace o typu přenášených dat, přenos seznamových datových struktur se může opřít o adresaci ve zprávě (obr. 9.2).



Obrázek 9.2: Přenos seznamové struktury

9.1.1 Notace ASN.1

Rozšířeným formátem dat v sítích je formát označovaný jako ASN.1, využívaný v protokolech správy. Jednotlivé položky přenášených dat jsou obvykle přenášeny jako trojice – (typ,délka,hodnota). Složitější datové struktury lze vytvářet pomocí konstruktorů, příklad struktury (poznáme ji podle identifikátoru typu 16), složené z řetězu znaků, celého čísla a logické hodnoty uvádí obr. 9.3.



Obrázek 9.3: Standardní formát ASN.1

Formát ASN.1 dovoluje, vedle univerzálních elementárních typů a konstrukce typů složitějších, vytvářet typy specifické pro konkrétní aplikaci, případně kontext. Tak lze snížit režii, kterou u složitějších struktur způsobuje předávání typové informace.

9.2 Komprese

Základním parametrem komunikačních kanálů je přenosová rychlost, efektivní využití omezené přenosové rychlosti má klíčový význam. Metody, dovolující snížit množství přenášených dat při zachování přenášené informace, jsou označovány jako kompresní metody.

Statistické metody

Budeme-li předpokládat, že přenášená data jsou posloupností symbolů abecedy $A = \{s_1, s_2, ..., s_n\}$, kde pravděpodobnosti jednotlivých symbolů $p(s_i)$ jsou odlišné, a budeme-li předpokládat nezávislost pravděpodobnosti symbolů v posloupnostech, můžeme dosáhnout efektivity přenosu, vyjádřené v počtu bitů potřebných k přenosu jednoho symbolu abecedy, blížící se entropii, vyjádřené jako

$$-\sum_{i=1}^{n} P_i log_2 P_i$$

.

Jednoduchou metodou, jak takové komprese dosáhnout, je prefixové *Hufmannovo kódování*. Jeho princip spočívá ve výstavbě binárního stromu nad listy, které odpovídají symbolům abecedy tak, že v každém kroku vybereme dva symboly/větve s nejnižší vahou a vytvoříme z nich větev složitější. V okamžiku, kdy dostaneme strom, jehož listy jsou všechny symboly abecedy, můžeme tento strom použít ke kódování.

Lepší aproximaci entropie efektivitou lze dosáhnout *aritmetickým kódováním*, které kóduje posloupnost znaků libovolně zvoleným číslem z intervalu reprezentující kódovanou posloupnost. Přidání symbolu k posloupnosti odpovídá rozdělení tohoto intervalu na části, jejich relativní velikost je úměrná pravděpodobnosti přidávaného symbolu. Prázdné posloupnosti odpovídá interval $< 0, 1 >$.

Často používanou statistickou metodou je *RLE* (Run Length Encoding), u které je místo opakování symbolu přenášena informace o symbolu a počtu jeho opakování. Metodu využívá například telefax.

O statistiku se opírají i historický pětibitový Baudotův kód a kódy využívané v teletextu. Rozdělují abecedu na stránky, přechody mezi stránkami řídí speciální symboly (číslicová a písmenová změna u dálnopisného kódu).

Diferenční metody

U některých datových přenosů je efektivnější přenášet informaci o změnách hodnot namísto hodnot samotných. To je typické pro přenos digitalizovaného zvukového a obrazového signálu, tedy pro přenosy v hovorových a videokonferenčních systémech.

Nejjednodušší metody *diferenciální PCM* přenášejí místo hodnoty rozdíl proti předchozí hodnotě, nebo proti lineární/polynomiální extrapolaci z více předcházejících hodnot. Efektivnější *adaptivní diferenciální PCM* navíc odvozuje velikost kvantizačního kroku z amplitudy signálu.

Slovníkové metody

U slovníkových metod jsou často se vyskytující posloupnosti symbolů nahrazovány indexem ve slovníku budovaném při statistické analýze textu. Nejznámější metodou komprese je *LZW* (Lempel-Ziv-Welsh). Její modifikovaná verze je základem kompresní metody *V.42bis* používané u telefonních modemů.

9.3 Kryptografie

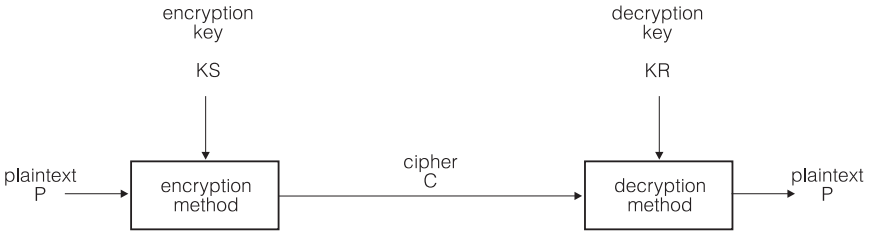
Přenos důvěrných informací veřejnými počítačovými sítěmi vyžaduje použít prostředky schopné zabránit:

-neoprávněnému přístupu a zneužití přenášených dat, a

-podvržení nebo modifikaci přenášených dat,

Používané prostředky mohou současně zajistit autentifikaci účastníků komunikace.

Metody utajování přenášených informací mají dlouhou historii a opírají se o obecné schéma (obr 9.4).



Obrázek 9.4: Model šifrovaného přenosu

Kódovaná data/test P jsou šifrována s použitím klíče KS, výsledkem tohoto postupu je šifrovaný text $C = E_{KS}(P)$. Ten je na straně příjemce dešifrován s použitím klíče KR, výsledkem jsou původní data/text $P = D_{KR}$. Musí tedy platit

$$D_{KR}(E_{KS}(P)) = P \ .$$

Funkce E_{KS} a D_{KR}, používané pro kódování, jsou obvykle známé, ale volené tak, aby zjištění textu předávané informace P z šifrovaného textu C bez znalosti klíče KR bylo velmi obtížné.

Starší, a výpočetně jednodušší techniky využívají shodného klíče pro kódování a dekódování, tedy $KR = KP$. Mluvíme o *symetrické kryptografii*, její nevýhodou je nutnost použití *tajného klíče* oběma komunikujícími partnery.

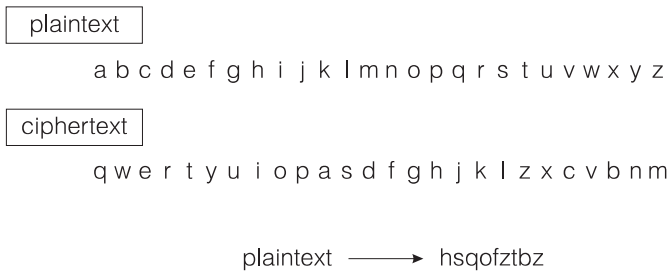
Příjemnou vlastností technik opírajících se o různé klíče KS a KR, kde určení klíče KR pro dekódování z klíče KS pro kódování (nebo naopak) je velice obtížné, je možnost vytvářet relativně jednoduchá a spolehlivá schémata pro autentizaci a pro distribuci klíčů efektivnějších symetrických metod. Mluvíme o *asymetrické kryptografii*, resp. o kryptografii využívající *veřejného klíče* (a pochopitelně i jeho protějšku — klíče soukromého).

9.3.1 Symetrická kryptografie

Symetrická šifra vyžaduje znalost tajného klíče na obou strnách kanálu, její výhodou je však poměrná jednoduchost a efektivita. Je obvykle založena na dvou jednoduchých principech — na substituci a transpozici.

Substituce

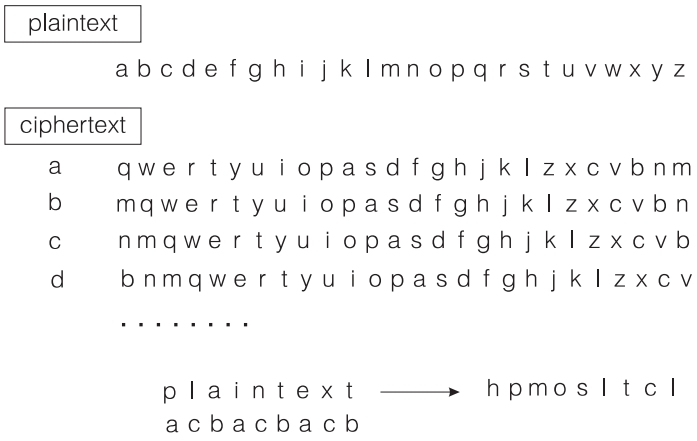
Čistou znakovou substituci používal pro utajení informací již Caesar, jeho metoda spočívala v náhradě původních znaků textu znaky z abecedy posunuté o pevný počet pozic. Obecná *monoalfabetická znaková substituce* se opírá o klíč, který definuje přiřazení jednotlivých znaků textu P znakům šifry C. Příklad monoalfabetické znakové substituce uvádí obrázek (obr. 9.5).



Obrázek 9.5: Monoalfabetická znaková substituce

Rozluštění této jednoduché šifry hrubou silou vyžaduje vyzkoušet až 26! možných klíčů; rozumnější metodou je využít statistických vlastností textu v daném jazyce a oblasti použití, které vede rychle k výsledku. Zvýšení odolnosti lze u znakové substituce dosáhnout zvětšením kódovaného bloku dat (např. kódování dvojic nebo trojic znaků), cenou je však velikost používaného klíče.

Podstatného zvýšení odolnosti dosahuje Vigenerova substituce, u níž pro jednotlivé znaky textu P používáme různá přiřazení znaků textuP znakům šifry C. Výběr konkrétního přiřazení je řízen tajným klíčem Příklad uvádí obrázek (obr. 9.6).



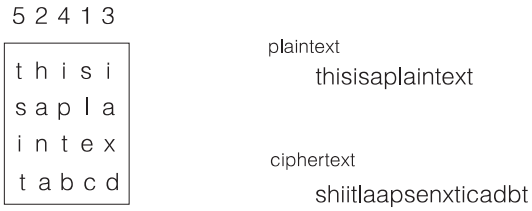
Obrázek 9.6: Vigenerova substituce

Odolnost Vigenerovy substituce závisí na délce klíče: je-li délka klíče rovna jedné, degeneruje Vigenerova substituce na substituci monoalfabetickou. Naopak, pokud je klíč alespoň stejně dlouhý jako text P, nemůže kryptoanalýza pro rozbití šifry nic nabídnout.

Nejjednodušší variantou Vigenerovy substituce je překódovávání jednotlivých bitů zprávy. K dispozici máme pouze dvě různá přiřazení bitů, výběr řídíme hodnotou jednoho bitu klíče (a prakticky realizujeme logickým prvkem XOR). Tím může být například dostatečně dlouhý záznam signálu náhodného binárního generátoru uložený na paměťovém médiu.

Transpozice

Znaková substituce zachovává pozici znaků, pouze mění jejich obraz. Transpozice naproti tomu zachovává obraz znaku, ale mění podle pravidla určeného klíčem jeho pozici v textu. Příklad znakové transpozice uvádí obrázek (obr. 9.7).

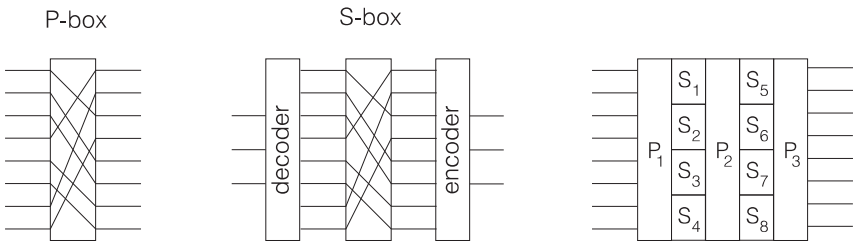


Obrázek 9.7: Znaková transpozice

Symetrické šifry

Obě popsané techniky, substituce a transpozice jsou základem metod prakticky využívaných v symetrické kryptografii. Tyto metody jsou v principu založeny na pouhé substituci, délka symbolu (64 bitů u DES, 128 bitů u kódů IDEA a Rijndael) však vylučuje uložení klíče pro přímou substituci na libovolném paměťovém médiu.

Řešením je opakování více substitučních a transpozičních kroků s menší délkou symbolu, řízené hodnotou klíče. Na tomto principu ((obr. 9.8)) pracuje většina používaných metod.



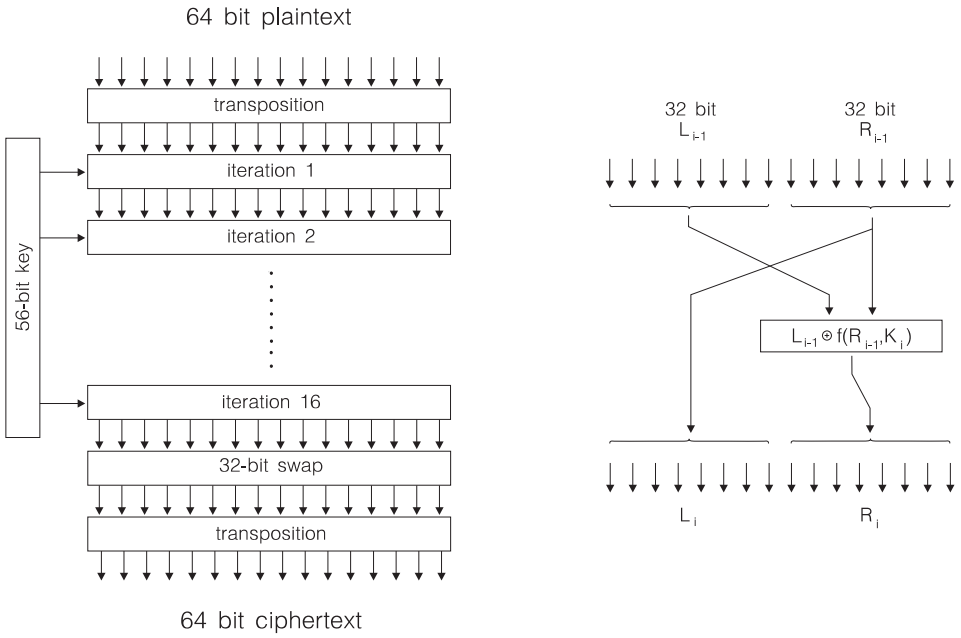
Obrázek 9.8: Princip iteračního výpočtu kódu

DES — Data Encryption Standard

Substituce je základem standardizované metody označované jako DES (Data Encryption Standard). Ta byla vytvořena na bázi původního návrhu: kódu Lucifer firmy IBM, který pracoval se 128-bitovými symboly a 128-bitovým klíčem. Na doporučení NSA (National Security Agency) byla metoda oslabena: výsledný kód DES používá 64-bitové synboly a 56-bitový klíč. Strukturu kódéru DES uvádí obrázek 9.9.

Vstupní 64-bitové slovo je, po úvodní transpozici, rozděleno na dvě 32-bitové části. Pravých 32 bitů R_{i-1} je použito jako levých 32 bitů v dalším kroku L_i, a současně transformováno funkcí $f(R_{i-1}, K_i)$. Výstup funkčního bloku je zkombinován s levými 32 bity L_{i-1} operací XOR. Tato operace se opakuje šestnáctkrát s různými hodnotami klíče K_i odvozenými ze základního 56-bitového klíče K. Celý postup uzavírá prohození 32-bitových částí slova a závěrečná transpozice.

Dekódér má stejnou strukturu jako kódér, pouze jednotlivé kroky jsou prováděny v opačné pořadí.



Obrázek 9.9: Výpočet kódu DES

Triple DES

Nepříjemnou vlastností základního kódu DES je jeho nižší odolnost proti dekódování. Byly hledány možnosti posílení: jednou je trojí opakování transformace (využívající stejného programového řešení nebo obvodu). Jedná se o transformaci

$$C = E_{K1}(D_{K2}(E_{K1}(P)))$$

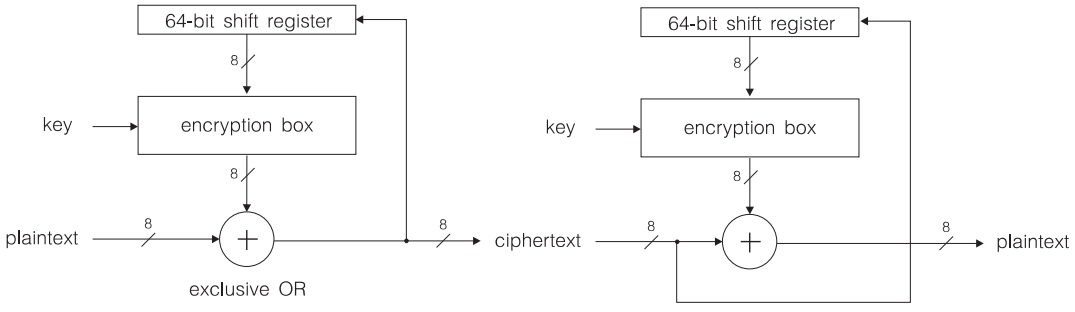
pro kódování a o transformaci

$$P = D_{K1}(E_{K2}(D_{K1}(C)))$$

pro dekódování. V případě, že potřebný 112-bitový klíč by byl nepraktický (např. proto, že existující systém distribuce počítá pouze s 56 bity), je možné použít 56-bitový klíč K ve funkci K_1 i K_2. Naopak dalšího posílení lze dosáhnout použitím 168-bitového klíče.

Proudová šifra DES

Útoky na blokové šifry, opírající se o znalost skutečnosti, že text je dělen na bloky, které jsou kódovány nezávisle, lze zkomplikovat *prodovou šifrou* získanou ve schématu na obrázku 9.10.



Obrázek 9.10: Proudová šifra DES

9.3.2 Kryptografie s veřejným klíčem

Použití shodného klíče pro kódování i dekódování komplikuje použití symetrické kryptografie v prostředí, kde kryptografického zabezpečení používá více partnerů. Každý komunikační kanál pak potřebuje vlastní klíč, nároky na generování klíčů a jejich distribuci rostou kvadraticky s počtem partnerů. Navíc je obtížné zajistit bezpečnou komunikaci, pokud je jeden z účastníků komunikace anonymní.

Řešení v těchto případech přináší asymetrické schéma, opírající se o kódování

$$C = E_B(P) \ ,$$

a o dekódování

$$P = D_B(C) \ .$$

Kódování E_B přitom využívá *veřejného klíče*, který přijímající partner B zpřístupní, zatímco sám pro dekódování D_B používá soukromý klíč, který si uchová v tajnosti. Metoda pochopitelně vyžaduje, aby z klíče E_B bylo velice obtížné odvodit klíč D_B, ať už přímo, nebo analýzou textů P a $E_B(P)$.

Příjemnou vlastností kryptografie se soukromým klíčem je i skutečnost, že ho lze snadno použít k autentizaci. Zprávu zakódovanou soukromým klíčem, tedy

$$C = D_B(P) \ ,$$

lze bez problémů dekódovat veřejným klíčem

$$P = E_B(D_B(P)) \ .$$

Lze ji však nesmírně obtížně modifikovat nebo podvrhnout.

RSA

Nejpoužívanější metodou veřejného klíče je algoritmu RSA (Rivest, Shamir, Adleman). Algoritmus se opírá o:

1- dvojici dostatečeně velkých prvočísel p a q (řádově 10^{100}),

2- čísla $n = p * q$ and $z = (p-1) * (q-1)$,

3- číslo d nesoudělné se z, a

4- číslo e takové, že $e * d = 1_{modz}$

Takto získaná čísla n, e a d jsou použita pro vlastní kódování

$$C = P^e_{modn} \ ,$$

a dekódování

$$P = C^d_{modn} \ .$$

Metoda RSA se opírá o vysokou složitost rozkladu velkých celých čísel a poskytuje dostatečnou při vhodné volbě délky klíče dostatečnou bezpečnost. Její nevýhodou je poměrně vysoká výpočetní náročnost kódování a dekódování, proto je využívána převážně pro předávání klíčů symetrikých šiter jako jsou DES, IDEA nebo Rijmdael.

Vedle faktorizace, o kterou se opírá RSA, jsou dnes využívány i další matematicky složité operace jako jsou výpočet diskrétních logaritmů, nebo eliptických funkcí.

Autentizace s využitím veřejného klíče

Veřejný klíč poskytuje poměrně bezpečný způsob vzájemné autentizace komunikujících partnerů. Mechanismus vzájemné autentifikace se opírá o posloupnost výměn zpráv:

$$E_B(A, R_A)$$

$$E_A(R_A, R_B, K_S)$$

$$K_S(R_B) \quad .$$

Příklad uvádí výměnu klíče K_S pro následující přenos dat zabezpečený symetrickým kódováním. Partner A (Alice) předává svou identifikaci A doplněnou o náhodnou posloupnost R_A kódovanou veřejným klíčem partnera B (Bob). Ten odpoví posloupností R_A, kterou dokázal dekódovat v přijaté zprávě, vlastní náhodnou posloupností R_B a klíčem pro symetrické kódování K_S. Vše kóduje veřejným klíčem partnera A. Posledním krokem postupu je potvrzení převzetí klíče K_S jeho použitím pro zakódování dekódované posloupnosti R_B.

Určitou nepříjemností metody je skutečnost, že se opírá o důvěru ve veřejný klíč partnera. Realizace metody vyžaduje použití důvěryhodné *certifikační autority*, organizace generující soukromé klíče pro své účastníky a poskytující informace o jejich veřejných klíčích.

Hashovací funkce

Často využívanou kryptografickou technologií jsou jednocestné *hashovací funkce* (*message digest*), které dovolí chránit přenášená data (i kryptograficky nekódovaná) proti změnám při přenosu sítí. Tyto funkce mají následující vlastnosti:

-pro přenášený text P lze snadno určit příznak $MD(P)$,

-pro dané $MD(P)$ není téměř možné zpětně zjistit P,

-je velice obtížné generovat P pro dané $MD(P)$.

V praxi jsou nejčastěji používány jednocestné hashovací funkce MD5 (Message Digest) a SHA-1 (Secure Hash Algorithm). Obě metody jsou založeny na rozkladu přenášeného textu na bloky o délce 512 bitů s tím, že poslední blok obsahuje případně doplněné výplňové bity a 64-bitový údaj o délce. Výstup funkce má v případě MD5 délku 128 bitů a je získán kombinací 128-bitových bloků vstupujícího textu s textem, získaným aplikací funkce *sin* (jde o zajištění důveryhodnosti faktu, že metodu nelze obejít). V případě SHA-1 je výpočet prováděn na 160 bitech (výstup SHA-1 má délku 160 bitů) a opírá se pouze o vstupní text.

Elektronický podpis

Kryptografický příznak lze společně s veřejnou kryptografií snadno využít pro elektronický podpis. Opíráme se zde o přenos originálního textu doplněného o příznak zakódovaný soukromým klíčem odesílatele

$$P, D_A(MD(P)) \quad .$$

Změna zprávy se projeví změnou potřebného příznaku, ale pro to je potřebný soukromý klíč odesílatele, který je velice obtížné získat.

10. Podpora distribuovaných aplikací

V minulé kapitole jsme se seznámili s rozhraním, které poskytuje typický operační systém pro vytváření distribuovaných aplikací. Knihovny podprogramů dovolující využít základní komunikační mechanismy sice postačí, pro snadný vývoj je výhodné mít vhodná jazyková primitiva ve vhodném jazyce.

10.1 Výměna zpráv

Libovolná komunikace mezi procesy umístěnými na různých uzlech distribuovaného systému je zprostředkována výměnou zpráv. Jako výměnu zpráv označujeme postup, při kterém jeden proces — vysílač S odesílá primitivou **send** E **to** R datovou strukturu E jinému procesu — přijímači R. Ten očekává předání struktury primitivou **receive** V (případně **receive** V **from** S, chceme-li explicitně označit odesílatele), kde V je označení proměnné vyhrazené pro její uložení.

Mechanismy výměny zpráv mezi dvěma procesy dělíme na dvě skupiny. Tyto skupiny se liší činností vysílače zprávy. U *synchronní výměny zpráv* musí vysílač počkat do okamžiku, kdy je k převzetí zprávy připraven i přijímač (samozřejmě předpokládáme, že přijímač musí počkat na vysílač, pokud je k převzetí zprávy připraven dříve). U *asynchronní výměny zpráv* může vysílač pokračovat ve výpočtu, zatímco zpráva je dočasně uložena kanálem a předána přijímači až v okamžiku, kdy je přijímač k převzetí zprávy připraven (přijímač však musí na zprávu v kanále počkat).

10.1.1 Asynchronní výměna zpráv

Asynchronní výměna zpráv mezi dvěma procesy — vysílajícím procesem S a přijímajícím procesem R využívá dvou primitiv send a receive, která mohou mít tvar:

send E **to** R

Vysílající proces S odesílá zprávu přijímajícímu procesu R. Po odeslání zprávy pokračuje vysílající proces S ve výpočtu.

receive V **from** S

Přijímající proces R očekává zprávu od vysílajícího procesu S. Po převzetí zprávy proces R pokračuje ve výpočtu.

Uvedená varianta je označována jako *přímá asynchronní komunikace* a je pro svou jednoduchost používána pro teoretické rozbory chování distribuovaných systémů. V praktických situacích je výhodné pojmenovat reálně existujícího prostředníka komunikace. Tímto prostředníkem je komunikační kanál C, při jeho explicitním začlenění do komunikačních primitiv mluvíme o *nepřímé asynchronní komunikaci*. Zůstaneme-li u komunikace našich dvou procesů S a R, mohou mít komunikační primitiva send a receive tvar:

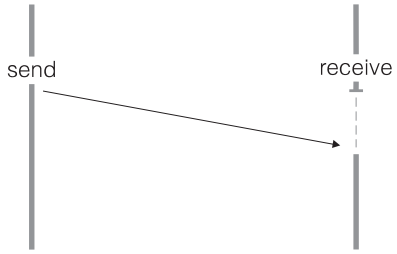
send E **to** C

Vysílající proces S předává zprávu pro přijímající proces komunikačnímu kanálu C. Po předání zprávy pokračuje vysílající proces S ve výpočtu.

receive V **from** C

Přijímající proces R očekává na komunikačním kanálu C zprávu od vysílajícího procesu. Po převzetí zprávy proces R pokračuje ve výpočtu.

Následující obrázek 10.1 ilustruje činnost asynchronního komunikačního kanálu při výměně jedné zprávy.



Obrázek 10.1: Asynchronní výměna zpráv

Nepřímá asynchronní komunikace podstatně rozšiřuje možnosti, které poskytuje komunikace přímá. Vypuštěním identifikace protějšího procesu z komunikačních primitiv (tedy identifikace příjemce R v primitivě send nebo odesílatele S v primitivě receive) získáme mechanismus, který dovolí různé formy multiplexu — příjem zpráv od více odesílatelů jedním příjemcem, příjem zpráv od jednoho odesílatele více příjemci nebo příjem zpráv od více odesílatelů více příjemci.

Poznámka
Z implementačních důvodů je obvykle kanál (resp. port) spojován s příjemcem. Takové omezení je většinou akceptovatelné.

10.1.2 Synchronní výměna zpráv

Synchronní výměna zpráv mezi dvěma procesy — vysílajícím procesem S a přijímajícím procesem R ve své jednodušší modifikaci využívá dvou primitiv.

send E **to** R

Vysílající proces S odesílá zprávu přijímajícímu procesu R. Pokračovat ve výpočtu může proces S *až po převzetí zprávy* procesem R.

receive V **from** S

Přijímající proces R očekává zprávu od vysílajícího procesu S. Pokračovat ve výpočtu může proces R až po převzetí zprávy od procesu S.

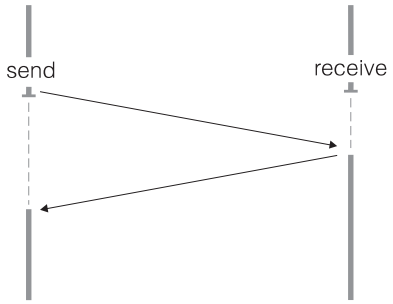
Tato varianta je označována jako *přímá synchronní komunikace* a je pro svou extrémní jednoduchost podobně jako přímá komunikace asychronní užívána především pro teoretické rozbory chování distribuovaných systémů. Je základem formalismu komunikujících sekvenčních procesů, zavedených Hoarem [8], najdeme ji v jazyce OCCAM 2. V obou případech se používá poněkud jiná notace, vyslání zprávy má formu

R!E ,

příjem zprávy má formu

S?V .

Podobně jako u asynchronní výměny zpráv je i u výměny synchronní v praxi výhodné pojmenovat prostředníka komunikace — komunikační kanál. Při explicitním začlenění komunikačního kanálu do komunikačních primitiv mluvíme o *nepřímé synchronní komunikaci*. Nepřímá synchronní komunikace, podobně jako nepřímá varianta komunikace asynchronní, podstatně rozšiřuje možnosti komunikace přímé.



Obrázek 10.2: Synchronní výměna zpráv

Modifikace

V řadě jazyků je konstrukce receive dále modifikována. Často požadujeme, aby přijímající proces mohl čekat zprávu současně na více kanálech. To dovolí například konstrukce:

select
  receive msg_1 **from** $chn_1 \Rightarrow stat_1$
  or receive msg_2 **from** $chn_2 \Rightarrow stat_2$
  or receive msg_3 **from** $chn_3 \Rightarrow stat_3$
end

kde označuje

msg_i	zprávu přijímanou na kanále chn_i,
chn_i	komunikační kanál a
$stat_i$	zpracování odpovídající příslušné variantě.

Výběr kanálů, na kterých očekáváme zprávu, lze omezit strážnou podmínkou, která může mít tvar:

select
  when g_1 **receive** msg_1 **from** $chn_1 \Rightarrow stat_1$
  or when g_2 **receive** msg_2 **from** $chn_2 \Rightarrow stat_2$
  or when g_3 **receive** msg_3 **from** $chn_3 \Rightarrow stat_3$
end

kde označuje

g_i	strážnou podmínku pro kanál chn_i.

Je-li splnění podmínky g_i závislé na obsahu zprávy msg_i (například na identifikaci odesílatele v textu zprávy) lze použít konstrukce:

select
  receive msg_1 **from** chn_1 **when** $g_1 \Rightarrow stat_1$
  or receive msg_2 **from** chn_2 **when** $g_2 \Rightarrow stat_2$
  or receive msg_3 **from** chn_3 **when** $g_3 \Rightarrow stat_3$
end

V řadě případů je vhodné mít konstrukci, která dovolí příjem od libovolného odesílatele nebo z libovolného komunikačního kanálu, například ve tvaru :

receive msg **from** any.

Uvedené konstrukce explicitně nedefinují strategii výběru zprávy v případě, že je přijatých zpráv více. Jde jednak o strategii výběru komunikačního kanálu nebo stráženého příkazu, jednak o strategii výběru jedné ze zpráv na jednom z kanálů.

Pokud jde o výběr kanálu nebo stráženého příkazu, většinou nepředpokládáme explicitní uspořádání, podle něhož je výběr kanálu realizován. Mluvíme o *nedeterministickém výběru* a prakticky necháváme výběr strategie na implementaci jazyka.

Výběr zpráv z jednoho kanálu obvykle odpovídá běžné strategii fronty (FIFO), modifikace této strategie je nutná, pokud například dovolíme strážený příkaz s výběrem podle obsahu zprávy.

10.2 Procedurální komunikace

Častým modelem komunikace je výměna dotazu a odpovědi mezi klientem a serverem. Takovou oboustrannou výměnu zpráv je výhodné vzít jako základní primitivu meziprocesové komunikace. Mluvíme o procedurální komunikaci a spolupráci procesů distribuovaného systému můžeme postavit na jejím základě. S procedurální komunikací se setkáváme u řady distribuovaných systémů. Dnes nejznámějšími jsou Amoeba, V-Kernel, Xerox Courier, SunOS NFS a Andrew.

10.2.1 Transakce v Amoebě

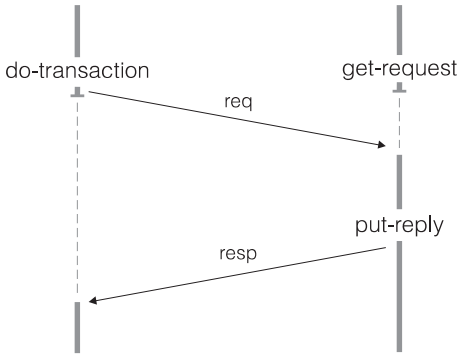
Jednodušší formu procedurální komunikace (z hlediska jazykové podpory) nalezneme u distribuovaného systému Amoeba [7], který je distribuovanou nadstavbou UNIXu. Komunikace procesů se opírá o transakce. Klient, který vyžaduje provedení funkce vzdáleného serveru, skládá z capability (což je autorizační přístupový mechanismus), z kódu požadované funkce a parametrů jejího volání zprávu. Tuto zprávu posílá serveru a pozastaví svůj výpočet. Server zprávu přebírá, provádí požadovanou funkci a vrací výsledek klientovi. Klient po převzetí odpovědi pokračuje ve výpočtu. Mechanismus se opírá o knihovnu procedur volatelných z jazyka C :

get-request(rq-header,rq-buffer,rq-size)
put-reply(rp-header,rp-buffer,rp-size)

do-transaction(rq-header,rq-buffer,rq-size,rp-header,rp-buffer,rp-size)

Hlavička požadavku a odpovědi obsahuje šestnáctiznakové pole capability, osmiznakový kód požadavku a osmiznakové pole dat, buffer je využíván pro předání delšího pole parametrů. Voláním procedury do-transaction žádá klient o odeslání požadavku, voláním procedury get-request očekává server požadavek, voláním procedury put-reply odesílá server odpověď.

Aby klient mohl dělat něco jiného souběžně s transakcí, případně aby bylo možné spustit více transakcí souběžně, lze jeden proces rozvětvit na nepreemptivně plánované "lightweight" procesy — *vlákna* (tento mechanismus nalezneme dnes prakticky u všech distribuovaných systémů odvozených z UNIXu).

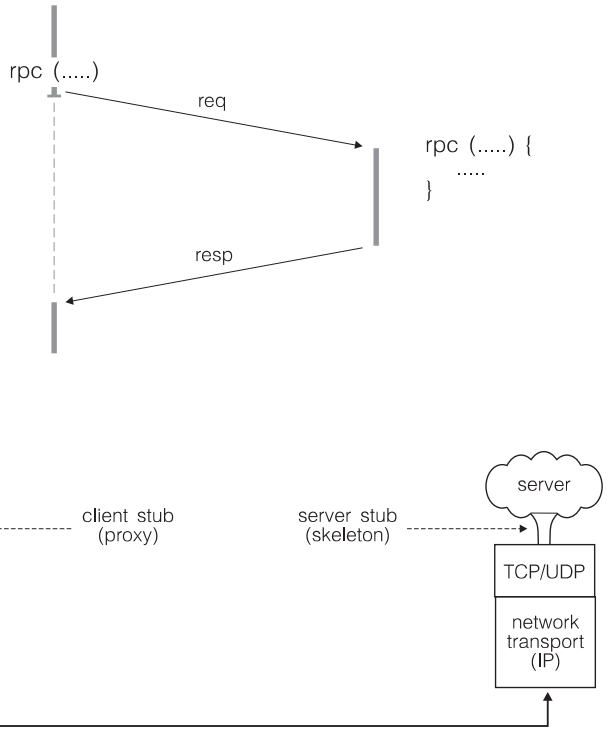


Obrázek 10.3: Transakce v Amoebě

Protokol Amoeby se opírá, na rozdíl od jiných distribuovaných rozšíření UNIXu, přímo o datagramovou komunikaci v lokální síti (a ne např. o protokoly Internetu — TCP nebo UDP). Výsledkem je velká efektivita.

10.2.2 RPC mechanismus

Procedurální komunikace připomíná běžnou proceduru a jedna z jejích forem je proto standardně nazývána voláním vzdálených procedur (Remote Procedure Call, RPC mechanismus). Ve své klasické formě zahrnuje RPC mechanismus kromě vlastní výměny žádosti mezi klientem a serverem, provedení funkce serverem a vrácení odpovědi klientovi ještě mechanismus, který dává procedurální komunikaci formu běžného volání procedury. Tento mechanismus je označován jako stub a popíšeme si ho ve tvaru, v jakém ho pro firmu Xerox navrhli Birrell a Nelson.



Obrázek 10.4: RPC mechanismus

Mechanismus RPC se skládá ze:

-stubu na straně klienta, který transformuje volání RPC procedury na zprávu předávanou komunikačním systémem vzdálenému serveru a zpětně transformuje odpověď serveru,

-stubu na straně serveru, který transformuje přijatou zprávu na volání RPC procedury a výsledek transformuje na odpověď odesílanou klientovi, a

-vlastního kódu RPC procedury na serveru.

Prvky RPC mechanismu vytváří překladač z popisu RPC procedury, pro překlad obou stubů je potřebná znalost seznamu parametrů.

Volání každé RPC procedury je zprostředkováno dvojicí stubů (na straně klienta a serveru), pro volání RPC procedury na vzdáleném serveru musíme mít k dispozici odpovídající stub.

Komunikační podpora RPC

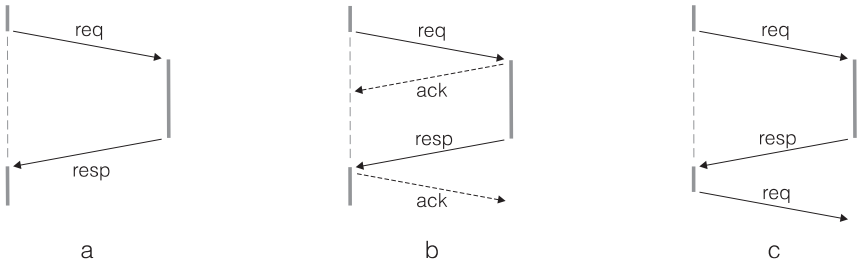
Mechanismus RPC je postaven nad komunikační systém, který vzájemně propojuje jednotlivé uzly a procesy distribuovaného systému. Na jeho vlastnostech závisí, jak dokonale podpoří procedurální komunikaci a nakolik ho budeme nuceni doplnit. Z našeho pohledu můžeme komunikační systémy rozdělit do tří skupin:

-virtuální kanál mezi klientem a serverem,

-datagramová komunikace zachovávající pořadí (např. lokální síť), a

-datagramová komunikace nezachovávající pořadí (např. internet).

Virtuální kanál, realizovaný například transportním komunikačním protokolem TCP, zabezpečí svými vnitřními mechanismy spolehlivé předání požadavku a odpovědi. Vlastní RPC mechanismus v takovém jednoduchém případě popisuje obr. 10.5a. Nemáme-li takový spolehlivý prostředek, nebo nám takový prostředek svou vysokou režií nevyhovuje, musíme doplnit méně spolehlivou datagramovou službu vlastním koncovým potvrzováním. Jednoduché potvrzování dovolí automaticky opakovat poškozený požadavek nebo odpověď a jeho funkci ilustruje obr. 10.5b. V uvedené formě ovšem znamená výměnu čtyř zpráv pro jedno RPC (proti dvěma u ideální komunikace). Existující implementace (Amoeba, V-Kernel) nahrazují kladné potvrzení požadavku odpovědí a kladné potvrzení odpovědi dalším požadavkem téhož klienta (obr. 10.5c). Počet výměn je tak redukován na dvě zprávy na RPC (za předpokladu absence chyby).



Obrázek 10.5: Výměny zpráv pro mechanismus RPC

Sémantika RPC

Procedurální komunikace RPC se podobá běžnému volání procedur a mechanismus RPC (stub) rozdíly záměrně zakrývá. Mechanismus RPC a běžné volání procedury se však liší svou sémantikou.

Rozdíl je způsoben tím, že každý z procesů, klient a server, může běžet na jiném počítači. Požadavek efektivity pak přirozeně omezuje typy parametrů, které RPC mechanismus používá. Nejsou povolené odkazy, seznamové struktury, procedury. Volání odkazem je převáděno na kopírování hodnot a důsledkem je posun sémantiky, ilustrovaný klasickým příkladem:

client

```
var x:integer;
begin
  x:=1;
  dbinc(x,x)
end;
```

server

```
RPC dbinc(var a,b:integer);
  begin
    a:=a+1;
    b:=b+1
  end
```

Na rozdíl od běžné procedury, která dvakrát inkrementuje tutéž buňku paměti, RPC mechanismus inkrementuje každou ze dvou kopií této buňky pouze jednou.

Dalším problémem, se kterým se musí RPC mechanismus a jeho uživatel vyrovnat, jsou důsledky ztrát požadavků a odpovědí a výpadky systémů, na nichž běží proces.

Ztráta požadavku má za následek, že server požadavek do časového limitu nepotvrdí, nebo nedodá odpověď. Čekajícího klienta můžeme uvolnit nebo požadavek automaticky opakovat, v prvém případě však server požadovanou funkci neprovede. Ztráta potvrzení požadavku, nebo ztráta odpovědi se na straně klienta projeví zcela shodně, automatické opakování požadavku však může vést na násobné provedení požadované funkce (nedokáže-li klient duplikát požadavku rozpoznat).

Důsledkem ztrát zpráv a výpadku procesů je další posun sémantiky. Ukončíme-li RPC mechanismus po ztrátě odpovědi vypršením časového limitu na straně klienta, je důsledkem sémantika označovaná jako *at-most-once*, server požadovanou funkci nemusí provést. Opakujeme-li naopak, po vypršení časového limitu, automaticky RPC žádost, je důsledkem sémantika označovaná jako *at-least-once*, server může požadovanou funkci provést opakovaně.

Výše uvedené sémantiky přirozeně respektují vlastnosti komunikačního podsystému. V řadě aplikací je můžeme připustit. V měřících systémech může být postačující sémantika at-most-once, pokud nás zajímá pouze poslední hodnota vysílaná sensorem — klientem. V regulačních systémech může postačovat sémantika at-least-once, pokud je serverem proces nastavující určitý ovládací prvek na zadanou hodnotu.

Pokud nejsme ochotni se s některou jednoduchou sémantikou smířit, musíme automatické opakování požadavku klientem doplnit o schopnost serveru rozeznat duplikáty požadavků. Běžně toho dosáhneme číslováním požadavků a kontrolou jejich posloupnosti. (Pro detekci duplikátu a jeho vyloučení je ovšem nutné uchovat nejen čísla předchozích požadavků a identifikaci klientů, ale také odeslané odpovědi.) Výsledkem je sémantika blízká sémantice *exactly-once*, na jakou jsme zvyklí u běžných procedur. Chceme-li dosáhnout čisté sémantiky exactly-once musíme zachovat stav serveru (spojený s RPC mechanismem) i při jeho výpadku.

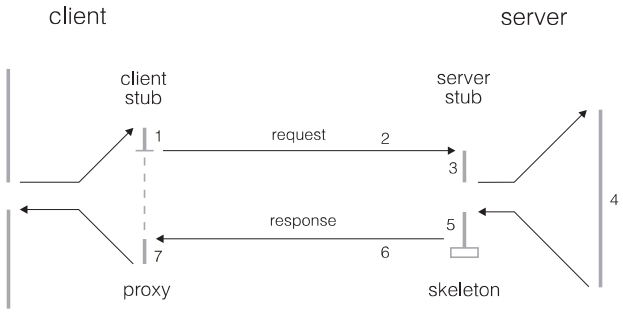
Kromě ztráty požadavku a odpovědi a výpadku serveru působí potíže i výpadek klienta. Odpovědi na požadavky vyslané před výpadkem klienta je nutné rozlišit od odpovědí na požadavky vyslané po jeho restartu.

Komunikace Remote Procedure Call (RPC) je podporovanou formou komunikace v řadě programévací prostředků/jazyků. Předpokládá nesymetrické postavení aplikačních procesů a nevyžaduje explicitní navazování spojení.

Model spolupráce aplikačních procesů, pro který se RPC používá označujeme obvykle jako model Client–Server. Klient žádá konkrétní server o provedení určité akce zasláním požadavku. Server, který na takovou žádost od libovolného klienta čeká, požadovanou akci provede. Po jejím ukončení odešle server odpověd, na kterou klient mezitím čekal.

Z popisu komunikace mezi klientem a serverem vidíme nesymetrii jejich postavení a absenci předchozího navazování spojení mezi nimi. Tato forma komunikace se podstatně liší od modelu ISO, ale je velmi blízká jinému modelu, na který jsme zvyklí z programovacích jazyků, totiž volání podprogramu. Na požadavek klienta se můžeme dívat jako na vstupní parametry volání podprogramu, na odpověd serveru se můžeme dívat jako na parametry výstupní. Jediným rozdílem je fakt, že na rozdíl od běžného podprogramu je akce serveru realizována na vzdáleném počítači, mimo prostředí klienta. To má sice mnohé podstatné sémantické důsledky, ale jednoduchost modelu je pádným důvodem pro to, aby byla komunikace RPC začleněna do řady operačních systémů, které počítají se spoluprací vzdálených aplikací.

V systémech SunOS a Xerox Courier (obojí je rozšířením UNIXu) zapisujeme v jazyce C komunikaci RPC stejně jako volání podprogramu. Při sestavování je však připojen speciální připravený podprogram označovaný jako stub. Ten převede parametry volání na zprávu, která je prostřednictvím transportní vrstvy předána vzdálené aplikaci. Tam je zpráva protějškem stubu převedena opět na seznam parametrů a zavolán podprogram, který provede požadovanou akci. Jeho výsledky uloží vzdálený stub do zprávy, která je po přenosu zpětné transformována na seznam parametrů. Ty jako výsledek konečně dostává volající aplikační program. Posloupnost akcí spojených s komunikací RPC uvádí obrázek 10.6.



Obrázek 10.6: Remote Procedure Call

Posloupnost akcí spojených s komunikací RPC může být narušena výpadkem některého z procesů. Jestliže vypadne server, klient neobdrží odpověď a aplikační program zůstane zablokovaný čekáním na ní. Z této situace se lze dostat stanovením časového limitu, potom je však nutné rozhodnout (a může to udělat stub nebo aplikační program), budeme–li požadavek opakovat.

Požadavek je opakovatelný bez jakýchkoliv problémů pouze, je–li příslušná operace serveru *idempotentní* (změna stavu serveru nezávisí na stavu předcházejícím operaci). V opačném případě se budeme snažit o to aby se operace provedla právě jednou (zdvojení operace výběru z bankovního konta nebude asi žádoucí), mluvíme o požadavku na *only once* sémantiku (která odpovídá přirozené sémantice volání běžného podprogramu). Snaha o tuto sémantiku komplikuje celý mechanismus i funkci serveru, proto se často omezíme na jednodušší řešení.

Sémantika *at most once* (operace je provedena nejvýše jednou) odpovídá jednoduchému ukončení komunikace RPC po vypršení timeoutu bez opakování žádosti. Sémantika *at least once* (operace je provedena alespoň jednou) odpovídá opakování až do příjmu prvé odpovědi a je výhodná pro idempotentní operace.

V předchozích odstavcích jsme si všímali důsledků výpadku serveru. Problémy však přináší i výpadek klienta, kdy aktivizovaný server osiří. Důsledky mohou být někdy pouze zanedbatelné (zbytečné ztráta času procesoru, nutnost zlikvidovat odpověď serveru), jindy mohou být značně nepříjemné (zablokování relace v databázi).

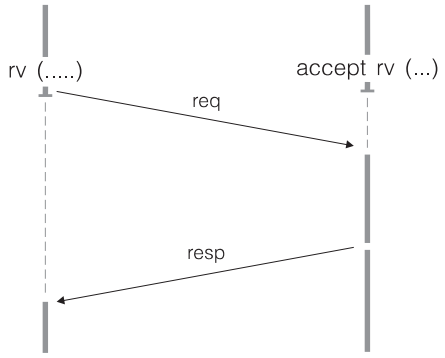
V obou případech, ať jde o výpadek serveru nebo klienta, je výhodné, má–li používaný programovací jazyk mechanismus výjimek.

10.2.3 Rendez-vous

Rendez-vous, které se stalo populárním díky jazyku Ada, je procedurálním mechanismem, který má ve své nejjednodušší podobě formu:

client	server
	accept pname (fplist);
call pname (aplist);	**begin**
	end;

Klient předává příkazem call serveru požadavek na provedení určité funkce. Server, takový požadavek očekává příkazem accept, provádí požadovanou funkci a vrací výsledek klientovi. Oba procesy po ukončení rendez-vous pokračují ve výpočtu. Příkaz accept v Adě připouští nedeterministický výběr z více čekajících požadavků (konstrukce select). Mechanismus rendez-vous uvádí obr. 10.7.



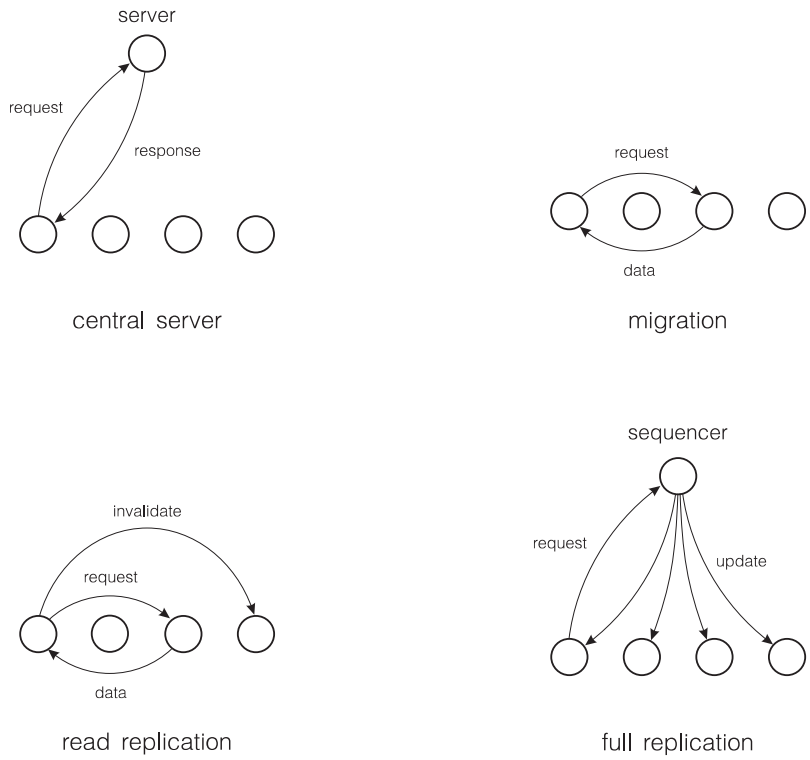
Obrázek 10.7: Rendez-vous

10.3 Distribuovaná sdílená paměť

Kromě předávání zpráv a procedurálních komunikačních mechanismů přichází v úvahu při programování distribuovaných systémů ještě další model spolupráce — *sdílení proměnných*. Spolupracující procesy používají pro přístup ke sdíleným proměnným operace READ a WRITE (podobně jako při přístupu k vlastním proměnným), případně doplněné o synchronizaci. Takový způsob spolupráce je přirozený v distribuovaných systémech s těsnou vazbou, ve kterých existuje nějaká forma sdílení fyzické paměti. Svůj význam má sdílení proměnných i v případech, kdy jakákoliv forma sdílení fyzické paměti je nedostupná, tedy v distribuovaných systémech s volnou vazbou.

Metody sdílení

Nejjednodušší formou sdílení je *pevné umístění proměnné* na jednom z uzlů distribuovaného systému. Operace READ a WRITE libovolného procesu nad sdílenou proměnnou jsou převedeny na zprávy obsahující žádost o čtení nebo zápis hodnoty a zaslány na uzel udržující sdílenou proměnnou. Provedení operace je procesu indikováno zprávou, která pro operaci čtení obsahuje hodnotu sdílené proměnné. Jednotlivé proměnné sdílené paměti mohou být umístěny na jednom uzlu nebo rozloženy na více uzlech, například s ohledem na jejich optimální dostupnost.



Obrázek 10.8: Metody sdílení proměnných

Poněkud složitějším způsobem sdílení je *migrační algoritmus*. Sdílené proměnné nebo jejich skupiny nemají v systému pevné místo. Libovolný proces operací READ nebo WRITE nad sdílenou proměnnou žádá o její přenesení na svůj uzel. Po přenosu je příslušná operace provedena obdobně jako operace nad vlastní proměnnou.

Obě metody se opírají o jedinou kopii sdílené proměnné. Jednoduchost těchto metod je vykoupena nutností přenášet hodnotu této proměnné, přesto, že se tato v průběhu výpočtu programu nemění. Pokud počet operací READ silně převažuje nad počtem operací WRITE, je

výhodnější sáhnout k některému z dále uvedených způsobů *sdílení s replikací*.

Jednodušší *částečná replikace* se opírá o jediný "originál" proměnné, který lze číst i aktualizovat a o řadu "read-only" kopií, které lze pouze číst. Operace READ nad sdílenou proměnnou může být předcházena žádostí o poskytnutí "read-only" kopie a jejím předáním, tato kopie je k dispozici pro následující operace READ až do aktualizace proměnné libovolným z procesů. Operace WRITE žádá o aktualizaci proměnné vlastníka "originálu" (v případě jeho pevného umístění), nebo žádá o přenesení "originálu" (u migrační varianty), aby mohla být aktualizace provedena lokálně. Před vlastní aktualizací je nutné zrušit všechny "read-only" kopie rozesláním příslušného požadavku jejich vlastníkům.

Komplikovanější než částečná replikace je *replikace úplná*, kdy v distribuovaném systému existuje více rovnocenných kopií sdílené proměnné. Operace READ pracuje nad lokální kopií, pokud tato není dostupná, lze o překopírování požádat některý z uzlů, který kopii vlastní. Operace WRITE realizují všichni vlastnící kopií proměnné, přičemž je nutné použít některý z algoritmů, který zabezpečí shodnou posloupnost aktualizací na všech uzlech (takovým algoritmům se budeme věnovat v následující kapitole).

Alternativou ke sdílení jednotlivých proměnných může být sdílení stránek paměťových prostorů mezi procesy na různých procesorech. S takovou distribuovanou sdílenou pamětí se můžeme setkat v operačních systémech Sprite a Mach [7].

10.3.1 Linda

Důležitou vlastností distribuované sdílené paměti je metoda identifikace hodnot a proměnných v ní uložených. Zajímavý mechanismus je použit v jazyce Linda: sdílená paměť je označována jako *tuple space* a dovoluje ukládat uspořádané n-tice (*tuples*). Nad sdílenou pamětí lze aplikovat tři atomické operace: operace *out* n-tici do sdílené paměti zapíše, operace *read* n-tici přečte a operace *in* n-tici po přečtení ze sdílené paměti odstraní.

Například n-tice ["john",31,true] je tvořena řetězem znaků, celým číslem a logickou hodnotou. Operace

$$\textbf{out}\,(\text{"}john\text{"}, 31, true)$$

uloží tuto n-tici do sdílené paměti, operace

$$\textbf{read}\,(\text{"}john\text{"}, \textbf{var}\ age, \textbf{var}\ maried)$$

ji přečte a odpovídající položky uloží do proměnných *age* a *maried*. Pokud je ve sdílené paměti více n-tic, jejichž první pole má hodnotu *"john"*, je vybrána jedna z nich nedeterministicky. Jako klíč pro výběr může sloužit libovolná podmnožina položek n-tice. Konečně operace

$$\textbf{in}\,(\text{"}john\text{"}, \textbf{var}\ age, \textbf{var}\ maried)$$

přečtenou n-tici ze sdílené paměti odstraní.

Operace *read* a *in* pozastaví výpočet procesu, pokud není ve sdíleném prostoru požadovaná n-tice k dispozici. Operace *in* nad určitou n-ticí je výlučná.

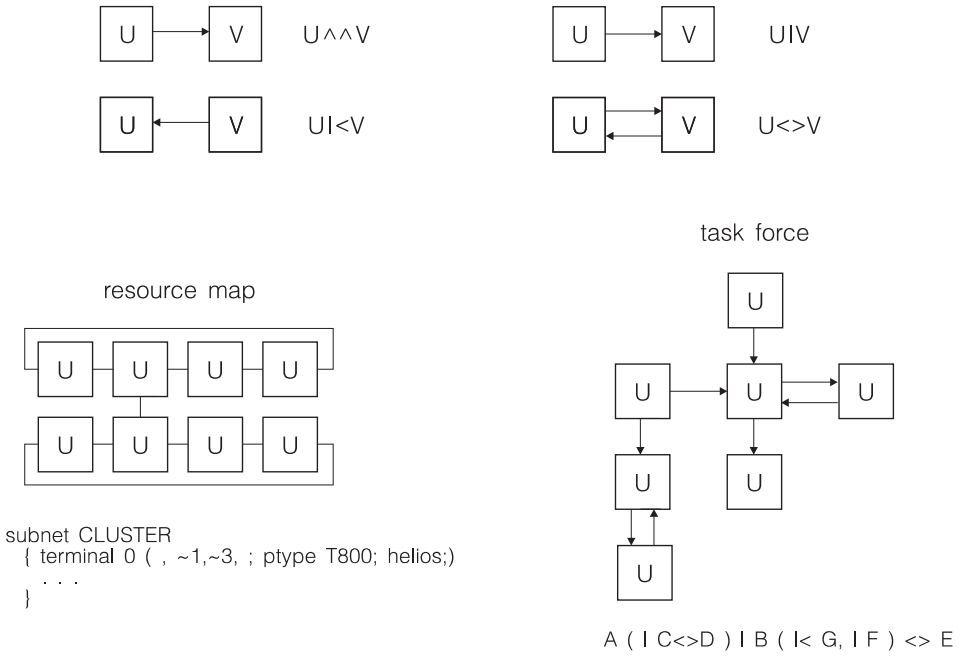
11. Distribuované algoritmy

Pod pojmem distribuovaný algoritmus budeme rozumět postup, při kterém skupina sekvenčních procesů vzájemně spolupracuje na řešení nějakého konkrétního problému.

Nejjednodušší formou spolupráce procesů je použití výstupu jednoho procesu jako vstupu procesu jiného, postupné zpracování informace posloupností procesů označované jako *pipeline*. Tuto formu spolupráce procesů dává k dispozici například operační systém UNIX, využití paralelismu tohoto typu většinou nepřináší podstatné změny ve způsobu programování aplikací a nevyžaduje zvláštní konstrukce pro meziprocesovou komunikaci.

Další typické schéma spolupráce je označováno jako *client-server*. Jeden z dvojice procesů, *klient*, žádá provedení určité služby druhým z dvojice, *serverem*. Server dokončení své služby oznámí klientovi současně s předáním výsledků. Na rozdíl od předchozího mechanismu je spolupráce obou procesů podstatně užší, bývá podporována jazykovými konstrukcemi (RPC, rendez-vous) a v případě pasivních serverů (server, který pouze reaguje na vnější požadavky a nemá svůj vlastní výpočet) je přímou obdobou monitorů [3].

Jednoduché formy distribuce jsou v praxi zajímavé a kladou (zvláště pipeline, kde vystačíme se sekvenčním zápisem procesů) minimální nároky na programování. Jako příklad systému, který se o tyto formy distribuce opírá, si uvedeme operační systém *Helios* (UNIXem inspirovaný operační systém vytvořený pro transputerové systémy, ale podporující i heterogenní hardwareové konfigurace). Ten zahrnuje mechanismy dovolující popsat globální strukturu programu složeného z takto komunikujících procesů; konfigurační jazyk *CDL* (*Component Distribution Language*) popisuje propojení procesů, a společně s popisem hardwareové architektury systému (*Resource Map*) definuje mapování procesů na procesory. Příklad konfigurace programu v systému Helios uvádí obr. 11.1.



Obrázek 11.1: Konfigurace distribuovaného programu a mapování na architekturu v Heliosu

V případě distribuce typu *pipeline* a *client-server* je chování systému definováno převážně sekvenčním výpočtem jednotlivých procesů, jejichž vzájemná spolupráce se omezuje na předávání mezivýsledků. Vzhledem k vyhraněné funkci jednotlivých procesů takové formy distribuce označujeme jako *asymetrické*. Protipólem asymetrických systémů jsou systémy, ve kterých jednotlivé procesy spolupracují na řešení problému, výsledek je produktem spolupráce více procesů, které jsou si podobné. Takové systémy a způsoby kooperace označujeme jako *symetrické*.

Typickými symetrickými algoritmy jsou algoritmy *difuzní*. Výpočet se opírá o sousednost procesů. Ta je definována strukturou vzájemného propojení procesů a budeme ji označovat jako *graf komunikací*.

Čistě difuzní výpočet má periodický charakter. Každý z procesů na základě informace, kterou obdržel od svých sousedů, provede krok výpočtu a rozešle výsledky, které jsou pro jeho sousedy vstupem v dalším kroku. Jednoduché algoritmy tohoto typu dovolují získat procesům informace o topologii vzájemných propojení (procesy si vyměňují a aktualizují matice grafu), o nejkratších cestách mezi procesy (procesy si vyměňují informace o odhadu vzájemných vzdáleností - Floyd-Fulkersonův algoritmus). Algoritmy tohoto typu jsou pro svou pravidelnou funkci někdy označovány jako *heart-beat* algoritmy.

Řada algoritmů využívá pouze podgrafu vzájemných propojení, nejčastěji kostry grafu (*difuzní* algoritmy) nebo Hamiltonovského cyklu (*token-passing* algoritmy).

Pro jednoduchou spolupráci procesů ve schématech *pipeline* a *client-server* je typické, že jednotlivé procesy plní odlišnou funkci a zpracovávají odlišný program. U *čistě difuzních algoritmů* jsou naopak jak zpracovávaný program tak i chování jednotlivých procesů shodné. Mezi těmito dvěma extrémy existuje řada forem přechodných.

O *plné symetrii* (striktní symetrii) mluvíme, jestliže jednotlivé procesy mají shodný text programu i chování. Příkladem jsou difuzní algoritmy.

U *textové symetrie* mají sice jednotlivé procesy týž program, jejich chování se však při výpočtu liší. Typickým příkladem jsou systémy s automatickou rekonfigurací, ve kterých určitou klíčovou funkci (např. funkci centrálního serveru) plní jeden z procesů, ale tento proces byl vybrán na základě volby mezi procesy. Každý z procesů má k dispozici potřebný kód, ale v určité fázi výpočtu si procesy funkce zvolí a jejich chování se od tohoto okamžiku liší.

Pojem *slabá symetrie* používáme v případech, kdy jednotlivé procesy procházejí delšími fázemi odlišného chování. Jako příklad mohou sloužit algoritmy s předáváním pověření (token passing), u kterých má v daném čase vždy jeden z procesů poněkud odlišné chování než procesy ostatní.

O *asymetrii* mluvíme, liší-li se jak kód programu, tak i výsledné chování jednotlivých procesů. Typickým příkladem je spolupráce mezi pevně určeným serverem a klienty.

11.1 Čas v distribuovaných systémech

Jedním z podstatných problémů spojených s distribuovanými systémy je skutečnost, že je značně obtížné získat informaci o globálním stavu systému. Informace, které může proces získat o chování jiných procesů, dostává se zpožděním. Roli hraje hlavně zprostředkované předávání informace mezi procesy, ale často musíme vzít v úvahu i technická omezení (omezená kapacita kanálů a režim jejich obsluhy, omezená doba rychlosti šíření signálů). Dosáhnout, aby proces mohl použít při rozhodování aktuální globální informace, může být obtížné.

Cestou, jak se přiblížit k získání informace o globálním stavu distribuovaného systému, je zavést do jeho popisu čas (tedy prvek, kterému se v klasickém programování raději vyhýbáme). Jednodušší *logický čas* se opírá o kauzalitu sekvenčního výpočtu procesu a o kauzalitu při předávání zpráv. *Fyzický čas* je snahou vytvořit na distribuovaném systému "hodiny", které dovolí mluvit o stavu v daném "časovém okamžiku".

11.1.1 Kauzální uspořádání událostí

Zavedení logického času využívá faktu, že z hlediska výpočetního chování systému (bez vnějšího pozorovatele) je možné globální astronomický čas nahradit jakýmkoliv uspořádáním, které zachová kauzalitu událostí.

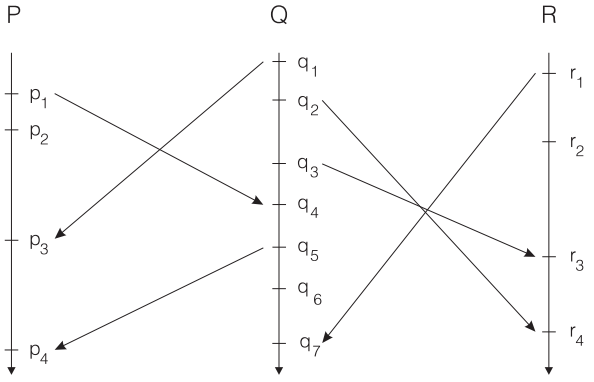
Pro vyjádření následnosti událostí zavádíme relaci $\rightarrow$. Relace definuje ireflexivní částečné uspořádání na množině událostí v systému. Události a a b jsou v relaci $\rightarrow$ (tuto skutečnost zapisujeme $a \rightarrow b$), jestliže buď

-a a b jsou události jediného procesu a událost a nastala před událostí b, nebo

-událost a je odeslání zprávy jedním procesem a událost b je příjem této zprávy jiným procesem, nebo

-existuje událost c taková, že $a \rightarrow c$ a $c \rightarrow b$.

Nejsou-li události a a b v právě definované relaci $\rightarrow$, pak je označujeme jako *souběžné* (concurrent) a tuto skutečnost vyjadřujeme zápisem $a \not\rightarrow b$ nebo $b \not\rightarrow a$.



Obrázek 11.2: Logický čas

Relaci $\rightarrow$ si lze pro konkrétní chování systému znázornit "časoprostorovým" diagramem systému (obr. 11.2). Jednotlivým procesům (v našem případě P, Q a R) odpovídají svislé časové osy, jejich událostem uzly (p_i, q_i a r_i) a předávání zpráv orientované hrany mezi nimi. Nahradíme-li si časové osy spojením bezprostředně následujících událostí jednotlivých procesů orientovanými hranami, pak události a a b jsou v relaci $a \rightarrow b$, jestliže mezi událostmi a a b existuje v diagramu systému orientovaná cesta mezi a a b (s počátkem v a a s koncem v b).

11.1.2 Skalární logický čas

Uspořádání $\rightarrow$ na množině událostí nám dovoluje zavést čas v systému. Přesněji, dovoluje nám pro každý proces P_i zavést funkci C_i (s číselným oborem hodnot), kterou budeme označovat jako *lokální čas* procesu P_i. Pro systém funkcí C_i musí platit

$$(a \rightarrow b) \Rightarrow (C_i(a) < C_j(b)), \text{ kde } a \in P_i \text{ a } b \in P_j \ .$$

Jestliže událost a předchází události b, pak lokální čas procesu P_i odpovídající události a musí být menší než lokální čas procesu P_j odpovídající události b.

Podmínku, kterou musí splňovat systém lokálních časů lze nejjednodušeji splnit zavedením čítače v každém procesu. Takový *čítač* bude inkrementován mezi každými dvěma událostmi příslušného procesu $\mathrm{P_i}$ (hodiny "tiknou") a podmínka je tak splněna pro události $a \in \mathrm{P_i}$ a $b \in \mathrm{P_i}$. Čítače různých procesů $\mathrm{P_i}$ a $\mathrm{P_j}$ budeme vzájemně synchronizovat tak, že při každém odeslání zprávy procesem $\mathrm{P_i}$ (událost $a \in \mathrm{P_i}$) předávanou zprávu doplníme o *časové razítko* Tm, které odpovídá okamžité hodnotě čítače $\mathrm{C_i}$. Jestliže při příjmu této zprávy procesem $\mathrm{P_j}$ (událost $b \in \mathrm{P_j}$) je hodnota časového razítka Tm větší nebo rovna hodnotě čítače $\mathrm{C_j}$, posuneme čítač $\mathrm{C_j}$ na hodnotu větší než je hodnota časového razítka (zajistíme tak platnost relace $\rightarrow$ pro předání zprávy). Před další událostí procesu $\mathrm{P_j}$ čítač $\mathrm{C_j}$ inkrementujeme.

Systém lokálních časů s uvedenou synchronizací obvykle označujeme jako logický čas nebo podle autora myšlenky jako *Lamportovy hodiny*. Relace definovaná logickým časem je rozšířením relace $\rightarrow$, pokud jsou události a a b v relaci $a \rightarrow b$, pak nutně $\mathrm{C_i}(a) < \mathrm{C_j}(b)$.

Pro funkci logických hodin není podstatné, zda jsou čítače inkrementovány o jedničku nebo o jinou hodnotu.

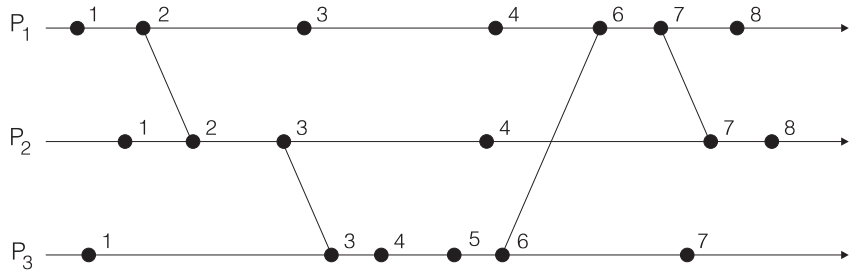
Úplné uspořádání

V řadě případů je potřebné úplně uspořádat události v systému a to relace $\rightarrow$ podporovaná výše definovanými časovými razítky nezajišťuje. Částečné uspořádání lze však na uspořádání úplné $\rightarrow$ převést velmi snadno. Procesy očíslujeme a časovým razítkem budeme rozumět dvojici ($\mathrm{C_i}$,$\mathrm{P_i}$), budeme tedy předávat vedle lokálního času i identifikace vysílače. Úplné uspořádání $\rightarrow$ je pak definováno takto:

$$\begin{array}{ll} \text{je} - \text{li } C_i(a) < C_j(b) & \text{pak } a \rightarrow b, \\ \text{je} - \text{li } (C_i(a) = C_j(b)) \& (i < j) & \text{pak } a \rightarrow b \ . \end{array}$$

Poznámka
Uspořádání zavedené logickým časem může být nepříjemné pro vnějšího účastníka. Zadá-li tento účastník požadavek a procesu P_i a později (z hlediska svého vlastního se systémem nekoordinovaného času) požadavek b procesu P_j je docela dobře možné, že tyto dva požadavky budou z hlediska systému uspořádány opačně. Každému z požadavků totiž bude přiřazen lokální čas příslušného procesu. Jediným způsobem, jak takovému anomálnímu chování zabránit, je zahrnout vnějšího účastníka do systému. Bude si udržovat svůj vlastní lokální čas, ten však musí koordinovat s procesy sledovaného systému.

Příklad synchronizace, opírajícího se o skalární logický čas, uvádí obrázek 11.3.

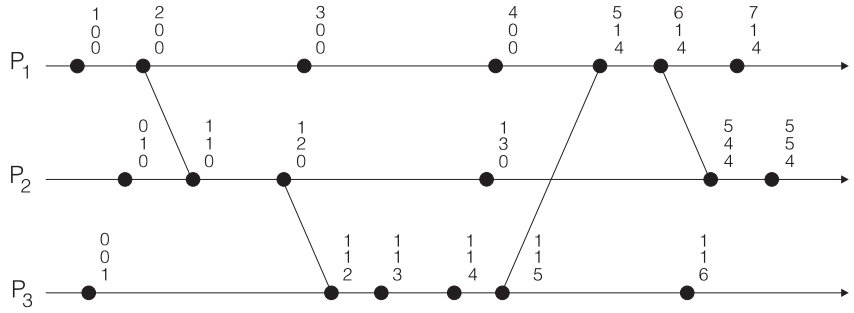


Obrázek 11.3: Skalární logický čas

11.1.3 Vektorový logický čas

O skalární logický čas je možné se opřít u algoritmů, kterým postačí jednoznačně uspořádat události distribuovaného výpočtu. Patří sem např. zajištnění výlučného přístupu. Algoritmům, které vyžadují kompletní informaci o kauzálním uspořádání, skalární čas nestačí; řešením je *vektorový logický čas*. Opírá se o vektory pro uložení lokální informace o událostech v distribuovaném systému a o vektory jako časové známky. Je využíván např. algoritmy podporujícími replikaci.

Příklad synchronizace, opírajícího se o vektorový logický čas, uvádí obrázek 11.3.



Obrázek 11.4: Vektorový logický čas

Konečně, v určitých situacích můžeme sáhnout po maticovém logické čase, ten využijeme např. v mechanismech, které podporují restart jednotlivých procesů po výpadcích.

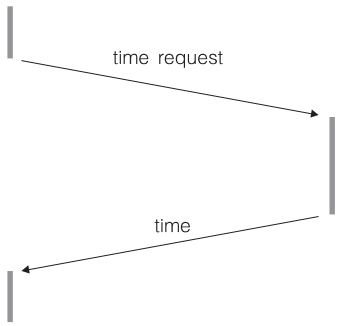
11.1.4 Fyzický čas

Běžnější než zahrnout vnějšího účastníka do vytváření logického času je svázat lokální čas procesů i vnějšího účastníka s časem, který se blíží času fyzikálnímu. Vnějšímu účastníku i procesu umožníme číst lokální hodinový čítač, který inkrementujeme (o určitý krok) po daném časovém intervalu. Lokální čítače procesů jsou řízeny vlastními kmitočtovými zdroji, důsledkem jsou určité rozdíly v rychlosti nárůstu hodnot. Periodickým předáváním služebních zpráv přenášejících časová razítka a dostavováním hodinových čítačů lze dosáhnout, že se hodinové čítače příliš nerozejdou.

Jednoduché nastavení lokálního hodinového čítače na maximum z jeho okamžité hodnoty a z přijatého časového razítka (zvýšeného o odhadnuté zpoždění při přenosu) vede na přizpůsobení se všech hodinových čítačů nejrychlejšímu z nich. Pokud chceme aby se výsledný čas systému

přizpůsobil vybranému hodinovému čítači — *time-serveru*, nebo aby odpovídal průměru lokálních časů, je potřebné rychlejší čítače zpomalovat a pomalejší zrychlovat. Čítač lze zpomalit pouze změnou hodnoty inkrementu, posunutí některého hodinového čítače zpět by porušilo základní podmínku kladenou na čas v systému.

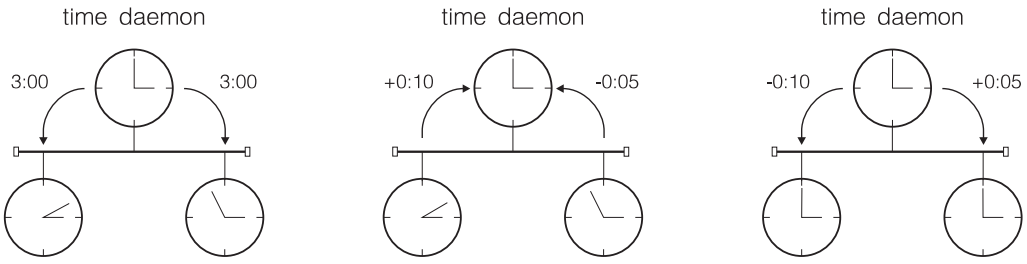
Rozptyl hodnot jednotlivých čítačů závisí na odchylkách v jejich rychlosti, na frekvenci služebních zpráv a na rozptylu zpoždění těchto zpráv při přenosu. Střední zpoždění služebních zpráv je nutné vyloučit. V případě centrálního time-serveru lze vyloučení těchto zpoždění dosáhnout statistickým vyhodnocováním doby, kterou vzdálený procesor potřebuje, aby dostal odpověď na svou žádost o poskytnutí časového údaje. Vliv zpoždění, která se projevují v systému s centrálním *time-serverem* (Cristianův algoritmus) ilustruje obr. 11.5.



Obrázek 11.5: Získání časového údaje od time-serveru

Centrální time-server, který definuje standardní čas v systému, se obvykle opírá o přesný časový zdroj. Tím může být vysílání časových standardů — stanic jako jsou WWV (Fort Collins, Colorado) nebo DCF(Braunschweig).

V systémech bez určeného time-serveru lze dosáhnout vzájemné synchronizace hodinových čítačů difuzním algoritmem, při kterém procesor (resp. specializovaný proces — time-daemon) periodicky rozesílá ostatním procesorům žádosti o poskytnutí hodnoty jejich hodinového čítače, vyhodnocuje odpovědi a rozesílá žádosti o seřízení čítačů. Takovýto algoritmus je použit v Berkeley UNIXu a jeho funkci ilustruje obr. 11.6.



Obrázek 11.6: Nastavování hodinových čítačů v Berkeley UNIXu

V praxi se lze setkat i s kombinací přístupů, kdy se opíráme při určování fyzického času o více časových zdrojů. Příkladem je OSF DCE (Open Software Foundation's Distributed Computing Environment), který dovoluje využívat více časových serverů (časových standardů).

11.2 Algoritmy zajišťující výlučný přístup

Algoritmy, které zabezpečují synchronizaci procesů a jejich výlučný přístup ke sdíleným prostředkům (do kritických sekcí) patří v distribuovaných systémech bez sdílené paměti k nejdůležitějším. Proto je jim také v literatuře věnována největší pozornost a my s nimi náš přehled začneme.

11.2.1 Lamport

Prvním známým algoritmem v této oblasti je algoritmus, který popsal Lamport jako aplikaci logických hodin [16]. Cílem algoritmu je zajistit výlučný přístup ke sdílenému prostředku skupinou N procesů a to při splnění následujících podmínek:

1Proces, kterému byl prostředek vyhrazen musí tento prostředek uvolnit před jeho přidělením dalšímu procesu.

2Požadavky procesů na vyhrazení prostředku musí být vyřizovány v pořadí jejich zadání (ve smyslu logického času).

3Pokud každý z procesů uvolní prostředek v konečném čase, pak je v konečném čase vyřízena i každá žádost o přidělení.

Lamportův algoritmus si nyní slovně popíšeme.

Proces P_i žádá o přidělení prostředku tím, že rozešle všem ostatním procesům $P_j, j \neq i$, zprávu $REQUEST$ obsahující časové razítko $Tm = (C_i, P_i)$. Proces P_j, který zprávu $REQUEST$ přijme, zařadí požadavek do své fronty přijatých požadavků podle hodnoty časového razítka. Příjem požadavku potvrdí proces P_j zprávou $REPLY$, která také obsahuje časové razítko; časové razítko přijatého potvrzení si proces P_i poznamená. (Proces P_j nemusí zprávu $REQUEST$ potvrzovat, pokud již dříve poslal procesu P_i nějakou zprávu s pozdějším razítkem, než je razítko přijatého požadavku).

Je-li konečně procesu P_i prostředek přidělen (podmínky přidělení si uvedeme dále) a proces P_i ho po konečné obsluze uvolňuje, učiní tak rozesláním zprávy $RELEASE$ všem ostatním procesům $P_j, j \neq i$. Na příjem zprávy $RELEASE$ reaguje proces P_j vymazáním požadavku procesu P_i ze své fronty požadavků. Časové razítko zprávy $RELEASE$ si proces P_j poznamená.

Klíčovým bodem algoritmu jsou podmínky, za kterých je procesu P_i vyhrazen přístup ke sdílenému prostředku. Pro Lamportův algoritmus tyto podmínky jsou:

1Vlastní požadavek procesu P_i je nejstarším požadavkem v jeho frontě požadavků.

2Proces P_i přijal od každého jiného procesu P_j nějakou zprávu s pozdějším razítkem.

Algoritmus vyžaduje, aby zprávy od každého z procesů byly doručovány v pořadí jejich odeslání. Proces P_i ví, že pokud přijal nějakou zprávu od procesu P_j pozdější než jeho vlastní požadavek, pak již přijal i všechny předchozí zprávy procesu P_j a mezi nimi i případný dosud nevyřízený požadavek. Každý z N procesů si proto udržuje dvě základní lokální struktury — frontu požadavků a pole přijatých časových razítek (indexované číslem procesu).

Nevýhodou Lamportova algoritmu je vysoký počet vyměňovaných zpráv, pro zajištění jednoho přístupu je potřeba vyměnit až $3 * (n - 1)$ zpráv.

Počet zpráv, které si mezi sebou procesy v Lamportově algoritmu vyměňují, je možné snížit, vynecháme-li zprávu $RELEASE$ (která slouží k informování všech účastníků o uvolnění prostředku) a dáme-li procesu P_j možnost pozdržet odpověď na požadavek procesu P_i až po splnění svého staršího požadavku. Zjednodušení je základem Ricart-Agarwalova algoritmu [17] a redukuje počet vyměňovaných zpráv na $2 * (n - 1)$.

11.2.2 Ricart-Agarwala

Proces žádá o vstup do kritické sekce tím, že rozešle ostatním procesům zprávu $REQUEST$. Na zprávu $REQUEST$ odpoví každý z procesů zprávou $REPLY$ buď bezprostředně, nebo odpověď pozdrží až do ukončení svého vlastního požadavku. Algoritmus je založen na skutečnosti, že každý proces může snadno rozhodnout, kterým došlým požadavkům ostatních procesů dá přednost před vlastním požadavkem na vstup do kritické sekce. Algoritmus si popíšeme pro i-tý z N procesů.

```
when request                                          { žádost i-tého procesu }
   P(S); Req[i]:=T; MyRq:=MaxRq+1; V(S);
   RpCnt:=0;
   for j:=1 to N do
      if j<>i then
         send REQUEST(MyRq,i) to j;
   wait RpCnt=N-1;

      { kritická sekce }

   Req[i]:=F;
   for j:=1 to N do                          { rozeslání pozdržených odpovědí }
      if Req[j] then
         begin
            Req[j]:=F;
            send REPLY to j
         end

when received REQUEST(k,j) do                  { přijata žádost j-tého procesu }
   begin
      MaxRq:=max(MaxRq,k);
      P(S); Delay:=Req[i] and ((k>MyRq) or (k=MyRq and j>i)) V(S);
      if Delay
      then
         Req[j]:=T
      else
         send REPLY to j
   end

when received REPLY do                                    { přijata odpověď }
   RpCnt:=RpCnt+1

begin                                                        { inicializace }
   MaxRq:=0; MyReq:=F;
   for j:=1 to N do
      Req[j]:=F
end
```

Každý z N procesů je jednoznačně označen identifikačním číslem procesu a udržuje si několik lokálních proměnných — sekvenční číslo vlastní žádosti $MyRq$, nejvyšší z vlastních a dosud přijatých sekvenčních čísel $MaxRq$ a pole registrující žádosti jednotlivých procesů na vstup do kritické sekce.

Přijme-li náš (i-tý) proces žádost jiného procesu j, upraví hodnotu proměnné $MaxRq$. Na základě poznámky o vlastní žádosti $Req[i]$ a na základě sekvenčního čísla požadavku k (a v případě rovnosti porovnáním identifikací procesů i a j) se rozhodne, zda žádost potvrdí, nebo zda dá přednost vlastnímu požadavku na přístup do kritické sekce a žádost pozdrží. Ve druhém případě si poznamená pozdrženou odpověď v proměnné $Req[j]$.

Vlastnímu požadavku našeho procesu je přiřazeno sekvenční číslo $MyRq$ vyšší, než sekvenční číslo libovolné dosud přijaté žádosti jiného procesu (náš proces nemůže předběhnout žádný z procesů, kterým již žádost potvrdil) a požadavek je zaregistrován v proměnné $Req[i]$. Žádost s číslem $MyRq$ je rozeslána ostatním procesům a náš proces je pozdržen až do přijetí všech odpovědí. Počet odpovědí na náš požadavek udržujeme v proměnné $RpCnt$. Do kritické sekce smí proces vstoupit až po příjmu odpovědí od všech $N-1$ zbývajících procesů.

Požadavky ostatních procesů přicházející během čekání jsou potvrzovány nebo pozdrženy. V době výpočtu našeho procesu v kritické sekci již žádný požadavek s nižším nebo stejným sekvenčním číslem přijít nemůže, požadavky s vyššími sekvenčními čísly musí počkat na výstup z kritické sekce. Po výstupu z kritické sekce náš proces zruší vlastní registraci v proměnné $Req[i]$ (to dovolí bezprostředně odpovídat na přicházející žádosti) a rozešle všechny pozdržené odpovědi (registrované v indexované proměnné Req).

Přístupy do kritické sekce jsou identifikovány neklesajícími sekvenčními čísly žádostí. Rovnost sekvenčních čísel je řešena ve prospěch procesu s nižším identifikačním číslem. Odpověď $REPLY$ je odeslána jinému procesu na jeho žádost $REQUEST$ teprve tehdy, když proces odsouhlasí dřívější vstup tohoto procesu do kritické sekce.

Sekvenční čísla žádostí spolu s identifikačními čísly procesů vytvářejí uspořádání na žádostech procesů o vstup do kritické sekce. Preference procesů s nižšími identifikačními čísly nevyvolá starvaci procesů s čísly vyššími.

Modifikace algoritmu

Na závěr popisu Ricart-Agarwalova algoritmu si uvedeme zajímavé modifikace, které dovolí snížit počet předávaných zpráv. Ten je při použití dvoubodově předávaných zpráv roven hodnotě $2 * (n-1)$.

Negativní potvrzování

Odpověď $REPLY$ na žádost o vstup do kritické sekce předává pouze jeden bit informace. Lze-li v síti zaručit spolehlivé doručení zprávy do nějakého rozumného časového limitu, lze pozitivní odpověď nahradit vypršením časového limitu na straně žádajícího procesu. Pokud tedy proces přijatou žádost akceptuje, pak neodpoví; pokud chce proces přijatou žádost pozdržet, pak odpoví specielní zprávou s významem "pozdržuji požadavek" a později zprávou s významem "souhlasím s přístupem". Počet zpráv vyměněných pro zajištění jednoho požadavku bude v intervalu mezi $1 * (n-1)$ a $3 * (n-1)$ a bude záviset na intenzitě požadavků. Při nízké intenzitě požadavků lze podstatně snížit počet předávaných zpráv.

Broadcast

Dovoluje-li přenosové médium při vyslání zprávy jedním z procesů její převzetí všemi ostatními procesy, lze rozesílání žádosti $REQUEST$ jednotlivým procesům nahradit vysláním této žádosti broadcastem. Počet zpráv vyměněných mezi procesy se tak sníží na N nezávisle na intenzitě požadavků.

Zkombinujeme-li obě uvedené modifikace, dovolíme-li tedy nahradit odpověď na žádost vyslanou broadcastem implicitním "mlčením", lze snížit počet vyměněných zpráv na 1 až $1+2*(n-1)$ pro jeden požadavek. Při nízké intenzitě požadavků se lze přiblížit ideální hodnotě jedné předané zprávy na požadavek. U Lamportova algoritmu lze podobně dosáhnout snížení počtu zpráv vysíláním žádostí a odpovědí při výstupu z kritické sekce broadcastem, minimální počet vyměněných zpráv na jeden požadavek je roven dvěma.

11.2.3 Carvalho-Roucairol

Algoritmus Ricart-Agarwaly lze (kromě uvedených modifikací využívajících vhodných vlastností přenosového média) dále zefektivnit, uvědomíme-li si následující skutečnost. Zprávou *REPLY* předává j-tý proces našemu i-tému procesu pověření ke vstupu do kritické sekce. Pokud chce tento j-tý proces získat toto pověření od našeho i-tého procesu zpět, požádá o to zprávou *REQUEST*. Postačující podmínkou pro vstup do kritické sekce je soustředění pověření od všech ostatních procesů, přičemž nezáleží na tom, zda procesu toto pověření zůstalo k dispozici od jeho posledního vstupu do kritické sekce, nebo o něj musel požádat zprávou *REQUEST*.

Oproti Ricart-Agarwalově algoritmu musí algoritmus popsaný Carvalho a Roucairolem [18] udržovat přehled o poskytnutých pověřeních, v našem popisu v indexované proměnné *Grant*. Algoritmus si opět popíšeme pro i-tý z N procesů.

```
when request                                              { žádost i-tého procesu }
   P(S); Req[i]:=T; MyRq:=MaxRq+1; V(S);
   for j:=1 to N do
      if j<>i and (not Grant[j]) then
         send REQUEST(MyRq,i) to j;
   wait (Grant[j]=T for j<>i);
   Req[i]:=F; InUse:=T;

      { kritická sekce }

   InUse:=F;
   for j:=1 to N do                          { rozeslání pozdržených odpovědí }
      if Req[j] then
         begin
            Grant[j]:=F; Req[j]:=F;
            send REPLY to j
         end

when received REQUEST(k,j) do                  { přijata žádost j-tého procesu }
   begin
      MaxRq:=max(MaxRq,k);
      P(S); Delay:=((k>MyRq) or (k=MyRq and j>i)) V(S);
      if InUse or (Req[i] and Delay) then
         Req[j]:=T;
      if not(InUse or Req[i])or(Req[i]and(not Grant[j])and(not Delay)) then
         send REPLY(i) to j;
      if (Req[i] and Grant[j] and (not Delay)) then
         begin
            Grant[j]:=F;
            send REPLY(i) to j;
            send REQUEST(MyRq,i) to j
         end
   end

when received REPLY from j do                  { přijata odpověď j-tého procesu }
   Grant[j]:=T

begin
   MaxRq:=0; MyRq:=0;                                          { inicializace }
   for j:=1 to N do
      begin Req[j]:=F; Grant[j]:=F end
end
```

Algoritmus potřebuje pro jeden vstup do kritické sekce výměnu 0 až $2 * (N - 1)$ zpráv. Dolnímu limitu se přibližuje, vstupuje-li do kritické sekce častěji pouhá část procesů, nebo opakují-li procesy své vstupy.

11.2.4 Ricart-Agarwala (Token Passing)

V předchozím odstavci popsaný Carvalho-Roucairolův algoritmus je velice blízký dalšímu algoritmu pro synchronizaci vstupů do kritické sekce, který popsali Ricart a Agarwala [19]. Jejich algoritmus se opírá o explicitní předávání jediného globálního pověření — *tokenu* mezi procesy.

```
when request do                                          { žádost i-tého procesu }
   if not TokenHeld then
      begin
         Clock := Clok+1;
         broadcast REQUEST(Clock,i);                     { rozeslání požadavku }
         receive TOKEN;                                    { čekání na pověření }
         TokenHeld:=T
      end;
   InUse:=true;

      { kritická sekce }

   Token[i]:=Clock;
   InUse:=F;
   j:=(i+1) mod N;
   while i<>j do
      begin
         if Req[j]>Token[j] and TokenHeld then
            begin                                  { předání pověření dalšímu procesu }
               TokenHeld:=F; send TOKEN to j;
               j:=(j+1) mod N
            end
      end

when received REQUEST(k,j) do                    { příjem požadavku j-tého procesu }
   begin
      Req[j]:=max(Req[j],k);
      if TokenHeld and not InUse then
         begin
            j:=(i+1) mod N;
            while i<>j do
               begin
                  if Req[j]>Token[j] and TokenHeld then
                     begin
                        TokenHeld:=F; send TOKEN to j;
                        j:=(j+1) mod N
                     end                           { předání pověření dalšímu procesu }
               end
         end
   end

begin                                                                { inicializace }
   for j:=1 to N do
      Req[j]:=0;
   Clock:=0
end
```

Každý z procesů je jednoznačně identifikován číslem i a udržuje si informaci o okamžitém držení pověření v proměnné $TokenHeld$ a o jeho aktivním využití při vstupu do kritické sekce v proměnné $InUse$. V indexované proměnné Req proces registruje požadavky ostatních procesů na poskytnutí pověření. Pole $Token$ je aktualizováno při každém vstupu do kritické sekce a je předáváno mezi procesy současně s pověřením.

Algoritmus se opírá o dva typy zpráv, žádosti $REQUEST$ a pověření $TOKEN$. Zprávou $REQUEST$ proces, který chce získat pověření ke vstupu do kritické sekce, žádá ostatní procesy o poskytnutí pověření. Zpráva $REQUEST$ kromě identifikace žádajícího procesu informuje o jeho logických hodinách *Clock*. Zprávou $TOKEN$ předává proces, který pověření okamžitě drží, toto pověření procesu, který o pověření požádal. Zpráva $TOKEN$ přenáší obsah pole *Token*, které udržuje informaci o vstupech do kritické sekce realizovaných jednotlivými procesy.

Pro získání pověření musí proces rozeslat všem ostatním procesům žádost a počkat si na jeho příchod. Opírá-li se algoritmus o dvoubodovou komunikaci, znamená to, že na obdržení pověření je nutné vyměnit N zpráv. Podobně jako v případě předchozích algoritmů lze tento počet podstatně zredukovat dovolí-li komunikační systém rozeslání žádosti jediným broadcastem. V takovém případě se počet zpráv potřebných pro získání potvrzení rovná dvěma.

Algoritmus v tom tvaru, ve kterém je popsán nevyžaduje rozesílání žádosti v případě, že proces již drží pověření (test hodnoty proměnné $TokenHeld$ ve druhé řádce). Výsledkem může být další snížení intenzity zpráv, samozřejmě opět závislé na režimu vstupů procesů do kritické sekce.

Další zajímavostí jsou logické hodiny procesu, realizované proměnnou *Clock*. Na rozdíl od předchozích algoritmů neslouží tyto hodiny k jednoznačnému uspořádání vstupů do kritické sekce a k vytvoření "společného času".

Cyklické předávání pověření

K velice často využívanému a jednoduchému algoritmu dospějeme, definujeme-li na množině procesů pevné pořadí (logický kruh), ve kterém si procesy pověření předávají. Proces, kterému pověření předáme, určíme velice snadno, je jím například vždy soused s nejblíže vyšším identifikačním číslem (pochopitelně modulo N). Tato snadnost je však vykoupena skutečností, že pověření na cestě k žadateli musí být předáno i celou řadou procesů, které o ně nežádaly (roste počet vyměněných zpráv při nízké intenzitě požadavků).

V praktickém použití bývá tento algoritmus ještě dále zjednodušen a to tím, že pověření je předáváno mezi sousedními procesy nezávisle na existenci požadavku, žádosti tedy procesy nerozesílají. Pověření "obíhá" logický kruh, právo na přístup do kritické sekce má jediný proces, který si na dobu práce v kritické sekci pověření podrží. Algoritmus vyžaduje alespoň jednu zprávu na jeden vstup do kritické sekce (pro rovnoměrně rozdělené a časté vstupy). Další úspora (více vstupů do kritické sekce na jedno pověření) není možná, chybějící žádosti by vedly k monopolu jediného procesu a ke starvaci procesů ostatních. Shora není počet vyměněných zpráv na jeden vstup do kritické sekce omezen, pověření "obíhá" i pokud žádný z procesů nepožaduje vstup do kritické sekce.

Cyklické předávání pověření, často většinou spojené i se stejnou strukturou komunikačního systému, je pro praktické použití ještě nutné doplnit určitými ochrannými mechanismy. Zaručit, že každá zpráva dojde ke svému adresátovi, v technickém systému nemůžeme. Připustíme-li ztrátu zprávy, v našem případě pověření, připustíme, že se nechráněný systém dostane do zablokovaného stavu. Žádný z procesů nemá pověření (to se ztratilo) a nemůže ho tedy ani žádnému dalšímu procesu předat. Pro vyvedení algoritmu ze zablokovaného stavu, regeneraci pověření, se obvykle využívá časového limitu, hrubého prostředku, který je nutné pečlivě sladit s časovým chováním systému procesů v běžném provozu. Kromě časového limitu však pro cyklicky předávané pověření existuje i čistě algoritmická metoda regenerace pověření, kterou si nyní uvedeme.

Regenerace pověření

Algoritmus, který navrhl Misra [20] se opírá o dvě pověření označená jako $PING$ a $PONG$, která "obíhají" logický kruh ve vzájemně opačném smyslu a vzájemně se zajišťují. Pověření přenášejí čítače počtu setkání $NPing$ a $NPong$. Při setkání obou pověření v některém procesu je čítač $NPing$ zvyšován o jedničku, čítač $NPong$ je o jedničku snižován. Za normálního provozu vždy platí

$$NPing + NPong = 0 \ .$$

Kromě hodnot $NPing$ a $NPong$ přenášených pověřeními si každý z procesů v logickém kruhu udržuje proměnnou M, nastavovanou při průchodu jediného z pověření procesem na hodnotu $NPing$ nebo $NPong$. Algoritmus je založen na faktu, že přijme-li kterýkoliv proces pověření s čítačem rovným M, pak to znamená, že se toto pověření při posledním průchodu logickým kruhem nesetkalo se svým protějškem, ani tento protějšek mezitím neprošel procesem. Tak lze testovat správnost průchodu obou pověření. V případě zjištění ztráty některého z pověření proces, který tuto ztrátu zjistil, ztracené pověření regeneruje.

Při zahájení algoritmu mají čítače pověření hodnoty $NPing = 1$ a $NPong = -1$, pro všechny procesy s výjimkou jednoho jsou indikátory M nulovány, zbylý proces bude počátečním držitelem obou pověření a jeho indikátor bude nastaven na hodnotu $M = 1$.

```
when received PING(NPing) do
   if M=NPing                          { ztráta pověření Pong a jeho regenerace }
   then
      begin
         NPing:=NPing+1;
         NPong:=-NPing
      end
   else
      M:=NPing

when received PONG(NPong) do
   if M=NPong                          { ztráta pověření Ping a jeho regenerace }
   then
      begin
         NPong:=NPong+1;
         NPing:=-NPong
      end
   else
      M:=NPong

when meeting (PING,PONG) do                          { setkání Ping a Pong }
   begin
      NPing:=NPing+1;
      NPong:=NPong-1
   end
```

Algoritmus ve tvaru, v němž byl popsán, se opírá o celá neomezená čísla $NPing$ a $NPong$, jejichž přenos nelze konečnou zprávou zajistit. Při praktické realizaci algoritmu musíme rozsah hodnot čítačů $NPing$ a $NPong$ omezit a výpočet následující hodnoty realizovat modulo P, kde $P \geq N + 1$ (počet procesů v logickém kruhu zvýšený o jedničku).

11.3 Algoritmy výběru

Doposud jsme se zabývali algoritmy, které splňovaly podmínku symetrie, každé distribuované rozhodnutí bylo realizováno procesy, které vyhodnocovaly stejný kód. Výhoda symetrie však byla zaplacena poměrnou komplikovaností (z hlediska návrhu a srozumitelnosti chování) a vysokým počtem vyměňovaných zpráv. Snížení počtu zpráv jsme dosáhli oslabením požadavku na symetrii u algoritmů s předáváním pověření. Dalšího snížení počtu vyměňovaných zpráv zřejmě dosáhneme, vyhradíme-li příslušnou funkci (například povolování vstupů do kritické sekce) jedinému z procesů, který pak označujeme jako *koordinátor* nebo obecně *server*.

Výhodou tohoto řešení je snadnost centralizované implementace, nevýhodou je citlivost celého systému na výpadek zvoleného procesu — serveru. Abychom tuto nevýhodu omezili na minimum, dovolíme realizovat příslušnou funkci každému procesu a pro volbu jediného z nich, který bude funkci zajišťovat použijeme symetrický distribuovaný algoritmus.

Jednoduchý difuzní algoritmus pro volbu serveru lze popsat následovně:

Proces, který chce být zvolen zahájí výběr rozesláním zprávy $REQUEST$ se svou vlastní identifikací svým sousedům a očekává návrat odpovědí $REPLY$. Každý proces si udržuje pole $Voting$ indikující snahu jednotlivých procesů o zvolení, pole $Parent$, indikující proces, od kterého zpráva $REQUEST$ přišla, a pole $NResp$ indikující počet očekávaných odpovědí. Po příchodu zprávy $REQUEST$ proces upraví příslušnou položku polí $Voting$ a $Parent$, rozešle zprávy $REQUEST$ ostatním sousedům a nastaví hodnotu položky pole $NResp$.

Proces, který zjistí opakovaný příchod zprávy $REQUEST$ (podle informace v poli $Voting$) vrátí odpověď $REPLY$ s hodnotou, která je maximem z hodnoty přenášené zprávou $REQUEST$ a z vlastní identifikace procesu. Stejně se zachová proces, který již nemá dalšího souseda. (Tento postup omezí výpočet algoritmu na kostru grafu komunikací.)

Proces, který obdrží odpověď $REPLY$ si poznamená výsledek v dané větvi do pole $Voting$ (maximum z původní a přijaté hodnoty). Po příjmu všech požadovaných odpovědí odpoví sám procesu identifikovanému položkou v poli $Parent$. Hodnota, kterou vrací ve zprávě $REPLY$, je přitom maximem z hodnoty v položce pole $Voting$ a z identifikace procesu.

Jakmile obdrží proces, který algoritmus odstartoval všechny požadované odpovědi, rozhodne, zda byl zvolen (všechny odpovědi obsahují jeho vlastní identifikaci) nebo nikoli.

Nevýhodou uvedeného algoritmu je skutečnost, že pro N procesů musíme vyměnit až $2*N*N$ zpráv, následující algoritmy počet zpráv redukují.

11.3.1 Chang-Roberts

Algoritmus publikovaný Changem a Robertsem [21] se opírá o jednoznačnou číselnou identifikaci procesů, které si předávají zprávy po logickém kruhu. Proces, který algoritmus výběru zahajuje, pošle svůj identifikátor svému levému sousedovi ve zprávě *ELECTION*. Proces, který zprávu *ELECTION* přijímá, porovná přenášenou hodnotu se svou vlastní identifikací. Je-li vlastní číselný identifikátor procesu větší než přijatá hodnota, proces odešle zprávu *ELECTION* s vlastním identifikátorem svému levému sousedovi. Je-li vyšší přijatá hodnota, proces odešle zprávu *ELECTION* beze změny. Rovnost hodnoty přijaté ve zprávě *ELECTION* s vlastním číselným identifikátorem indikuje skutečnost, že daný proces je identifikován nejvyšším identifikátorem mezi všemi procesy kruhu a že jeho výběr všechny ostatní procesy schválily.

Proces, který uvedeným způsobem zjistil svou volbu, odešle svému levému sousedovi zprávu *ELECTED* se svou identifikací, touto zprávou ho informuje o výsledku volby. Proces, který zprávu *ELECTED* přijímá, si poznamená identifikaci nového koordinátora v proměnné *Coordinator* a předá zprávu *ELECTED* levému sousedovi. Po návratu zprávy *ELECTED* ke zvolenému procesu je algoritmus výběru ukončen. Proměnné Voting jednotlivých procesů indikují probíhající hlasování.

```
when decision INITIATE_ELECTION do                    { rozhodnutˇ volit }
   begin
      Voting:=T;
      sendl ELECTION(i)
   end

when received ELECTION(j) do                          { pc˘jem zpr vy ELECTION }
   begin
      if j>i then
         begin
            sendl ELECTION(j);
            Voting:=T
         end;
      if j<i and not Voting then
         begin
            sendl ELECTION(MyNumber);
            Voting:=T
         end;
      if j=i then
         begin
            sendl ELECTED(i)
         end
   end

when received ELECTED(j) do                           { pc˘jem zpr vy ELECTED }
   begin
      Coordinator:=j;
      Voting:=F;
      if j<>i then sendl ELECTED(j)
   end

begin                                                 { inicializace }
   Voting:=F; Coordinator:=0
end
```

Z hlediska počtu předávaných zpráv $ELECTION$ (počet zpráv $ELECTED$ je vždy roven počtu procesů N) je nejvýhodnější situace, kdy algoritmus zahajuje proces s nejvyšším číselným identifikátorem. Minimem je N předaných zpráv. Nejhorší situace odpovídá uspořádání procesů podle stoupajících hodnot identifikátorů a současnému startu algoritmu ve všech procesech. Počet vyměněných zpráv je pak roven hodnotě $0.5 * n * (n + 1)$. Průměrný počet vyměněných zpráv pro všechny možné konfigurace a způsoby zahájení výpočtu je $n * logn$.

11.3.2 Hirschberg-Sinclair

O kruhovou komunikaci mezi procesy se opírá i algoritmus publikovaný Hirschbergem a Sinclarem [22]. Stejně jako Chang-Robertsův algoritmus využívá jednoznačnou číselnou identifikaci procesů, předpokládá ale obousměrný kruh s těmito komunikačními primitivami:

sendLRodeslání zprávy oběma sousedům,

passpředání zprávy přijaté od jednoho souseda druhému sousedovi,

respondodpověď sousedovi na přijatou zprávu.

Algoritmus je založen na následující myšlence: Proces P_i, který zahajuje volbu se prohlásí za kandidáta a svou kandidaturu rozešle svým dvěma sousedům, procesům P_j a P_k, primitivou *sendLR*. Ti porovnají číselnou identifikaci odesílatele zprávy se svou vlastní identifikací a vrátí primitivou *respond* odpověd s větším číslem procesu P_i. Pokud má některý z procesů P_j a P_k vyšší identifikaci než proces P_i stává se kandidátem pro další kolo (mohou jimi být i oba). Pokud tomu tak není, nevstupují do dalších kroků volby, ale pouze předávají zprávy obou základních typů primitivou pass. Podobně proces P_i, pokud daném kroku volby zvítězil, je kandidátem pro další kolo. Pokud prohrál, předává až do ukončení volby zprávy primitivou pass.

Volba probíhá tak dlouho, než jsou konzultovány všechny procesy na kruhu. Vítězný proces rozpozná svou volbu příjmem vlastní žádosti o zvolení, průchod této žádosti celým kruhem indikuje, že s volbou každý jiný proces na kruhu souhlasí.

Algoritmus se sice opírá o složitější komunikační primitiva než algoritmus předcházející, snižuje však nejvyšší počet vyměňovaných zpráv na $O(n.logn)$.

Nižšího maximálního počtu vyměňovaných zpráv lze dosáhnout i na jednosměrném kruhu. Algoritmus, jehož autory jsou Dolev, Klawe a Rodeh, se opírá o předchozí metodu, krok volby opět vybírá mezi třemi aktivními sousedy, s tím, že rozhodování provádí nejlevější ze všech tří procesů za proces střední.

11.4 Prevence a detekce zablokování

Vážným problémem kooperace více procesů je vzájemné zablokování (deadlock) p)i přístupu ke sdíleným prostředkům, nebo při vzájemné (meziprocesové) komunikaci. Setkáváme se s ním i v centrálně spravovaných operačních systémech, kde se pro řešení využívá centralizovaných algoritmů [23]. V tomto textu nás budou zajímat symetrické distribuované algoritmy.

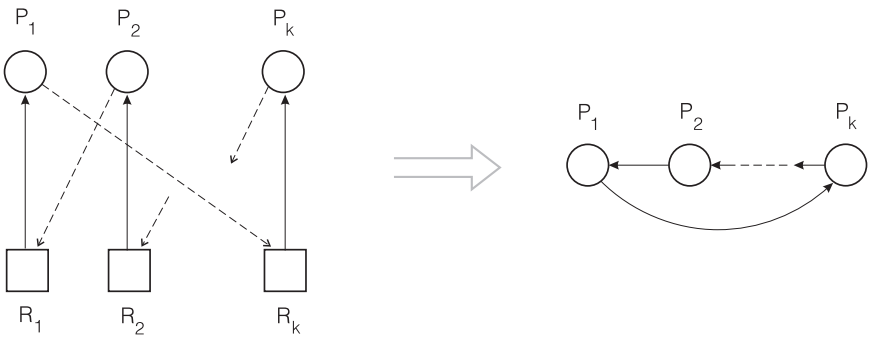
Nejdříve si zopakujeme několik základních poznatků spojených se vzájemným zablokováním procesů.

11.4.1 Zablokování při přístupu ke sdíleným prostředkům

Sdílí-li více procesů nějaký prostředek a využívání tohoto prostředku je výlučné v čase, dochází zákonitě ke kolizím požadavků. Kolizí přitom rozumíme skutečnost, že v době, kdy byl prostředek přidělen některému z procesů, může o přidělení prostředku požádat jiný proces. Kolizi požadavků řešíme běžně ve prospěch toho procesu P, kterému je prostředek právě přidělen. Proces Q, který o prostředek žádá v době jeho vyhrazení jinému procesu P, je pozdržen a musí čekat na uvolnění prostředku z iniciativy procesu P. Uvedenou relaci, kterou jsme si ilustrovali na procesech P a Q, nazýváme *závislostí*. Říkáme, že proces Q závisí na procesu P.

Ve skupině procesů $P_1, P_2, ..., P_k$ sdílejících určité prostředky může dojít k situaci, kdy například proces P_2 závisí na procesu P_1, proces P_3 závisí na P_2, a tak dále, až konečně P_k závisí na P_{k-1} a P_1 závisí na P_k. Proces P_1, aby mohl pokračovat ve výpočtu, musí počkat na uvolnění některého prostředku procesem P_k, ale vzhledem k celému řetězu závislostí musí počkat na uvolnění některého prostředku sebou samým. Cykl na skupině procesů definovaný závislostí způsobí zablokování skupiny procesů P_1 až P_k, situaci označujeme jako *deadlock* (zablokování, smrtelné objetí).

Situaci z předchozího odstavce si můžeme graficky znázornit orientovaným grafem. Uzly grafu budeme reprezentovat procesy, orientovanou hranou směřující od procesu P_i k procesu P_{i-1} budeme vyjadřovat závislost procesu P_i na procesu P_{i-1}. Příklad cyklu na grafu závislosti uvádí následující obrázek 11.7.



Obrázek 11.7: Graf závislosti

Tento graf závislostí zahrnuje pouze procesy. Chceme-li situaci popsat přesněji, musíme do grafu závislosti zahrnout i sdílené prostředky. Graf závislosti pak dostává podobu bipartitního orientovaného grafu. Jeden typ uzlů reprezentuje procesy, druhý typ uzlů reprezentuje sdílené prostředky. Hrany orientované od prostředků k procesům (plné) odpovídají přidělení prostředků, hrany orientované opačně (čárkované) odpovídají neuspokojeným požadavkům.

Závěrem si připomeňme formální podmínky, které musí být splněny, aby mohlo na skupině procesů k zablokování dojít:

1Prostředek nemůže být současně přidělen více procesům (jeho přidělování je výlučné v čase).

2Existuje situace, kdy proces, který čeká na přidělení určitého prostředku, má již jiný prostředek přidělen.

3Přidělený prostředek může proces uvolnit pouze z vlastní iniciativy.

4V grafu závislosti může vzniknout cykl.

Poznámka
Zablokování se pochopitelně nemusí týkat všech procesů distribuovaného systému, ale pouze určité skupiny procesů. Měli bychom tedy přesněji mluvit o zablokování na množině procesů.

K zablokování procesů může dojít i při výměně zpráv. Čekání jednoho procesu na zprávu odesílanou jiným procesem je obdobou dříve uvedené relace závislosti při sdílení prostředků. Vznikne-li mezi několika procesy cykl vzájemného čekání na zprávu (přičemž předpokládáme, že v daném okamžiku není žádná zpráva "předávána"), může i zde dojít k zablokování.

Řešení zablokování

Problém zablokování lze řešit dvěma způsoby. Buď preventivně zabráníme vzniku podmínek nutných pro zablokování nebo zablokování dodatečně detekujeme a celý systém vrátíme do vhodného stavu předcházejícího zablokování.

Metody odpovídající prvému přístupu označujeme jako *pesimistické* (apriorní). Zablokování považujeme za natolik vážný problém, že se mu musíme věnovat při každém přidělování prostředku, před každým vyhrazením prostředku testujeme možnost budoucího zablokování. Testování se opírá o dosud existující závislosti a může být časově náročné. Protože vyloučíme posloupnosti přidělování, které by mohly (ale nemusely) vést k zablokování, snižujeme úroveň sdílení. Pesimistické metody mají své místo tam, kde je obtížné vrátit systém do stavu předcházejícího zablokování.

Metody odpovídající druhému přístupu můžeme označit za *optimistické* (aposteriorní). Zablokování považujeme za výjimečnou situaci, kterou řešíme až v okamžiku, kdy k ní skutečně dojde. Odblokování systému procesů předpokládá, že některému z procesů lze dříve přidělený prostředek odebrat (porušujeme zde podmínku 3 předchozího odstavce), a vrátit systém do stavu před přidělením tohoto prostředku. Optimistické metody dovolují plně využít možnosti sdílení, předpokládají však možnost vrácení některého z procesů do stavu předcházejícího zablokování.

11.4.2 Apriorní metody

Nejsnazším způsobem, jak zabránit zablokování v distribuovaném systému, je pověřit správou prostředků jeden proces, předávat mu žádosti o přidělení prostředků a nechat ho rozhodnout na základě znalosti globálního stavu. Takové řešení je výrazně nesymetrické a nás budou zajímat symetrické modifikace algoritmu, v nichž rozhodují všechny procesy bez zvláštní centrální autority.

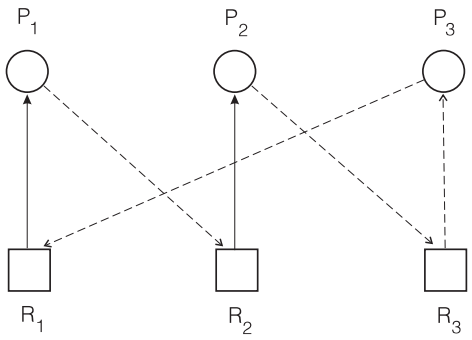
Lometovy algoritmy

Jako příklad algoritmů bránících zablokování si uvedeme dva Lometovy algoritmy využívané v distribuovaných databázových systémech. Pro prvý z nich uvádíme centralizovanou i distribuovanou verzi.

Lometovy algoritmy vyžadují, aby bylo možné rozložit procesy na úseky, ve kterých bude využíván jeden nebo více sdílených prostředků, a na úseky, ve kterých se přístup ke sdíleným prostředkům nevyžaduje. Před vstupem do úseku, ve kterém bude využíván jeden nebo více sdílených prostředků, musí proces předběžně informovat o svých požadavcích. Autorita, která o přidělení prostředků rozhoduje, na základě aktuálně přidělených prostředků a na základě předběžných oznámení rozhodne, zda požadované prostředky může přidělit nebo zda proces musí počkat.

Rozhodnutí se opírá o analýzu *modifikovaného grafu závislosti*, ve kterém registrujeme přidělení prostředků a *předběžné žádosti* (ty se v průběhu výpočtu mění na žádosti aktuální, registrované v základním grafu závislosti). Metodu si budeme ilustrovat na následujícím příkladě:

Tři procesy P_1, P_2 a P_3 sdílejí tři prostředky R_1, R_2 a R_3 takovým způsobem, že proces P_i pro provedení svého úseku T_i vyžaduje současně prostředky R_i a R_{i+1} (indexujeme modulárně). Představme si situaci, kdy všechny procesy již předběžně oznámily své požadavky, procesu P_1 již byl vyhrazen prostředek R_1 a procesu P_2 již byl vyhrazen prostředek R_2. Požádá-li za tohoto stavu proces P_3 o vyhrazení prostředku R_3, nemůže být jeho žádost akceptována, vyhrazení prostředku R_3 by totiž vedlo na uzavření cyklu na modifikovaném grafu závislosti. Situaci v okamžiku, kdy testujeme akceptovatelnost žádosti P_3 uvádí obr. 11.8.



Obrázek 11.8: Lometova analýza na modifikovaném grafu závislosti

Algoritmus je ve své centralizované formě používán pro výlučné vyhrazení datových oblastí v databázích. Elementárním úsekem, pro který je výlučný přístup k datovým oblastem zajišťován, je *transakce*.

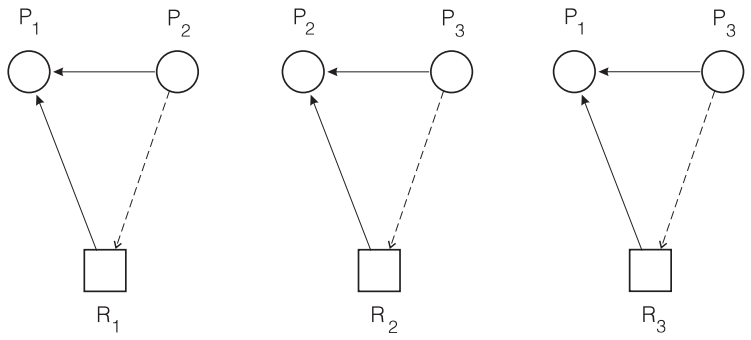
Poznámka
Transakcí budeme rozumět úsek procesu, na který bude omezeno vyhrazení prostředku, přičemž budeme vyžadovat možnost vrátit se během výpočtu transakce na její začátek (to je důležité pro aposteriorní metody).

Centralizovanou verzi algoritmu lze velice snadno převést na verzi distribuovanou, opřemeli se o algoritmus výlučného přístupu. Kritickou sekcí je přidělovací algoritmus, který je včetně potřebných datových struktur replikován v každém z procesů. (Alternativně lze spojit algoritmus s obsluhou prostředků — se servery). Algoritmus výlučného přístupu (například základní Lamportův) zajistí, že vstupy do kritické sekce jsou uspořádány. Operace nad grafem závislosti jsou realizovány ve všech kopiích v takto definovaném pořadí. Rozhodnutí, které dává lokální autorita je zcela shodné s rozhodnutím, které by dala jediná centrální autorita. Výpočet

všech lokálních autorit je totiž vzhledem k synchronizaci zavedené algoritmem výlučného přístupu totožný.

Nevýhodou plně replikované verze Lometova algoritmu je skutečnost, že každý proces (resp. server) musí udržovat úplnou informaci o globálním přidělení prostředků, a hlavně velké množství vyměňovaných zpráv. Pro server by bylo výhodnější udržovat pouze lokální informaci, spojenou s přidělovaným prostředkem, a na základě této informace zabránit vzniku cyklu v globálním grafu závislosti (jehož model ovšem server vytvářet nebude). Tato myšlenka je základem druhého Lometova algoritmu.

Druhý Lometův algoritmus se (stejně jako prvý) opírá o skutečnost, že předběžné informace o požadavcích jednotlivých procesů jsou plně uspořádané (díky podpoře jejich předávání algoritmem výlučného přístupu). Pro každý prostředek je vytvářen lokální graf, který registruje předběžné požadavky procesů a aktuálně přidělené prostředky. Hrany, které přísluší předběžným rezervacím a realizovaným přidělením odpovídají globálnímu grafu závislosti, žádající procesy navíc propojíme hranami orientovanými v souladu s časovými razítky předběžných rezervací. Příklad konstrukce jednotlivých lokálních grafů uvádí obr. 11.9, který odpovídá dřívějšímu příkladu.



Obrázek 11.9: Lokální grafy závislosti

Lze ukázat, že pokud zabráníme vzniku cyklu v každém z lokálních grafů závislosti, je vyloučen i vznik cyklu v globálním grafu závislosti. Bohužel podmínka na lokálním grafu je silnější, sdílení prostředků je méně efektivní.

Funkci druhého Lometova algoritmu si budeme prezentovat na našem předchozím příkladu. Předpokládejme, že předběžně o svých požadavcích na rezervaci prostředků informují procesy v pořadí P_1, P_2 a P_3. Prostředek R_1 může být vyhrazen procesu P_1, protože předběžná rezervace procesu P_1 předchází předběžné rezervaci procesu P_3, a vyhrazení prostředku R_1 procesu P_1 nemůže vyvolat vznik cyklu v lokálním grafu závislosti G_1. Prostředek R_2 však procesu P_2 vyhradit nelze, protože předběžné rezervaci procesu P_2 předchází předběžná rezervace procesu P_1, následující požadavek procesu P_1 na přidělení prostředku by vedl na vznik cyklu v lokálním grafu G_2. Podobně je tomu i v případě prostředku P_3.

Poznámka
Lometovy algoritmy brání zablokování definováním uspořádání na předběžných požadavcích. Vzhledem k symetrii problému lze zabránit zablokování také definováním úplného uspořádání přidělovaných prostředků a žádat o jejich přidělení jen v odpovídajícím pořadí. Tento postup (Havenderova metoda) však vyžaduje dodržení pevné disciplíny v procesech.

11.4.3 Aposteriorní metody

Alternativou k apriornímu vyloučení posloupnosti rezervací, která vede k zablokování, jsou aposteriorní metody. Tyto metody nezajišťují prevenci, pouze řeší vzniklé kolize. Protože nemusíme znát dopředu předběžné požadavky transakcí na sdílené prostředky, jsou algoritmy vhodné pro situace, kdy přidělování prostředků závisí na průběhu transakce (přidělování je dynamické).

Aposteriorní metody neomezují přidělování tak silně jako metody apriorní a jsou proto většinou efektivnější. Pro svou funkci však vyžadují, aby procesy bylo možné vrátit do stavu, který předchází detekovanému zablokování. Obvykle se vyžaduje omezit vyhrazení prostředků na transakce, na jejichž začátek se lze vracet. Algoritmy totiž musí při detekovaném zablokování vybrat jeden z procesů jako oběť a jeho výpočet vrátit.

Primitivní centralizovaný algoritmus testující uzavření cyklu na základním grafu závislosti lze podobně jako v případě prvého Lometova algoritmu replikovat. My si zde ale uvedeme pouze dva distribuované algoritmy, které se opírají o minimální lokální informaci. Tou jsou časové známky přidělené jednotlivým transakcím při jejich (prvém) zahájení.

Předpokládejme, že ve dvou transakcích T_1 a T_2 využíváme sdílený prostředek R_1, a že tento prostředek je v daném okamžiku přidělen transakci T_1. Předpokládejme, že transakce jsou označeny časovými známkami $e(T_1)$ a $e(T_2)$. Žádost transakce T_2, která dojde v době vyhrazení prostředku R_1 transakci T_1 můžeme vyřešit tímto postupem:

```
if e(T2)<e(T1) then halt T2
               else kill T2
```

Pokud je transakce T_2 "starší" než transakce T_1, pak její výpočet pozastavíme (operace *halt* T_2) do uvolnění prostředku R_1 "mladší" transakcí T_1. Je-li naopak transakce T_2 "mladší" než transakce T_1, pak musí T_2 uvolnit všechny již přidělené prostředky (operace *kill* T_2). Proces, který zahájil násilně ukončenou transakci T_2 se pokusí transakci zopakovat s původní časovou známkou. Vzhledem k zachovávání časové známky opakovanou transakcí a vzhledem k uspořádání na množině transakcí vzhledem k časovým známkám je vyloučena starvace některého procesu (jeho transakce se po určité době stane "nejstarší" transakcí a bude dokončena).

Alternativou předchozího postupu je uvolnění všech prostředků vyhrazených "mladší" transakci T_1 při kolizi způsobené příchodem požadavku "starší" transakce T_2. Postup lze vyjádřit takto:

```
if e(T2)<e(T1) then kill T1
               else halt T2
```

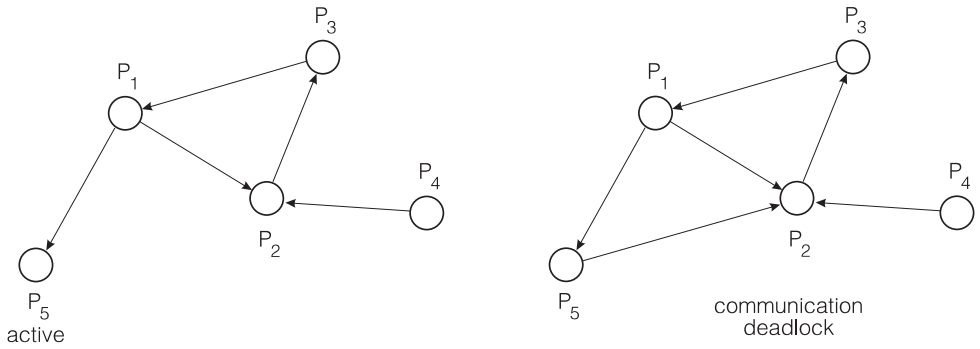
"Starší" transakce T_2 nikdy nečeká na ukončení "mladší" transakce T_1, která musí vrátit dosud vyhrazené prostředky a pokusit se o výpočet později s původní časovou známkou. Stejně jako u předchozího postupu nemůže dojít k nekonečné starvaci některého procesu.

Oba uvedené algoritmy, podobně jako druhý Lometův algoritmus, testují podmínku nutnou pro zablokování. Tato podmínka je silnější než podmínka zablokování (postačující podmínka), transakce jsou zbytečně ukončovány i v případech, kdy by k zablokování nedošlo. Z tohoto pohledu se nejedná o čistě aposteriorní algoritmy, zasahují často dříve než k zablokování skutečně dojde.

11.5 Zablokování při komunikaci

Vedle kolizí při sdílení prostředků je dalším zdrojem zablokování v distribuovaném systému komunikace. Podobně jako u sdílení prostředků lze i pro předávání zpráv mezi procesy zavést relaci závislosti a to následovně: Proces Q bude záviset na procesu P, pokud bude proces Q čekat na zprávu odesílanou procesem P. Podobně jako u přidělování prostředků budeme znázorňovat situaci v distribuovaném systému grafem závislosti: s procesy jako uzly a hranami jako kanály, na nichž očekáváme zprávu.

Srovnáme-li si zablokování při komunikaci se zablokováním při přidělování prostředků, všimneme si patrně podstatného rozdílu. U přidělování prostředků potřebuje proces, aby mohl pokračovat ve výpočtu, přidat jeden konkrétní sdílený prostředek k těm, které již má vyhrazené. Naproti tomu u komunikace proces často čeká na příchod jedné ze zpráv od různých procesů, kterákoliv z nich mu dovolí pokračovat ve výpočtu. Prostý cykl na grafu závislosti ještě neznamená zablokování, podmínka, kterou je nutné testovat je složitější. (Podobnou situaci bychom museli řešit i u přidělování jednoho z více použitelných prostředků.) Příklady grafu závislosti pro komunikaci, které rozdíl ilustrují, uvádí obr. 11.10.



Obrázek 11.10: Graf závislosti

V prvém případě nejde o zablokování, procesy P_1, P_2 a P_3 jsou sice spojené cyklem vzájemných čekání, ale zpráva odeslaná procesem P_5 uvolní proces P_1 a cykl odstraní. Ve druhém příkladě se o zablokování jedná, proces P_5 je zde blokován čekáním na zprávu od procesu P_2 a nemůže zprávu procesu P_1 odeslat.

Proces P distribuovaného systému označíme jako *pasivní*, jestliže čeká na zprávu od jiného (jiných) z procesů distribuovaného systému. V opačném případě ho označujeme jako *aktivní*. Množinu procesů, od kterých proces P očekává v daném stavu zprávu označujeme jako *množinu závislosti* (dependancy set). Množina závislosti je pro aktivní procesy prázdná. S využitím těchto termínů lze pak *zablokování na množině* S procesů distribuovaného systému definovat jako splnění následujících podmínek:

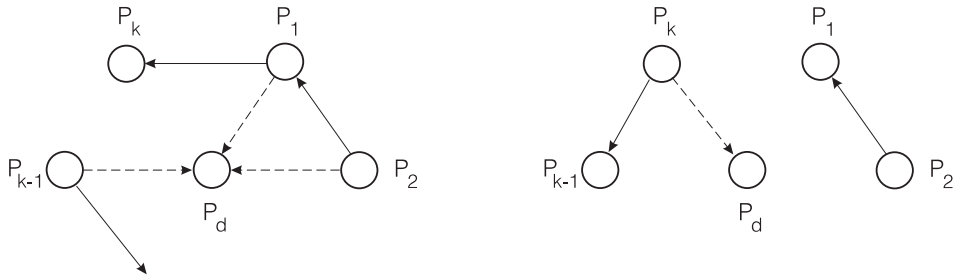
1každý z procesů $P \in S$ je pasivní,

2pro množinu závislosti $DS(P)$ každého procesu $P \in S$ platí, že je podmnožinou S, $DS(P) \subset S$.

Podgraf grafu závislosti, který odpovídá množině S, bývá označován jako *zauzlení* (knot).

Zdánlivé zablokování

Určitým problémem, který je spojen s detekcí zablokování při komunikaci je nemožnost zjištění okamžitého globálního stavu distribuovaného systému. Tento problém si budeme ilustrovat na jednoduchém případě. Mějme skupinu procesů P_1, P_2,...,P_k a proces P_d, který sbírá informace o čekání procesu na zprávu. Předpokládejme, že graf závislosti procesů P_1, P_2,...,P_k odpovídá následujícímu obrázku 11.11 a procesy P_1, P_2,...,P_{k-1} indikují procesu P_d čekání. Proces P_k odešle zprávu procesu P_1 a bude očekávat zprávu od P_{k-1}, o svém čekání bude informovat proces P_d. Ten na základě informací, které obdržel, vyhodnotí uvedenou situaci jako zablokování.



Obrázek 11.11: Zdánlivé zablokování

Algoritmy vyhodnocující zablokování musí být schopné zdánlivé zablokování rozpoznat.

11.5.1 Chandy-Misra-Hass

Algoritmus dovoluje procesu zjistit, zda je prvkem množiny procesů, na níž došlo k zablokování. Je příkladem difusního algoritmu, procesy odesílají dotazy svým sousedům a na základě odpovědí od nich vyhodnocují situaci. Opírá se přitom o původní komunikační strukturu distribuovaného systému, služební zprávy si procesy vyměňují s týmiž sousedy jako zprávy aplikace.

Testu zablokování může zahájit každý pasivní proces P_k. Test začíná rozesláním žádosti $REQUEST$ všem procesům, od kterých proces P_k očekává zprávu aplikace, tedy všem procesům z množiny závislosti DS_k. Žádost má formát

$$REQUEST(k, m, j, i) \ ,$$

kde

kje číslo procesu, který test zahájil,

mje pořadové číslo tohoto jeho testu,

jje číslo odesílatele žádosti, a

ije číslo adresáta žádosti.

Počet rozeslaných žádostí si proces poznamená v proměnné $Number[k]$.

Proces P_i, který přijme žádost $REQUEST(k, m, j, i)$ od jiného procesu P_j a je pasivní se zachová následovně: Jestliže se jedná o prvou kopii m-té žádosti k-tého procesu (tedy, nemá-li ji náš proces dosud registrovanou v proměnné $Last[k]$), rozešle proces P_i žádost $REQUEST(k, m, i, l)$ všem procesům P_l, které patří do množiny DS_i. V proměnné $Number[k]$ si poznamená počet procesů v množině DS_i a očekává od nich odpovědi $ANSWER(k, m, l, i)$. Jakmile náš proces P_i shromáždí odpovědi od všech procesů z DS_i, odešle odpověď $ANSWER(k, m, i, j)$ procesu P_j. Obdrží-li proces P_i další kopii zprávy $REQUEST(k, m, j, i)$, odpoví proces P_i zprávou

$ANSWER(k, m, i, j)$ okamžitě. Je-li proces P_i aktivní (schopný pokračovat ve výpočtu aplikace), pak zprávu $REQUEST(k, m, j, i)$ ignoruje.

```
when decision START_DETECTION and State=PASSIVE do { start algoritmu }
   begin
      Last[i]:=Last[i]+1;
      Wait[i]:=T;
      for j in DSet do
         send QUESTION(i,Last[i],i,j) to j
      Number[i]:=card(DSet)
   end

when received ANY_OTHER_MESSAGE do                  { zpráva aplikace }
   begin
      State:=ACTIVE;
      for i:=1 to N do
         Wait[i]:=F
   end

when received QUESTION(k,m,j,i) and State=PASSIVE do   { příjem dotazu }
   if m>Last[k] then
      begin
         Last[k]:=m;
         Parent[k]:=j;
         Wait[k]:=T;
         for r in DSet do
            send QUESTION(k,m,i,r) to r;
         Number[k]:=card(DSet)
      end
   else
      if Wait[k] and m=Last[k] then
         send ANSWER(k,m,i,j) to j

when received ANSWER(k,m,r,i) and State=PASSIVE do   { příjem odpovědi }
   if m=Last[k] and Wait[k] then
      begin
         Number[k]:=Number[k]-1;
         if Number[k]=0 then
            if k=i
            then

               { Pi je zablokován }

            else
               send ANSWER(k,m,i,Parent[k]) to Parent[i]
               Wait[k]:=F
      end

begin                                               { inicializace }
   for i:=1 to n do
      begin
         Last[i]:=0; Wait[i]:=F
      end
```

Proces P_k, který test inicializoval tak do ohraničené doby dostane určitý počet odpovědí $ANSWER(k, m, l, k)$ od procesů P_l množiny DS_k. Je-li počet přijatých odpovědí shodný s počtem procesů v množině DS_k, nebyl v množině procesů, na kterých proces P_k i zprostředkovaně závisí, nalezen žádný aktivní proces. Test v takovém případě indikuje zablokování na množině, do které proces P_k patří.

Pro práci algoritmu si každý proces udržuje informace o ukončených a rozpracovaných

testech zablokování v indexovaných proměnných $Last$, $Wait$, $Parent$ a $Number$. Proměnná $Last$ udržuje ve svém prvku $Last[k]$ pro každý proces P_k pořadové číslo posledního testu zablokování, který byl inicializován tímto procesem P_k. Proměnná $Wait$ udržuje ve svém prvku $Wait[k]$ pro každý proces P_k informaci o aktivitě tohoto procesu. Proměnná $Parent$ udržuje ve svém prvku $Parent[k]$ informaci o číselné identifikaci procesu P_j, který zaslal našemu procesu P_i žádost procesu P_k o test zablokování. Konečně proměnná Number udržuje ve svém prvku $Number[k]$ počet dosud nezodpovězených dotazů, které rozeslal náš proces P_i procesům, na kterých závisí, jako reakci na žádost procesu P_k o test zablokování.

Dovolíme-li v systému existenci *ukončených* procesů, je třeba doplnit předchozí algoritmus o reakci ukončených procesů na žádost $REQUEST$. Ukončený proces nemůže vyslat procesu, který na něm závisí, žádnou zprávu (týkající se aplikace), musí proto na žádost $REQUEST$ bezprostředně odpovědět zprávou $ANSWER$.

Je potřebné si uvědomit, že algoritmus netestuje (ne)existenci zablokování v systému N procesů, ale pouze příslušnost konkrétního procesu P_k k množině vzájemně zablokovaných procesů. Otestování (ne)existence zablokování v systému vyžaduje inicializovat test ve všech pasivních procesech P_k systému.

Za poznámku stojí i skutečnost, že testu zablokování se účastní pouze pasivní procesy, test zablokování tedy není vyhodnocován na úkor aktivních procesů (neuvažujeme-li sdílení procesorů a přídavnou zátěž komunikačních kanálů).

Difuzní algoritmus, který jsme uvedli, byl schopen rozhodnout, zda daný proces je prvkem skupiny procesů zablokovaných při vzájemné komunikaci. S minimální modifikací lze algoritmus použít i pro detekci skupiny procesů zablokovaných při sdílení prostředků.

Poznámka
Algoritmus, doplněný o obsluhu ukončených procesů, lze poměrně snadno rozšířit na test ukončení práce skupiny procesů (které se budeme samostatně věnovat v další kapitole. Ke skupině procesů E přidáme proces P_s a jako množinu závislosti DS_s definujeme množinu všech procesů skupiny E. Pozitivní výsledek testu inicializovaného procesem P_s indikuje ukončení aktivity všech procesů skupiny E a tedy (při vhodné definici ukončení) ukončení práce skupiny procesů.

11.6 Ukončení výpočtu

Algoritmy detekující ukončení výpočtu jsou algoritmům detekujícím zablokování v řadě rysů podobné. U zablokování nás zajímá situace, kdy se *některé procesy* systému udržují vzájemně v pasivním stavu; na zbývajících procesech systému výpočet může pokračovat. U algoritmů detekujících ukončení výpočtu na systému nás zajímá situace, kdy jsou v pasivním stavu *všechny procesy* systému současně.

Podobně jako u předchozích úloh lze nalézt algoritmy pracující na rozdílných principech. Jako příklad si uvedeme test ukončení difuzního algoritmu, a test ukončení libovolného distribuovaného algoritmu ve dvou formách.

11.6.1 Dijkstra-Scholten

Algoritmus detekující ukončení výpočtu, který si uvedeme jako prvý, je vlastně pouhým doplňkovým testem k libovolnému difuznímu algoritmu. Připomeňme si, že difuzní algoritmus je startován jedním z procesů, který je kořenem stromu definovaným relací následnosti (o níž se difuzní program opírá). Tento proces rozesílá požadavky svým následníkům a očekává jejich odpovědi. Následníci reagují rozesláním požadavků svým následníkům a očekávají odpovědi; po soustředění všech odpovědí odpovídají procesu, který jejich aktivitu vyvolal. Konečně procesy v listech stromu definovaného relací následnosti odpovídají přímo na požadavky.

Následující test doplňuje difuzní algoritmus o akce spojené s odesíláním a příjmem požadavků ($MESSAGE$) a odpovědí ($SIGNAL$). Pro každý proces udržujeme proměnnou $DefIn$ (deficit na vstupu), která udává rozdíl v počtu přijatých požadavků a odeslaných potvrzení a proměnnou $DefOut$ (deficit na výstupu), která udává rozdíl v počtu odeslaných žádostí a přijatých odpovědí. Proměnná $Parent$ identifikuje proces, od kterého náš proces přijal prvou žádost, proměnná $Others$ ukládá záznamy o dalších žádajících procesech (s možností zaznamenat libovolný proces násobně).

```
receiving MESSAGE from j do                          { příjem žádosti aplikace }
    begin
        if DefIn=0
        then Parent:=j
        else Others:=Others+j;
        DefIn:=DefIn+1
    end

receiving SIGNAL from j do                           { příjem odpovědi aplikace }
    DefOut:=DefOut-1

sending MESSAGE to j { possible if DefIn>0 } do
    DefOut:=DefOut+1;                                { odeslání žádosti aplikace }

sending SIGNAL to (Oth=any of Others) { possible if (DefIn>1 } do
    begin                                            { odeslání odpovědi aplikace }
        Others:=Others-Oth;
        DefIn:=DefIn-1
    end;

sending SIGNAL to Parent { possible if (DefIn=1 and DefOut=0) } do
    DefIn:=DefIn-1                                   { odeslání odpovědi aplikace }

begin                                                { inicializace }
    Defin:=0; Defout:=0; Others:=
end
```

Proces, který startuje algoritmus vyznačíme počáteční hodnotou $DefIn = 1$ a dovolíme mu tak rozeslat žádosti následovníkům. Důsledkem je nenulová hodnota proměnné $DefOut$ tohoto procesu. Výpočet algoritmu je ukončen po přijetí všech odpovědí, tedy v okamžiku, kdy se hodnota $DefOut$ vrátí na nulu.

11.6.2 Dijkstra-Feijen-Van Gasteren

Tento algoritmus dovoluje testovat ukončení libovolného distribuovaného výpočtu. Na množině procesů definujeme uspořádání, například očíslováním procesů $P_0, P_1, ..., P_k$. Proces P_0 má v algoritmu významné postavení: startuje algoritmus a rozhoduje o ukončení výpočtu na distribuovaném systému.

Procesy $P_0, P_1, ..., P_k$ si (kromě komunikace související s vlastním výpočtem distribuovaného programu) předávají zvláštní zprávu — $TOKEN$, která slouží pouze detekci ukončení. Tato zpráva nese jednobitovou informaci — údaj o barvě, může být bílá nebo černá ($WHITE$, $BLACK$). Podobně každý z procesů si udržuje proměnnou $Color$, která může nabývat hodnot odpovídajících bílé a černé barvě ($WHITE$, $BLACK$).

```
receiving MESSAGE do                                   { p©ˇjem zpr vy aplikace }
   State:=ACTIVE

waiting MESSAGE or State=TERMINATED do          { ‡ek nˇ na zpr vu aplikace }
   State:=PASSIVE

sending MESSAGE to j begin                  { odesl nˇ zpr vy procesu s indexem j>i }
   if i<j then Color:=BLACK

when received TOKEN(ct) from i+1 do                    { p©ˇjem zpr vy TOKEN }
   begin
      TPresent:=T;
      TColor:=ct;
      if i=0 then
         if Color=WHITE and TColor=WHITE
         then TERMINATION_DETECTED
         else TColor:=WHITE
   end

when TPresent and State=PASSIVE do  { p©ed nˇ zpr vy TOKEN n sledn ˇkovi }
   begin
      if Color=BLACK then TColor:=BLACK;
      TPresent:=F;
      send TOKEN(TColor) to i-1;
      Color:=WHITE
   end

begin                                                          { inicializace }
   TPresent:=F; Color:=WHITE
end
```

Test ukončení startuje proces P_0 odesláním zprávy $TOKEN(WHITE)$ procesu P_k, zpráva $TOKEN$ je postupně předávána procesy $P_k, P_{k-1}, ..., P_1$, až se vrátí k procesu P_0. Změna barvy přenášené touto zprávou na černou ($BLACK$) indikuje aktivitu některého z procesů a proces P_0 odstartuje test ukončení znovu.

Důvodem pro změnu barvy zprávy $TOKEN$ z hodnoty $WHITE$ na hodnotu $BLACK$ při "průchodu" libovolným procesem je aktivita tohoto procesu nebo skutečnost, že v době od posledního "průchodu" tento proces odeslal zprávu aplikace nějakému procesu s vyšším indexem (který mohl být díky čekání na tuto zprávu nalezen v pasivním stavu). Algoritmus tak rozliší

zdánlivé ukončení výpočtu od ukončení skutečného.

11.6.3 Misra

Předchozí algoritmus má dvě omezení. Vyžaduje speciální funkci procesu P_0 a předpokládá, že zpoždění zpráv při výměně je zanedbatelné. Modifikace algoritmu popsaná Misrou obě omezení odstraňuje. Kruh propojující procesy nahrazuje cyklem C, který obsahuje všechny hrany grafu komunikací. Pokud zpráva $TOKEN$ projde všemi procesy cyklu C a nalezne je v pasivním stavu ($State = PASSIVE$) a nedostane informaci o aktivitě procesu od předchozího "průchodu" ($Color = BLACK$), dosáhne hodnota nb přenášená zprávou $TOKEN$ počtu procesů v cyklu a indikuje ukončení výpočtu.

```
when received MESSAGE do                          { p©ˇjem zpr vy aplikace }
   begin
      State:=ACTIVE;
      Color:=BLACK
   end

when waiting MESSAGE do                           { ‡ek nˇ na zpr vu aplikace }
   State:=PASSIVE

when received TOKEN(j) do                         { p©ˇjem zpr vy TOKEN }
   begin
      nb:=j;
      TPresent:=T;
      if nb=Size(C) and Color=WHITE then
         TERMINATION DETECTED
   end

when TPresent and State=PASSIVE                   { odesl nˇ zpr vy TOKEN }
   begin
      if Color=BLACK then nb:=0
                     else nb:=nb+1;
      send TOKEN(nb) to Succesor(C,i);
      Color:=WHITE;
      TPresent:=F
   end

begin                                             { inicializace }
   Color:=BLACK; TPresent:=F; nb:=0
end
```

Délka cyklu je určena hodnotou $Size(C)$, následnost procesů v cyklu funkcí $Succesor(C, i)$.

Za předpokladu, že kanály zachovávají pořadí zpráv, algoritmus odstraňuje nebezpečí, že v systému po oběhu zprávy $TOKEN$ zůstane nedoručená zpráva aplikace.

11.7 Ochrana proti výpadkům

Většina algoritmů, kterými jsme se dosud zabývali, předpokládala bezchybnou funkci všech spolupracujících procesů. Reálné systémy musí počítat s výpadky procesů a komunikačních kanálů, důležité jsou proto postupy dovolující zajistit výpočet i v situaci, kdy k výpadkům dochází.

-Nejsnáze zvládnutelným výpadkem je nefunkčnost procesu od okamžiku, kdy distribuovaný algoritmus odstartoval. Proces je označován jako *initialy dead*.

-Běžnější a složitější na obsluhu je ukončení funkce procesu po určité části výpočtu, situace je označována jako *crash*. Mechanismy ochrany jsou schopné zvládnout obvykle až t výpadků typu *crash* ve skupině N procesů, kde $2t < N$.

-Konečně nejsložitějším případem nekorektního chování je nesprávná zpráva, kterou proces v průběhu výpočtu odešle, nebo zpráva chybějící. Chování procesu, které vede k takovému důsledku, označujeme jako *byzantské*. Patří sem vedle záměrně chybově se chovajícího procesu i ztráty a poškození zpráv při přenosu. Mechanismy ochrany jsou schopné zvládnout obvykle až b procesů s byzantským chováním ve skupině N procesů, kde $3b < N$.

Algoritmy dovolující tolerovat omezený počet výpadků lze rozdělit do dvou základních tříd - na odolné algoritmy a algoritmy samostabilizující:

-*Odolné* (robust) algoritmy korektně pracují až do daného počtu nekoretně fungujících procesů, typickým příkladem jsou algoritmy opírající se o výběr podmnožiny procesů – *quora* před spuštěním výpočtu.

-*Samostabilizující* (self-stabilizing) algoritmy reagují na výpadek procesu přechodným stavem, po jehož odeznění jsou schopné poskytovat korektní službu. Typickými příklady samostabilizujících algoritmů jsou algoritmy pro vytváření směrovacích tabulek v komunikačních sítích.

11.7.1 Quorum algoritmy

U řady algoritmů (např. algoritmů výlučného přístupu) postačí, pokud se na výsledku domluví určitá podmnožina procesů, výpadek ostatních procesů výsledek neovlivní. Takovou podmnožinu nazýváme *quorum* a vyžadujeme, aby nebylo možné vytvořit dvě takové disjunktní podmnožiny současně.

-Nejjednodušším typem quora je podmnožina s nejméně $(n + 1)/2 prvky$, takové quorum označujeme jako *majoritní*. Existují však i quora, počet jejichž prvků je nižší.

-Maekawova quora mají počet prvků roven $\sqrt{n}$ pro n procesů, příbuzné *maticové quorum* tvořené procesy v i-tém sloupci a j-tém řádku matice procesů má $2\sqrt(n) - 1$ prvků.

-Další zajímavou třídou quor jsou *stromová quora*. Quorem je cesta mezi kořenem a listem binárního stromu, jehož uzly jsou procesy. Pokud je kterýkoliv z procesů na této cestě nefunkcní, mohou ho zastoupit jeho *"synové"* a cesty od nich k listům.

O quorum mechanismy, které dovolují vybrat množinu procesů schopných provést krok výpočtu, se opírají mechanismy, které dovolí zajistit konzistenci *replikovaných* dat na všech procesech výpočtu.

Literatura

[1]Janeček J., Bílý M. *Lokální sítě.* Ediční středisko ČVUT Praha 1997.

[2]Janeček J., Kubr. J., Červený M. *Distribuované systémy - cvičení.* Vydavatelství ČVUT Praha 2000.

[3]Tanenbaum A.S. *Computer Networks.* Prentice Hall 1996.

[4]Stallings W. *Data and Computer Communication.* Prentice Hall 1997.

[5]Pužmanová R. *Moderní komunikační sítě od A do Z.* Computer Press 1998.

[6]Kállay F., Peniak P. *Počítačové sítě a jejich aplikace.* Grada 1999.

[7]Stevens W.R.: *UNIX Network Programming. Vol.1: Networking APIs: Sockets and XTI.* Prentice-Hall 1998.

[8]Stones R., Matthew N. Linux - začínáme programovat. Computer Press 2000.

[9]Coulouris G., Dollimore J., Kindberg T. *Distributed Systems.* Addison-Wesley 1994.

[10]Raynal M. *Distributed Algorithms and Protocols.* John Willey 1985.

[11]Lynch N.A. *Distributed Algorithms.* Morgan Kaufmann 1996.

[12]Tel G. *Introduction to Distributed Algorithms.* Cambridge University Press 1994.

Odkazy

[13]Kleinrock L. *Queueing Systems. Vol.2: Computer Applications.* John Willey 1976.

[14]Hoare C.A.R. *Communicating Sequential Processes*, CACM 21(8), Aug. 1978, 666–677.

[15]Hansen P.B. *Distributed Processes: A Concurrent Programming Concept*, CACM 21(11), Nov. 1978, 934–941.

[16]Lamport L. *Time, Clocks and the Ordering of Events in a Distributed System*, CACM 21(7), July 1978, 558–565.

[17]Ricart G.,Agarwala A.K. *An Optimal Algorithm for Mutual Exclusion in Computer Networks*, CACM 24(1), Jan. 1981, 9–17.

[18]Carvalho O.,Roucairol G. *On Mutual Exclusion in Computer Networks*, CACM 26(2), Feb. 1983, 146–147.

[19]Ricart G.,Agarwala A.K. *Author's response to "On Mutual Exclusion in Computer Networks" by Carvalho and Roucairol*, CACM 24(1), Feb. 1983, 147–148.

[20]Misra J. *Detecting Termination in Distributed Computation Using Markers*, Proc. 2nd ACM Conf.on Principles of Distributed Computing. Montreal 1983, 290–394.

[21]Chang E.G.,Roberts R *An Improved algorithm for Decentralized Extrema-Finding in Circular Configuration of Processors*, CACM 22(5), May 1979, 281–383.

[22]Hirschberg D.S.,Sinclair J.B *Decentralized extrema finding in Circular Configurations of Processors*, CACM 23(11), Nov. 1980, 627–628.

[23]Plášil F. *Operační systémy.* *Ediční středisko ČVUT Praha 1989.

[24]Chandy K.M.,Misra J.,Hass L.M. *Distributed Deadlock Detection*, ACM TOCS, 1(2) May 1983, 144–156.

[25]Dijkstra E.W.,Scholten C.S. *Termination Detection for Distributed Computation*, Inf.Processing Letters 11(1), Aug. 1980, 1–4.

[26]Dijkstra E.W.,Feijen W.H.J.,Van Gasteren A.J.M. *Derivation of a Termination Detection Algorithm for Distributed Communication*, Inf.Proc.Letters 16, June 1983, 217–319.

Index