

Kapitola 1

Architektura sítí, adresování, konfigurace TCP/IP

V této kapitole se podíváme, jak vypadá architektura počítačových sítí. Správné pochopení vrstevné architektury počítačových sítí je nezbytným základem, jak porozumět činnosti síťových služeb. Každá síťová služba (např. elektronická pošta) využívá služeb dalších síťových vrstev (zajištění správného doručení dat, kódování, oprava chyb) pro svou činnost. Zejména při správě sítí či odhalování chyb je nezbytné zjistit, na které úrovni počítačové komunikace došlo k chybě. Jde o chybu aplikace (špatná konfigurace, chybné heslo), chybu přenosové vrstvy (spojení je pomalé, padá) či chybu média?

Druhou podstatnou oblastí, kterou se zabýváme v této kapitole, je adresování. Čtenář se seznámí se základními druhy adres (fyzická adresa, IP adresa, doménová adresa), překlady adres, jejich klasifikací a přidělováním.

Závěr kapitoly tvoří zmínka o implementaci síťových vrstev v počítači ukázané na příkladech systémů Linux, FreeBSD a Windows XP.

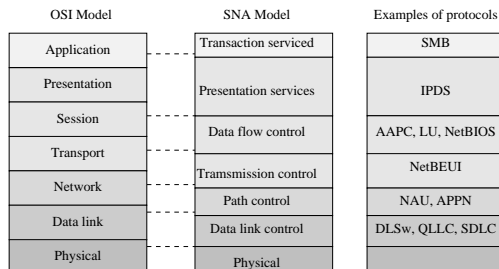
1.1 Vrstvové modely počítačových sítí

Architektura počítačových sítí byla od svého počátku navržena tak, aby odlišovala tři základní části komunikačního systému:

1. technologie pro přenos signálu (metalická kabeláž, rádiové vlny, optická vlákna),
2. vrstvu pro zajištění spolehlivého přenosu (TCP/IP, IPX/SPX, AppleTalk, SNA, DECnet) a
3. aplikační vrstvu, která poskytuje služby uživateli.

Během historického vývoje počítačových sítí vzniklo několik modelů architektury počítačové sítě postavených na firemních protokolech. V současné době se používají v oblasti Internetu dva modely — model ISO/OSI, který slouží jako referenční model (RM) a model TCP/IP založený na protokolech TCP (Transmission Control Protocol) a IP (Internet Protocol). Model TCP/IP je standardem Internetu a dnes je implementován jako přenosová vrstva většiny počítačových sítí.

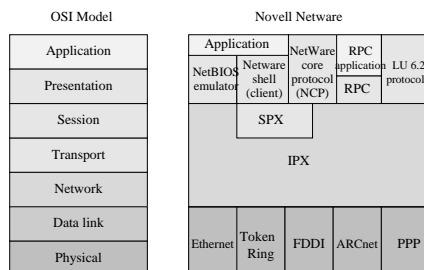
V praxi se stále ještě můžeme setkat s modelem IBM SNA (Systems Network Architecture), který původně navrhla roku 1974 firma IBM pro komunikaci terminálu se stanicí mainframe. Tento model obsahuje šest vrstev, viz obr. 1.1.



Obrázek 1.1: Architektura IBM SNA

Některé protokoly z tohoto modelu se stále používají. Např. protokol SMB (Server Message Block) pro sdílení souborů, tiskáren a dalších zařízení se stal základem protokolu CIFS (Common Internet File System) od Microsoftu a používá se dnes v aplikaci Samba (viz kapitola 6.14). Protokol NetBIOS (Network Basic Input Output System)[19] zajišťuje služby relační a transportní vrstvy společně s protokolem NetBEUI (NetBIOS Extended User Interface). Protokoly NetBIOS/NetBEUI zajišťují přenos data mezi aplikací a přenosovým médiem. Lze je použít například jako ekvivalent protokolů TCP/IP. Protokoly pro fyzickou vrstvu model SNA nedefinuje, ale využívá jiné standardy, např. IEEE.

Další firmou, která navrhla vlastní síťovou architekturu, tzv. protokolový profil (protocol suite)¹, je firma Novell s architekturou z roku 1980, která vycházela z návrhu XNS (Xerox Network Systems). Model Novell NetWare definuje pět horních vrstev RM OSI, viz obr. 1.2. Tento model byl navržený pro síťové operační systémy a poskytoval vzdálený přístup

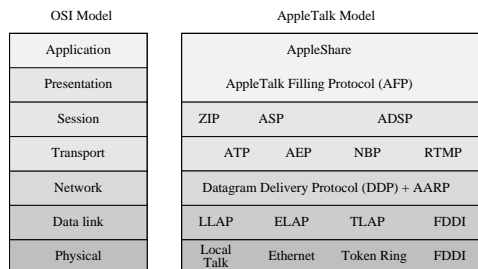


Obrázek 1.2: Architektura Novell NetWare

¹Protokolový profil (protocol suite) popisuje využití protokolů v daném síťovém modelu. Pokud daný profil implementujeme, hovoříme o protokolovém zásobníku (protocol stack)

k souborům, sdílení síťových služeb, tiskáren, databází či podporu aplikací typu přenos elektronické pošty. V současné době je tento systém na ústupu, i když s protokoly IPX/SPX se stále můžeme setkat.

Protokolový profil AppleTalk vyvinutý firmou Apple Computer v osmdesátých letech je spojený se světem počítačů MacIntosh. Systém AppleTalk propojoval počítače a umožňoval sdílení souborů, tiskáren apod. Model definuje čtyři základní komponenty pro výstavbu sítě – schránky (sockets), uzly, síť a zóny. Komunikaci zajišťují protokoly zobrazené na obr. 1.3.



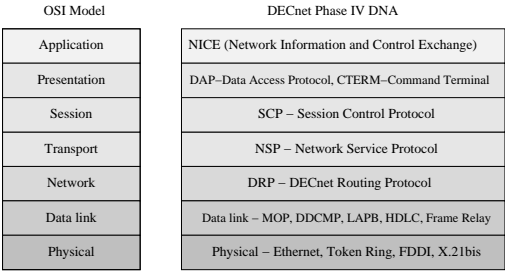
Obrázek 1.3: Architektura AppleTalk

Model obsahuje hierarchickou strukturu protokolů, které zabezpečují přenos. Kromě připojení k médiu (rozhraní LocalTalk, Ethernet, FDDI či Token Ring) je zajímavá síťová vrstva s protokolem DDP (Datagram Delivery Protocol). Na síťové vrstvě pracuje také protokol AARP (AppleTalk Address Resolution Protocol) sloužící pro asociaci síťových adres typu AppleTalk (16 bitů síť, 8 bitů uzel, 8 bitů socket, celkem 32 bitů) s hardwarovými adresami. Princip je podobný jako u dnes používaného protokolu ARP. Na transportní vrstvě se používá protokol pro přenos ATP (AppleTalk Transaction Protocol) či směrovací protokoly RTMP a AURP. V dnešní době dochází k ústupu počítačových sítí postavených na technologii AppleTalk.

Posledním z historických modelů architektury počítačové sítě je architektura DECnet vyvinutá a podporovaná firmou Digital Equipment Corporation (DEC, později součást HP). První verze architektury byla vytvořena v roce 1975 tak, aby odpovídala standardu OSI. Verze DECnet plus podporuje mnoho protokolů standardů OSI. Architektura sítě DECnet, Digital Network Architecture (DNA), verze Phase IV, je zobrazena na obr. 1.4.

Architektury SNA, Novell, AppleTalk či DECnet uvádíme spíše z historického hlediska. Dnes se s nimi málokterý čtenář setká. Nicméně demonstují vývoj počítačových sítí, architektury služeb i rozličné přístupy ke komunikaci, včetně různého způsobu adresování. Zájemcům o další studium mohou doporučit Přehled síťových protokolů [12] nebo knihu Internetworking Technologies Handbook [4], kde jsou tyto architektury podrobněji popsány.

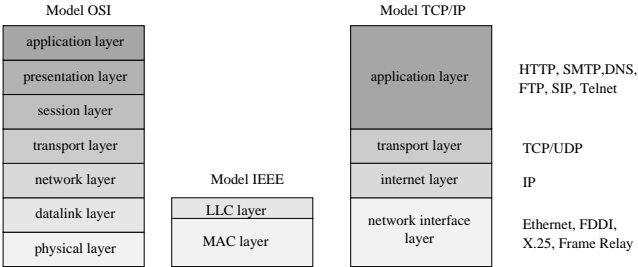
V současnosti je dominující architekturou v oblasti počítačových sítí model TCP/IP, který se prosadil zejména díky otevřeným standardům. Používá se nejen na Internetu a v oblasti lokálních počítačových sítí, ale proniká i do průmyslových sítí. S touto architekturou budeme pracovat v následujících kapitolách.



Obrázek 1.4: Architektura DECnet

1.2 Modely TCP/IP a ISO/OSI

Pro popis síťové architektury se používá referenční model OSI (Open Systems Interconnect Reference Model) definovaný Mezinárodní standardizační organizací ISO (International Standards Organization) v roce 1987 jako rámec standardů pro komunikaci po síti mezi různými zařízeními a aplikacemi různých výrobců. Model OSI lze dnes považovat za primární architekturu pro počítačovou komunikaci. Mnoho dnešních protokolů má strukturu vytvořenou právě podle modelu OSI. V praxi se implementovala pouze část tohoto modelu, plná implementace nikdy nebyla ve větší míře nasazena do provozu.



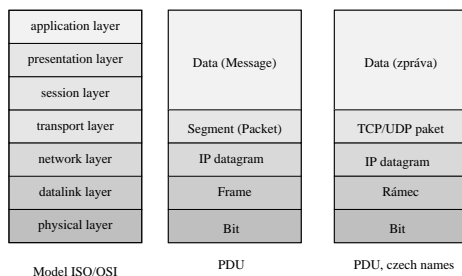
Obrázek 1.5: Porovnání modelů OSI, IEEE a TCP/IP

Referenční model OSI tvoří sedm vrstev, které definují služby příslušné vrstvy a odpovídající protokoly, obr. 1.5. Každá vrstva modelu OSI popisuje funkce pro přenos dat mezi procesy stejné vrstvy (tzv. partnerská komunikace, peer-to-peer communication). Vrstva nedefinuje pouze jeden protokol, nýbrž funkce datové komunikace, které mají provádět různé protokoly dané vrstvy. Při komunikaci využívá daná vrstva služeb nižších vrstev, aniž by musela znát, jak se tato vrstva chová a pomocí jakých protokolů byla implementována. Přenos po síti je z pohledu dané vrstvy záležitostí nejbližší vrstvy.

Na obrázku 1.6 jsou zobrazeny základní datové jednotky na jednotlivých úrovních vrstvo-

vého modelu (PDU - Process Data Unit). Pokud např. probíhá komunikace mezi aplikací, dochází k přenosu dat (zpráv) mezi komunikujícími programy. Tyto programy se nestarají o adresování, vyhledání cesty, navázání spojení apod., což je úkol nižších vrstev.

Například uživatel posílá data pomocí aplikačního programu. Data jsou transportní vrstvou rozdělena na pakety TCP či UDP a předána síťové vrstvě k doručení. Síťová vrstva zapouzdří pakety do IP datagramů a předá je síťovému rozhraní (např. síťové kartě v počítači). Síťové rozhraní převede data do rámců (přidá fyzickou adresu) a запиše na přenosové médium (metalická kabeláž, optický kabel, bezdrátová WiFi síť). Přenosové médium přenáší data ve formě bitů zakódovaných na určitou fyzikální veličinu podle typu datového média (např. hladinu elektrického napětí, frekvenci rádiové záření, vlnovou délku světelného toku).



Obrázek 1.6: Datové entity (PDU) v jednotlivých vrstvách

Názvy jednotek PDU se v různých standardech liší. V původní definici TCP/IP se píše: „*Jméno datové jednotky toku internetu závisí na tom, kde se objevuje v protokolovém profilu. Stručně řečeno: pokud je na Ethernetu, nazývá se Ethernet frame, pokud je PDU mezi řadičem Ethernetu a IP modulem, nazývá se IP paket, pokud je to mezi modulem IP a UDP, nazývá se UDP datagram, pokud je to mezi modulem IP a TCP, nazývá se TCP segment; pokud je součástí síťové aplikace, nazývá se aplikační zprávou. Tyto definice však nejsou dokonalé. Aktuální pojmenování se liší v různých publikacích.*“ [28, 2.2] V českém názvosloví se používají termíny bit, rámec (frame), datagram (IP datagram), paket (UDP a TCP) či data/zpráva.

1.2.1 Vrstvy modelu OSI

V téměř každé publikaci o počítačových sítích se setkáme s popisem referenčního modelu OSI. Zde si uvedeme pouze stručný popis jednotlivých vrstev, podrobnější informace lze najít např. v [27], [10].

- **Fyzická vrstva** definuje fyzické vlastnosti linky, např. napěťové charakteristiky, počet a umístění pinů konektoru, kódování úrovně napětí pro logickou 0 a 1, fyzickou rychlost přenosového média apod. Mezi protokoly fyzické vrstvy patří standardy RS-232C (sériové rozhraní), V.35, IEEE 802.3 (CSMA/CD), IEEE 802.11 (WiFi), 802.5 Token Ring, FDDI, X.25 a další.
- **Linková vrstva** popisuje přenos dat po konkrétní datové lince, vytváření rámců, adresování na linkové vrstvě. Popisují ji standardy 802 (fyzická i linková), ISO 8802/2,

8802/3, 8802/4 a 8802/5 (odpovídá standardům IEEE 802.x), protokoly Ethernet, PPP, Token Ring, Frame Relay, SDLC, HDLC a další.

- **Síťová vrstva** zajišťuje adresování a směrování dat. Vrstva definuje protokol X.25 a standardy ISO 8878, ISO 8473. Na této vrstvě pracují směrovací protokoly ES-IS (End System-to-Intermediate System) a IS-IS. Spojované služby jsou popsány protokoly CONP (Connection-Oriented Network Protocol) a CMNS (Connection-Mode Network Service), nespojované služby používají protokol CLNP (Connectionless Network Protocol) a službu CLNS (Connectionless Network Service).

Příkladem adresy ISO počítače na síťové vrstvě podle standardu ISO 10589 je např. 49.0005.AA00.0301.16CD.00, kde první oktet (49) určuje identifikaci a formát adresy, hodnota 0005 definuje doménu, AA00 je číslo oblasti v rámci autonomního systému, dalších osm oktetů tvoří identifikátor systému (v tomto případě MAC adresu) a poslední oktet určuje službu na síťové vrstvě. Tyto adresy ISO se dnes příliš nepoužívají s výjimkou směrovacího protokolu IS-IS.

- **Transportní vrstva** garantuje spolehlivý přenos mezi koncovými uzly. Standard ISO 8073 definuje protokoly transportní služby typu A (spolehlivá síťová služba), B (spolehlivá síťová služba s vyznačením chyb), C (nespolehlivá síťová služba). Vrstva implementuje spojované a nespojované transportní protokoly, které jsou podle typu přenosu a způsobu fragmentace dat rozděleny do pěti protokolových tříd Transport Protocol Class (TP), TP0 až TP4. Tyto třídy byly implementovány např. jako součástí softwarového balíku ISODE.
- **Relační vrstva** slouží k udržování relací mezi komunikujícími aplikacemi. Služby zahrnují vytvoření, zrušení relace, obsluhu dialogů (při poloduplexním spojení), synchronizaci apod. Standard ISO 8326 definuje služby relační vrstvy a standard ISO 8327 protokol relační vrstvy.
- **Presentační vrstva** zajišťuje zobrazení (prezentaci) dat mezi různými aplikacemi a architekturami. Zahrnuje odlišné formáty dat (ASCII, EBCDIC, binární data), kompresi dat a kódování. Standard ISO 8822 definuje služby prezenční vrstvy, standard ISO 8823 popisuje protokoly. Standard ISO 8824 definuje abstraktní syntaktickou notaci ASN.1 (Abstract Syntax Notation One), která se používá např. pro prezentaci dat v síťových hardwarových zařízeních (databáze MIB, viz kapitola 11.4.4.2, LDAP apod.).
- **Aplikační vrstva** definuje uživatelské procesy a aplikace komunikující po síti. Patří sem např. služby pro obecné aplikace CASE (Common Application Service Elements) popsané standardy ISO 8649 (služby CASE) a ISO 8650 (protokoly CASE). Mezi tyto aplikace patří:
 - Elektronická pošta MHS (Message Handling System) – standard ITU-T X.400.
 - Adresářové služby – standard ISO 9594 a ITU-T X.500
 - Virtuální terminál (VT) – ISO 9040 (služby) a ISO 9041 (protokoly).
 - Přenos souborů, přístup a zpracování – FTAM (File Transfer Access and Management).

1.2.2 Model TCP/IP

Vznik modelu TCP/IP a jeho rozšíření v Internetu souvisí s obdobím studené války v 60. letech 20. století a s vypuštěním sovětské sondy Sputnik v roce 1957. Jako reakci na sovětskou přítomnost na oběžné dráze vytvořil americký úřad obrany, US Department of Defence (DoD), agenturu DARPA (Defence Advanced Research Project Agency). V roce 1969 začal projekt ARPAnet, jehož cílem bylo vyvinout spolehlivou komunikační technologii, která by obstála i v případě narušení pozemních komunikačních cest. Síť měla být decentralizovaná, robustní, nezávislá na přenosovém médiu a snadno implementovatelná. V případě narušení přenosové cesty měla tato síť využít jiné komunikační linky pro doručení dat. Principy této architektury vedly o několik let později ke vzniku modelu TCP/IP.

1.2.2.1 Krátký přehled historie Internetu

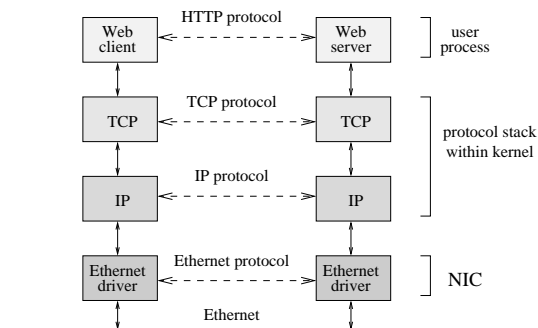
- V roce 1969 agentura ARPA financuje projekt na vytvoření experimentální sítě s přepínáním paketů. Vzniká síť ARPAnet.
- V roce 1975 je na síti ARPAnet zahájen běžný provoz, dochází k vývoji základních protokolů TCP/IP. Protokoly IP a TCP byly standardizovány v roce 1981.
- V roce 1983 byly protokoly TCP/IP implementovány v operačním systému Berkeley (BSD) Unix.
- V téže roce byla síť ARPAnet rozdělena na MILNET a část ARPAnet. Termín Internet se používá pro veškerou síť (ARPAnet a MILNET).
- V roce 1985 vzniká nadace National Science Foundation (NSF) a začíná projekt sítě NSFNet propojující akademické a výzkumné organizace v USA k Internetu. Rychlost páteřních linek sítě je 56-kbit/s.
- V roce 1987 vytváří NSF páteřní síť Internetu s linkami T1 (1.5 Mbit/s) a definuje třívrstvou topologii sítě (páteřní síť, regionální sítě, lokální sítě). Od roku 1991 tvoří páteřní spoje linky T3 (45 Mbit/s). V roce 1995 přestává být NSFNet primární páteřní sítí Internetu. Páteřní spoje jsou tvořeny komerčními subjekty např. internetMCI, PSINet, SprintLink či ANSNet.

1.2.2.2 Architektura TCP/IP

Architektura TCP/IP je výrazně jednodušší než referenční model OSI. Model TCP/IP spojuje služby prezentační a relační do jediné vrstvy, aplikační. Na úrovni fyzické přenosu bitů spojuje vrstvy fyzickou a linkovou do vrstvy fyzického rozhraní, která je obvykle implementovaná na síťové kartě. Srovnání obou modelů je na obrázku č. 1.5. Model TCP/IP odstraňuje nedostatky, které měla komplikovaná struktura vrstev modelu OSI a které nebyly plně nikdy implementovány.

Vlastní implementace architektury TCP/IP je rozdělena do tří částí. Nejnižší část, vrstva fyzického rozhraní, je implementována v síťové kartě (NIC, Network Interface Card) a ovladači karty (driver). Vyšší vrstvy – internetová a transportní – jsou součástí síťových modulů operačních systémů. Jejich nastavení a ruční instalace je popsána v kapitole 1.4.

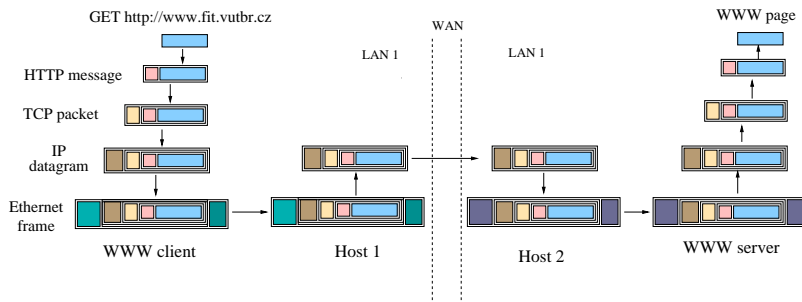
Příklad aplikace modelu TCP/IP pro službu WWW (protokol HTTP) lze graficky znázornit na obrázku č.1.7. Podobně i pro další služby je možné sestavit obdobný model, který



Obrázek 1.7: Komunikace klienta a serveru v prostředí WWW.

pro jednotlivé vrstvy modelu definuje používaný protokol. Protokol nejnižší vrstvy závisí na zvolené přenosové technologii (Ethernet, bezdrát, optika) a pro danou službu je transparentní.

Fyzický přenos dat je realizován pomocí datových jednotek PDU, o kterých jsme se zmínili v předchozí kapitole. Při přenosu po síti dochází k *zapouzdření dat* (*encapsulation*) vyšších vrstev do PDU nižších vrstev na straně odesílatele a *rozbalení dat* (*decapsulation*) na straně příjemce. Graficky je přenos dat znázorněn na obrázku č. 1.8. Na tomto obrázku jsou data z



Obrázek 1.8: Přenos dat služby WWW počítačovou sítí přes WAN síť.

klienta WWW nejprve vložena do protokolové jednotky protokolu HTTP (příkaz GET) na aplikační vrstvě. Poté se přidá hlavička transportní vrstvy a vytvoří se paket TCP. K této datové jednotce je připojena hlavička IP s IP adresou zdroje a cíle. Datagram IP je vložen do ethernetovského rámce a poslán na přenosové médium.

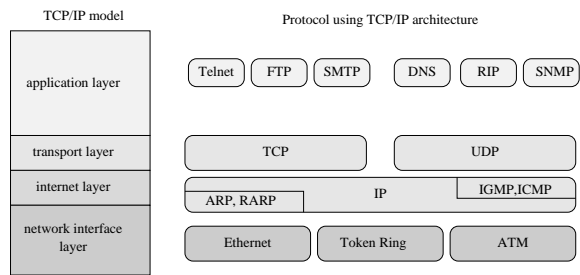
Datové rámce procházejí obvykle mezilehlými uzly², které sice nemění datovou část aplikačního protokolu, ale mohou zasáhnout nižší přenosové protokoly. Například *Host 1* se chová

²síťovými zařízeními, které jsou součástí přenosové cesty (například směrovače)

jako vstupní brána sítě, která např. převádí data z veřejné adres používané na Internetu do privátní sítě lokální sítě. Na obrázku je znázorněn změněný formát IP hlavičky a také nové hlavičky ethernetovského rámce. Samotné tělo IP datagramu – transportní a aplikační data – se nemění. Po průchodu počítačovou sítí je rámec načten počítačem, kde běží cílový server WWW, který obsahuje požadovaná data. Ethernetovský rámec je odstraněn síťovou kartou počítače a IP data jsou předána síťovému modulu. Ten provede kontrolu adresy a kontrolního součtu. Pokud jsou IP datagram i vnořený TCP paket v pořádku, dojde k vypouzdření dat a předání zprávy HTTP aplikaci, v tomto případě serveru WWW. Server provede akci, např. načtení požadované stránky. Požadovanou stránku WWW pošle obdobným způsobem zpět klientovi.

Vrstvy modelu TCP/IP

Protokolový profil (protocol suite) TCP/IP je základem Internetu. Je postaven na čtyřvrstvě modelu – viz obrázek č.1.9. Tento model skrývá funkce nižších vrstev tj, fyzické rozhraní a internetová vrstva, a soustředí se na transport dat.



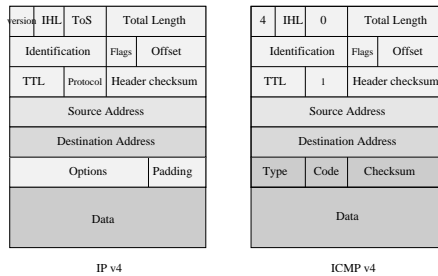
Obrázek 1.9: Architektura a protokoly TCP/IP

1. **Vrstva fyzického rozhraní (Network Access Layer, Network Interface Layer)** popisuje standardy pro fyzické médium a elektrické signály, např. Ethernet (viz 1.10), Token Ring, FDDI, X.25, Frame Relay. Kromě fyzických technologií (metalická kabeláž, bezdrátový přenos, optika, rádiové vlny) zahrnuje také funkce pro přístup k fyzickému médium (ovladače síťových karet). Tato vrstva také zajišťuje zapouzdření IP datagramů do rámců.

Length/ Type					
Preamble	Destination address	Source address		Data+Pad	FCS
7B+1B	6B	6B	2B	46–1500B	4B

Obrázek 1.10: Formát rámce Ethernetu IEEE 802.3

2. **Internetová vrstva (IP Layer)** vytváří datagramy, adresuje je a směřuje na místo určení. Zajišťuje tak zvané doručení s největším úsilím (best-effort delivery), což znamená, že vrstva IP se snaží doručit poslaná data nejvhodnější cestou. Pokud dojde ke ztrátě dat (výpadek linky, přeplněné fronty na směrovačích), vysílající uzel je o ztrátě datagramu informován, nicméně musí sám zajistit opětovné přenesení dat. Internetová vrstva kromě přenosového protokolu IP (Internet Protocol) [16] používá protokoly ARP (Address Resolution Protocol) [6] a RARP (Reverse ARP) [24], které slouží k mapování IP adres na MAC adresy. Dále jsou to protokoly ICMP (Internet Control Message Protocol) [15] pro řízení toku a detekci nedosažitelných uzlů a protokol IGMP (Internet Group Management Protocol) [25, 1] pro přihlašování se do multicastových skupin. Tyto protokoly musí být součástí implementace modulu IP v operačním systému. Bez těchto protokolů nebude komunikace na síťové vrstvě korektně pracovat.



Obrázek 1.11: Struktura protokolu IPv4 a ICMP

Ukázka protokolu IP verze 4 a ICMP verze 4 je na obrázku č. 1.11. Na internetové vrstvě modelu TCP/IP se používá adresování pomocí 32-bitové IP adresy. Délku datagramu (Total Length) obsahuje 16-bitová hodnota v hlavičce protokolu, proto maximální délka datagramu IPv4 včetně hlavičky je $2^{16} - 1 = 65535$ bytů (64 kB). V praxi se maximální délka nepoužívá, protože na nižší vrstvě, Ethernetu, je délka dat v rámci omezena na 1 500 bytů. V případě delších IP datagramů by docházelo k fragmentaci na lokálních sítích. Proto se i délka IP datagramů volí kratší. Při průchodu IP datagramu uzlem sítě se hodnota kontrolního součtu hlavičky (Header Checksum) musí znovu přepočítat, neboť se dekrementuje hodnota TTL (Time To Live). K přepočítání dochází také při použití privátních adres a překladu adres NAT (viz kapitola 1.3.3.4).

Mezi základní operace vrstvy IP patří

- definice datagramu a způsobu adresování,
- přenos dat mezi internetovou vrstvou a fyzickým rozhraním a
- směrování datagramu na vzdálený počítač.

IP vrstva neprovádí žádné kontroly správnosti přenosu ani opravu poškozených dat. Proto se protokol IP někdy nazývá nespolehlivý protokol. Kontrolu obsahu a opravy chyb je nutné provádět na vyšších vrstvách modelu TCP/IP.

Internetová vrstva využívá také linkový protokol ARP [6], který je implementován přímo nad Ethernetem či jinou lokální sítí (ARCNET, Frame Relay, LocalTalk, FDDI apod.). ARP slouží k překladu fyzických adres na IP adresy a obráceně.

Hardware Type		Protocol Type
HLen	PLen	Operation
Sender Hardware Address		
Sender Protocol Address		
Target Hardware Address		
Target Protocol Address		

ARP format

Obrázek 1.12: Struktura protokolu ARP, Address Resolution Protocol

Dalším protokolem, který je součástí vrstvy IP, který a musí být implementován v každém modulu IP, je protokol ICMP [15]. ICMP je ve skutečnosti součástí protokolu IP, který v poli hlavičky **Protocol** obsahuje hodnotu 1. ICMP slouží k řízení a kontrole toku dat na síťové úrovni, blíže viz kapitola 11.2.

3. **Transportní vrstva (Transport Layer)** přenáší data z aplikace na zdrojovém počítači do aplikace na cílovém počítači. Vytváří tak zvané *logické spojení* mezi procesy (narozdíl od protokolu IP, který vytváří logické propojení uzlů). Transportní protokoly rozdělují aplikační data na menší jednotky (pakety), které posílají po síti.

Source Port			Destination Port		
Sequence Number					
Acknowledgement Number					
Offset	Reserved		Flags	Window	
Checksum			Urgent Pointer		
Options					Padding
Data					

TCP format

Source Port	Destination Port
Length	Checksum
Data	

UDP format

Obrázek 1.13: Formát protokolu TCP a UDP

Mezi základní činnosti transportní vrstvy patří

- segmentace aplikačních dat (TCP, UDP),
- posílání dat z jednoho koncového zařízení na druhé koncové zařízení (TCP, UDP),
- ustavení spojení (pouze TCP),
- řízení toku dat mechanismem posuvné okno (sliding window) (pouze TCP) či

- spolehlivý přenos pomocí sekvenčních čísel a potvrzování (pouze TCP).

Transportní služba slouží k předávání dat mezi koncovými uzly. Protokoly transportní vrstvy zajišťují logické spojení mezi aplikačními procesy běžícími na různých počítačích. Z pohledu aplikace to vypadá, jako by počítače, na nichž běží tyto procesy, byly přímo spojeny. Zatímco transportní vrstva vytváří *logické spojení mezi procesy*, síťová vrstva *logické spojení mezi počítači*. Základní protokoly transportní vrstvy jsou protokoly TCP a UDP, viz obr. 1.13.

Protokol TCP (Transmission Control Protocol) [17] načítá data z aplikační vrstvy jako proud dat, které seskupuje do jednotlivých paketů, a které posílá po síti k cíli. Na cílové straně dochází k přeskládání paketů podle pořadí do datového proudu pomocí sekvenčních čísel, které určují, na které místo v toku dat patří data v daném paketu.

TCP zajišťuje spolehlivý přenos dat: řízení toku, řazení paketů, potvrzování či řízení zahlcení (congestion control). Prostřednictvím vyrovnávací paměti pro odeslaná data (sending window) řídí rychlost přenosu – hodnota `Window` v hlavičce protokolu.

Protokol UDP (User Datagram Protocol) [13] umožňuje rychlý přenos paketů bez zajišťování spolehlivého doručení. Implementuje tak zvané *nespojované služby* (connectionless). Zdrojový uzel musí sám zajistit potvrzování (neboť paket se může ztratit) a také doručení ve správném pořadí. Spojení UDP je oproti TCP rychlejší, neboť odpadá režie s vytvořením a zrušením spojení a také režie spojená s opakovaným doručením ztracených paketů. Přenos UDP se používá zejména u aplikacích přenášející menší objemy dat v paketu a s důrazem na rychlost doručení, např. aplikace pro správu a řízení sítě (SNMP, DNS) či u multimediálních přenosů. Pokud je síť hodně vytížená nebo na ni dochází k častým ztrátám, je spojení TCP velmi pomalé, protože dynamicky při ztrátách vysílající uzel snižuje rychlost přenosu.

Aplikace	Aplikační protokol	Transportní protokol
Elektronická pošta	SMTP	TCP
Vzdálené přihlášení	telnet	TCP
Zabezpečené přihlašování	ssh	TCP
Vzdálený tisk	lpr	TCP
Služba WWW	HTTP	TCP
Přenos souborů	FTP	TCP
Sdílení souborů	NFS	UDP, TCP
Multimediální aplikace	RTP/RTCP	UDP
IP telefonie	SIP,H.323	UDP, TCP
Správa sítě	SNMP	UDP
Směrování	RIP	UDP
Adresace	DNS	UDP, TCP

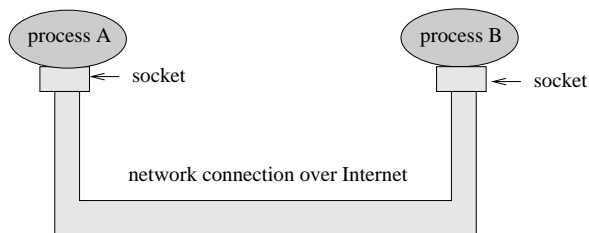
Tabulka 1.1: Příklady aplikací pracujících nad TCP a UDP

Protokoly TCP (spolehlivé spojované služby), UDP (nespolehlivé nespojované služby) pracují nad protokolem IP (síťová vrstva), který zajišťuje *doručení s největším úsilím* (best-effort delivery). To znamená, že IP se snaží doručit paket mezi komunikujícími

uzly, ale bez záruky. Nezaručuje doručení dat, pořadí přenesených datagramů či poškození dat (obsahuje pouze kontrolní součet hlavičky, nikoliv dat!).

Podle typu mohou aplikace pracovat nad protokoly TCP či UDP, případně nad oběma transportními protokoly. Např. standard HTTP umožňuje implementovat HTTP zprávy nad UDP, v praxi se tato implementace nepoužívá. Příklady aplikací pracujících nad TCP a UDP jsou uvedeny v tabulce 1.1.

Při programování komunikace po síti obvykle nepotřebujeme implementovat transportní protokoly. Většinou nám jde o komunikaci mezi aplikacemi. Proto můžeme využít již implementované knihovny funkcí, které poskytují přenos TCP/IP. Jednou z nejrozšířenějších implementací je jsou tak zvané *schránky* (sockets)- viz obr. 1.14. Schránky si můžete představit jako brány, přes něž se přenáší data po síti. Aplikace se tedy nemusí starat o komunikaci po síti. Jednoduše si nastaví parametry spojení a zapíše data do speciální datové struktury spojené se schránkou. Vlastní přenos zajistí knihovna pro práci se schránkami.



Obrázek 1.14: Komunikace aplikací po síti přes rozhraní *socket*

Schránky jsou v rámci sítě jednoznačně identifikované - adresují síťová zařízení, která se účastní komunikace (IP adresa), a aplikace, která si vyměňují data (pomocí čísla portu). Při vytváření logického spojení mezi schránkami je potřeba, aby schránky na obou stranách znaly svou IP adresu a své číslo portu, dále pak IP adresu komunikujícího partnera a číslo portu, kde běží vzdálená aplikace, s níž si bude vyměňovat data. Nejběžnější implementací schránek je knihovna BSD sockets. Principy programování komunikace a použití těchto schránek se budeme zabývat v kapitole 2.

4. **Aplikační vrstva (Application Layer)** je tvořena procesy a aplikacemi, které komunikují po síti. Vrstva zajišťuje zpracování dat na nejvyšší úrovni včetně reprezentace dat, kódování i řízení dialogu.

Aplikační protokoly můžeme rozdělit do dvou kategorií na

- (a) *uživatelské protokoly*, které vykonávají služby přímo uživateli (např. Telnet, FTP, SMTP), a na
- (b) *systémové protokoly*, které zajišťuje síťové funkce (např. SNMP, BOOTP, DNS).

Mezi běžné aplikační protokoly patří

- Telnet (Terminal Emulation) pro vzdálený přístup na počítač přes síť,
- FTP (File Transfer Protocol) pro spolehlivý interaktivní přenos souborů nad TCP,
- TFTP (Trivial File Transfer Protocol) zajišťující nespojovaný přenos souborů nad UDP. Tento protokol se používá např. při přenosu konfiguračních dat na aktivní síťové prvky či bezdiskové stanice.
- SMTP (Simple Mail Transfer Protocol), který zajišťuje přenos elektronické pošty,
- HTTP (Hypertext Transfer Protocol) pro přenos webových stránek,
- DNS (Domain Name System), což je služba zajišťující např. mapování IP adres na doménová jména,
- OSPF (Open Shortest Path First), což je směrovací protokol (routing protocol) pro výměnu směrovacích informací mezi uzly na síti,
- SNMP (Simple Network Management Protocol) se využívá pro monitorování a řízení síťových zařízení, jejich konfiguraci, zjišťování statistických informací, sledování výkonu a bezpečnosti zařízení apod., a
- NFS (Network File System), což je distribuovaný souborový systém, který slouží zejména pro sdílení souborů na více uzlech v síti.

Podrobnější popis těchto protokolů je náplní dalších kapitol, proto se zde spokojíme pouze s výčtem základních aplikačních protokolů a jejich stručnou charakteristikou.

1.3 Způsoby adresování u TCP/IP

Jednou z nejdůležitějších oblastí, kterou potřebujeme pro správné pochopení počítačové komunikace, je oblast adresování. Adresováním zde rozumíme *způsob identifikace adresáta (cíle) pomocí jednoznačné informace—adresy*. Podobně jako musíme znát adresu bydliště kamaráda, kterého jdeme navštívit nebo telefonní číslo přítelkyně, s níž chceme mluvit, tak potřebujeme znát i adresu počítače, procesu či služby, se kterými budeme komunikovat po síti.

Adresování zahrnuje stanovení formátu adresy (čísla, řetězce, oddělovače, délka adresy), rozdělení adres do skupin (např. na individuální, speciální), způsob přidělování adres (na lokální síti, globální autorita), uchování adres (pevné adresy, adresy přidělené pouze dočasně, lokálně závislé adresy) a mapování adres (převod jednoho typu adresy na jiný).

V počítačové komunikaci se používá více druhů adres. Můžeme se setkat s hardwarovou adresou síťové karty (např. 00:0C:6E:77:CE:22), asi všichni známe IP adresy (např. 147.229.12.91), doménové adresy (např. www.fit.vutbr.cz), emailové adresy (např. matousp@fit.vutbr.cz) či adresy stránek WWW (např. www.fit.vutbr.cz/study/courses/ISA).

Připomeňme si, že architektura počítačové komunikace je tvořena jako vrstevná, kde každá vrstva je nezávislá na vyšších vrstvách. Komunikace probíhá vždy na dané vrstvě s využitím služeb nejbližší nižší vrstvy. Na adresování lze také nahlížet podle jednotlivých vrstev modelu TCP/IP.

• Adresování na vrstvě fyzického rozhraní

Na fyzické vrstvě používáme tzv. *fyzickou adresu*. U technologií založených na Ethernetu (802.3, 803.11) se jedná o 48-bitovou adresu, která se také nazývá ethernetovská nebo hardwarová adresa. Tato adresa jednoznačně identifikuje síťové rozhraní počítače (kartu NIC). Prvních 24 bitů určuje výrobce karty (OUI, Organizational Unique

Identifier), dalších 24 bitů je číslo karty přidělené výrobcem. Seznam OUI spravuje organizace IEEE. Např. adresa 00:0c:6e:77:ce:22 nám říká, že jde o síťovou kartu vyrobenou firmou Asustek Computer Inc. z Taiwanu (00 0c 6e) s identifikačním číslem 77 ce 22. Např. prefix 00:11:17 má registrovaný provozovatel české akademické sítě CESNET (www.cesnet.cz). Seznam registrovaných identifikátorů OUI lze najít na adrese <http://standards.ieee.org/regauth/oui>. Cena za přidělení OUI je 1 650 USD (2006).

Tato adresa je trvale uložena v paměti ROM na síťovém zařízení. Po spuštění počítače a inicializaci síťové karty je zkopírována do paměti RAM. Některé operační systémy umožňují fyzickou adresu změnit. Jde však o změnu softwarovou, tj. změnu softwarové kopie. Fyzická adresa by se neměla měnit. Neuvážené změny mohou narušit či omezit činnost lokální sítě – např. duplikace MAC adres na LAN segmentu, změna MAC adresy na přepínané síti apod.

Fyzická adresa se používá pro adresování rámců v lokálních sítích. Zařízení na druhé vrstvě OSI modelu (například mosty a přepínače) si ukládají fyzické adresy v přepínacích tabulkách (switching tables, MAC tables). Přepínací tabulka obsahuje mapování čísla portu přepínače na připojené zařízení, konkrétně na jeho MAC adresu. Např. obsahuje informaci, že k portu FastEthernet č. 1 je připojen počítač s MAC adresou 00:16:36:53:EC:8B.

Podle fyzické adresy může počítač či jiné síťové zařízení zjistit, zda je zpráva určena jemu. Pokud se cílová adresa rámce liší, zařízení přijatou zprávu ignoruje. Speciální adresa FF:FF:FF:FF:FF:FF (samé jedničky) se nazývá adresa všesměrového vysílání (broadcast). Rámec s touto adresou cíle je určen pro všechna síťová zařízení v lokální síti.

• Adresování na IP vrstvě

K identifikaci počítače na internetové vrstvě (u OSI modelu na síťové) slouží *IP adresa*. Tato adresa je ve verzi 4 protokolu IP 32-bitová, u verze 6 dokonce 128-bitová. IP adresa se na rozdíl od hardwarové adresy neomezuje pouze na adresování v sítích LAN. IP adresa je jednoznačná v celém Internetu. Protože je přidělování IP adres rozděleno geograficky, lze na základě IP adresy úspěšně směřovat data a vytvářet spojení. Příkladem IP adres je např. adresa 147.229.12.91.

Kontrolu IP adres provádějí zařízení na internetové vrstvě, tj. na třetí vrstvě OSI (též vrstvě L3). Nastavení IP adresy provádí správce síťového zařízení. U uživatelských stanic se přidělování provádí automaticky ze serveru DHCP. IP adresy mohou být přiděleny staticky, tj. manuálně, kdy se jejich hodnota se nemění, nebo lze IP adresu přidělit pouze na dobu komunikace a po uplynutí určitého času ji použít pro jiný počítač. Tomu se říká dynamické přidělování IP adres. Formát IP adres, jejich rozdělení, způsoby přidělování a použití popíšeme podrobněji v následující kapitole.

Každé síťové rozhraní (NIC) má kromě fyzické adresy přiděleno alespoň jednu IP adresu. Ve výjimečných případech lze mít na jednom fyzickém rozhraní více IP adres. Toto se týká spíše aktivních síťových prvků, např. směrovačů.

K převodu IP adresy na fyzickou adresu se používá protokol ARP [6]. Protokol RARP (reverzní ARP) [24] se používá pro opačný převod. V případě výpadku ARP komunikace nelze v LAN síti doručovat žádná data a daný segment LAN je nedostupný.

• Adresování na transportní vrstvě

Na transportní vrstvě neadresujeme počítače (např. síťové karty) ale služby, které na těchto zařízeních běží. Příkladem může být např. emailový server, služba DNS apod. Adresu služby přenáší ve své hlavičce protokol TCP či UDP ve formě 16-bitového čísla portu. Číslo portu identifikuje službu (aplikační proces), která data posílá (port odesílatele) či cílový aplikační proces (číslu portu adresáta). Např. webový klient posílá žádost z IP adresy 147.229.12.91 a portu 54690 na server WWW, který běží na počítači s adresou 147.229.2.90 na portu 80. Přenosový kanál pro komunikaci dvou aplikací nad transportní vrstvou je určen typem transportního protokolu, IP adresami síťového rozhraní a čísly portů.

Port	Protokol	Služba
21	ftp	Přenos souborů
22	ssh	Bezpečné přihlašování
23	telnet	Vzdálené přihlašování
25	smtp	Přenos elektronické pošty
53	dns	Překlad doménových adres
67	dhcp	DHCP server
80	http	Přenos WWW stránek
110	pop3	Čtení elektronické pošty
143	imap	Čtení elektronické pošty
161	snmp	Správa sítě
389	ldap	Adresářové služby
443	https	Zabezpečený přenos WWW stránek
585	imap4ssl	Zabezpečené čtení el.pošty

Tabulka 1.2: Přehled některých rezervovaných portů.

Všimněte si, že číslo portu adresáta (WWW serveru) je jiné než číslo portu odesílatele (WWW klienta). Je to proto, že identifikace adresáta (většinou nějaké obecně známé služby, např. WWW serveru, poštovního serveru, DNS serveru apod.) musí být obecně známá. K čemu by bylo, kdybychom znali IP adresu serveru www.liberouter.org, ale služba WWW by běžela na jiném portu než standardním? Proto jsou známým službám (well-known services) přiděleny porty centrálně standardizační organizací IANA. Pro některé účely má smysl používat WWW server na jiném portu – potom ale musí uživatel zadat ve svém prohlížeči nejen adresu serveru, ale i číslo portu. Pokud ho nezadá, doplní prohlížeč standardní číslo služby, tj. 80 pro WWW. Změna portu WWW serveru se využívá např. pro proxy servery či neveřejné servery.

Číslo portů se dělí na porty rezervované (0–1023), registrované (1024–49151) a dynamické (49452–65535). Rezervované porty přiděluje organizace IANA (Internet Assigned Numbers Authority) [14] pro standardní služby, například FTP, SNMP, HTTP apod., viz tabulka 1.2.

Registrované porty nejsou přidělovány IANA, ale IANA je registruje a zobrazuje ve výpisu. Používají se pro uživatelské procesy. Patří tam většinou firemní aplikační protokoly. Např. 6000–6063 využívá server XWindow pro své protokoly, port 2049 je vyhrazen

pro server NSF a tak dále. Dynamické porty (dynamic, private ports) jsou určeny pro volné, krátkodobé využití. Na různých systémech mohou identifikovat jinou službu.

• Adresování na aplikační vrstvě

Adresování na aplikační úrovni závisí na konkrétní aplikaci. Např. u elektronické pošty se používají adresy ve formátu (`user@host`, např. `matousp@fit.vutbr.cz`)[21], u systému WWW jsou to adresy ve formátu URL (Uniform, Resource Locator)[3], např. `http://www.fit.vutbr.cz/study/courses/ISA/`.

Důležitý význam mají tzv. *doménové adresy* (host names, domain names), které se používají k identifikaci počítačů v síti. Narozdíl od IP adres jsou tvořeny snadno zapamatovatelnými řetězci oddělenými tečkami (např. `www.google.com`). Pro datovou komunikaci je tento formát nevhodný. Doménové adresy se kromě případů speciálního použití mapují na IP adresy v poměru 1:1.

Doménová jména se používá pouze na aplikační vrstvě. To znamená, že před každou komunikací se musí doménová adresa převést na IP adresu. K tomu slouží služba DNS (Domain Name System), která provádí mapování doménových jmen na IP adresy. Tento překlad je pro uživatele transparentní a mnohdy si uživatelé neuvědomují, že službu DNS používají. Popisu služby DNS je věnována část kapitoly 3.1.

V této části jsme si uvedli různé způsoby adresování při datové komunikaci. V Internetu se obvykle adresuje pomocí doménové adresy, která se převádí na IP adresu (DNS). Pro doručení na konkrétní fyzické rozhraní počítače je potřeba mapování na fyzickou adresu (ARP). Pokud správně nepracují překlady DNS a ARP, nelze navázat spojení a úspěšně komunikovat.

1.3.1 Příklad komunikace

Jak tedy probíhá odeslání dat po síti TCP/IP? Například načtení úvodní stránky WWW na VUT v Brně obsahuje následující kroky:

1. Uživatel zadá prohlížeči WWW adresu požadované stránky `http://www.fit.vutbr.cz`.
2. Webový prohlížeč (tj. klient služby WWW) pošle pomocí síťových knihoven požadavek na server DNS s žádostí o překlad doménové adresy na IP adresu. Klient obdrží IP adresu 147.229.2.90. Pokud ji nedostane, nahlásí uživateli chybu a skončí komunikaci.
3. Prohlížeč následně vytváří spojení z lokálního počítače na počítač 147.229.2.90 a port 80 (služba WWW). Pomocí síťové knihovny vytvoří paket TCP (s příznakem SYN) s cílem vytvořit kanál TCP mezi prohlížečem a webovým serverem.
4. Síťový modul TCP/IP počítače zapouzdří paket TCP do datagramu IP a přidá IP adresy odesílatele a cíle. Pomocí protokolu ARP zjistí, jaká fyzická adresa přísluší IP adrese cíle. IP datagram předá k odeslání síťové kartě.
5. Síťová karta vytvoří ethernetový rámec, vloží do něj požadovaná data, MAC adresu zdroje a získanou MAC adresu cíle. Vytvořený rámec запиše na přenosové médium.
6. Počítač `www.vutbr.cz` připojený ve stejné síti LAN přijme došlý rámec. Porovná cílovou MAC adresou s adresou síťové karty a tím zjistí, zda je rámec určen pro něho. Zkontroluje správnost dat pomocí kontrolního součtu FCS, odstraní postupně hlavičky

protokolů IP a TCP a předá aplikační data (ve formátu protokolu HTTP) webovému serveru, který běží na portu 80. Server požadavek zpracuje a podobným způsobem odešle odpověď.

1.3.2 Přidělování IP adres

Každé síťové zařízení na třetí vrstvě modelu ISO (počítač, brána, směrovač) má přiřazenu jedinečnou 32-bitovou IP adresu, která se skládá ze síťové části (netid) a identifikátoru počítače (hostid). IP adresy jsou centrálně spravovány organizací IANA (Internet Assigned Numbers Authority, www.iana.org). IP adresy přidělují regionální registrátoři (Regional Internet Registry, RIR). RIR jsou rozděleny podle světových regionů:

- AfriNIC (African Network Information Center) — Afrika (od srpna 2005)
- APNIC (Asia-Pacific Network Information Center) — Asie, Tichomoří
- ARIN (American Registry for Internet Numbers) — Severní Amerika
- LACNIC (Regional Latin-American and Caribbean IP Address Registry) — Latinská Amerika a Karibik (od listopadu 2002)
- RIPE NCC (Réseaux IP Européen) — Evropa, Blízký východ, střední Asie

Regionální registrátoři přidělují části svého adresového prostoru provozovatelům Internetového připojení (ISP, Internet service providers) v jednotlivých zemích či dalším organizacím, které přidělují získané adresy svým zákazníkům v dané lokalitě. Samotný uživatel může obvykle získat IP adresu pouze od svého ISP.

Pro Českou republiku je výpis lokálních registrátorů (LIR, Local Internet Registry) na stránce <http://www.ripe.net/membership/indices/CZ.html>. Mezi české registrátory patří např. Český Telecom, Chello, UPC, Eurotel, Tiscali či např. sdružení CZ.NIC (srpen 2006).

Určitý rozsah IP adres je rezervován pro speciální použití mimo Internet – patří tam např. privátní adresy definované v RFC 1918 [30] a RFC 5735 [5]. Žádná z těchto adres se nesmí objevit na vnějším rozhraní lokální sítě. V tabulce 1.3 jsou uvedeny lokální privátní adresy, které se používají pro adresování v lokálních sítích. Mezi další rezervované adresy patří například adres 127.0.0.0/8, 169.254.0.0/16, 240.0.0.0/4, apod.

Počáteční adresa	Koncová adresa	Prefix
10.0.0.0	10.255.255.255	10/8
172.16.0.0	172.31.255.255	172.16/12
192.168.0.0	192.168.255.255	192.168/16

Tabulka 1.3: Hodnoty privátních adres

Během vývoje Internetu a nárůstu počtu uživatelů Internetu bylo nutné postupně revidovat existující způsoby přidělování IP adres tak, aby se nevyčerpal adresový prostor. K vyčerpání prostoru IPv4 nakonec došlo 3. února 2011, kdy IANA přidělila poslední bloky /8 všem pěti regionálním registrátorům. Tito registrátoři ještě mohou rozdělit tyto bloky na podbloky (např. /20, /24) a přidělit je žadatelům ve svém regionu. Další IPv4 adresy však už

nemohou získat. Podle statistiky na serveru www.potaroo.net/tools/ipv4 spravují RIR následující kapacitu adresového prostoru: AfriNIC (5%), APNIC (21,5%), ARIN (32.9%), RIPE (19,2%) a LACNIC (7.8%). Zbýlých cca 13.6% adres spravuje IANA a tvoří je adresy třídy D,E a speciální adresy.

Jak se ukázalo během historie Internetu, původní rozdělení IP adres do čtyř tříd A,B,C,D se ukázalo jako neefektivní. Vznikly proto nové postupy jak, efektivně využívat IP adresy pro připojená zařízení – tzv. schémata přidělování IP adres.

1.3.3 Schémata přidělování IP adres

Mezi schémata pro přidělování IP adres patří [10]:

1. rozdělení IP adres do tříd,
2. vytváření podsítí (subnetting),
3. beztrždní adresování (CIDR),
4. překlad adres NAT a
5. nástup protokolu IPv6.

V této kapitole se podíváme na první čtyři způsoby, protokol IPv6 bude popsán na jiném místě.

1.3.3.1 Rozdělení IP adres do tříd

32-bitová IP adresa se pro lepší čtení zapisuje jako čtyři byty oddělené tečkou. Skládá se z části, která identifikuje síť (netid), na které se počítač nachází a z části identifikující počítač (hostid). Podle toho, kolik bitů se použije na adresu sítě a kolik na adresu počítače, rozdělujeme adresy do tří základních skupiny - viz tabulka 1.4. Maska sítě udává počet bitů, které jsou použity pro část *netid* a pro část *hostid*. U tohoto schématu je maska pevně daná pro každou pro třídu, např. 255.0.0.0 pro třídu A.

Pro rychlejší určení třídy (délky adresy sítě) se používá vyjádření sítě ve tvaru *IP adresa/prefix sítě*. Například pro IP adresu 147.229.12.97 s maskou 255.255.0.0 se použije vyjádření 147.229.12.97/16. Číslo 16 (prefix sítě) označuje počet jedničkových bitů v masce sítě.

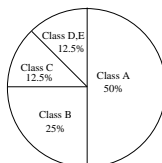
Třída adres	Začátek adresy		Maska podsítě	Prefix sítě
Třída A	0	1-126	11111111 00000000 00000000 00000000	/8=255.0.0.0
Třída B	10	128-191	11111111 11111111 00000000 00000000	/16=255.255.0.0
Třída C	110	192-223	11111111 11111111 11111111 00000000	/24=255.255.255.0
Třída D	1110	224 - 239	multicast	
Třída E	1111	240 - 254	rezervováno pro experimentální účely	

Tabulka 1.4: Třídy adres IPv4

Použití tříd je zřejmé. Rozsáhlé sítě např. ISP dostaly přiděleny adresy typu A, kde je možné využít až $2^{24} - 2$ IP adres, tj. přes 16 mil. adres. Takových sítí je možné vytvořit pouze $2^7 - 2 = 126$. Sítě s adresami třídy B mohou adresovat až $2^{16} - 2 = 65\,534$ počítačů. Těchto sítí lze vytvořit $2^{14} - 2$, tj. 16 382. Nejvíce sítí, $2^{21} - 2$ (přes 2 mil. adres), lze vytvořit ve

třídě C. Do sítě typu C lze připojit pouze $2^8 - 2$ počítačů, tj. 254, což je z dnešního pohledu nedostatečné. Malé firmy tento rozsah nevyužily plně, větším firmám nestačil a buď musely mít více adres typu C nebo požádat o adresy typu B. V dubnu 1992 bylo přiděleno 38% adres typu A (49), 45% adres typu B (7354) a 2% adres typu C (44014)[31]. Z tohoto výčtu je vidět neefektivní využití adresového prostoru.

Rozdělení IP adres verze 4 do tříd není rovnoměrné - adresy tříd A a B zabírají 75% adresového prostoru, viz obrázek 1.15.



Obrázek 1.15: Rozdělení adres IPv4 do tříd

Určité IP adresy daného rozsahu jsou rezervované a nelze je volně přiřadit síťovým zařízením. Tyto adresy jsou uvedeny v tabulce 1.5. Pokud uvažujeme například síť 195.180.11.0/24,

Použití	Typ adresy	Příklad
Adresa sítě	bity <i>hostid</i> jsou nulové	195.180.11.0 (síť třídy C)
Broadcastová adresa na síti	bity <i>hostid</i> jsou jedničkové	147.229.255.255 (pro síť třídy B)
Stejná síť	bity <i>netid</i> jsou nulové	0.0.0.12 (pro síť třídy C)
Broadcastová adresa na LAN	bity adresy jsou jedničkové	255.255.255.255 (DHCP výzva)

Tabulka 1.5: Speciální použití některých IP adres

pak lze v této síti adresovat 254 počítačů adresami v rozsahu 195.180.11.1-195.180.11.254. Kromě privátních IP adres a adres třídy D a E definují dokumenty RFC ještě další vyhrazené adresy - viz tabulka 1.6 [11].

Složení IP adresy ze dvou částí (adresa sítě a adresa počítače) vytváří dvouúrovňové (též hierarchické) adresování. V první části zjišťujeme, kde se nachází daná síť a přeposíláme paket do této sítě.

V druhé fázi je paket doručen cílovému počítači v dané síti. Tento dvouúrovňový model zpracování IP adres se využívá pro směrování. Hierarchické uspořádání totiž říká, že daný uzel v síti nemusí znát cestu ke všem počítačům. Stačí znát adresy k sítím (nikoliv konkrétním počítačům) a doručit datagram na síťový prvek v dané síti, který zná cílový počítač. Tento hraniční uzel (obvykle směrovač) zajistí doručení v lokální síti.

Směrnice pro rozdělování IP adres mezi regiony jsou popsány v RFC 1466[8] a RFC 2050[20]. Aktuální informace lze najít v databázi [v databázi IANA](#).

Adresy třídy D jsou rezervovány pro skupinové vysílání (multicast). Zde mluvíme o multicastovém vysílání na třetí vrstvě, tj. v rámci IP sítě. Pro posílání multicastového provozu do sítě LAN se používají multicastové MAC adresy, které se vytváří z multicastové skupiny na IP vrstvě. O nich budeme hovořit později.

Při multicastovém vysílání přijímají data všechny počítače, které jsou připojeny do dané multicastové skupiny, tj. mají nastavenou multicastovou adresu. Oproti klasickému přenosu

Blok adres	Použití	Odkaz
0.0.0.0/8	"This" Network	RFC 1700, page 4
10.0.0.0/8	Private-Use Networks	RFC 1918
14.0.0.0/8	Public-Data Networks	RFC 1700, page 181
24.0.0.0/8	Cable Television Networks	
39.0.0.0/8	Reserved but subject to allocation	RFC 1797
127.0.0.0/8	Loopback	RFC 1700, page 5
128.0.0.0/16	Reserved but subject to allocation	
169.254.0.0/16	Link Local	
172.16.0.0/12	Private-Use Networks	RFC 1918
191.255.0.0/16	Reserved but subject to allocation	
192.0.0.0/24	Reserved but subject to allocation	
192.0.2.0/24	Test-Net	
192.88.99.0/24	6to4 Relay Anycast	RFC 3068
192.168.0.0/16	Private-Use Networks	RFC 1918
198.18.0.0/15	Network Interconnect Device Benchmark Testing	RFC 2544
223.255.255.0/24	Reserved but subject to allocation	
224.0.0.0/4	Multicast	RFC 3171
240.0.0.0/4	Reserved for Future Use	RFC 1700, page 4

Tabulka 1.6: Souhrn speciálních adres IPv4, RFC 3330

typu unicast se stejná data nezasílají skupině cílových počítačů opakovaně, ale jsou zaslána všem najednou. Multicastového přenosu se využívá např. pro zaslání směrovacích informací (protokoly RIPv2, OSPF), synchronizace času na síťových zařízeních (služba NTP, Network Time Protocol), rezervaci zdrojů (protokol RSVP) či pro přenos multimediálních dat (streaming, Video on Demand, apod).

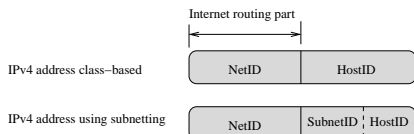
Rozsah adres třídy D je 224.0.0.0 – 239.255.255.255. První čtyři bity adresy třídy D definují typ třídy (třída D má prefix 1110), dalších 28 bitů identifikuje multicastovou skupinu. Podrobnější rozdělení adres uvedené v tabulce 2.8 v kapitole 2.9 a je definováno normou RFC 2365 [7]. Např. skupina 224.0.0.0 – 224.0.0.255 je rezervována pro využití směrovacími protokoly a dalšími služebními protokoly. Speciálními multicastovými adresami je například 224.0.0.1, která je přiřazena všem zařízením na síti (počítače, směrovače, tiskárny), adresa 224.0.0.2 adresuje všechny směrovače v síti. Registraci multicastových adres provádí IANA. Aktuální seznam zaregistrovaných multicastových skupin lze najít na <http://www.iana.org/assignments/multicast-addresses>. Pro lokální použití doporučuje IETF (Internet Engineering Task Force) použít adresy z rozsahu 232.0.1.0 – 232.255.255.255.

1.3.3.2 Vytváření podsítí (subnetting)

Vytváření podsítí je dalším typem schématu pro správu IP adres. Hlavní motivací vytváření podsítí bylo efektivní využití adresového prostoru. Narozdíl od předchozího schématu, kdy maska sítě závisí na třídě adresy (například třída C má délku masku /24), umožňuje subnetting rozdělit část adresy určenou pro počítače (dle typu třídy) ještě na adresu podsítě (subnet) a adresu počítače v této síti libovolně podle aktuálních potřeb, viz obr. 1.16. Např. lokální síť o velikost dvaceti počítačů by potřebovala podle schématu (1) adresu třídy C (která umožňuje adresovat až 254 počítačů), zatímco pro adresování počítačů této sítě stačí pouze pět bitů, které umožňují adresovat až třicet počítačů. Zatímco v prvním případě zůstalo nevyužito 234

adres, v případě použití subnettingu by to bylo pouze 12.

Dělení do podsítí umožňuje vytvářet další sítě v rámci LAN. Tento přístup má význam především u větších sítí, které si administrátor rozdělí přidělený adresový prostor na menší části. Tyto podsítě lze pak lépe spravovat, sledovat jejich využití a také jasněji definovat stanice, které jsou do nich připojeny. Např. školní síť lze rozdělit na podsítě využívanou studenty, zaměstnaneckou podsít, podsít obsahující servery a tiskárny apod. Pro jednotlivé podsítě pak můžeme nastavit např. různé oprávnění přístupy do Internetu apod.



Obrázek 1.16: Adresování podsítí

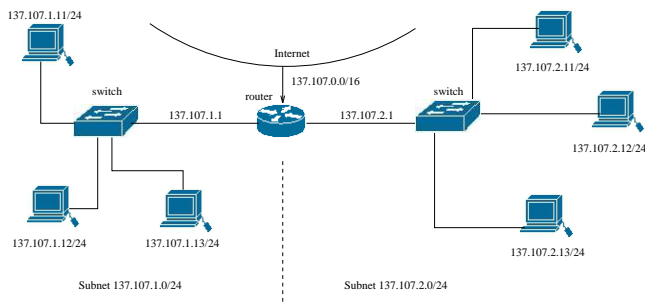
Při vytváření podsítí měníme masku sítě tak, že si vypůjčíme bity s části **hostid** a označíme je jako adresa podsítě, viz obr. 1.16. Minimálně si můžeme vypůjčit jeden bit, pomoci něhož lze vytvořit dvě nové podsítě. Maximálně si můžeme vzít *hostid* – 2 bitů, protože vždy potřebuje alespoň dva bity na adresu počítačů v podsíti. Například pro adresy sítě 137.107.0.0 (třída B) lze použít prvních osm bitů části **hostid** (tzn. třetí byte adresy) na adresování podsítě. Nové podsítě budou mít adresu 137.107.1.0/24, 137.107.2.0/24 apod. Těchto podsítí lze vytvořit až 254 ($2^8 - 2$). V každé podsíti může být 254 počítačů (obr. 1.17). Pokud bychom potřebovali více než 254 počítačů v podsíti, vypůjčíme si z části **hostid** méně bitů, např. tři. Se třemi bity můžeme vytvořit osm podsítí a v každé podsíti mít až 8 190 počítačů. Adresa jedné z podsítí by mohla být např. 137.107.33.0/19. Principy vytváření podsítí v Internetu jsou popsány ve standardu RFC 950 [18].

Při použití této metody se nemění délka adresy sítě **netid**, což je důležité pro směrování. Podle této adresy určí směrovače, kam se datagram přepoše. Poslední na cestě směrovač předá datagram do lokální sítě, kde se doručí cílové stanici. Metoda vytváření podsítí zachovává třídy adres i směrování podle tříd, pouze **netid** se už nevztahuje k jedné síti LAN, ale spíše k lokalitě (např. firma, universita apod.). Tato lokalita obvykle obsahuje více podsítí LAN vytvořených pomocí techniky subnetting.

Rozdělování počítačů do podsítí se nejvíce používá u adresy třídy A a B, kde počet počítačů v podsíti je 16 mil. či 65.534, což je mnohem více než obvyklé potřeby organizací. Příklad vytvoření podsítě je na obrázku 1.17. Na tomto obrázku komunikují počítače podsítí mezi sebou. Pokud chtějí něco poslat druhému počítači, posílají zprávu na implicitní bránu (default gateway) 131.107.1.1, což je směrovač, který předá zprávu do dané podsítě. Z pohledu Internetu se počítače tváří, jako by byly v jedné podsíti. Směrovač však rozlišuje jednotlivé podsítě a přeposílá data tam, kde jsou potřeba.

Výhody použití rozdělování do podsítí:

- logické či geografické dělení sítí (např. jednotlivá oddělení firmy mají vlastní podsít)
- snížení zátěže na síti - omezení přenosů typu broadcast (např. při dotazech ARP).
- bezpečnost při oddělení podsítí



Obrázek 1.17: Vytvoření podsítě pro adresy třídy B

- jednodušší administrace

Pokud rozdělují adresový prostor, který mám přidělený (např. 147.229.0.0/16) na podsítě, nemusí mít každá podsít stejný počet počítačů. Lze vytvořit více podsít, které se liší i délkou masky. Tato technice se říká *vytváření podsít s proměnnou délkou masky*, VLSM (Variable Length Subnet Masking). Princip této techniky je demonstrován na následujícím příkladu.

Příklad vytvoření sítě:

Zadání: Navrhněte rozdělení počítačů v lokální síti s adresou 192.168.1.0 do sedmi podsít tak, aby první podsít A obsahovala 21 počítačů, druhá síť B 9 počítačů, třetí síť C 29 počítačů, čtvrtá síť D dva počítače, pátá síť E 47 počítačů, šestá síť F 10 počítačů a sedmá síť G 5 počítačů. Využijte co nejefektivněji adresový prostor.

Řešení:

1. Pro zadané podsítě určím počet potřebných bitů pro adresování v daném podsíti (zarovnání adres) - viz následující tabulka. Při zarovnání je potřeba si uvědomit, že každá jednotlivá podsít obsahuje kromě adres počítačů také dvě další adresy - adresu podsítě a broadcastovou adresu.

Název podsítě	A	B	C	D	E	F	G
Počet počítačů	21	9	29	2	47	10	5
Zarovnání	32	16	32	4	64	16	8

2. Vytvořím tabulku reprezentující graficky rozdělovaný adresový prostor. Tabulka bude obsahovat okna odpovídající rozsahům přidělovaných adres jednotlivým podsítím. Začátek rozsahu (adresa podsítě) bude napsán v levém horním rohu okna, konec adres podsítě (adresa pro broadcast) je v pravém dolním rohu.

Při rozdělování postupuji následovně:

- (a) Vezmu největší požadovanou podsít a podívám se, jestli tabulka adresového prostoru obsahuje okno této velikosti. Pokud ano, alokuji toto okno pro danou podsít.

- (b) Pokud je potřebný alokovaný prostor menší než existující okna, vyberu nejbližší větší volné okno a rozdělím ho na polovinu. Pokud nové okno odpovídá požadované velikosti podsítě, alokuji prostor. V opačném případě opakuji půlení okna tak dlouho, než dostanu požadovaný prostor.
- (c) Vezmu další podsítě podle velikosti a podle výše uvedeného algoritmu pro ně alokuji okna v tabulce adresového prostoru.

0	16	20	128
	19	D	
	15	24	G
		31	
32	48		E
F	47	B	191
		63	
64			192
			A
			223
			224
			C
			255
		127	

3. Získané rozdělení adresového prostoru přepíšu do tabulky podsítí s příslušnými IP adresami a délkou síťové masky:

Název podsítě	A	B	C	D
Počet počítačů	21	9	29	2
Adresa nové podsítě	192.168.1.192/27	192.168.1.48/28	192.168.1.224/27	192.168.1.20/30
Adresa broadcastu	192.168.1.223/27	192.168.1.63/28	192.168.1.255/27	192.168.1.23/30
Začátek adres.prostoru	192.168.1.193/27	192.168.1.49/28	192.168.1.225/27	192.168.1.21/30
Konec adres.prostoru	192.168.1.2228/27	192.168.1.62/28	192.168.1.254/27	192.168.1.22/30
Název podsítě	E	F	G	
Počet počítačů	47	10	5	
Adresa nové podsítě	192.168.1.128/26	192.168.1.32/28	192.168.1.24/29	
Adresa broadcastu	192.168.1.191/26	192.168.1.47/28	192.168.1.31/29	
Začátek adres.prostoru	192.168.1.128/26	192.168.1.31/28	192.168.1.25/29	
Konec adres.prostoru	192.168.1.190/26	192.168.1.46/28	192.168.1.30/29	

1.3.3.3 Beztrídní adresování (CIDR)

Další metodou jak plně využít adresový prostor, je technika beztrídního adresování, která se objevila v polovině 90.let a je definovaná v RFC 1519 [29]. Pomocí beztrídního adresování může mít síťová část IP adresy libovolný počet bitů. Formát beztrídní adresy je $w.x.y.z/n$, kde n udává počet bitů síťové části IP adresy. Pokud firma požaduje např. blok adres pro tisíc počítačů, bude mít adresu typu $w.x.y.z/22$. Posledními deseti bity adresy totiž můžeme adresovat $2^{10} = 1024$ počítačů, čímž efektivně využijeme adresový prostor. V případě třídního adresování by organizace dostala adresu typu B, kde je možné mít $2^{16} = 65536$ počítačů. Což je příliš, když máme maximálně tisíc počítačů. Zbylé adresy v síti se nevyužijí. Systém tříd adres jemnější dělení adres neumožňoval.

Při použití beztrídního adresování dochází k efektivnímu využití adresového prostoru, na druhou stranu nám tento přístup komplikuje směrování. Využívá se proto tak zvané beztrídní směrování (CIDR, Classless Inter-domain Routing) [29], podle něhož se nazývá i tato metoda.

Postup vytváření je podobný jako u podsítí. Zatímco u podsítí rozdělujeme adresový prostor adres třídy A a B na podsítě, zde spojujeme sousední bloky adres třídy C a vytváříme síť umožňující adresovat více jak 254 počítačů. U podsítí se rozšiřuje maska sítě na úkor počtu počítačů. Zde se naopak zmenšuje maska podsítě a vytváří se nová síť s větším počtem počítačů. Vznikají tzv. *supersítě* (*supernets*).

Princip vytváření supersítě je ukázán na obrázku 1.18. Nejprve se vezme blok sousedních IP adres třídy C a hledá se část **netid**, která je společná pro oba bloky. V našem případě se použije maska /21 místo implicitní hodnoty /24. Vznikne nám tak supersíť 207.46.168.0/21.

		Supernet ID (21 bits)	Host ID (11 bits)
one network ID	207.46.168.0	11001111 00101110 1010	1000 00000000
	207.46.169.0	11001111 00101110 1010	1001 00000000
	207.46.170.0	11001111 00101110 1010	1010 00000000
	207.46.171.0	11001111 00101110 1010	1011 00000000
	207.46.172.0	11001111 00101110 1010	1100 00000000
	207.46.173.0	11001111 00101110 1010	1101 00000000
	207.46.174.0	11001111 00101110 1010	1110 00000000
Subnet mask	207.46.175.0	11001111 00101110 1010	1111 00000000
	255.255.248.0	11111111 11111111 1111	0000 00000000

Obrázek 1.18: Rozdělení bloku adres C do supersítě

Tuto síť lze rozdělit do několika podsítí různé velikosti. Například potřebujeme mít v jedné logické jednotce sto počítačů, v jiné padesát, v další dvacet. Toto nelze uspokojivě rozdělit pro pevnou velikost masky podsítě - museli bychom pro každou podsíť využít masku /25, tj. 126 počítačů na podsíť.

Jak už jsme se zmínili, vytváření supersítí vyžaduje složitější způsob směrování. Proč? V případě klasického třídního adresování mohl každý směrovač jednoduchým způsobem získat adresu sítě z cílové IP adresy. Stačilo podle prvních tří bitů cílové adresy určit třídu adresy a pak vyhledat ve směrovací tabulce cílové rozhraní pro danou síť.

U techniky CIDR musí směrovače uchovávat pro každou síť také masku. Maska se přenáší i ve směrovacích protokolech, které slouží ke zjišťování nových sítí a hledání optimální cesty k cílové síti. Některé starší směrovací protokoly, např. RIP verze 1 či IGRP, techniku CIDR nepodporují a proto je nelze použít v sítích, kde je beztřídní adresování použito.

1.3.3.4 Překlad adres (NAT)

Network Address Translation (NAT) [9] je způsob mapování a překladu IP adres z jedné skupiny IP adres na jinou skupinu adres. Tento překlad umožňuje např. skrýt adresy firemní sítě při použití privátních adres. Překlad adres může být typu 1:1 (jedna adresa vnitřní sítě se mapuje na předem danou adresu vnější sítě), typu M:N (M adres vnitřní sítě se mapuje na N adres vnější sítě, obvykle $M \leq N$), případně N:1 (N adres vnitřní sítě se mapuje na jednu adresu vnější sítě)[22]. Posledně jmenovanému překladu se někdy říká *PAT (Port Address Translation)* nebo *NAPT (Network Address Port Translation)*, neboť hodnota portu slouží k rozlišení spojení. V praxi se často se výraz NAT (mapování více vnitřních adres na více

vnějších adres) zaměňuje s pojmem PAT (mapování více vnitřních adres na jednu vnější adresu). Překladu bez změny portu se pak říká tradiční NAT (traditional NAT, basic NAT).

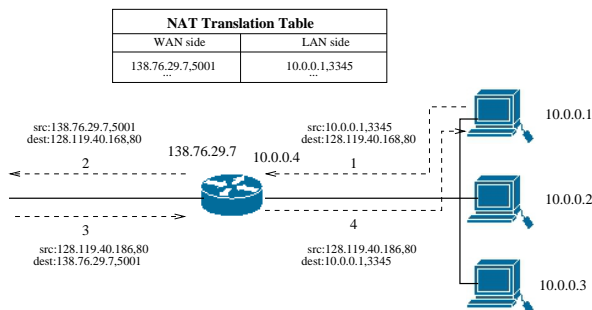
PAT šetří adresový prostor, protože připojená síť obsahuje pouze jedinou veřejnou adresu pro všechny počítače komunikující z lokální sítě ven. Pro adresování počítačů v rámci lokální sítě se využívají privátní (neveřejné) adresy, viz tabulka 1.3. Adresování PAT využívají často poskytovatelé internetového připojení pro připojení koncových uživatelů.

Pro překlad adres se využívá směrovač NAT (NAT router), který překládá adresy. Směrovač NAT může být součástí směrovače nebo můžeme použít software, např. **natd**. Mezi výhody překladu adres NAT patří:

- NAT Šetří IP adresy. Mnoho spojení v rámci organizace vyžaduje pouze interní komunikaci. Pouze malá část počítačů se v určitý okamžik připojuje na Internet.
- Redukuje útoky typu “spoofing”, kdy se útočníkův počítač tváří jako lokální. NAT používá pro vnitřní systém privátní adresy, které se nepoužívají v Internetu. Pokud se vyskytnou v Internetu, směrovač je zahodí.
- Odstraňuje potřebu změnit IP adresy počítačů, když dojde ke změně ISP.

Směrovač NAT se z pohledu vnější sítě tváří jako koncový uzel, který přijímá data na svou IP adresu. Ve skutečnosti přeposílá přijatá data do vnitřní sítě podle mapování, které má uloženo ve své překladové tabulce. Tabulka pro překlad PAT obsahuje položky ve tvaru {lokální IP adresa, lokální port, vnější IP adresa, vzdálený port}.

U každého IP datagramu přicházejícího z vnitřní sítě je adresa odesílatele přeložena na adresu směrovače NAT a je také upraven zdrojový port v hlavičce TCP či UDP. Původní hodnoty se spolu s novými hodnotami uloží do překladové tabulky. Pokud přijde odpověď z vnější sítě, určí se podle čísla portu adresáta cílový počítač. Provede se překlad cílové IP adresy a portu na hodnoty uložené v NAT tabulce. Paket se pak pošle do vnitřní sítě. Příklad použití překladu NAT je na obrázku 1.19. Počítač 10.0.0.1 chce komunikovat s WWW serverem (port 80), který je na adrese 128.119.40.168. Komunikace probíhá následujícím způsobem:



Obrázek 1.19: Překlad adres - NAT

1. Počítač 10.0.0.1 vytvoří paket a pošle ho v lokální síti na implicitní bránu (směrovač). Tento směrovač zároveň pracuje jako směrovač NAT. NAT přijme požadavek a přeloží

IP adresu odesílatele na IP adresu vnější sítě. Zároveň vloží do hlavičky paketu nové číslo portu (např. 5001). Původní číslo portu si uloží spolu s IP adresou do překladové tabulky NAT. Nové číslo portu slouží ke zpětné identifikaci odesílatele po přijetí odpovědi.

2. Přeložený paket je poslán do vnější sítě. Směrovač NAT se tváří jako odesílatel paketu. Paket obsahuje IP adresu vnějšího rozhraní sítě s novým číslem portu. Cílová IP adresa a port se při překladu nemění.
3. Odpověď od webového serveru s IP adresou 128.119.40.186 je poslána na vnější rozhraní směrovače NAT. Podle zdrojové IP adresy, zdrojového a cílového portu vyhledá NAT v překladové tabulce příslušný záznam. Podle něho nahradí cílovou IP adresu a port původní IP adresou a portem původního odesílatele dotazu. Při překladu se nemění adresa a číslo portu odesílatele.
4. Přeložený paket je přeposlán do lokální sítě. Pro počítače z vnitřní sítě i pro počítače z vnější sítě je tento postup transparentní.

Obvykle se činnost NAT spojuje s filtrováním provozu pomocí pravidel firewallu. Tato pravidla stanoví, které pakety se přijmou a budou se překládat, případně které pakety se zahodí (filtrují).

Dnes je NAT poměrně rozšířený, zejména v kombinaci s dynamickým přidělováním IP adres (DHCP). Využívají ho například poskytovatelé internetového připojení pro připojení domácností. Hlavní předností použití NAT je úspora veřejných IP adres.

Překlad adres NAT lze nastavit jako dynamický nebo statický. Statický překlad je definován předem zadanou překladovou tabulkou, která mapuje vnější IP adresy a porty na předem vybrané vnitřní adresy a porty. U dynamického překladu se překladová tabulka vytváří na základě požadavků. Většinou se využívá mapování 1:N, tj. jedna veřejná IP adresa se mapuje na více vnitřních adres. Pro rozlišení jednotlivých spojení se používá číslo portu. Použití NAT má i své nevýhody:

- NAT vyžaduje nároky na hardware či software při překladu adres a udržování překladové tabulky. Při překladu se rovněž musí přepočítávat kontrolní součet. Musí se také překládat IP adresy ve vyšších protokolech, například v hlavičce protokolu SIP, která obsahuje IP adresu odesílatele či číslo portu v protokolu FTP. Překlady se týkají každého IP datagramu, který prochází směrovačem NAT.
- Stanice, které jsou umístěny ve vnitřní síti, nelze v případě použití PAT adresovat. Lze pouze odpovědět na dotaz přicházející z vnitřní sítě LAN. To je například problém dvou stanic, které jsou v různých sítích LAN používajících PAT. Částečně lze tento problém řešit statickým překladem NAT nebo vytvořením virtuálních privátních sítí VPN.
- PAT používá čísla portů pro adresování počítačů, nikoliv procesů, což je původní význam čísel portů. Porušuje tak původní návrh NAT, který se týkal pouze překladu IP adresy [9].
- Směrovače by měly zpracovávat pouze informace do třetí úrovně modelu TCP/IP. PAT však zasahuje i do hlaviček TCP a UDP na transportní vrstvě, čímž porušuje spojení koncových uzlů (end-to-end connection). Koncové uzly mají spolu komunikovat přímo na síťové a transportní vrstvě bez toho, aniž by mezilehlé uzly upravovaly IP adresy a čísla portů.

- NAT omezuje použití kryptování a autentizace. Autentizace, která započítává původní IP hlavičku do kontrolního součtu, nebude pracovat korektně s překladem PAT.
- Současný nedostatek IP adres řeší 128-bitové IP adresy verze 6 (kapitola ??). Neustálé používání NAT brzdí rozšiřování protokolu IPv6.

Podobný překlad jako NAT provádějí i *proxy servery*. Narozdíl od překladu NAT se jejich překlad týká pouze konkrétní aplikace. Proxy servery slouží jako aplikační brány, lze je pokládat za jeden z typů firewallu. Proxy servery jsou aplikačně závislé, to znamená, že pro každý typ aplikace se používá specializovaný server (např. FTP proxy server, WWW proxy apod.). NAT překládá adresy bez ohledu na aplikace, proxy server se zaměřuje pouze na jeden typ aplikací.

Zavedením překladu NAT se ztrácí transparentnost komunikace — počítače na vnitřní síti nejsou z vnějšího Internetu adresovatelné, což znamená, že komunikace se dá zahájit pouze z vnitřní sítě ven. To přináší omezení službám, které potřebují přímou komunikaci mezi uživateli — např. ICQ, IP telefonie, videokonference apod. Nacházejí-li oba komunikující partneři v sítích za NAT, nemohou spolu navázat spojení jinak než přes třetího prostředníka, který není v síti s překladem adres. Toho využívá například aplikace Skype při vytváření spojení.

1.3.4 Dynamické přidělování adres

Manuální přidělování IP adres počítačům se s rostoucím počtem připojených stanic ukázalo jako nereálné a už v roce 1985 se objevuje první způsob načítání IP adres a konfigurace počítače ze vzdáleného řídicího serveru. Využívá přitom protokol Bootstrap (BOOTP)[2], který slouží k načtení konfigurace při startu počítače bez nutnosti ručního nastavení administrátorem.

Protokol BOOTP přenáší IP adresu počítače, IP adresu serveru BOOTP a jméno souboru, který se načte do paměti a spustí. Protokol používá pro svůj běh dva porty — port 67 pro server a port 68 pro klienta. Protože při spuštění nezná počítač svou IP adresu ani IP adresu serveru, ze kterého má načíst konfiguraci, posílá dotazy všesměrovým vysíláním typu broadcast. Pro přidělování adres definuje služba BOOTP statické mapování hardwarových adres počítačů na IP adresy, tj. každá stanice dostane vždy stejnou IP adresu.

Nástupcem protokolu BOOTP je protokol DHCP (Dynamic Host Configuration Protocol) [23], který rozšiřuje možnosti dynamické konfigurace. Umožňuje přidělovat IP adresy, aniž by bylo nutné definovat statické mapování hardwarové adresy na IP adresu. Stanice si mohou načíst adresu DNS serveru, WINS serveru a další konfigurační parametry. DHCP umožňuje přidělovat adresy na omezenou dobu (lease time). Po ukončení spojení umí stejnou IP adresu přidělit dalšímu zájemci. Protokol je zpětně kompatibilní s protokolem BOOTP.

Komunikace

Protokol DHCP pracuje na principu klient-server. Klientem je počítač, který se připojuje do sítě a žádá informace o IP adresu a další konfigurační parametry. Server je program, který spravuje databázi dostupných IP adres a dalších konfiguračních parametrů pro danou síť, odpovídá na dotazy klientů a posílá jim požadovaná data. Komunikaci dynamického přidělování adres lze rozdělit na čtyři kroky podle posílaných příkazů:

1. DHCP discover

Klient DHCP posílá první dotaz na port 67 formou broadcastu. Jako zdrojovou IP adresu uvede adresu 0.0.0.0. Dotaz obsahuje identifikaci zprávy, podle které klient rozpozná příslušnou odpověď od serveru.

2. DHCP offer

Zpráva **DHCP offer** je odpovědí serveru na zprávu **DHCP discover**. Pokud je na síti více běžících serverů, klient si vybere jednu z došlých odpovědí. Nemusí to být ta první, záleží na konfiguraci klienta. Zpráva od serveru obsahuje nabízenou IP adresu, síťovou masku, dobu zapůjčení adresy a další parametry. Pokud se server rozhodne adresu nepřidělit – například je adresa vázána na pevnou hardwarovou adresu nebo nemá žádné volné IP adresy – zamítne žádost zprávou **DHCP NAK**.

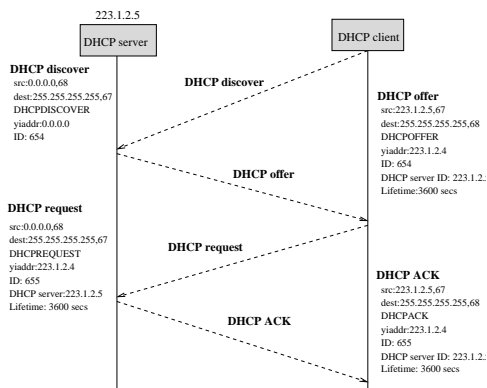
3. DHCP request

Klient na nabídku odpoví příkazem **DHCP request**, který opět posílá broadcastem, aby ho slyšely všechny DHCP servery. Server, který zprávu poslal, volbu potvrdí. Ostatní DHCP servery zruší své nabídky. Příkaz **DHCP request** používá klient také v případě, že došlo k expiraci doby zapůjčení IP adresy. Server pak znovupřidělí stejnou IP adresu na nový časový interval.

4. DHCP ACK

Touto zprávou server potvrzuje přijetí volby a přidělení IP adresy na danou dobu.

Kromě těchto základních příkazů definuje protokol DHCP ještě další příkazy, například **DHCP RELEASE** pro uvolnění IP adresy, **DHCPINFORM** pro poslání dalších konfiguračních informací od serveru atd. Příklad komunikace DHCP klienta se serverem je na obrázku 1.20.



Obrázek 1.20: Příklad komunikace DHCP

Nastavení serveru

Na unixových systémech se používá server **dhcpcd**, jehož konfigurace je uložena v souboru `/etc/dhcpcd.conf`. Konfigurační soubor DHCP obsahuje deklarace, které definují topologii sítě (shared-network), podsítě (subnet), skupin (group) a počítačů (host). Tyto deklarace se mohou objevovat v souboru opakovaně. Struktura deklarací je hierarchická, síť obsahuje podsítě, podsítě obsahují uzly. S každým příkazem (deklarací) mohou být spojeny parametry a volitelná nastavení. Parametry definují vlastnosti serveru a protokol, volitelná nastavení určují konfiguraci klienta. Základní typy deklarací jsou:

group – vytváří skupinu sdílených sítí, podsítí a počítačů, pro něž se použijí zadané parametry a nastavení.

shared-network – příkaz se použije pouze tehdy, pokud více IP podsítí sdílí jednu fyzickou síť.

subnet – definuje IP adresu a adresu masky každé podsítě DHCP serveru. Každá síť, která je fyzicky připojená k serveru, musí obsahovat deklaraci **subnet**.

host – definuje parametry a volitelná nastavení pro jednotlivé klienty. Každý klient BOOTP obsahovat tuto deklaraci. Pro klienty DHCP je tato položka volitelná.

Server může pro jednotlivé sekce využívat následující konfigurační parametry:

range - rozsah IP adres, které jsou k dispozici pro dynamické přidělování.

default-lease-time - implicitní doba přidělení adresy, pokud klient nežádá jinou dobu.

max-lease-time - maximální doba přidělení adresy, bez ohledu na požadavek klienta.

get-lease-hostname - tento parametr říká DHCP serveru, že má poskytovat každému klientovi i doménovou adresu získanou z DNS.

hardware - definuje klientovu fyzickou adresu.

server-name - doménová adresa DHCP serveru, kterou vrátí klientovi

fixed-addresses - přiřadí jednu či více IP adres počítači.

allow, deny - určuje, zda server odpoví či nikoliv na určité typy dotazů.

Příklad konfigurace DHCP serveru, který přiděluje IP adresy v síti 10.10.10.0/24, je na následujícím obrázku:

```
subnet 10.10.10.0 netmask 255.255.255.0 {
    option routers 10.10.10.1;
    option ip-forwarding off;
    option domain-name "fit.vutbr.cz";
    option domain-name-servers 10.10.10.1;
    pool {
        max-lease-time 1800;
        range 10.10.10.101 10.10.10.109;
        allow unknown clients;
    }
}
```

Nastavení klienta

Nastavení klienta `dhclient` v systému je uloženo v souboru `/etc/dhclient.conf`. Tento soubor může být i prázdný, pokud není potřeba speciální nastavení. V souboru lze nastavit následující volby:

timeout – maximální délka čekání na odpověď od serveru, implicitně je nastaveno 60 sekund.

Pokud klient odpověď nedostane, po uplynutí času pošle opakovaný dotaz.

retry – definuje dobu, po které se klient znovu dotazuje DHCP serveru. Implicitní hodnota je pět minut.

reboot – pokud je klient restartován, nejprve žádá adresu, kterou už měl (stav INIT-REBOOT).

Pokud je připojen ke stejné síti, je to nejrychlejší způsob, jak dostat adresu. Příkaz **reboot** stanoví čas, který musí uplynout od doby, kdy klient poprvé žádal poslední adresu. Po uplynutí této doby vyše klient žádost o novou IP adresu. Implicitní hodnota je deset sekund.

request – tento příkaz definuje, jaké informace požaduje klient po serveru. Může to být např. maska podsítě, adresa všesměrového vysílání, směrovače, doménové jméno a další.

require – tato volba určuje nastavení, která jsou požadována po serveru, aby žádost byla klientem akceptována. Odpovědi od serveru bez uvedených hodnot klient odmítne.

reject – zde se specifikují servery, od kterých má klient odmítnout odpovědi.

Příklad konfigurace DHCP klienta:

```
timeout 60;
retry 60;
reboot 10;
select-timeout 5;
initial-interval 2;
reject 192.33.137.209;
```

1.4 Konfigurace a testování síťového připojení

V této části se podrobněji podíváme na nastavení síťového spojení na počítačích. Ukážeme si způsoby, jak otestovat spolehlivost síťového spojení a jak postupovat při hledání chyb (angl. troubleshooting). Uvedené příklady konfigurace a testování závisí na operačním systému počítače. V tomto textu uvedeme příklad operačního systému Windows XP a unixových systémů FreeBSD a Linux, které lze považovat za typické zástupce současných nejvíce rozšířených operačních systémů.

1.4.1 Instalace a konfigurace TCP/IP

Jak jsme si ukázali v kapitole 1.2.2.2, str. 10, jsou služby síťové a transportní vrstvy součástí operačního systému (OS). Obvykle nejsou implementovány přímo v jádře OS, ale tvoří je knihovny a moduly, které jsou přidány k operačnímu systému. Při instalaci OS je obvyklé, že se tyto služby nainstalují automaticky s OS. Úkolem uživatele a administrátora je pouze nastavit vlastnosti síťového připojení, zejména IP adresu, síťovou masku, bránu, abychom mohli komunikovat s počítačem po síti.

Konfigurace TCP/IP v systému FreeBSD

Unixové systémy byly již v roce 1978 vybaveny základním komunikačním rozhraním s názvem UUCP (Unix to Unix copy) [26]. Tento systém sloužil pro jednoduché spojení dvou počítačů a výměnu dat. V roce 1980 vzniká první implementace síťových komunikačních protokolů TCP/IP pro systémy 4.xBSD. Implementace TCP/IP se od té doby stává nedílnou součástí unixových operačních systémů. Tato konfigurace zahrnuje následující činnosti.

1. Konfigurace jádra.

Konfigurace jádra OS je spíše úkolem systémového administrátora než správce sítě. Některá nastavení pro úspěšnou komunikaci v systému FreeBSD je však potřeba provést na úrovni jádra OS. Konfigurační soubor jádra u FreeBSD je v adresáři `/usr/src/sys/i386/conf/`. Implicitní konfigurace je v souboru `GENERIC`. Pro úspěšné zavedení síťových služeb je třeba mít povolené alespoň volby `INET` a `INET6` (pro IPv6):

```
options INET # InterNETworking
options INET6 # IPv6 communications protocols
...
options NFSCLIENT # Network Filesystem Client
options NFSSERVER # Network Filesystem Server
```

Při změně voleb v konfiguraci jádra (např. zapnutí NFS klienta) je potřeba znovu jádro přeložit.

2. Dynamicky linkované moduly jádra.

Používání dynamicky linkovaných modulů je pružnější než konfigurace jádra. Jádro OS může načíst podporu pro nový hardware, jakmile byl hardware detekován. Pro výpis nainstalovaných modulů slouží příkaz `kldstat [-v]`. Instalace nového modulu (např. ovladače síťové karty) se provádí příkazem `kldload`. Pokud chceme modul načíst při bootování operačního systému, je možné nastavení trvale uložit do souboru `/boot/loader.conf`.

```
kldstat [-v] # dynamicky nahrané moduly, (volba -v vypíše moduly v jádře OS)
```

Id	Refs	Address	Size	Name
1	7	0xc0400000	3939e4	kernel
2	1	0xc0794000	45a8	snd_ich.ko
3	2	0xc0799000	1d768	snd_pcm.ko
4	1	0xc07b7000	1aac50	nvidia.ko
5	1	0xc0962000	4a458	acpi.ko

3. Připojení k síti.

Pro úspěšnou konfiguraci síťového připojení počítače musíme zadat následující informace:

- jméno síťového rozhraní (na zařízení může být více síťových rozhraní)
- IP adresu síťového rozhraní (IP address)
- síťovou masku (netmask)
- adresu výchozí brány (default gateway)

- adresu DNS serveru

Jméno síťového rozhraní (např. `fxp0`) je symbolický název, který popisuje rozhraní, které nastavujeme. IP adresa a síťová maska identifikují počítač připojený do sítě. Adresa výchozí brány určuje, na který počítač (síťové zařízení) se budou směřovat IP datagramy, které nepatří do lokální sítě. Bez zadání výchozí brány můžeme komunikovat pouze v rámci lokální sítě, přesněji řečeno adresového prostoru, do kterého patří naše IP adresa. Pokud správně nastavíme adresu DNS serveru, můžeme komunikovat nejen pomocí IP adres, ale i doménových jmen.

Nastavení konfigurace lze provést ručně příkazy `ifconfig` a `route`, např.

```
ifconfig fxp0 inet 147.229.10.12 netmask 255.255.254.0
route add default 147.229.10.4
```

Toto nastavení není perzistentní (stálé) a po restartu počítače se zahodí. Běžnější je uložit síťovou konfiguraci do souboru `/etc/rc.conf`:

```
hostname="pcmatousek.fit.vutbr.cz"
ifconfig_fxp0="inet 147.229.10.12 netmask 255.255.254.0"
# ifconfig_fxp0="DHCP" -- v případě načtení adresy z DHCP serveru
defaultrouter="147.229.10.4"
inetd_enable="YES"
ipv6_enable="YES"
```

4. Spuštění aplikačních služeb.

Důležitým úkolem při konfiguraci počítače připojeného do sítě, je nastavení a spuštění aplikačních služeb, např. webového serveru **Apache**, poštovního serveru **sendmail**, FTP serveru **ftpd** apod. Existují dva základní způsoby, jak aplikační procesy spustit:

- (a) *Spuštění aplikace během bootování systému z inicializačního skriptu.*

Síťové aplikace (např. webový server, poštovní server, služba FTP apod.) je možné spustit pouze jednou (např. při bootování systému) a nechat je běžet po celou dobu běhu systému. Pokud přijde požadavek na danou síťovou službu, vytvoří aplikace pomocí funkce `fork()` synovský proces, který požadavek obslouží. Po vyřízení požadavku se synovský proces ukončí, rodičovská aplikace však běží dále a čeká na další požadavky. Nastavení aplikací je uloženo v souboru `/etc/rc.local`.

- (b) *Spuštění aplikace při příchodu požadavku službou `inetd`.*

Pokud není síťová služba často využívána, je zbytečné, aby neustále běžela v paměti počítače. Výhodnější je službu spustit, až tehdy, kdy se požadavek objeví. K tomu se využívá speciální proces `inetd` (Internet Daemon), který naslouchá na zadáných portech. Pokud přijde požadavek na nějakou službu, spustí příslušnou aplikaci a předá jí příchozí požadavek.

Příkaz `inetd.enable=YES` z předchozího příkladu povoluje spuštění tohoto procesu. Seznam aplikací, které `inetd` zavolá, je definován v konfiguračním souboru `/etc/inetd.conf`. Po modifikaci této konfigurace (např. po přidání nového procesu) je potřeba zaslat běžícímu procesu `inetd` signál `SIGHUP` pro načtení konfigurace programu `inetd`.

Konfigurace TCP/IP v systému Linux

Konfigurace síťového připojení v systému Linux je velmi podobná konfiguraci v systému FreeBSD. Uvedeme si zde přehled nastavení, principy popsané v předchozím oddíle platí i pro Linux.

1. Konfigurace jádra – podobné jako FreeBSD.
2. Dynamicky linkované moduly jádra.

Pro kontrolu dynamicky načítaných modulů je možné využít příkaz `lsmod`.

```
lsmod
Module                Size  Used by
nls_iso8859_1          4256   0
isofs                  29124   0
vfat                   15488   0
fat                    48352   1 vfat
nls_base               8064   4 nls_iso8859_1,isofs,vfat,fat
i2c_sensor             4032   1 w83627hf
i2c_isa                 2240   0
i2c_core               24768   3 w83627hf,i2c_sensor,i2c_isa
processor              19232   0
button                 6704   0
nfs                    117484   7
e1000                  89028   0
nfsd                   103016  17
exportfs               7296   1 nfsd
```

Pro dynamické zavedení ovladače je možné použít příkaz `insmod <filename> [args]`. Tento příkaz nezjišťuje síťové závislosti. Pro manuální načtení ovladače síťové karty (pokud ho systém sám nedetekuje a nenačte), lze použít příkaz `modprobe <module>`. Pokud požadovaný ovladač síťové karty není součástí instalace, je nutné ho najít a přeložit se správnými knihovnami pro dané jádro. Po zkompilování je potřeba nakopírovat přeložený modul (soubor `obj`) do správného adresáře `/lib/modules` a načíst příkazem `mprobe`. Ovladače ve formátu RPM (Redhat Package Manager) nepotřebují přeložení. Všechny síťové ovladače jsou uloženy např. v adresáři `/lib/modules/<system>/kernel/drivers/net`, např.

```
ls /lib/modules/2.6.8-rc3/kernel/drivers/net/
3c59x.ko  bsd_comp.ko  e1000      ppp_deflate.ko  slhc.ko
8139too.ko  dummy.ko    mii.ko     ppp_generic.ko  tg3.ko
bonding    e100.ko     ppp_async.ko  ppp_synctty.ko  tun.ko
```

3. Připojení k síti.

Statickou konfiguraci síťového připojení u Linuxu zapisujeme do souboru `/etc/sysconfig/networkscripts/ifcfg-<interface>`. Ruční konfiguraci lze provést příkazy

```
ifconfig em0 inet 147.229.10.12 netmask 255.255.254.0
route add default gw 147.229.10.4
```

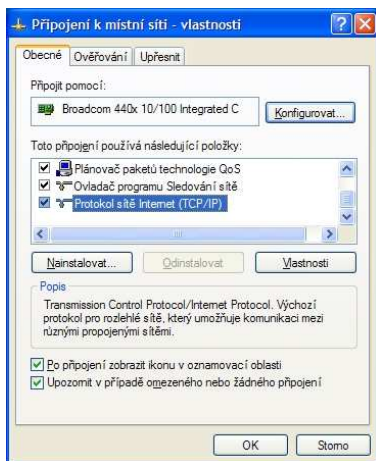
4. Spuštění aplikačních služeb.

Pro spouštění aplikačních služeb při příchodu požadavku se v Linuxu využívá program `xinetd` s konfigurací v souboru `/etc/xinetd.conf`.

Konfigurace v systému Microsoft Windows XP

1. Instalace protokolů TCP/IP.

Instalace protokolů TCP/IP se provádí v sekci Nastavení/Síťová připojení/Připojení k místní síti/Vlastnosti, kde musíme mít nainstalovaný protokol TCP/IP, viz obr. 1.21.



Obrázek 1.21: Instalace modulu pro podporu TCP/IP

2. připojení k síti

Konfigurace IP adresy, síťové masky, brány a DNS serveru se zadává ve vlastnostech TCP/IP spojení, viz obr. 1.22. Nastavení může být buď automatické (přiřazení adresy DHCP serverem) nebo manuální (statická adresa).

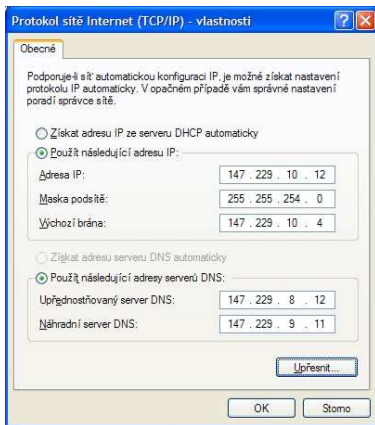
V případě, že není k dispozici DHCP server, lze nastavit alternativní konfiguraci – buď vložím statické adresy nebo automatickým získáním privátní IP adresy, kterou přiděluje operační systém.

Připojení k síti lze u Windows XP nastavit také pomocí příkazové řádky příkazem `netsh`.

```
# nastavení statické IP adresy a brány
netsh interface ip set address "připojení k místní síti" static
    147.229.10.12 255.255.254.0 147.229.10.4 1

# nastavení statické adresy DNS serveru
netsh interface ip add dns "připojení k místní síti" 147.229.8.12
netsh interface ip add dns "připojení k místní síti" 147.229.9.11

# nastavení dynamického získávání IP adresy přes DHCP
```



Obrázek 1.22: Konfigurace připojení k síti u Windows XP

```
netsh interface ip set address "připojení k místní síti" dhcp
```

```
# nastavení dynamického získávání DNS serveru přes DHCP
netsh interface ip delete dns "připojení k místní síti" all
netsh interface ip set dns "připojení k místní síti" dhcp
```

1.5 Nástroje pro zjišťování síťové konfigurace

Nezbytnou součástí konfigurace síťového připojení je zjišťování aktuálního nastavení a hledání chyb, pokud spojení nepracuje správně. V této části uvedeme základní nástroje (příkazy a aplikace), které je možné využít k odstraňování chyb. Pro správnou analýzu chybového stavu je důležité rozumět architektuře komunikace a síťových služeb. Pokud rozumíme, jak počítačové sítě a služby pracují, stačí provést jednotlivé kroky vedoucí k detekci chyb a nalezené chyby odstranit.

Kroky pro zjišťování aktuální konfiguraci a detekci chyb

- *Kontrola nastavení síťového modulu (TCP/IP stacku).*

Nejprve je potřeba ověřit, zda máme správně nainstalovaný síťový modul. Pro zjištění je možné použít příkazy z předchozí kapitoly. Tato část závisí na druhu operačním systémem a jeho konkrétním nastavení.

- *Zjištění aktuálního síťového nastavení.*

Pokud máme nějaké problémy s komunikací, je užitečné si nejprve vypsát aktuální nastavení a zjistit název komunikujícího rozhraní, IP adresu, síťovou masku, adresu implicitní brány či adresy DNS serverů. Výpis můžeme provést příkazy `ifconfig` (unixové systémy) nebo `ipconfig` (Windows).

```

ifconfig
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    inet6 fe80::20c:6eff:fe77:ce22%fxp0 prefixlen 64 scopeid 0x1
    inet 147.229.12.91 netmask 0xfffffe00 broadcast 147.229.13.255
    ether 00:0c:6e:77:ce:22
    media: Ethernet autoselect (100baseTX <full-duplex>)
    status: active

```

První hodnotou výše uvedeného výpisu je název síťového rozhraní. Informace o dostupných síťových zařízeních lze získat z manuálových stránek, např. `man 4 fxp`.

```

ipconfig /all
Konfigurace protokolu IP systému Windows
    Název hostitele . . . . . : nbisa
    Primární přípona DNS. . . . . :
    Typ uzlu . . . . . : hybridní
    Povoleno směrování IP . . . . . : Ne
    WINS Proxy povoleno . . . . . : Ne
    Prohledávací seznam přípon DNS. . : fit.vutbr.cz

Adaptér sítě Ethernet Připojení k místní síti:
    Přípona DNS podle připojení . . . : fit.vutbr.cz
    Popis . . . . . : Broadcom 440x 10/100 Controller
    Fyzická Adresa. . . . . : 00-14-38-10-56-3D
    Protokol DHCP povolen . . . . . : Ano
    Automatická konfigurace povolena : Ano
    Adresa IP . . . . . : 147.229.12.44
    Maska podsítě . . . . . : 255.255.254.0
    Výchozí brána . . . . . : 147.229.12.1
    Server DHCP . . . . . : 147.229.9.22
    Servery DNS . . . . . : 147.229.8.12
                           147.229.9.11
    Primární server WINS. . . . . : 147.229.8.12
    Zapůjčeno . . . . . : 23. srpna 2006 10:59:29
    Zápůjčka vyprší . . . . . : 23. srpna 2006 11:29:29

```

Informace o překlad fyzických adres na IP adresy můžeme zjistit výpisem ARP tabulky:

```

arp -a ... zobrazí aktuální obsah ARP tabulky
arp -d ... zruší záznam v ARP tabulce
arp -s ... přidá statický záznam do ARP tabulky
arp -N ... zobrazí ARP záznamy pro konkrétní fyzickou adresu

```

- *Způsob síťového nastavení.*

Pokud síťové nastavení není korektní, je potřeba zjistit, zda se jedná o nastavení statické (ruční konfigurace) nebo dynamické (informace získané ze serveru DHCP). Pokud očekáváme síťové nastavení z DHCP a nelze jej dostat, je potřeba zjistit, zda vůbec nějaký DHCP server na síti existuje a odpovídá. Pro sledování síťové komunikace je vhodné použití např. programu `tcpdump` (unixové systémy) nebo síťový analyzátor – např. Microsoft Network Monitor, Netdiag (Windows) či případně Wireshark (Windows, Unix).

- *Testování základní komunikace (pomocí zasílání ICMP zpráv).*

Pokud síťová komunikace nefunguje správně, je potřeba zjistit, v jakém místě nastala chyba. K testování komunikace se používají programy `ping` a `tracert` (`tracert` u Windows). Vhodné je zkontrolovat spojení v následujícím pořadí:

```
ping 127.0.0.1 - otestuje správné nastavení TCP/IP driveru
ping <local_IP> - otestuje, zda je lokální adresa správně nastavená
ping <gateway_IP> - otestuje, zda je funkční spojení s bránou
ping <147.229.8.12> - otestuje, zda je funkční spojení se vzdáleným počítačem
ping <kazi.fit.vutbr.cz> - otestuje, zda správně funguje překlad DNS
```

Program `ping` posílá na počítač ICMP datagramy typu Echo a zobrazuje došlé odpovědi (Echo Reply). Pokud nedostaneme žádnou odpověď, může být chyba v nastavení síťového připojení, filtrování datagramů ICMP nebo špatném směrování.

- *Kontrola směrování.*

Jestliže nelze vytvořit spojení do jiných sítí LAN (na počítače s IP adresou, která je různá od IP adres v lokální síti), jde pravděpodobně o chybu směrování. Pro kontrolu je potřeba vypsat směrovací tabulky příkazem `netstat -rn` (pro Unix i Windows).

Unix: `/home/matousp/test> netstat -rn`
Routing tables

```
Internet:
Destination      Gateway          Flags    Refs      Use  Netif  Expire
default          147.229.12.1    UGS      0        11578  fxp0
127.0.0.1        127.0.0.1       UH       0         4552  lo0
147.229.12/23    link#1          UC       0          0  fxp0
147.229.12.1     00:04:96:1d:34:20 UHLW     2          56  fxp0  1200
147.229.12.36    00:11:2f:63:ec:07 UHLW     1          31  fxp0   264
```

Windows: `netstat -rn`

```
=====
Seznam rozhraní
0x1 ..... MS TCP Loopback interface
0x2 ...00 14 38 10 56 3d ..... Broadcom 440x 10/100 Integrated Controller
=====
```

Aktivní směrování:

Cíl v síti	Síťová maska	Brána	Rozhraní	Metrika
0.0.0.0	0.0.0.0	147.229.12.1	147.229.12.44	20
127.0.0.0	255.0.0.0	127.0.0.1	127.0.0.1	1
147.229.12.0	255.255.254.0	147.229.12.44	147.229.12.44	20
147.229.12.44	255.255.255.255	127.0.0.1	127.0.0.1	20
147.229.255.255	255.255.255.255	147.229.12.44	147.229.12.44	20
224.0.0.0	240.0.0.0	147.229.12.44	147.229.12.44	20
255.255.255.255	255.255.255.255	147.229.12.44	147.229.12.44	1

Výchozí brána: 147.229.12.1

Trvalé trasy:

Žádné

Z výpisů můžeme vidět, na která síťová rozhraní se posílají datagramy s příslušnou cílovou IP adresou. Směrovací tabulka obvykle obsahuje i záznam obsahující adresu výchozí brány pro připojení do jiných sítí.

Mezi další prostředky pro výpis cesty patří programy `pathping`, `tracert` (Windows), `traceroute`, `tcptraceroute` (Unix).

```
/home/matousp/test> traceroute www.seznam.cz
traceroute to www.seznam.cz (212.80.76.3), 64 hops max, 40 byte packets
 1 bda-boz (147.229.12.1) 5.053 ms 0.842 ms 1.051 ms
 2 bd-kou.net.vutbr.cz (147.229.254.217) 1.071 ms 0.718 ms 0.532 ms
 3 fw-kou.net.vutbr.cz (147.229.252.33) 0.177 ms 0.264 ms 0.171 ms
 4 r43-bm.cesnet.cz (147.229.253.177) 43.958 ms 19.540 ms 9.937 ms
 5 r84-r105.cesnet.cz (195.113.156.165) 9.996 ms 9.283 ms 3.959 ms
 6 nix.tgnet.cz (194.50.100.75) 4.139 ms 13.539 ms 4.098 ms
 7 * * *
 8 * *
```

Při testování spojení službou `traceroute` je potřeba si uvědomit, že `traceroute` používá jiný typ datagramů ICMP než `ping`, tj. zprávy typu Time Exceeded Message. Tyto zprávy mohou být blokovány, tzn. ne vždy ukazuje tento program, zda je spojení funkční.

- *Testování dostupnosti služeb.*

Pokud nejsou problémy s připojením na vzdálený počítač, ale nefunguje například odesílání elektronické pošty nebo načtení webové stránky, je potřeba otestovat dostupnost služeb. Pro testování služeb nestačí se pouze připojit na cílový počítač. Často to ani nejde, protože na něm nemáme účet pro přístup. Jak tedy zjistit, zda daná služba na počítači běží? Služba je adresována číslem portu. Funkčnost služby typu TCP můžeme otestovat programem `telnet <host> <port>`.

```
home/matousp/test> telnet mail.fit.vutbr.cz smtp
Trying 147.229.7.10...
Connected to mail.fit.vutbr.cz.
Escape character is '^]'.
220 mail.fit.vutbr.cz ESMTP Sendmail 8.13.8/8.13.7; Wed, 23 Aug 2006 13:28:04
+0200 (CEST)
```

Pokud služba korektně funguje (viz příklad výše), začneme komunikaci s aplikačním protokolem. Jestliže služba na daném portu neběží (např. výpadek webového serveru), dostaneme např. následující odpověď:

```
/home/matousp/skola/site/skripta> telnet pc1.fit.vutbr.cz http
Trying 147.229.1.3...
telnet: connect to address 147.229.1.3: Connection refused
telnet: Unable to connect to remote host
```

- *Sledování otevřených síťových spojení.*

Někdy potřebujeme zjistit, zda na našem počítači běží daná služba a zda je k ní někdo připojen. Pro zjištění běžících lokálních služeb lze použít prostředky popsané v následujících odstavcích.

Výpis síťových spojení sockstat (Unix) a netstat -a (Windows)

Programy **sockstat** a **netstat** zobrazí aktuální otevřená spojení, včetně typu protokolu (TCP/UDP), čísla portu příchozího i odchozího spojení a stavu spojení, viz následující ukázka:

Windows: Aktivní připojení

Proto	Místní adresa	Cizí adresa	Stav
TCP	nbisa:5679	nbisa:0	NASLOUCHÁNÍ
TCP	nbisa:http	nbisa:0	NASLOUCHÁNÍ
TCP	nbisa:1033	nbisa:0	NASLOUCHÁNÍ
TCP	nbisa:netbios-ssn	nbisa:0	NASLOUCHÁNÍ
TCP	nbisa:1076	kazi.fit.vutbr.cz:microsoft-ds	NAVÁZANO
TCP	nbisa:1098	pcmatousek.fit.vutbr.cz:microsoft-ds	NAVÁZANO
UDP	nbisa:microsoft-ds	*:*	
UDP	nbisa:isakmp	*:*	

Unix:

```
/home/matousek/test> sockstat
USER      COMMAND  PID  FD PROTO  LOCAL ADDRESS    FOREIGN ADDRESS
matousek  firefox-bi 56896 603 tcp4    147.229.12.101:44792 74.125.232.206:80
root      smbd      54847 28 tcp4    147.229.12.101:445 147.229.14.56:55137
matousek  ssh       52875 3 tcp4    147.229.12.101:47477 147.229.8.12:22
root      rpcbind   801 10 udp4    *:889          *:*
```

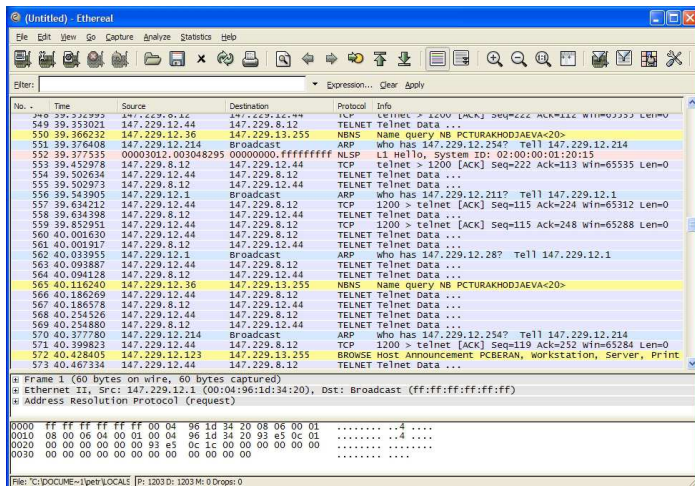
Z výše uvedeného výpisu můžeme vyčíst, že na lokálním unixovém systému běží aplikace **firefox** s navázaným spojením na server 74.125.232.206, dále aplikace **smbd** (Samba), spojení **ssh**. Dále jsou tam služby **rpcbind**, které běží a očekávají na příchozí spojení.

Sledování toku dat na síťovém rozhraní

Sledování toku dat a jejich rozbor je náročnější způsob analýzy. Na druhou stranu nám tento přístup nabízí ucelený a jednoznačný pohled na aktuálně probíhající komunikace. Při analýze vidíme obsahu příchozích i odchozích paketů. Protože se jedná o citlivé aktivity, může tyto prostředky pro sledování používat pouze uživatel s právy administrátora systému. V prostředí unixových operačních systému se používá program **tcpdump**, který sleduje přenos na zadaném rozhraní.

```
tcpdump -s0 -X -i fxp0
13:53:02.136977 IP pcmatousek.fit.vutbr.cz.62047 > kazi.fit.vutbr.cz.domain:
 8123+ PTR? 254.12.229.147.in-addr.arpa. (45)
    0x0000: 4500 0049 6233 0000 4011 dc3f 93e5 0c5b E..Ib3..@..?..[
    0x0010: 93e5 080c f25f 0035 0035 23ea 1fbb 0100 ....._5.5#....
    0x0020: 0001 0000 0000 0000 0332 3534 0231 3203 .....254.12.
    0x0030: 3232 3903 3134 3707 696e 2d61 6464 7204 229.147.in-addr.
13:53:04.458742 IP pckolar.fit.vutbr.cz.ipp>147.229.13.255.ipp: UDP, length
0x0000: 4500 00cb dec6 4000 4011 1935 93e5 0c5d E....@..5...[
0x0010: 93e5 0dfc 0277 0277 00b7 e27a 3330 3136 .....w...z3016
0x0020: 2033 2069 7070 3a2f 2f70 636b 6f6c 6172 .3.ipp://pckolar
0x0030: 3a36 3331 2f70 7269 6e74 6572 732f 4850 :631/printers/HP
0x0040: 3433 3030 5053 2022 4c61 7365 724a 6574 4300PS."LaserJet
```


V prostředí Windows i Unix je velmi oblíbený grafický protokolový analyzátor Wireshark (dříve Ethereal), obr. 1.23, případné programy Network Monitor, Network Diagnostik a další.



Obrázek 1.23: Protokolový analyzátor Wireshark

Blíže si o způsobu analýzy dat povíme v kapitole 11 Správa sítě.

Shrnutí

Architektura počítačových se skládá ze tří základních částí – a) fyzické části, která zahrnuje přenosové technologie, b) z prostředků pro zajištění spolehlivého přenosu, a c) ze síťových aplikací. Struktura jednotlivých částí sítě se popisuje tzv. vrstevnými modely. V dnešní době se pro systémy pracujícími na bázi Internetu používá model TCP/IP. Pro porovnávání různých modelů slouží referenční model ISO OSI.

Na každé vrstvě modelu jsou definovány služby, které zajišťují určité služby pro vyšší vrstvy. Při přenosu dat se data z vyšších vrstev zapouzdřují do protokolových jednotek nižších vrstev. Z pohledu přenosu dat probíhá komunikace vždy mezi procesy na stejné vrstvě.

Architekturu Internetu popisuje model TCP/IP. Ten definuje dva základní typy přenosů – spolehlivé spojované služby TCP a nespolehlivé nespojované služby UDP. Aplikační procesy TCP/IP komunikují přes schránky (sockets), což jsou programové struktury tvořící API pro přenos dat po síti. Schránka je identifikovaná adresou počítače a adresou služby. V systému jsou schránky implementovány prostřednictvím síťové knihovny, např. BSD sockets.

Pro identifikaci zdrojů, služeb a uzlů na síti se používají adresy. V Internetu se můžeme setkat s různými způsoby adresování, které se liší podle vrstvy, na níž se používají. Pro

identifikaci síťového rozhraní se používají MAC adresy (Ethernet, hardwarové adresy), pro adresování uzlů v Internetu slouží IP adresy, na aplikační úrovni používáme doménová jména. Pro adresování služeb se využívá čísel portů. Aplikace mají své vlastní způsoby adresování, např. webová služba používá adresy typu URL (Uniform Resource Locator), poštovní služby využívají emailové adresy, atd.

Pro práci s IP adresami se používá pět schémat přístupu k adresování: (1) rozdělení IP adres do tříd (třídní adresování), (2) vytváření podsítí (subnetting), (3) beztřídní adresování (CIDR), (4) překlad adres (NAT) či (6) použití IP adres verze 6. Tato schémata vznikla v reakci na nedostatek IP adres v polovině 90. let 20. století. Dnes se můžeme setkat se všemi pěti způsoby používání IP adres, které lze často kombinovat.

Při připojení síťového zařízení do Internetu je potřeba nainstalovat služby TCP/IP (jako modul či součást jádra operačního systému), nastavit IP adresu počítače včetně síťové masky, nakonfigurovat výchozí bránu a adresy serverů služby DNS. Pokud jsme správně nastavili síťové připojení, můžeme nainstalovat a spustit síťové služby. Služby mohou běžet na počítači neustále (např. poštovní server), nebo se spustí až při příchodu požadavku (FTP server).

Pokud dojde k chybě při komunikaci, existuje spousta diagnostických prostředků pro detekci chyby a konfiguraci spojení. Pro kontrolu nastavení IP adres lze použít program `ifconfig/ipconfig`, k otestování přenosu je vhodný program `ping` či `traceroute/tracert`. Pro datovou analýzu se používá program `tcpdump` či Wireshark.

Otázky

1. Popište jednotlivé vrstvy modelu TCP/IP.
2. Co je potřeba zajistit pro úspěšné připojení počítače k síti?
3. K čemu slouží modul TCP/IP z pohledu aplikací?
4. Které aplikace využívají TCP/IP?
5. Jaký je rozdíl mezi UDP a TCP? Jaké jsou minimální délky paketů TCP a UDP?
6. Co jsou to schránky (sockets) a k čemu slouží?
7. Uveďte rozdělení IP adres do tříd. Které adresy jsou privátní?
8. Kolik počítačů lze adresovat v síti 195.180.11.0/8?
9. Uveďte příklad adres na linkové, síťové, transportní a aplikační vrstvě.
10. Co je to CIDR a k čemu slouží?
11. Vysvětlete, jak funguje NAT. Co obsahuje překladová tabulka NAT?
12. Uveďte postup při zjišťování nedostupnosti webového serveru.
13. Jaké znáte nástroje pro: zjištění konfigurace síťových rozhraní, sledování paketů, testování dostupnosti sítě, zobrazení otevřených spojení, výpisu tabulky ARP, výpisu směrovací tabulky?

Doporučená literatura a standardy

- [1] B.Cain, S.Deering, I.Kouvelas, B.Feener, and A.Thyagarajan. *Internet Group Management Protocol Version 2*. RFC 3376, October 2002.
- [2] B.Croft and J.Gilmore. *Bootstrap Protocol (BOOTP)*. RFC 951, September 1985.
- [3] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945, May 1996.
- [4] Cisco. *Internetworking Technologies Handbook*. Cisco Press, 4th edition, 2004.
- [5] M. Cotton and L. Vegoda. *Special Use IPv4 Addresses*. RFC 5735, January 2010.
- [6] D.C.Plummer. *An Ethernet Address Resolution Protocol*. RFC 826, November 1982.
- [7] D.Meyer. *Administratively Scoped IP Multicast*. RFC 2365, July 1998.
- [8] E.Gerich. *Guidelines for Management of IP Address Space*. RFC 1466, May 1993.
- [9] K. Egevang and P. Francis. *The IP Network Address Translator (NAT)*. RFC 1631, May 1994.
- [10] F. Halsall. *Computer Networking and the Internet*. Addison-Wesley, 5th edition, 2005.
- [11] IANA. *Special-Use IPv4 Addresses*. RFC 3330, September 2002.
- [12] Javvin. *Network Protocols Handbook*. Javvin Technologies, Inc., 2nd edition, 2005.
- [13] J.Postel. *User Datagram Protocol*. RFC 768, August 1980.
- [14] J.Postel. *An Assigned numbers*. RFC 790, September 1981. Od roku 1994 převedeno do databáze na IANA na <http://www.iana.org/>.
- [15] J.Postel. *Internet Control Message Protocol*. RFC 792, September 1981.
- [16] J.Postel. *Internet Protocol*. RFC 791, September 1981.
- [17] J.Postel. *Transmission Control Protocol*. RFC 793, September 1981.
- [18] J.Postel and J.Mogul. *Internet Standard Subnetting Procedure*. RFC 950, August 1985.
- [19] K.Auerbach. *Protocol Standard for a NetBIOS Service on a TCP/UDP Transport: Concepts and Methods*. RFC 1001, March 1987.
- [20] K.Hubbard, M.Kosters, D.Conrad, D.Karrenberg, and J.Postel. *Internet Registry IP Allocation Guidelines*. RFC 2050, November 1996.
- [21] P.Resnick. *Internet Message Format*. RFC 2822, April 2001.
- [22] P.Srisuresh and K.Egevang. *Traditional IP Network Address Translator (Traditional NAT)*. RFC 3022, January 2001.
- [23] R.Droms. *Dynamic Host Configuration Protocol*. RFC 2131, March 1997.

- [24] R.Finlayson, T.Mann, J.Mogul, and M.Theimer. *A Reverse Address Resolution Protocol*. RFC 903, June 1984.
- [25] S.Deering. *Host Extensions for IP Multicasting*. RFC 1112, August 1989.
- [26] L. Skočovský. *Principy a problémy operačního systému Unix*. Science, 1993.
- [27] W.R. Stevens. *Programování sítí operačního systému Unix*. Science, Veletiny, 1994.
- [28] T.Socolofsky and C.Kale. *A TCP/IP Tutorial*. RFC 1180, January 1991.
- [29] V.Fuller, T.Li, J.Yu, and K.Varadhan. *CIDR: an Address Assignment and Aggregation Strategy*. RFC 1519, September 1993.
- [30] Y.Rekhter, B.Moskowitz, D.Karrenberg, G.J.de Groot, and E.Lear. *Address Allocation for Private Internets*. RFC 1918, February 1996.
- [31] Z.Wang and J.Crowcroft. *A Two-Tier Address Structure for the Internet: A Solution to the Problem of Address Space Exhaustion*. RFC 1335, May 1992.

Kapitola 2

Programování sítí TCP/IP

V této kapitole se podíváme na další zajímavou část počítačových sítí – vytváření síťových aplikací. Dosud jsme se bavili o principech, na kterých počítačová komunikace stojí, jako jsou adresování, komunikace na různých vrstvách a implementace síťové části komunikujícího uzlu.

Nyní se budeme zabývat otázkou tvorby síťových aplikací z pohledu programátora. Jak uvidíme, není až tak složité vytvořit aplikace typu klient-server, které si mezi sebou vyměňují data. Když potřebujeme vyřešit specifický přenos dat, např. přenos dat z centrální databáze do lokální aplikace, využití programových nástrojů na tvorbu klienta a serveru nám ušetří čas. Navíc postupy pro tvorbu síťových aplikací jsou víceméně známé a je možné na nich stavět.

Při vytváření aplikací využijeme vrstvého modelu TCP/IP, který odděluje jednotlivé funkce přenosových vrstev. Z pohledu programátora nepotřebujeme implementovat komunikaci TCP, přenos IP či skládání ethernetovských rámců. Stačí využít služeb knihoven, které implementují tyto služby a které jsou součástí většiny programovacích jazyků a operačních systémů.

Existuje více způsobů, jak zajistit komunikaci mezi procesy na síti. Mezi historicky nejstarší (80. léta 20. století) patří například unixové roury (pipes), komunikace UUCP (unix to unix copy) či volání vzdálených procedur RPC (remote procedure call). Unixový operační systém BSD (Berkeley Software Distribution) byl první, který ve své verzi 4.2 BSD roku 1983 implementoval komunikační rozhraní TCP/IP prostřednictvím knihovny *BSD sockets* (*schránky BSD*) [9].

Jedná se o knihovnu funkcí pro jazyk C, která obsahuje funkce pro vytvoření spojení na transportní úrovni, adresování vzdáleného počítače, nastavení parametrů spojení a přenos dat. Pokud použijeme tuto knihovnu, nemusíme vytvářet vlastní TCP pakety a zapisovat je na síťovou kartu. To vše zajistí knihovna. Přístup k síťovému spojení je stejný jako k souborům – přes souborové deskriptory (file descriptor). Souborové deskriptory jsou číselné identifikátory souboru (či zařízení) souborového systému, které jednoznačně identifikují dané zařízení pro zápis a čtení. Existují deskriptory typu soubor (file), adresář (directory), schránka (sockets), roura (pipe) a další. Přístup k zařízením pro zápis či čtení dat přes souborové deskriptory je transparentní, tzn. stejné operace pro zápis dat do souboru na disku můžeme použít i pro zápis dat po síti (například funkci *write()*).

Knihovny pro komunikaci se postupně vyvíjely i na dalších operačních systémech a pro různé jazyky. Jedna z prvních implementací komunikačního API pro MS Windows, která vycházela z existujícího rozhraní BSD sockets, se objevila v roce 1992 pod názvem Winsock verze 1. Také implementace komunikačních rozhraní (schránek) v jazycích Perl, Java apod. vychá-

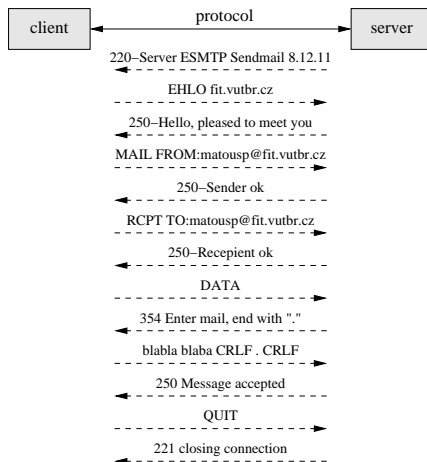
zejí z principů původních BSD schránek. Tyto knihovny se sice liší v názvem a parametrech síťových funkcí, nicméně programovací principy zůstávají stejné.

Proto se pokusíme v této části popsat původní a stále používanou knihovnu BSD schránek a její vlastnosti: funkce, principy pro vytváření konkurentních serverů, techniky neblokujícího chování, možnosti přenosu multicastu, broadcastu a další. Ukážeme si také, jak je možné vytvářet jednotky PDU na nižších vrstvách, například IP datagramy a ethernetové rámce, a jak odchyťávat veškerý provoz na síťové kartě.

Tyto znalosti jsou důležité pro implementaci komunikace mezi dvěma aplikacemi. Pro komunikaci je také nutný protokol, který nám říká, jaké příkazy si aplikace posílají (syntax) a jaké chování očekávají (sémantika). Uvedeme také některé způsoby popisu komunikačních protokolů.

2.1 Model klient-server, protokol

Standardní schéma komunikace mezi dvěma procesy se nazývá *model klient-server*. Klient i server jsou aplikační procesy, které komunikují přes síťové rozhraní. Může jít i o procesy běžící na stejném počítači. Základní činností klienta je posílat žádosti o nějakou síťovou službu. Server čeká na příchozí požadavky, přijímá je, zpracovává a posílá zpět odpovědi. Příkladem je například webový server (například aplikace Apache) a webový klient (aplikace Mozilla). Webový klient pošle žádost o webovou stránku – příkazem `GET http://www.fit.vutbr.cz/HTTP/1.1`. Server požadavek přijme, vyhledá příslušnou stránku na lokálním počítači a zašle ji zpět klientovi protokolem HTTP.



Obrázek 2.1: Příklad komunikace prostřednictvím protokolu SMTP.

Důležité je, že zpracování požadavku probíhá výhradně na straně serveru. Klient pouze předává požadavek a zobrazuje odpověď. Převádí požadavek z uživatelského formátu (napří-

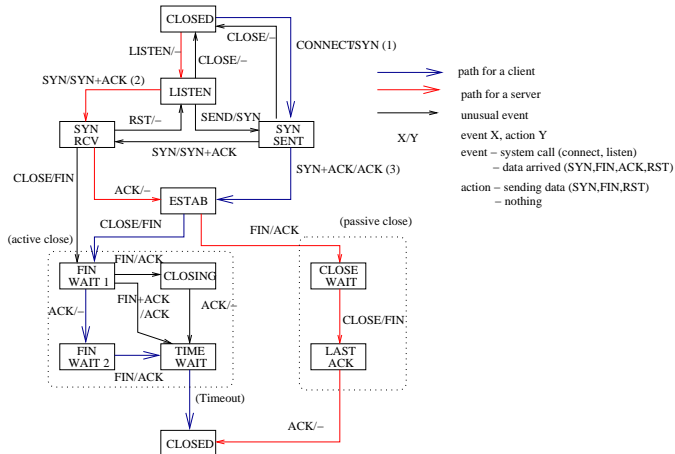
klad zápis URL `www.fit.vutbr.cz` na stavovém řádku prohlížeče) do formátu komunikačního protokolu (v našem případě protokolu HTTP). Tento příkaz pak odešle. Po přijetí odpovědi v jazyce HTML zobrazí klient došlá data v uživatelsky příjemném tvaru, například jako webovou stránku s grafickými prvky.

U modelu klient-server komunikaci obvykle začíná klient. Server čeká na dotazy a odpovídá je. Často běží v nekonečné smyčce. Většinou umožňuje konkurentní (souběžný) přístup, tj. současné zpracování více požadavků. Pro implementaci konkurentního serveru se využijí vlastnosti programovacího jazyka pro paralelní programování, např. funkce `fork()` pro vytváření procesů či funkce pro vytváření vláken.

Komunikace mezi klientem a serverem je popsána tzv. *protokolem* – viz obrázek č. 2.1. Protokol je soubor syntaktických a sémantických pravidel určujících výměnu informace mezi nejméně dvěma entitami. Protokol zahrnuje proceduru navázání spojení, adresování, přenosu dat, zpracování chyb, řízení toku komunikace, přidělování prostředků.

Protokol lze popsat neformálně (například slovně, viz standardy RFC) nebo formálně. Mezi nejčastěji používané formální popisy protokolů patří

- Stavové automaty – například konečné automaty,
- Gramatiky – formální jazyky, jazyk SDL (Structure Definition Language),
- Grafové modely – Petriho sítě, sekvenční diagramy MSC,
- Algebraické prostředky – CCS (Calculus of Communication Systems)[5], stopy (traces).



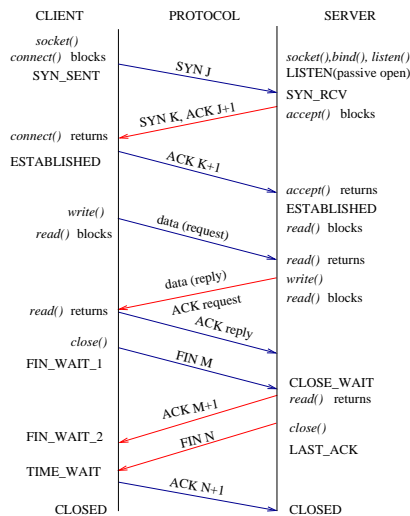
Obrázek 2.2: Popis funkce TCP pomocí stavového automatu

Příklad popisu protokolu TCP pomocí stavového automatu lze najít na obrázku 2.2. Automat obsahuje stavy a přechody pro klienta i pro server. Například server se po spuštění nachází

ve stavu **CLOSED**, odkud přechází po volání funkce `listen()` do stavu **LISTEN**, ve kterém čeká na příchozí požadavky. Po přijetí požadavku **SYN** odpoví potvrzující sekvencí **SYN+ACK** a přejde do stavu čekající na ukončení ustavující fáze (stav **SYN RCV**). Po přijetí potvrzení je spojení navázáno (stav **ESTAB**) a dochází k výměně dat. V případě pasivního uzavření spojení přijme server příkaz **FIN**, který potvrdí **ACK** a přechází do stavu **CLOSE WAIT**. Zde po systémovém volání `close()` vyšle příkaz **FIN** a po jeho potvrzení (přijetí **ACK**), přechází do původního stavu **CLOSED**.

Z pohledu klienta vypadá chování trochu jinak. Po zavolání funkce `connect()` vyšle klient příkaz **SYN** s žádostí o ustavení spojení. Přechází přitom ze stavu **CLOSED** do stavu **SYN SENT**. Pokud obdrží od serveru zprávu **SYN** s potvrzením **ACK**, potvrdí tuto komunikaci zprávou **ACK** a přejde do stavu **ESTAB**. Zde dochází k výměně dat. Pro ukončení spojení (aktivní uzavření), pošle klient voláním funkce `close()` příkaz **FIN** serveru a přejde do stavu **FIN WAIT 1**. Pokud na to server odpoví potvrzením **ACK**, přejde klient do stavu **FIN WAIT 2**. Zde pak čeká na uzavření ze strany serveru, tj. na příkaz **FIN**. Pokud žádost přijde, klient ji potvrdí (pošle příkaz **ACK**) a přejde do stavu **TIME WAIT**. Zde čeká nějakou dobu na doručení tohoto potvrzení a po uplynutí časového limitu `timeout` přejde do původního stavu **CLOSED**.

Výše uvedený protokol pomocí stavového automatu jednoznačně definuje pořadí zpráv (tzn. jazyk) a chování protokolu. Totéž chování protokolu komunikace TCP můžeme popsat také sekvenčním časovým diagramem MSC, jak je vidět na obrázku 2.3. Narozdíl od konečného



Obrázek 2.3: Popis funkce TCP pomocí sekvenčního diagramu MSC

automatu nám tento prostředek popisuje pouze jeden scénář chování TCP (z pohledu teorie jazyků bychom řekli jednu větu jazyka). Neukazuje nám, jak se bude systém chovat v případě, že nedojde ke korektnímu navázání spojení, například když server pošle zprávu **RST**.

V diagramu se objevují názvy funkcí `socket()`, `connect()`, `listen()`, `bind()`, které tvoří programové rozhraní BSD schráněk. Podrobněji si je popíšeme v další kapitole. Příznaky protokolu TCP SYN, ACK, FIN, RST jsou vysvětleny v definici protokolu TCP [4].

Komunikační protokol může být standardizovaný, tj. definovaný standardy mezinárodních organizací typu IETF¹ (standardy RFC), IEEE² (standardy 802.3, 802.1q), organizací CCITT/ITU-T³ (např. H.323, G.711) či ISO⁴ (např. X.500, IS-IS) apod.). Protokoly mohou být také firemní (proprietární), např. CIFS (od firmy Microsoft), EIGRP (firma Cisco) či SMB (firma IBM).

Pokud si vytváříme vlastní síťovou aplikaci, můžeme si také navrhnout vlastní protokol. Výhodou je, že tím přesně implementujeme naše požadované vlastnosti. Výsledná aplikace bude rychlá a jednoduchá. Na druhou stranu pro použití této aplikace budeme muset vytvořit klienty a také jasně popsat protokol tak, aby i další uživatelé mohli k aplikaci přistupovat. Běžné protokoly na Internetu (například HTTP, DNS, SMTP, IMAP) jsou popsány standardy IETF RFC (Request for Comments). Jedná se často o doporučení než závazné standardy, nicméně je výrobci síťových zařízení a aplikací berou vážně. Podle toho popisu si můžeme vytvářet například vlastního klienta či server pro službu WWW.

Při tvorbě klienta či serveru pro standardizovanou službu je potřeba zajistit tzv. konformanci (přizpůsobení se) vůči standardu. Musíme ukázat, že nově vytvořený klient/server splňuje požadavky dané standardem a chová se tak, jak od klienta či serveru dané služby očekáváme. Toto zajišťuje tzv. testování konformance (conformance testing).

K základním úlohám serveru kromě vlastního zpracování dotazu patří

- ověření identity uživatele (autentizace),
- autorizace uživatele (má daný uživatel právo žádat tuto službu),
- zajištění bezpečnosti dat na serveru,
- privátnost dat (uživatel má právo přistupovat pouze ke svým datům),
- ochrana zdrojů (zabezpečení systémových zdrojů před zneužitím přes komunikační síť).

2.2 Programové rozhraní pro transportní vrstvu

Základním programovým prostředkem pro vytváření aplikací komunikujících po síti jsou *schránky* (*sockets*). Schránky tvoří aplikační programové rozhraní (API) pro komunikující procesy po síti. Vytvářejí tzv. koncový komunikační bod. Fyzicky se jedná o abstraktní datovou strukturu, která obsahuje nezbytné údaje pro komunikaci a jednoznačnou identifikaci v síti: IP adresu pro identifikaci počítače a číslo portu pro určení aplikace.

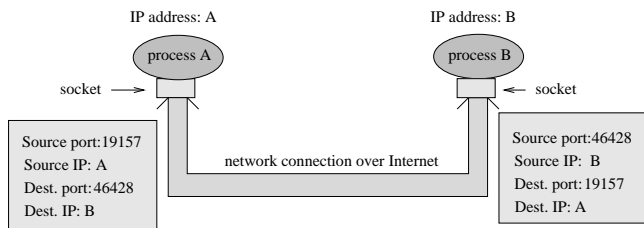
Pro vytvoření spojení je potřeba propojit obě schránky na komunikujících uzlech. Pro úspěšnou komunikaci musí obě schránky znát jednak svou svou IP adresu a číslo portu své aplikace, dále pak IP adresu a číslo portu komunikujícího partnera, viz obrázek 2.4. Bez znalosti údajů partnerské schránky by lokální schránka nemohla vyplnit údaje do hlaviček IP a TCP/UDP.

¹Internet Engineering Task Force, www.ietf.org

²Institute of Electrical and Electronics Engineers, www.ieee.org

³Comité Consultatif International Téléphonique et Télégraphique, přejmenovaná v roce 1993 na International Telecommunication Union Telecommunication Standardization Sector (ITU-T), www.itu.int/ITU-T

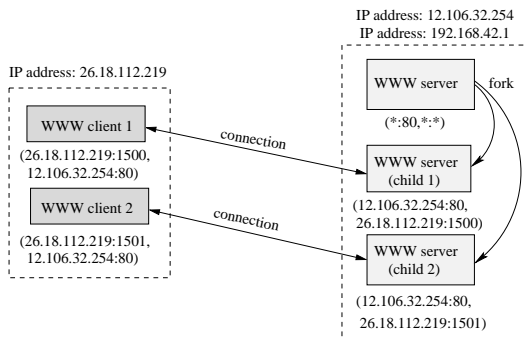
⁴International Organization for Standardization, www.iso.org



Obrázek 2.4: Komunikace pomocí schránek.

Informace typu lokální IP adresy, lokální port, vzdálená IP adresa a vzdálený port se ukládají v datové struktuře schránka (jedná se o složený datový typ záznam). Po vyplnění těchto údajů lze zahájit komunikaci. Zápis těchto údajů do struktury schránka zajišťují funkce síťové knihovny, které si za chvíli představíme.

Použití schránek při komunikaci mezi aplikacemi pro službu WWW je na obrázku č. 2.5. Webový server komunikuje se dvěma klienty, kteří běží na stejném počítači. TCP neumí rozlišit více souběžných spojení na stejném síťovém rozhraní pouze podle IP adresy. Potřebuje k takomu číslu portů obou komunikujících stran. Spojení tedy plně identifikují čtyři položky: IP adresy a porty zdroje, IP adresy a porty cíle. Pokud přijde požadavek od klienta č. 1 (port 1500) jsou předána procesu serveru č. 1. Pokud přijdou z portu 1501, jsou předána procesu serveru č. 2. Servery č. 1 a 2 vzniknou voláním funkce `fork()`, která vytváří paralelní procesy.



Obrázek 2.5: Konkurentní WWW server.

V praxi si můžeme všimnout, že na typickém serveru, běží více lokálních komunikujících procesů (viz např. výpis příkazem `sockstat`). Jde o procesy, které buď obsluhují konkrétního klienta, například (26.18.112.119:1500), nebo o procesy, které čekají na příchozí spojení. Tyto naslouchající procesy lze ve výpisu odlišit tak, že obsahují nedefinovanou adresu klienta ve formátu (*:*)

2.2.1 Datová struktura schránka

Datová strukturu schránka pro protokol IPv4 se nazývá `sockaddr_in` a je definovaná v knihovně `<netinet/in.h>`. Její struktura je následující:

```
<netinet/in.h>
struct in_addr{
    in_addr_t s_addr;          /* 32-bitová adresa IPv4, network byte order */
};
struct sockaddr_in{
    uint8_t      sin_len;      /* délka struktury */
    sa_family_t   sin_family; /* typ adresy - AF_INET */
    in_port_t     sin_port;    /* 16-bitové číslo portu, network byte order */
    struct in_addr sin_addr;    /* 32-bitová adresa IPv4, network byte order */
    char          sin_zero[8]; /* nepoužito */
};
```

Pro vytvoření a inicializaci schránky se používá funkce

`socket(int family, int type, int protocol)`, kde

family specifikuje rodinu protokolů (doménu, jmenný prostor), ve kterém se bude komunikovat. Jmenný prostor je určen typem protokolového profilu (např. TCP/IP). Schránky mohou obsahovat typy adres uvedené v tabulce 2.1. V některých implementacích se místo prefixu `AF_` (address family), používá prefix `PF_` (protocol family). Čísla rodin protokolů (address family numbers) lze najít na

<http://ftp.iana.org/assignments/address-family-numbers>.

Rodina protokolů	Specifikace
AF_INET	protokoly Internetu IPv4
AF_INET6	protokoly Internetu IPv6
AF_LOCAL	interní komunikace v Unixu, dříve AF_UNIX
AF_ROUTE	schránky pro směrování
AF_KEY	schránky pro správu klíčů

Tabulka 2.1: Přehled rodin protokolů (adresových typů).

type určuje typ schránky, tj. způsob komunikace, viz `<sys/socket.h>`. Mezi základní typy schránek patří schránky pro komunikace TCP, UDP, sekvenční pakety a přímý přístup na síťovou vrstvu.

Typ schránky	Specifikace
SOCK_STREAM	spolehlivá, sekvenční komunikace, spojení TCP
SOCK_DGRAM	nespolehlivá datagramová komunikace, spojení UDP
SOCK_RAW	přístup k rozhraní síťové vrstvy, pouze pro uživatele <code>root</code>
SOCK_SEQPACKET	spolehlivé, sekvenční pakety, pouze pro typ <code>AF_NS</code>
SOCK_RDM	spolehlivé doručované datagramy, neimplementováno

Tabulka 2.2: Typy schránek

protocol určuje typ transportního protokolu - `IPPROTO_TCP`, `IPPROTO_UDP`, `IPPROTO_SCTP`. Hodnota 0 označuje implicitní protokol. Typ protokolu závisí na typu komunikace (předchozí parametr).

Protokol	Specifikace
<code>IPPROTO_TCP</code>	transportní protokol TCP, RFC 793 [4]
<code>IPPROTO_UDP</code>	transportní protokol UDP, RFC 768 [3]
<code>IPPROTO_SCTP</code>	transportní protokol SCTP, RFC 4960 [10]

Tabulka 2.3: Typy protokolů

Ne všechny kombinace rodiny protokolů a typů protokolů jsou přípustné. Pro základní komunikaci UDP a TCP se doporučuje ponechat implicitní hodnotu 0, která způsobí přiřazení vhodného protokolu. Možné kombinace hodnot **family** a **protocol** jsou uvedeny v tab. 2.4.

	<code>AF_INET</code>	<code>AF_INET6</code>	<code>AF_LOCAL</code>	<code>AF_ROUTE</code>	<code>AF_KEY</code>
<code>SOCK_STREAM</code>	TCP, SCTP	TCP, SCTP	Ano		
<code>SOCK_DGRAM</code>	UDP	UDP	Ano		
<code>SOCK_SEQPACKET</code>	SCTP	SCTP	Ano		
<code>SOCK_RAW</code>	IPv4	IPv6		Ano	Ano

Tabulka 2.4: Vybrané kombinace typu rodiny protokolů a protokolu

Protokol SCTP (Stream Control Transmission Protocol) [10] nabízí podobné služby jako transportní protokoly UDP a TCP. SCTP vytváří asociace mezi klienty a servery. Při komunikaci zajišťuje spolehlivé spojení, řazení paketů, řízení toku dat a duplexní přenos podobně jako TCP. Místo výrazu "spojení (connection)" používá termín "asociace (association)", neboť spojení se uvažuje většinou mezi dvěma síťovými prvky identifikovanými IP adresami. Asociace označuje komunikaci mezi dvěma systémy, což může znamenat připojení na více IP adres (systémy s více adresami, multihoming). Narozdíl od TCP je SCTP orientováno na zprávy. Zajišťuje jejich spolehlivé doručení v pořadí tak, jak následují za sebou.

SCTP vytváří vícenásobné toky dat (streams) mezi koncovými body spojení. Každý tok má svou službu spolehlivého doručení při zachování řazení zpráv. Ztráta zprávy v jednom z těchto proudů nemá vliv na rychlost doručení zprávy jiného proudu. To je velký rozdíl oproti TCP, kde ztráta v jednom toku dat blokuje doručení následujících dat tohoto toku do té doby, než bude obnoven ztracený paket.

SCTP podporuje existenci více IP adres na koncovém síťovém zařízení (multihoming). Tato vlastnost zvyšuje robustnost proti chybám. Koncový bod může mít více redundantních spojení, kde každé rozhraní má jiné spojení do Internetu. SCTP může pracovat i v případě chyby jedné sítě nebo cesty přes Internet tak, že přepne na jinou adresu asociovanou pomocí SCTP asociace.

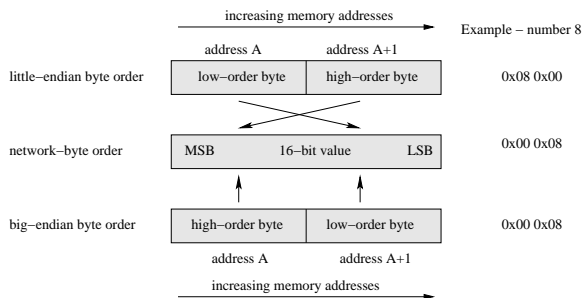
Kromě struktury `sockaddr_in` pro IPv4 obsahuje knihovna `<netinet/in.h>` také strukturu `sockaddr_in6` pro IPv6, generickou schránku `sockaddr_storage`, schránku pro unixové systémy `sockaddr_un` a další.

Síťový formát pro přenos bitů a bytů

Při používání schránek BSD si můžeme všimnout, že u několika položek ve struktuře `sockaddr_in` je uveden zápis čísla ve tvaru *Network Byte Order*, což definuje reprezentaci čísla na síti. Tento

zápis udává pořadí bitů pro vyjádření IP adres či čísel portů v hlavičkách protokolů.

Uvažujme číslo ve formátu 16-bitový integer. Toto číslo je tvořeno dvěma byty. Existují dva způsoby, jak uložit dvoubytové číslo v paměti procesoru. První je uložení nižšího bytu na nižší adresové místo, kdy hodnota se čte zprava doleva. Tato reprezentace se nazývá *little-endian*. Druhou možností je uložit vyšší byte na nižší adresové místo (a číst hodnotu zleva doprava). Tato reprezentace se nazývá *big-endian*, viz obrázek 2.6.



Obrázek 2.6: Příklad uložení 16-bitového čísla v reprezentaci *little-endian* a *big-endian*.

Názvy *big-endian* a *little-endian* byly převzaty z díla Jonathana Swifta Gulliverovy cesty, kdy se dva národy, Liliputáni a Blefuskové hádali, zda se má vajíčko rozbít na menším konci (pozice Liliputánů přezdívaných *Little-endians*) či na širším konci, což vyžadovali Blefuskové přezdívaní *Big-endians*.

Z obrázku 2.6 můžeme vidět, že například dekadické číslo 8 se v 16-bitové reprezentaci uloží jako 0x08 0x00 u architektury *little-endian* zatímco u architektury *big-endian* to bude v opačném pořadí, tj. jako 0x00 0x08. Různé druhy procesorů používají různou reprezentaci čísel. Například procesory Motorola používají reprezentaci *big-endian*, u procesorů Intel x86 či VAX můžeme najít architekturu *little-endian*. Pro komunikaci na síti mezi různými architekturami je potřeba zvolit jeden formát.

Pro přenos dat po síti (network byte order) byl vybrán formát **big-endian**, tzn. zápis zleva doprava s nejvýznamnějším bytem vlevo (pořadí odpovídající čtení textu v angličtině) a nejvýznamnějším bitem (MSB, Most Significant Bit) také vlevo. Znamená to, že čísla používaná například v hlavičkách protokolů, musí být reprezentována v tomto formátu, aby byly správně interpretována přijímající stranou.

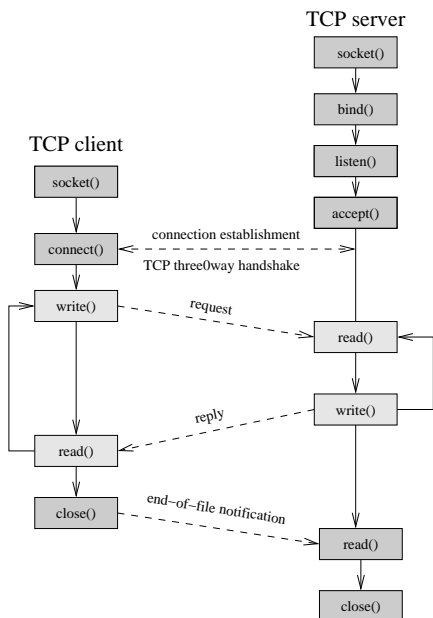
Proto se pro zadávání například IP adres a čísel portů používají funkce, které hodnotu převádějí na síťový tvar. U architektur *big-endians* neprovádí tyto funkce nic, u architektur *little-endians* se provádí bytovou konverzi. Aby byla síťová aplikace univerzální, je potřeba každou hodnotu převést do síťového tvaru. Jinak se nám může stát, že například 16-bitové číslo portu 80 (0x00 0x50) se bude interpretovat na jiné architektuře jako 20480 (0x50 0x00).

Pro převod do síťové reprezentace se používají následující funkce:

```
#include <netinet/in.h>
uint_16_t htons(uint16_t short_n); /* host to network byte order short */
uint_32_t htonl(uint32_t long_n);  /* host to network byte order long */
uint_16_t ntohs(uint16_t short_n); /* network byte order to host short */
uint_32_t ntohl(uint32_t long_n);  /* network byte order to host long */
```

2.2.2 Programování komunikace nad TCP

Nyní si popíšeme schéma komunikace TCP pro aplikace typu klient-server, dále implementaci konkurentního serveru a funkce pro vytváření spojení TCP. Protože spojení TCP je perzistentní, tj. trvá po celou dobu přenosu, je vytváření složitější než u UDP. Architekturu klienta a serveru je možné z pohledu komunikace graficky znázornit jako posloupnost funkcí síťové knihovny, viz obrázek 2.7.



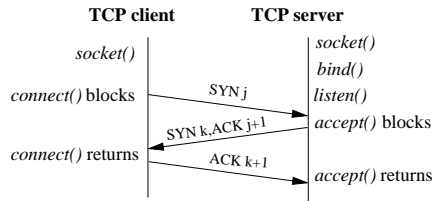
Obrázek 2.7: Komunikace klient-server přes TCP.

Činnost komunikace TCP lze rozdělit na tři základní části:

1. Vytvoření spojení TCP.
2. Komunikace, výměna dat.
3. Uzavření spojení.

Vytvoření spojení TCP

TCP vytváří spojení pomocí mechanismu *třířázové synchronizace* zvané též "třířázové podávání ruky", angl. *three-way handshake*, který zahrnuje výměnu paketů SYN, SYN+ACK, ACK, viz obrázek 2.8. Spojení vyvolává blokující funkce `connect()` na straně klienta TCP.



Obrázek 2.8: Ustavení TCP spojení.

Tato funkce způsobí vyslání paketu TCP s příznakem SYN. Na straně serveru čeká proces na přijetí spojení funkcí `accept()`. Po výměně synchronizačních paketů je ustaveno spojení.

Zde můžeme pozorovat jeden z problémů spojení TCP – blokování. Během blokování klientský proces aktivně čeká na potvrzení po síti. V případě dlouhé prodlevy či chyby při komunikaci, může dojít k výraznému zpomalení běhu celé aplikace. Zejména u konkurentních serverů je potřeba problém blokování řešit. Více si o tom povíme v kapitole 2.3. Pro vytvoření spojení se používají následující funkce:

socket() – vytvoření datové struktury schránka, inicializace schránky

```
int socket(int family, int type, int protocol)
```

Funkce vytvoří schránku pro danou rodinu protokolů či doménu (`family`, `domain`), typ komunikace a protokol. Při úspěšném vytvoření vrátí funkce deskriptor schránky.

```
#include <sys/socket.h>
int s = socket(AF_INET, SOCK_STREAM, 0);
```

bind() – svázání schránky na straně serveru s konkrétním portem (tzv. pasivní otevření)

```
int bind(int sockfd, const struct sockaddr *myaddr, socklen_t addrlen)
```

Funkce `bind()` propojí schránku s lokální adresou a portem zadanými druhým parametrem. Tato funkce se využívá především u serverů. Pokud místo adresy zadáme obecnou adresu `INADDR_ANY`, vybere operační systém jakoukoliv platnou IP adresu počítače. Toho se dá využít v případě, že počítač má více síťových rozhraní. Při použití `INADDR_ANY` naslouchá server na každém připojeném síťovém rozhraní.

Pokud je schránka využívána jinou aplikací, vrátí volání funkce chybu `EADDRINUSE` "Address already in use".

```
#include <sys/socket.h>
struct sockaddr_in sin;
bind(s, (struct sockaddr*) &sin, sizeof(sin));
```

Na straně klienta TCP není nutné funkci `bind()` volat, neboť při žádosti o komunikaci přidělí síťová knihovna klientovi první volný port pro komunikaci. U serveru je naopak nutné port explicitně zadat, aby klient věděl o jakou službu (jaké číslo portu) má žádat. Pokud chceme připojit server na rezervovaný port (0–1023), musíme volat funkci `bind()` s oprávněním privilegovaného uživatele. Běžný uživatel nemůže vytvořit server na rezervovaném portu.

connect() – aktivní otevření na straně klienta

```
int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen)
```

Tato funkce provede připojení klienta TCP k serveru. Druhý parametr funkce obsahuje IP adresu a port serveru. Klient nemusí zadávat vlastní IP adresu a číslo portu – to mu přidělí systém. U některých speciálních aplikací, kdy i klient potřebuje běžet na konkrétním portu, je možné tuto hodnotu nastavit funkcí `bind()`.

Funkce `connect()` vysílá příkaz SYN. V případě, že žádaný server neodpoví (například není připojen na daném portu, neodpoví do určité doby, cílový počítač je nedostupný), vrátí funkce chybu (odpověď od ICMP):

- `ETIMEDOUT` – po zadanou dobu se nepodařilo nastavit spojení.
- `ECONNREFUSED` – adresovaný počítač odmítl spojení (uzavřel).
- `ENETDOWN`, `EHOSTDOWN` – síť nefunguje, cílový počítač se neozývá.
- `ENETUNREACH`, `EHOSTUNREACH` – síť nenalezena, cílový počítač neznámý.

```
#include <sys/socket.h>
struct sockaddr_in server;
connect(s, (struct sockaddr*) &server, sizeof(server));
```

listen() – pasivní otevření na straně serveru, server čeká na spojení

```
int listen(int sockfd, int backlog)
```

Funkce nastaví pasivní režim komunikace TCP pro zadanou nepřipojenou schránku, což znamená, že jádro OS přijímá přicházející požadavky na spojení a směřuje je do této schránky.

Naslouchající schránka vytvoří v jádru systému dvě fronty:

- *Frontu neúplných spojení*, která obsahuje záznam pro každý příkaz SYN od klientů. Server se nachází ve stavu `SYN_RCV` a očekává dokončení vytvoření spojení TCP.
- *Frontu úplných spojení*, která obsahuje záznam pro každého klienta, který dokončil ustavení spojení TCP. Server je ve stavu `ESTABLISHED`.

Parametr `backlog` udává maximální počet požadavků na spojení. Je to součet záznamů fronty neúplných a fronty úplných spojení.

Inicializaci spojení TCP včetně práce nad frontami zajišťuje jádro OS ve spolupráci s knihovnou BSD sockets. Serverová aplikace se o to nemusí starat.

```
#include <sys/socket.h>
listen(s, 5);
```

accept() – přijetí spojení, ustavení komunikace

```
int accept(int sockfd, struct sockaddr *cliaddr, socklen_t *addrlen)
```

Funkce `accept()`, kterou volá server, vybere požadavek z fronty čekajících spojení a vytvoří spojení klient-server. Pokud je fronta požadavků prázdná, serverový proces je převeden do stavu `sleep`. Funkce `accept()` je blokující.

Důležité jsou parametry funkce. Prvním parametrem funkce je schránka, na které server poslouchá (tzv. listening socket). Druhým parametrem funkce `accept()` vrací adresu a

port klienta, který se připojil k danému serveru. Takto můžeme například zjistit, odkud se daný klient připojil a tuto informaci vypsát.

Návratovou hodnotou funkce `accept()` je deskriptor nové schránky (tzv. connected socket), kterou automaticky vytvoří systém pro právě navázané spojení. Tato schránka zaniká poté, co se klient odpojí od serveru. Původní schránka však zůstává připojena a čeká na další spojení.

Pokud nepotřebujeme zjistit adresu a port připojeného klienta, nastavíme hodnotu druhého a třetího parametru na `NULL`. Server TCP může obecně získat informace o klientovi (IP adresu a číslo portu) pomocí parametrů funkce `accept()`, případně funkci `getpeername()`. Informaci o aktuální schránce (například číslo lokálně přiděleného portu) lze získat funkcí `getsockname()`.

```
#include <sys/socket.h>
int fd, newsock;
char buff[MAXLINE];
pid_t pid;
newsock = accept(fd, (struct sockaddr *)&from, &len);
if (newsock > 0)
{
    printf("Connection from %s, port %d\n", inet_ntop(AF_INET, &from.sin_addr,
        buff, sizeof(buff)), ntohs(from.sin_port));
    if ((pid=fork()) == 0) /* child process */
    {
        close(fd);          /* decrement no. of references on fd */
        doit(newsock);
        close(newsock);
        exit(0);
    }
    close(newsock);        /* server process */
                          /* decrement no. of references on newsock */
}
```

Výměna dat

Pro přenos dat se používají funkce `read()` a `write()`, které se používají i pro čtení a zápis do souboru. Pro čtení a zápis dat do schránek lze použít i další funkce, například `recv()` a `send()`, které oproti funkcím `read()` a `write()` dovolují nastavit pokročilé parametry spojení, viz tabulka 2.5.

Příznak	Specifikace
MSG_DONTROUTE	přeskoč směrování, cíl je na lokální síti
MSG_DONTWAIT	neblokující operace čtení/zápisu
MSG_OOB	čtení/zápis data mimo pásmo (out-of-band), pouze 1B pro TCP
MSG_PEEK	nahlédnutí do příchozí fronty bez odebrání dat
MSG_WAITALL	při čtení se čeká na všechna data

Tabulka 2.5: Příznaky pro funkce `recv()`, `send()`.

Při čtení dat ze schránky TCP si musíme uvědomit, že komunikace TCP představuje je proud bytů, kde nejsou žádné bloky. Odpověď z funkce `read()` může být kratší, než je

velikost bufferu, do které se data ukládají. Proto je třeba provádět čtení opakovaně, dokud funkce nevrátí hodnotu 0 (EOF, end of file) nebo -1 (chyba čtení).

Také při zápisu dat je důležité kontrolovat návratovou hodnotu funkce `write()` či `send()`. V případě přerušení již navázaného spojení bychom zapsali data do schránky, ale ta by nebyla odeslána, protože se spojení přerušilo. Bez kontroly návratové hodnoty toto nezjistíme a program pak může čekat ve funkci `read()` na odpověď od druhé strany. Druhá strana samozřejmě odpověď nepošle, neboť dotaz nebyl vůbec odeslán.

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t nbytes);
ssize_t write(int fd, const void *buf, size_t nbytes);

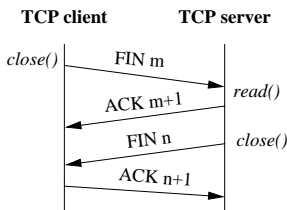
#include <sys/types.h>
#include <sys/socket.h>

ssize_t recv(int sockfd, void *buf, size_t len, int flags);
ssize_t send(int sockfd, const void *msg, size_t len, int flags);
```

Ukončení spojení

Ukončení spojení TCP obvykle iniciuje klient (tzv. aktivní uzavření). Z pohledu serveru se jedná spíše o jedná pasivní uzavření komunikace, tj. když klient požádá o ukončení, tak mu to umožním.

Pro uzavření se používají funkce `close()` a `shutdown()`, na něž někteří autoři síťových aplikací zapomíná. Pokud dojde k nekorektnímu uzavření spojení (například ukončíme klienta signálem KILL), zůstávají stále alokovány systémové prostředky pro komunikaci (například používaný port). Toto trvá tak dlouho, než dojde k vypršení časového limitu čekající na odpověď od druhé strany. Velikost tohoto limitu označovaného jako MSL (maximum segment lifetime) závisí na implementaci. Doporučená hodnota je dvě minuty [6]. Výpočet hodnoty závisí na více parametrech a může se pohybovat od jedné do čtyř minut podle typu spojení. Korektní ukončení TCP spojení zahrnuje výměnu čtyř paketů TCP s příznaky FIN m , ACK $m+1$, FIN n , ACK $n+1$, viz obrázek 2.9.



Obrázek 2.9: Korektní ukončení spojení TCP.

close() – uzavření shránky

```
int close(int sockfd)
```

Funkce **close()** uzavře shránku a ukončí spojení TCP. Ukončení spojení může trvat delší dobu, protože na cestě mohou být ještě nějaká data. Teprve po standardní uzavírací proceduře (sekvenci čtyř paketů TCP s výše uvedenými příznaky) je spojení skutečně ukončeno. Pokud počet referencí na shránku je větší než nula, spojení se fyzicky neuzavře, pouze se sníží počet referencí na shránku.

```
#include <unistd.h>
close(s);
```

Server uzavírá spojení funkcí také funkcí **close()**. Poslední paket s příznakem FIN se předá aplikaci jako znak EOF (End Of File), po němž funkce **read()** vrátí hodnotu 0.

shutdown() – okamžité uzavření shránky

```
int shutdown(int sockfd, int howto)
```

Narozdíl od funkce **close()** iniciuje funkce **shutdown()** normální uzavření spojení TCP (příkazem FIN) bez ohledu na počet referencí na shránku. Funkce **close()** uzavírá obě strany spojení TCP, které je plně duplexní (tj. souběžně probíhá čtení i zápis v rámci jednoho spojení). Funkce **shutdown()** dovoluje uzavřít pouze jednu stranu komunikace. Závisí to na parametru **howto**, který může obsahovat následující hodnoty:

SHUT_RD: Uzavření shránky pro čtení. Žádná další data nemohou být přijata do shránky.

SHUT_WR: Uzavření shránky pro zápis. Data určená k poslání jsou odeslána spolu s ukončovací sekvencí (FIN).

SHUT_RDWR: Uzavření shránky pro čtení i zápis.

```
#include <sys/socket.h>
shutdown(s, SHUT_RDWR);
```

Při vytváření serveru TCP rozlišujeme dva typy serverů: iterativní server TCP, který zpracovává požadavky postupně, a konkurentní server TCP, který je zpracovává souběžně. V praxi se většinou vytváří konkurentní servery, které umožňují zpracovat více příchozích požadavků najednou pomocí duplikace procesu serveru funkcí **fork()**.

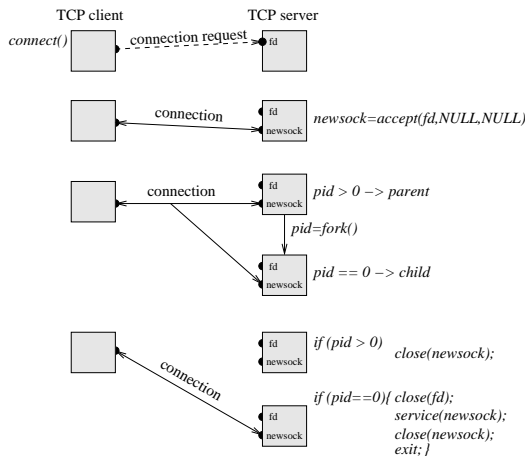
Algoritmus práce iterativního serveru TCP

1. Otevři shránku a spoj jej s požadovaným portem – **bind()**.
2. Přepni shránku do pasivního režimu (pasivní čekání) – **listen()**.
3. Přijmi nový požadavek a vytvoř novou shránku pro spojení s klientem – **accept()**.
4. Vyměňuj si zprávy s klientem – **read()**, **write()**.
5. Ukonči spojení – **close()** – a čekej na nový požadavek (návrat na krok 3).

Algoritmus práce konkurentního serveru TCP

1. Otevři schránku a spoj jej s požadovaným portem – `bind()`.
2. Přepni schránku do pasivního režimu (pasivní čekání) – `listen()`.
3. Přijmi nový požadavek funkcí, vytvoř nové spojení – `accept()`.
4. Pro nové spojení vytvoř nový proces – `fork()`.
5. Vyměňuj si zprávy s klientem – `read()`, `write()`.
6. Uzavři schránku a ukonči synovský proces – `close()`, `exit()`.

Funkce `fork()` je standardní systémovou funkcí knihovny `<sys/types.h>` a `<unistd.h>`. Její volání způsobí vytvoření nového procesu. Nový proces je přesnou kopií volajícího (rodičovského procesu) až na číslo procesu (PID, process identification) a číslo procesu otce (PPID, parent process identification). Při použití je potřeba si uvědomit jednu důležitou skutečnost – přestože zavoláme funkci `fork()` v programu jenom jednou, funkce vrátí na jedno zavolání dvě různé návratové hodnoty. Jednu hodnotu předá volajícímu rodičovskému procesu, kde vrací PID nově vytvořeného procesu. Druhá návratová hodnota je určena pro vzniklého potomka a má hodnotu PID 0.



Obrázek 2.10: Navazování spojení u konkurentního serveru TCP

Z pohledu komunikace je důležité připomenout, že všechny deskriptory otevřené rodičovským procesem před voláním funkce `fork()` jsou předány i novému potomku. Ve výše uvedeném příkladě jde například o nově vytvořenou schránku `newsock`. Proto je obvyklé, že otcovský proces novou schránku uzavře, neboť ji nepoužívá. Na druhé straně se očekává, že

také synovský proces uzavře původní schránku `s`, na které naslouchá rodičovský proces, a bude pracovat pouze s nově vytvořenou schránkou `newsock`, viz obrázek 2.10.

V případě vidíme, že klient TCP se připojuje (voláním funkce `connect()` na čekající schránku `fd()`). Nedojde však k úplnému navázání spojení na tuto schránku, neboť funkce `accept()` vrátí novou schránku, která bude obsluhovat nově příchozí spojení. K nově vytvořené schránce `newsock` je navázáno spojení od klienta. Sserver i klient přejdou do stavu `ESTABLISHED`. Po zavolání funkce `fork()` se vytvoří kopie procesu (včetně obou otevřených schránek). Původní proces pak uzavře novou schránku, neboť ji používá nový proces, který obsluhuje připojeného klienta TCP.

Zpracování signálů `SIGCHLD` při ukončování paralelních procesů

Obecně platí, že pokud nějaký proces ukončí svou činnost, uloží se stav jeho ukončení do systémové datové struktury. Ta obsahuje PID ukončeného procesu, stav ukončení a využití zdrojů (CPU, paměť apod.). Zároveň pošle synovský proces signál `SIGCHLD` svému rodičovskému procesu, kterým oznamuje změnu stavu, tj. své ukončení. Pokud tento signál rodičovský proces nepřevzme a nezpracuje, nemůže se synovský proces korektně ukončit. Je převeden do speciálního stavu `zombie` či `defunct`, ve kterém zůstává do doby, než si otcovský proces vyzvedne data o ukončení potomka pomocí funkce `wait()` nebo `waitpid()`. Funkce `wait()` je blokující a čeká na ukončení zadaného procesu. Zavolá se při ukončení prvního potomka. V následujícím příkladu je ukázán způsob obsluhy signálu `SIGCHLD`:

```
void sig_chld(int signo){
    pid_t pid;
    int stat;
    pid = wait(&stat);
    printf("child %d terminated\n",pid);
    return;
}
```

Funkce `waitpid()` je obecnější, neboť umožňuje definovat, na jaký proces má funkce čekat a zda má být čekání blokující či ne (volba `WNOHANG`).

Pokud skončí otcovský proces dříve, než zavolá funkci `wait()`, nastaví si jeho potomci `PPID` (PID rodičovského procesu) na hodnotu 1, což označuje inicializační řídicí proces `init`. Proces `init` počká na skončení těchto procesů a zruší jejich stav `zombie`. Jestliže se chceme vyhnout vzniku procesů `zombie`, které zůstávají v paměti a blokují uvolnění již nepotřebných systémových prostředků, musíme po funkci `fork()` provést funkci `wait()`, která nastaví zpracování signálu `SIGCHLD`, viz výše.

Funkce `wait()` čeká pouze na ukončení prvního potomka. Pokud skončí více potomků současně (souběh více signálů `SIGCHLD`), zpracuje se funkcí `sig_chld()` pouze první. Ostatní se zahodí, neboť Unix neukládá zasláné signály do fronty. To znamená, že takto ukončené procesy neskončí korektně a zůstávají ve stavu `zombie`. V této situaci je pak korektní použít funkci `waitpid()`, která čeká v neblokující smyčce na všechny své potomky (parametr `-1`), dokud se neukončí, viz následující příklad:

```
void sig_chld(int signo){
    pid_t pid;
    int stat;
    while (pid = waitpid(-1,&stat,WNOHANG)) > 0)
        printf("child %d terminated\n",pid);
    return;
}
```

Další možností je nastavit ignorování signálu SIGCHLD, který také zamezí blokování ukončovaného procesu ve stavu zombie – viz následující příklad:

```
...
pid_t pid; long p;
struct sigaction sa;
sa.sa_handler = SIG_IGN;    // způsob obsluhy signálu - ignorování
// sa.sa_handler = sig_chld; // možnost obsloužit signál např. funkcí sig_chld()
sa.sa_flags = 0;
sigemptyset(&sa.sa_mask);
if (sigaction(SIGCHLD, &sa, NULL) == -1) // obsluha signálu SIGCHLD
    err(1, "sigaction()");
...
if ((connectfd = accept(listenfd, NULL, NULL)) == -1)
    err(1, "accept()");
if ((pid = fork()) > 0) { /* parent */
    printf("close(connectfd)\n");
    close(connectfd);
} else if (pid == 0) { /* child */
    p = (long)getpid();
    ...
    exit(0);
} else
    err(1, "fork()");
}
```

Příklad klienta TCP

```
...
#define BUFFER (1024)
int main(int argc, char *argv[])
{
    int connectfd;
    struct sockaddr_in server;
    struct hostent *hostent;
    struct servent *servent;
    if (argc != 3)
        errx(1, "Usage: %s <address> <port>", getprogname());
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    if ((hostent = gethostbyname(argv[1])) == NULL)
        errx(1, "gethostbyname(): %s", hstrerror(h_errno));
    memcpy(&server.sin_addr, hostent->h_addr, hostent->h_length);
    if ((servent = getservbyname(argv[2], "tcp")) != NULL)
        server.sin_port = servent->s_port;
    else
        server.sin_port = htons(atoi(argv[2]));
    if ((connectfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        err(1, "socket()");
    if (connect(connectfd, (struct sockaddr *)&server, sizeof(server)) == -1)
        err(1, "connect()");
    service(STDIN_FILENO, connectfd);
    close(connectfd);
    return 0;
}
```

Příklad konkurentního serveru TCP

```

...
#define BUFFER (1024)
#define QUEUE (2)
int main(int argc, char *argv[])
{
    int listenfd, connectfd;
    struct sockaddr_in server;
    struct hostent *hostent;
    struct servent *servent;
    pid_t pid; long p;
    struct sigaction sa;

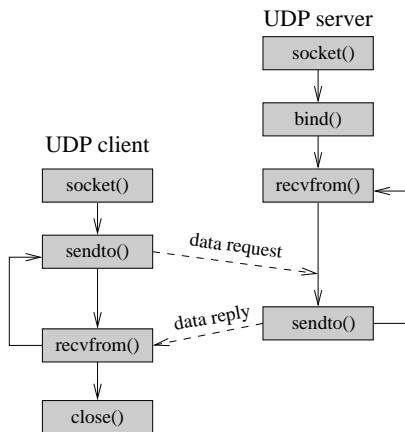
    if (argc != 3) errx(1, "Usage: %s <ANY|address> <port>", getprogname());
    sa.sa_handler = SIG_IGN; sa.sa_flags = 0;
    sigemptyset(&sa.sa_mask);
    if (sigaction(SIGCHLD, &sa, NULL) == -1) err(1, "sigaction()");
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    if (strcasecmp(argv[1], "any") == 0)
        server.sin_addr.s_addr = htonl(INADDR_ANY);
    else {
        if ((hostent = gethostbyname(argv[1])) == NULL)
            errx(1, "gethostbyname(): %s", hstrerror(h_errno));
        memcpy(&server.sin_addr, hostent->h_addr, hostent->h_length);
    }
    if ((servent = getservbyname(argv[2], "tcp")) != NULL)
        server.sin_port = servent->s_port;
    else
        server.sin_port = htons(atoi(argv[2]));
    if ((listenfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        err(1, "socket()");
    if (bind(listenfd, (struct sockaddr *)&server, sizeof(server)) == -1)
        err(1, "bind()");
    if (listen(listenfd, QUEUE) == -1)
        err(1, "listen()");
    while (1) {
        if ((connectfd = accept(listenfd, NULL, NULL)) == -1)
            err(1, "accept()");
        if ((pid = fork()) > 0) { /* parent */
            close(connectfd);
        } else if (pid == 0) { /* child */
            p = (long)getpid();
            close(listenfd);
            service(connectfd);
            close(connectfd);
            exit(0);
        } else
            err(1, "fork()");
    }
    close(listenfd);
    return 0;
}

```

Všimněte si na příkladu, že u každé síťové funkce (např. `connect()`, `bind()`, `listen()`) se testuje návratová hodnota. Důvod je jednoduchý – pokud dojde během komunikace k přerušení linky, bez testování návratové hodnoty nepoznáme, že operace neproběhla úspěšně a můžeme čekat na data z nefunkční linky.

2.2.3 Programování komunikace nad UDP

Spojení UDP je jednodušší než TCP. Klient nemusí vytvářet spojení se serverem. Místo toho pouze vyšle datagram UDP na server pomocí funkce `sendto()`. Podobně ani server nepotřebuje očekávat na otevření spojení od klienta. Zavolá pouze funkci `recvfrom()`, která čeká, dokud data nepřijdou od klienta. Funkce vrací kromě dat také adresu klienta, kam může server poslat odpověď. Schéma komunikace klient-server přes UDP je na obrázku č. 2.11.



Obrázek 2.11: Komunikace klient-server přes UDP.

Pokud klient explicitně nepožádá o přidělení lokálního portu (funkce `bind()`), jádro systému mu přidělí volný port při prvním volání funkce `sendto()` a tento port mu zůstává i nadále.

Mezi hlavní výhody spojení UDP patří:

- *Není potřeba ustanovit spojení.* TCP používá tříázový synchronizační mechanismus, jenž je náročný na vytvoření.
- *Neexistuje stav spojení.* Stav spojení vyžaduje paměť pro odesílání a přijímání, parametry pro řízení zahlcení, parametry pro nastavení sekvenčních čísel a čísel pro potvrzování. Server UDP, který toto neobsahuje, může obsloužit více klientů než server TCP.
- *Menší režie v hlavičce paketu.* TCP obsahuje 20 bytů informací pro spojení v hlavičce (sekvenční číslo, potvrzení, příznak, velikost posuvného okna, kontrolní součet, volba), zatímco UDP pouze 8 bytů (délku a kontrolní součet).

- *Rychlejší řízení odesílání dat aplikací.* Aplikace připraví data UDP a síťová vrstva je okamžitě odešle. Komunikace TCP obsahuje mechanismus řízení zahlcení, který zpožaluje spojení, pokud je některá linka zahlcena. TCP opakovaně posílá pakety, dokud nedostane potvrzení o jejich doručení.

Funkce pro čtení a zápis - `recvfrom()`, `sendto()`

```
#include <sys/types.h>
#include <sys/socket.h>
struct msghdr {
    caddr_t msg_name;      /* optional from address */
    u_int  msg_namelen;    /* size of address */
    struct iovec *msg_iov; /* scatter/gather array of I/O buffers */
    u_int  msg_iovlen;     /* # elements in msg_iov */
    caddr_t msg_control;    /* ancillary data, see below */
    u_int  msg_controllen; /* ancillary data buffer len */
    int    msg_flags;      /* flags on received message */
};
ssize_t recvfrom(int sockfd, void * buf, size_t nbytes, int flags,
    struct sockaddr * from, socklen_t * addrlen);

ssize_t sendto(int sockfd, const void *buf, size_t nbytes, int flags,
    const struct sockaddr *to, socklen_t addrlen);

ssize_t recvmsg(int s, struct msghdr *msg, int flags);
ssize_t sendmsg(int s, const struct msghdr *msg, int flags);
```

První tři parametry funkce jsou stejné jako `read()`, `write()` u TCP. Adresa `from` u čtení je adresa odesílatele. Odpovídá to argumentům funkce `accept()`, které nám říkají, kdo paket poslal. Adresa `to` u funkce `sendto()` označuje adresu cíle a odpovídá argumentům používaným u funkce `connect()`. Hodnota příznaků (`flags`) odpovídá hodnotám v tabulce č.2.6.

Funkce `recvmsg()` a `sendmsg()` umožňují také obsluhovat více bufferů a posílat pomocná data (ancillary data). Jsou to vlastně řídicí informace – např. max. počet skoků, adresa odesílatele, směrovací hlavička, třída přenosu (traffic class), informace o paketu apod.

Pomocí funkce `recvfrom()` může server UDP získat IP adresu a číslo portu odesílatele. Cílovou IP adresu může příjemce získat použitím funkce `recvmsg()`, číslo cílovou portu pomocí funkce `getsockname()`. Funkce `recvfrom()` je blokující a čeká, dokud nepřijdou požadovaná data.

Algoritmus práce iterativního serveru UDP

Většina UDP serverů je iterativních. Čekají na požadavek klienta, zpracují ho, pošlou odpověď a čekají na dalšího klienta. Algoritmus práce iterativního serveru UDP lze popsat následujícím způsobem:

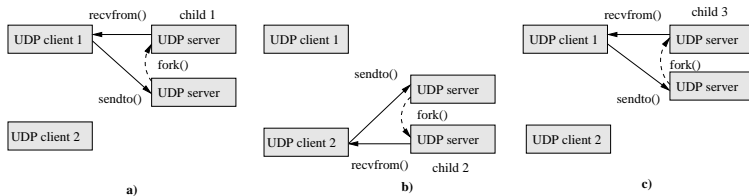
1. Otevří schránku a spoj ji s požadovaným portem – `bind()`.
2. Čti ve smyčce požadavky klientů. Zpracuj požadavek a odešli odpověď klientovi – `recvfrom()`, `sendto()`.

Príznak	Význam
MSG_DONTROUTE	přeskoč směrování, cíl je na lokální síti
MSG_DONTWAIT	neblokující operace čtení/zápisu
MSG_OOB	čtení/zápis data mimo pásmo (out-of-band)
MSG_PEEK	nahlédnutí do příchozí fronty bez odebrání dat
MSG_WAITALL	při čtení se čeká na všechny data
MSG_EOR	Príznak je nastaven, pokud data tvoří konec logického záznamu
MSG_OOB	out-of-band data
MSG_BCAST	Datagram UDP poslán jako broadcast
MSG_MCAST	Datagram UDP poslán jako multicast
MSG_TRUNC	Datagram byl zkrácen
MSG_CTRUNC	Pomocná data (ancillary) byla zkrácena
MSG_NOTIFICATION	Došlá zpráva je oznámení, nikoliv klasická data (SCTP)

Tabulka 2.6: Příznaky pro funkce `recvfrom()`, `sendto()`.

Protože zpracování může trvat delší dobu, určitá forma konkurentnosti je vhodná. U TCP je to jednoduché – server TCP zavolá funkci `fork()` a nový proces zpracuje příchozí požadavek. Není problém se směrování příchozích dat. Spojení je identifikováno jednoznačně.

U jednoduchého konkurentního serveru UDP to není složité – server načte požadavek a vytvoří nový proces. Tento nový proces zpracuje požadavek a vrátí odpověď. Co se však stane, pokud klient pošle svůj požadavek či data ve více paketech UDP? V takovém případě je každý nový paket považován za nový požadavek a vytvoří se pro něj nový proces. Server UDP totiž nedokáže rozlišit, zda klient posílá nový požadavek (který by měl zpracovat v novém procesu) nebo se jedná ještě o data patřící k předchozím datům, viz obrázek 2.12. Na tomto



Obrázek 2.12: Jednoduchý konkurentní server UDP

obrázku můžeme vidět požadavek od klienta č.1, který po příchodu na server je zpracován potomkem č.1 (a). Pokud jiný klient pošle svůj požadavek, je tento požadavek zpracován další kopií serveru, tj. potomkem č.2 (b). Oba potomci se po zpracování požadavku ukončí. Pokud by klient č.1 chtěl poslat ještě další část požadavku v druhém paketu UDP, vytvoří se pro něho nový potomek (c). To je však problém, protože tento potomek nemá nic společného s potomkem, který zpracoval předchozí část dotazu.

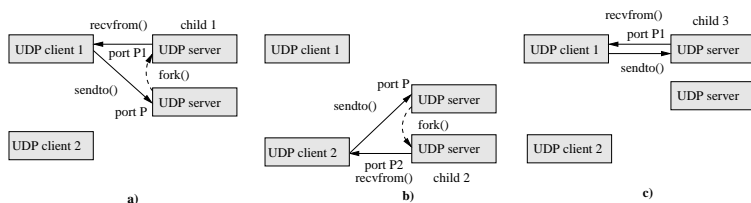
Pokud bychom tedy nerozlišovali spojení a pro každý nový paket UDP vytvořili nový proces, pak jeden požadavek (například uložení souboru na serveru) bude řešit více procesů a každý bude sám o sobě zapisovat jinou část příchozích dat. Což asi není chování, které bychom očekávali.

Jak z toho ven? Pokud víme, že požadavek je obsažen pouze v jednom datagramu (například žádost o zaslání dat), máme vyhráno, neboť požadavek lze zpracovat souběžně za použití volání funkce `fork()` pro obslužný proces. Tento přístup používají například služby DNS či SNMP. Algoritmus zpracování jednoduchého konkurentního serveru je následující.

Algoritmus práce konkurentního UDP serveru

1. Otevři schránku a spoj ji s požadovaným portem – `bind()`.
2. Pomocí funkce `recvfrom()` vyčkej na požadavek klienta.
3. Vytvoř nový proces a novou schránku na novém portu – `fork()`, `socket()`, `bind()`.
4. Z nového procesu odešli odpověď – `sendto()`.
5. Další požadavky přijímej pomocí této nové schránky – `recvfrom()`, `sendto()`.
6. Po ukončení komunikace uzavři schránku a ukonči proces potomka – `close()`, `exit()`.

Jestliže však požadavek na server UDP je obsažen ve více datagramech, používá se jiný přístup. Server vytvoří pro každého nově příchozího klienta novou schránku a pomocí funkce `bind()` ji připojí k novému portu. Obvykle se nejedná o rezervovaný port, na který se původně připojují klienti, ale dynamický port. Tato nová schránka na novém portu se použije pro odpověď klientovi. Od klienta to však vyžaduje, aby se podíval na číslo portu v odpovědi serveru UDP a veškeré další data posílal právě na tento nový port. Toto chování můžeme najít například u TFTP serveru. Příklad chování je na obrázku 2.13. Zde vidíme, že po



Obrázek 2.13: Složitější konkurentní server UDP

navázání spojení v obrázku (a) je zpracování přeneseno na potomka P1, který komunikuje s klientem č.1. Další požadavky od jiných klientů vyřizuje na portu P původní server (b), zatímco požadavky od prvního klienta směřují na server naslouchající na portu P1 (c).

Použití funkce `connect()` u spojení UDP

Další nevýhodou komunikace UDP je skutečnost, že funkce `recvfrom()` je blokující. Když např. klient pošle dotaz na server, který neběží, je klientský proces blokován při čekání na odpověď. To se nazývá jako *asynchronní chyba*. Uváznutí lze použitím funkce `connect()`. Pomocí této funkce můžeme vytvořit tzv. *spojovanou schránku UDP* (connected UDP socket). Použití funkce nemá nic společného s vytvářením třířázové synchronizace, která známe od

TCP. Místo toho jádro pouze otestuje aktuální dostupnost komunikujícího partnera, doplní jeho IP adresu a číslo portu a vrátí řízení volajícímu procesu.

Použití funkce `connect` s sebou přináší určité odlišnosti:

- Při posílání dat nelze zadávat cílovou IP adresu a číslo portu. Místo funkce `sendto()` je potřeba použít funkce `write()` nebo `send()`. Vše, co zapíšeme do vytvořené schránky, je automaticky posláno na adresu a port zadaný ve funkci `connect()`.
- Data lokálně přijímáme funkcí `read()` nebo `recvmsg()` místo `recvfrom()`. Do schránky přicházejí pouze pakety UDP jdoucí z adresy a portu, který byl uveden ve funkci `connect()`. Toto omezuje komunikaci pouze na jednoho partnera.
- Schránka přijímá asynchronní chyby, narozdíl od nespojené schránky UDP.

Obecně řečeno je použití funkce `connect()` pro komunikaci UDP vhodné pouze tehdy, když komunikujeme pouze s jedním partnerem po celou dobu. Toho se využívá například u služby TFTP. V ostatních případech je vhodné se funkcí `connect()` u UDP vyhnout. Je ovšem možné zavolat funkci `connect()` opakovaně, pokud chceme změnit partnera pro komunikaci.

Příklad klienta UDP (s použitím funkce `connect`)

```
#include <sys/types.h>
...
int main(int argc, char *argv[])
{
    int connectfd;
    struct sockaddr_in server;
    struct hostent *hostent;
    struct servent *servent;
    if (argc != 3)
        errx(1, "Usage: %s <address> <port>", getprogname());
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    if ((hostent = gethostbyname(argv[1])) == NULL)
        errx(1, "gethostbyname(): %s", hstrerror(h_errno));
    memcpy(&server.sin_addr, hostent->h_addr, hostent->h_length);
    if ((servent = getservbyname(argv[2], "udp")) != NULL)
        server.sin_port = servent->s_port;
    else
        server.sin_port = htons(atoi(argv[2]));
    if ((connectfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
        err(1, "socket()");
    if (connect(connectfd, (struct sockaddr *)&server, sizeof(server))
        == -1)
        err(1, "connect()");
    service(STDIN_FILENO, connectfd);
    close(connectfd);
    return 0;
}
```

Příklad konkurentního serveru UDP s jednoduchými požadavky

```

#include <sys/types.h>
...
#define BUFFER (1024)
void service(int fd)
{
    int n, r;
    char buf[BUFFER];
    struct sockaddr_in client;
    int length;
    pid_t pid; long p;

    length = sizeof(client);
    if ((n = recvfrom(fd, buf, BUFFER, 0,
        (struct sockaddr *)&client, &length)) >= 0) {
        if ((pid = fork()) > 0) { /* parent */
            /* empty */
        } else if (pid == 0) { /* child */
            p = (long)getpid();
            for (r = 0; r < n; r++)
                if (islower(buf[r])) buf[r] = toupper(buf[r]);
            else if (isupper(buf[r])) buf[r] = tolower(buf[r]);
            r = sendto(fd, buf, n, 0, (struct sockaddr *)&client, length);
            if (r == -1)
                err(1, "sendto()");
            if (r != n)
                errx(1, "sendto(): Buffer written just partially");
            exit(0);
        } else
            err(1, "fork()");
    }
    if (n == -1)
        err(1, "recvfrom()");
}

int main(int argc, char *argv[])
{
    int listenfd;
    struct sockaddr_in server;
    struct hostent *hostent;
    struct servent *servent;
    struct sigaction sa;

    if (argc != 3)
        errx(1, "Usage: %s <ANY|address> <port>", getprogname());

    sa.sa_handler = SIG_IGN;
    sa.sa_flags = 0;
    sigemptyset(&sa.sa_mask);
    if (sigaction(SIGCHLD, &sa, NULL) == -1)
        err(1, "sigaction()");
}

```

```

memset(&server, 0, sizeof(server));
server.sin_family = AF_INET;
if (strcasecmp(argv[1], "any") == 0)
    server.sin_addr.s_addr = htonl(INADDR_ANY);
else {
    if ((hostent = gethostbyname(argv[1])) == NULL)
        errx(1, "gethostbyname(): %s", hstrerror(h_errno));
    memcpy(&server.sin_addr, hostent->h_addr, hostent->h_length);
}
if ((servent = getservbyname(argv[2], "udp")) != NULL)
    server.sin_port = servent->s_port;
else
    server.sin_port = htons(atoi(argv[2]));
if ((listenfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    err(1, "socket()");
if (bind(listenfd, (struct sockaddr *)&server, sizeof(server)) == -1)
    err(1, "bind()");
while (1) {
    service(listenfd);
}
close(listenfd);
return 0;
}

```

2.2.4 Další funkce pro programování síťových aplikací

Funkce pro práci s byty

Pro práci se schránkami potřebujeme funkce, které do nich přiřadí hodnoty, aniž by je interpretovali, to jest, aby k nim přistupovali jako k binárním (neinterpretovaným) datům. Např. IP adresa může obsahovat hodnotu ASCII 0, což jazyk C (a spousta programů) interpretuje jako ukončení řetězce.

Zde uvedeme seznam funkcí, které jsou součástí standardní knihovně `<string.h>` a slouží k manipulaci s byty v jazyce C.

bzero(), bcopy(), bcmp(), memset(), memcpy(), memcmp() – manipulace s byty

```

#include <strings.h>
void bzero(void *dest, size_t nbytes);
void bcopy(const void *src, void *dest, size_t nbytes);
int bcmp(const void *ptr1, const void *ptr2, size_t nbytes);
void *memset(void *dest, int c, size_t len);
void *memcpy(void *dest, const void *src, size_t nbytes);
int memcmp(const void *ptr1, const void *ptr2, size_t nbytes);

```

Funkce **bzero()** nastaví specifikovaný počet bytů na hodnotu ASCII 0. Toho se využívá převážně pro inicializaci schránek. Funkce **bcopy()** kopíruje specifikovaný počet bytů do cílové struktury. Funkce **bcmp()** slouží k porovnání dvou řetězců bytů (neinterpretovaných řetězců).

Funkce **memset()** nastaví daný počet bytů na hodnotu uvedenou v parametru **c**. Funkce **memcpy()** kopíruje byty, podobně jako **bcopy()**. Funkce **memcmp()** porovnává řetězce bytů.

Jak je vidět, tak funkce `memset()`, `memcpy()` a `memcmp()` provádějí stejnou činnost jako funkce `bzero()`, `bcopy()`, resp. `bcmp()`, které patří mezi původní funkce knihovny. Dnes se doporučuje používat funkce `memset()`.

inet_aton(), inet_addr(), inet_ntoa(), inet_pton(), inet_ntop() – konverze IP adres

```
#include <arpa/inet.h>
int inet_aton(const char *str, struct in_addr *addr);
in_addr_t inet_addr(const char *str);
char *inet_ntoa(struct in_addr inaddr);

int inet_pton(int family, const char *str, void *addr);
const char *inet_ntop(int family, const void *addr, char *str, size_t len);
```

Tyto funkce slouží pro převod IP adres (verze 4) v tečkové notaci (např. 147.229.8.12) na 32-bitové číslo v síťovém uspořádání tak, jak je reprezentované v hardwaru. Funkce `inet_aton()` konvertuje řetězec ASCII do síťového tvaru (aton, ASCII to network). Funkce `inet_addr()` dělá to stejné, v současné době se už ale nepoužívá.

Funkce `inet_ntoa()` provádí opačnou konverzi, tj. převádí IP adresu v síťovém tvaru na řetězcovou reprezentaci s tečkami.

Funkce `inet_pton()` a `inet_ntop()` dělají totéž, pracují však s adresami IPv4 i IPv6. Prvním argumentem funkce je typ `AF_INET` nebo `AF_INET6`.

Funkce pro zjišťování IP adres, doménových adres, portů

Pro konverzi doménových jmen a IP adres se využívá zejména systému doménových jmen DNS (Domain Name System), který uchovává mapování těchto adres mezi sebou. Tyto informace lze také zjistit z jiných zdrojů – např. souboru `/etc/hosts`, ze systému NIS (Network Information System) či LDAP (Lightweight Directory Access Protocol).

gethostbyname(), gethostbyaddr() – převod doménové adresy a IP adresu

Funkce `gethostbyname()` zjišťuje IP adresu z doménového jména. V podstatě se dotazuje na záznam typu A v DNS. Funkce `gethostbyaddr()` provádí opačnou konverzi, tj. z IP adresy získá doménové jméno. Načítá tedy záznam PTR z DNS.

```
#include <netdb.h>
struct hostent {
    char    *h_name;           /* official (canonical) name of host */
    char    **h_aliases;       /* alias list */
    int     h_addrtype;         /* host address type: AF_INET */
    int     h_length;           /* length of address: 4 */
    char    **h_addr_list;     /* list of IPv4 addresses from name server */
};

struct hostent * gethostbyname(const char *name);
struct hostent * gethostbyaddr(const char *addr, socklen_t len, int
family);

/* příklad volání */
struct servent *sptr;
sptr=getservbyname("domain","udp"); /* dotaz na port DNS */
sptr=getservbyname("ftp",NULL);      /* dotaz na port ftp */
sptr=getservbyport(htons(53),"udp"); /* dotaz na službu DNS */
```

getservbyname(), getservbyport() – převod jména služby a čísla portu

Funkce **getservbyname()** zjišťuje číslo portu ze jména služby podle informací uvedených v `/etc/services`. Prvním parametrem je jméno služby (např. "ftp"), druhým je typ protokol ("tcp", "udp"). Pokud má služba pro oba typy protokolu stejné číslo portu, uvádí se místo protokolu hodnota NULL.

```
#include <netdb.h>
struct servent {
    char    *s_name;           /* official name of service */
    char    **s_aliases;       /* alias list */
    int      s_port;           /* port number, network byte order */
    char     s_proto;          /* protocol to use */
};
struct servent *getservbyname(const char *name, const char *proto);
struct servent *getservbyport(int port, const char *proto);
```

getaddrinfo – obecný převod jmen a adres

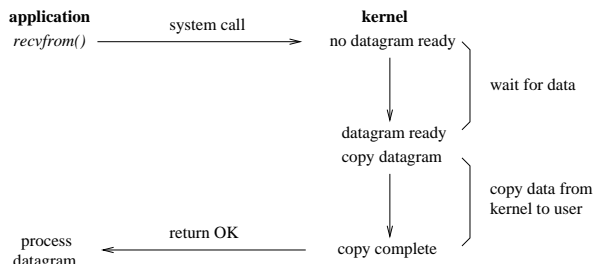
Funkce provádí překlad doménového jména na IP adresu (verze 4 i 6) a převod služby na port.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
struct addrinfo {
    int      ai_flags;         /* AI_PASSIVE, AI_CANONNAME, AI_NUMERICHOST */
    int      ai_family;        /* PF_xxx */
    int      ai_socktype;      /* SOCK_xxx */
    int      ai_protocol;      /* 0 or IPPROTO_xxx for IPv4 and IPv6 */
    size_t   ai_addrlen;       /* length of ai_addr */
    char     *ai_canonname;    /* canonical name for nodename */
    struct sockaddr *ai_addr;  /* binary address */
    struct addrinfo *ai_next;  /* next structure in linked list */
};
int getaddrinfo(const char *nodename, const char *servname,
    const struct addrinfo *hints, struct addrinfo **res);
```

2.3 Vytváření neblokujících aplikací

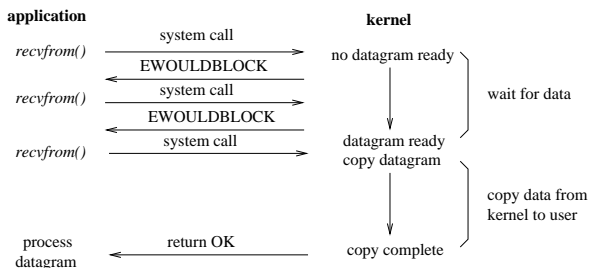
Základní vlastností serveru by mělo být, že při obsluze aktuálního požadavku neodmítne další požadavek od nového klienta. Kritickou operací na straně serveru je funkce pro příjem spojení **accept()** u TCP nebo čekání na příchozí paket funkcí **recvfrom()** u UDP. Dále jsou funkce pro čtení dat **read()** a **recv()** na straně klienta a serveru. Všechny tyto funkce, tj. **accept()**, **recvfrom()**, **read()**, **recv()** nazýváme *blokující funkce*, neboť způsobují blokování aplikací.

Princip blokování je znázorněn na obr. 2.14. Po zavolání např. funkce **recvfrom()** se předá řízení jádru operačního systému, který čeká na data na síťové kartě určená pro aplikaci. Když data přijdou, jádro je zkopíruje do fronty příchozích dat schránky a předá řízení zpět procesu, který funkci **recvfrom()** zavolal.

Obrázek 2.14: Blokující operace `recvfrom()`

2.3.1 Neblokující schránky

Jedním ze způsobů, jak předejít zablokování aplikace například při operaci čtení ze schránky, je nastavit schránku jako *neblokující*. Tím říkáme jádru operačního systému, aby proces, který čeká na data, jež nejsou na síťovém rozhraní k dispozici, místo převedení do stavu *sleep* zaslal procesu chybovou zprávu `EWouldBlock` a předal mu zpět řízení, viz obr. 2.15.



Obrázek 2.15: Tok řízení u neblokující schránky

Pro načtení dat může proces opakovaně zavolat operaci čtení ze schránky. Opakovanému volání funkce `recvfrom()` v neblokující schránce se říká vyzývání (polling). Tato operace je neblokující, nicméně spotřebovává procesorový čas. Proto se tento způsob zpracování příliš nepoužívá. Lze ho však využít třeba u jiných funkcí, např. `connect()` či `accept()`, kde by čekání na spojení mohlo způsobit blokování.

Neblokující chování schránky nastavujeme funkcí `fcntl()`. Tato funkce obecně nastavujeme vlastnosti deskriptoru (viz `man fcntl`). Pomocí příkazu `F_SETFL` ve funkci `fcntl()` nastavíme požadované vlastnosti (flags). Neblokující schránka musí mít nastavenou vlastnost `O_NONBLOCK`, jak je ukázáno v následujícím příkladu.

```
funkce int fcntl(int fd, int cmd ...);
```

Příklad:

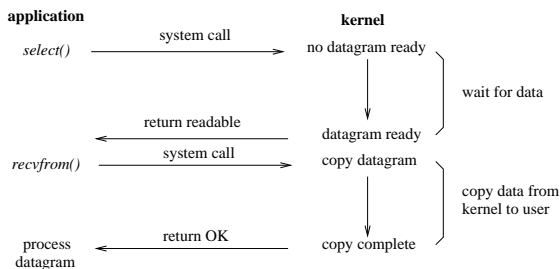
```
#include <fcntl.h>
...
int flags;
int s = socket(AF_INET, SOCK_STREAM, 0);
...
if ((flags=fcntl(s, F_GETFL, 0)) < 0) // zjistíme aktuální nastavení příznaků
    // error
flags |= O_NONBLOCK;                // přidáme příznak neblokující schránky
if (fcntl(fd, F_SETFL, flags) < 0)
    // error
...
```

2.3.2 Souběžný přístup k více schránkám – multiplexing

Druhým způsobem, jak vytvářet neblokující aplikace, je využití techniky souběžného přístupu k více schránkám, tzv. *multiplexing*. K tomu lze použít funkce `select` nebo `poll`. Obě funkce umožňují sledovat více schránek v rámci jednoho procesu, aniž by došlo k blokování. Multiplexing se často používá v těchto případech:

- Klient obsluhuje více deskriptorů (např. standardní vstup a komunikační schránky).
- TCP server současně obsluhuje naslouchající schránku a komunikující schránku.
- Server zpracovává současně spojení TCP a UDP.
- Server zpracovává více služeb a více protokolů (např. `inetd`).

Funkce `select()` a `poll()` předchází blokování tím, že před zavoláním funkce pro čtení otestují, zda jsou připravena data ke čtení, viz obrázek 2.16. Funkce `select()` oznámí jádru operačního systému, že čeká na I/O události na zařízeních specifikovaných deskriptory. Pokud se na některém z těchto zařízení objeví určitá událost, funkce vrátí řízení procesu. Pokud ne, řízení se vrací po vypršení časového limitu funkce `select()` zadaného při jejím volání.



Obrázek 2.16: Multiplexování

Funkce `select()` a `poll()` řeší blokování funkcí čtení, nikoliv konkurentní zpracování požadavků. Pokud chceme vytvořit konkurentní server, který zpracovává více požadavků z více schránek najednou, musí server po výběru schránky s připravenými daty zavolat ještě funkci `fork()`.

```
#include <sys/select.h>
int select(int nfd, fd_set *readfds, fd_set *writefds, fd_set *exceptfds,
           struct timeval *timeout);
FD_SET(fd, &fdset); // makro pro nastavení schránky v množině schránek
FD_CLR(fd, &fdset); // makro pro zrušení výběru schránky v dané množině
FD_ISSET(fd, &fdset); // makro pro otestování nastavení schránky
FD_ZERO(&fdset); // makro pro vynulování množiny sledovaných schránek
```

Na výše uvedeném příkladu vidíme, že funkce `select()` pracuje nad množinou (polem) schránek, přesněji řečeno nad množinou deskriptorů. Funkce testuje, zda jsou tyto schránky připraveny pro čtení `readfd`, zápis `writefd` či zpracování dat mimo pásmo (out-of-band data) `exceptfd`. Argument `nfd`s udává počet testovaných schránek. Jeho hodnota je odvozena z hodnoty maximálního deskriptoru plus jedna. Hodnota `timeout` udává, jak dlouho funkce `select()` čeká na odpověď od schránek. Jsou tři možnosti:

blokování – funkce vrací řízení, až když je některá ze schránek připravena k operaci čtení či zápis; argument `timeout` je `NULL`

čekání určenou dobu – pokud není žádná ze schránek připravena, počká maximálně `timeout` sekund; `timeout.tv_sec = max_sec`

zpracování bez čekání – funkce vrátí řízení okamžitě po otestování schránek; `timeout.tv_sec = 0`, `timeout.tv_usec=0`

Funkce `select()` vrátí počet vybraných schránek. Pokud dojde k vypršení časového limitu, vypíše hodnotu 0, v případě přerušení či chyby hodnotu -1. Množinu schránek vybraných funkcí `select()` je potřeba otestovat např. makrem `FD_ISSET`. To vrátí nenulovou hodnotu v případě, že daná schránka je připravena pro zadanou operaci, nebo hodnotu nula, pokud není.

Příklad aplikace, která čte ze dvou schránek:

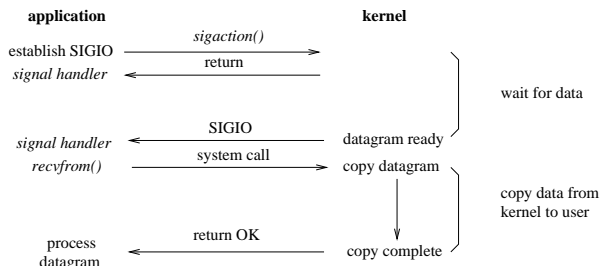
```
fd_set rset; // množina sledovaných deskriptorů
int s1,s2; // schránky
struct timeval wait; // timeout

for (;;) {
    wait.tv_sec = 1; // timeout nastaven na 1 sekundu
    wait.tv_usec = 0;
    FD_ZERO(&rset); // vynulování množiny sledovaných deskriptorů
    FD_SET(s1,&rset); // nastavení schránky s1 do množiny sledovaných deskriptorů
    FD_SET(s2,&rset);
    nb = select(FD_SETSIZE,&rset,NULL,NULL,&wait);
    if (nb <= 0) {
        // error
    }
    if (FD_ISSET(s1,&rset)) {
        // čti ze schránky s1
    }
    if (FD_ISSET(s2,&rset)) {
        // čti ze schránky s2
    }
}
```

2.3.3 Řízení I/O operací pomocí signálu SIGIO

Posledním zmiňovaným modelem, který řeší neblokující komunikaci, je použití signálu SIGIO pro zaslání zprávy o příchozích datech.

Nejprve se pomocí funkce **sigaction** se nastaví obsluha signálu, tj. určí se, která funkce se zavolá po přijetí signálu (tzv. **handler**). V případě příchodu I/O požadavku vygeneruje jádro operačního systému událost (signál), která je zachycena obsluhující funkcí. Tato funkce pak zavolá např. funkci **recvfrom** pro načtení připravených dat - viz obr. 2.17



Obrázek 2.17: Neblokující čtení ze schránky pomocí signálu SIGIO

Použití signálu SIGIO zahrnuje tři kroky:

1. nastavení obslužné rutiny signálu pomocí funkce **signal()** či **sigaction()**,
2. nastavení vlastníka schránky, tj. procesu, který obdrží signál SIGIO – funkce **fcntl()**
3. povolení přijímání asynchronních I/O signálů – funkce **fcntl()**

Následující příklad demonstruje použití signálu pro řešení blokování. V příkladu je použit starší zápis obsluhy signálu pomocí funkce **signal()**. Novější standard POSIX doporučuje používání funkce **sigaction()**, jejíž struktura je obecnější a přehlednější, než použití funkce **signal()**. Příklad použití funkce **sigaction** je například u konkurentního UDP serveru na stránce 72.

```

#include <fcntl.h>
...
int io_handler(int signo); // obslužná rutina pro zpracování signálu
...
signal(SIGIO,io_handler); // nastavení obsloužení signálu SIGIO funkcí io_handler()
...
if (fcntl(fd, F_SETOWN, getpid()) < 0) { // nastavení vlastníka schránky
    perror("fcntl F_SETOWN");
    exit(1);
}
...
if (fcntl(fd, F_SETFL, O_ASYNC) < 0) { // nastavení přijetí asyn. signálů
    perror("fcntl F_SETFL, O_ASYNC");
    exit(1);
}
  
```

Použití řízení I/O operací u komunikace UDP je vcelku jednoduché. Signál je generován, pokud přijde nový datagram do schránky nebo když se objeví asynchronní chyba. Obě informace nám vrátí volání funkce `recvfrom()`. V praxi se využívá odchylení signálu SIGIO například u serveru NTP (Network Time Protocol).

U komunikace TCP je použití signálu SIGIO složitější a proto se nedoporučuje. Důvodem je, že tento signál je generován během příchodu dat příliš často. Při zpracování pak nevíme, co ho způsobilo. Signál je generován, když přijdou nová data, když přijde potvrzení o odeslaných datech, když byla odeslána odeslána, když přišel požadavek na uzavření schránky a podobně. Bohužel neexistuje způsob, jak odlišit jednotlivé události.

2.4 Komunikace typu broadcast a multicast

Prozatím jsme se bavili o programování komunikace TCP a UDP vytvořené mezi dvěma entitami (tj. unicastový přenos). Schránky BSD umožňují také zasílat a přijímat data se skupinovým adresováním (multicast) či všesměrovým adresováním (broadcast) na úrovni IP vrstvy. Samozřejmě musí přenosy typu broadcast i multicast podporovat síťové médium, což je běžné u Ethernetu. U technologií typu Frame Relay to může být problém.

Pro vytvoření broadcastové či multicastové komunikace se používají převážně schránky typu `SOCK_DGRAM`. Pro vytvoření broadcastové a multicastové komunikace musíme u schránku nastavit speciální chování. Využívají se proto funkce `getsockopt()` a `setsockopt()`, které si popíšeme v následujícím textu.

2.4.1 Funkce pro nastavení vlastností schránek

Pomocí funkcí `setsockopt()`, resp. `getsockopt()` je možné nastavit, resp. načíst různé vlastnosti schránek. Volání funkcí má následující parametry:

```
#include <sys/socket.h>
int getsockopt(int s,int level,int optname,void *optval,socklen_t *optlen);
int setsockopt(int s,int level,int optname,const void *optval, socklen_t optlen);
```

kde `s` je deskriptor schránky, `level` specifikuje protokolovou vrstvu, na níž se nastavená vlastnost uplatní, `optname` je název vlastnosti, `optval` je hodnota (nastavovaná či získaná) a `optlen` je délka hodnoty.

název vrstvy	popis
SOL_SOCKET	úroveň schránky (libovolný protokol)
IPPROTO_IP	úroveň protokolu IPv4
IPPROTO_ICMPV6	úroveň protokolu ICMPv6
IPPROTO_IPV6	úroveň protokolu IPv6
IPPROTO_TCP	úroveň protokolu TCP
IPPROTO_SCTP	úroveň protokolu SCTP

Tabulka 2.7: Nastavení vlastností schránek pro různé úrovně protokolů

Protokolová vrstva `level` určuje, na jaké úrovni bude schránka pracovat. Možné hodnoty jsou uvedeny v tabulce 2.7. Např. u nejběžnějšího typu `SOL_SOCKET` definovaném v

knihovně `<sys/socket.h>` můžeme nastavit nahrávání ladících informací (`SO_DEBUG`), znovupoužití adresy (`SO_REUSEADDR`), ignorování směrování (`SO_DONTROUTE`), posílání broadcastových zpráv (`SO_BROADCAST`), nastavení velikosti bufferů (`SO_SNDBUF`, `SO_RCVBUF`) a další (viz `man setsockopt`). U protokolu IP (typ `IPPROTO_IP`) můžeme nastavit například vlastnosti pro multicastový přenos `IP_MULTICAST_IF`, `IP_MULTICAST_TTL`, `IP_ADD_MEMBERSHIP`, hodnoty položek v hlavičce IP, např. `IP_TOS`, `IP_TTL`. Pro schránku TCP lze nastavit například maximální délku segmentu `TCP_MAXSEG`. Podrobný výčet jednotlivých možností nastavení může čtenář najít například v [9].

Pro zjištění nastavených vlastností schránky se využívá funkce `getsockopt()`. Na následujícím příkladu je ukázka použití této funkce. Funkce zjišťuje typ zadané schránky (zda je určena pro přenos UDP či TCP). Výsledný typ schránky (např. `SOCK_DGRAM`) bude po zavolání funkce uložen v parametru `type`.

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

int type, size;
size = sizeof(int);
if (getsockopt(s, SOL_SOCKET, SO_TYPE, (char *)&type, &size) < 0)
    //error
...
```

2.4.2 Komunikace typu broadcast

Pomocí funkce `setsockopt()` můžeme u schránky typu `SOCK_DGRAM` nastavit vlastnost `SO_BROADCAST`. Tím vytvoříme schránku, která posílá pakety UDP na adresu všesměrového vysílání (broadcast):

```
int s = socket(AF_INET, SOCK_DGRAM, 0);
int on = 1;                // nastavení povolení broadcastu - hodnota 1
setsockopt(s, SOL_SOCKET, SO_BROADCAST, &on, sizeof(on));
...
sendto(sockfd, buf, strlen(buf), 0, sockaddr, slen); // sockaddr obsahuje
...                                                // broadcastovou adresu
```

Cílová adresa, na níž posíláme zprávu, musí být typu broadcast. Pro broadcast na lokální síti můžeme použít konstatní adresu `INADDR_BROADCAST` z knihovny `<netinet/in.h>`.

Při programování komunikace typu broadcast je potřeba si uvědomit, že tyto zprávy přijímá každá stanice na lokální síti, což jsou obvykle stovky počítačů. Posílání broadcastových zpráv vytváří velkou zátěž pro danou síť. Broadcast se typicky používá pro nalezení zdroje, jehož adresu neznáme (např. DHCP server) nebo pro zaslání směrovacích informací všem stanicím na síti.

2.4.3 Komunikace typu multicast

Jak jsme se již zmínili, unicastová adresa identifikuje jedno síťové rozhraní na vrstvě IP, broadcastová adresa všechny síťové rozhraní v lokální síti a *multicastová adresa identifikuje množinu síťových rozhraní*. Multicastová data se posílají těm účastníkům, kteří o ně mají zájem, nikoliv všem, jako u broadcastového vysílání. Posílání zpráv na broadcastovou adresu je limitováno na lokální síť LAN (Local Area Network). U multicastového přenosu takové omezení není a lze směřovat data i rozsáhlých sítí WAN.

Adresování

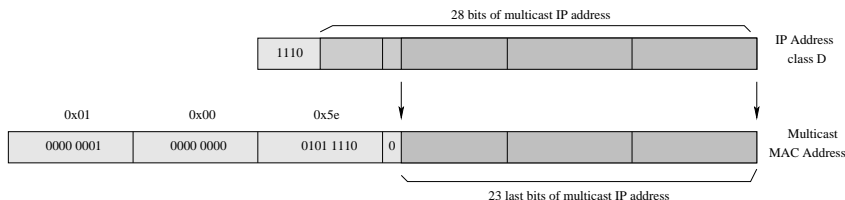
Multicast (skupinové vysílání) přenáší IP datagramy na skupinu počítačů, které jsou identifikované jedinou IP adresou třídy D (rozsah 224.0.0.0 – 239.255.255.255). Nižších 28 bitů adresy tvoří identifikaci multicastové skupiny. Maximální počet multicastových skupin tedy je $2^{28} - 1$, což je cca 268 mil. adres. Standardy RFC a IANA dávají doporučení, jaké multicastové IP adresy používat z pohledu jejich dosahu (viditelnosti), viz tabulka 2.8.

Rozsah	IPv6	TTL	adresy podle RFC 2365 [2]
lokální rozhraní (interface-local)	1	0	
lokální podsít (link-local)	2	1	224.0.0.0 – 224.0.0.255
daná lokalita (site-local)	5	<32	239.255.0.0 – 239.255.255.255
organizace (organization-local)	8		239.192.0.0 – 239.195.255.255
globální (global)	14	≤255	224.0.1.0 - 238.255.255.255

Tabulka 2.8: Rozsahy skupinového vysílání [9, str. 553]

Pro přenos multicastových dat v lokální síti je potřeba nastavit mapování multicastového IP datagramu na linový rámec. Jakou cílovou MAC adresu má mít takový rámec? U unicastu se odpověď dozvíme jednoduše dotazem na službu ARP, která ke každé unicastové IP adrese v lokální síti přiřadí odpovídající MAC adresu.

U multicastových IP datagramů je to jiné. Mapování multicastové IP adresy na adresu linkové rámce je pro Ethernet popsáno standardem RFC 1112 [8]. Nově vzniklá 48-bitová MAC adresa se skládá ze speciálního prefixu definovaného standardem IEEE 01:00:5e (24 bitů), další bit je 0 a zbylých 23 bitů se vezme s multicastové IP adresy (nižších 23 bitů), viz obrázek 2.18.



Obrázek 2.18: Mapování multicastové IP adresy na MAC adresu.

Při tomto převodu je vidět, že mapování multicastových IP adres na multicastové MAC adresy není v poměru 1:1. Pět horních bitů identifikace multicastové skupiny na vrstvě IP se totiž nepoužije, což sebou přináší překrytí adres. V praxi to znamená, že na jednu ethernetovou multicastovou adresu se může namapovat $2^5 - 1$ multicastových IP adres. To může vést k situaci, kdy dvě multicastové skupiny na vrstvě IP mají stejnou multicastovou MAC adresu a tudíž data poslaná na jednu skupinu jsou doručena i zařízením ve druhé skupině. Například uzel ve skupině 225.0.1.1 přijme rámec určený pro skupinu 224.0.1.1, neboť obě skupiny mají stejnou MAC adresu 01:00:5e:00:01:01. Tomuto způsobu doručování se říká *nedokonalé filtrování* (imperfect filtering), neboť se při doručení dat porovnává pouze část IP adresy umístěné v MAC adrese. Přijatý rámec je po kontrole MAC adresy předán vyšší vrstvě IP,

kteřá porovná celou cílovou IP adresu s nastavenou multicastovou IP adresou. Protože na této vrstvě se už porovnávají všechny bitů adresy (všech 32 bitů), jde o tzv. *dokonalé filtrování* (perfect filtering). V případě, že cílová IP adresa paketu je jiná, než nastavená multicastová IP adresa, datagram se zahodí.

Členství v multicastové skupině

Datagram zasílaný na multicastovou adresu je doručen všem členům skupiny stejným způsobem jako unicastový datagram, tzn. není garantováno spolehlivé doručení, zachování pořadí apod. Aby počítač mohl přijímat multicastová dat, musí se připojit do multicastové skupiny. Poté, co se rozhodne data dále nepřijímat, odhlásí se ze skupiny. Členství ve skupině je tedy dynamické. Skupina může být stálá (má administrativně přiřazenou IP adresu) nebo přechodná. Počítač může být součástí i více multicastových skupin, tj. má více multicastových IP adres.

Přihlašování a odhlašování uzlů do multicastové skupiny zajišťuje protokol IGMP (Internet Group Management Protocol) [8, 1]. Pro doručování multicastového přenosu mimo lokální síť, musí přenos multicastu také podporovat všechny propojující L3 prvky (směrovače a L3 přepínače).

Pokud se uzel připojuje do multicastové skupiny, pošle v příkazu JOIN protokolu IGMP žádost o připojení, ve které specifikuje adresu skupiny (tj. adresu třídy D), do které se chce přihlásit. Po úspěšném přihlášení se na rozhraní komunikujícího uzlu nastaví IP adresa této skupiny, např. 224.0.1.1. Zároveň řekne síťová vrstva linkové, aby přijímala data s cílovou adresou 01:00:5e:00:01:01, což je odpovídající multicastová MAC adresa. Na transportní vrstvě se pomocí čísla portu nastaví aplikace, která bude multicastová data zpracovávat.

Připojený uzel může data přijímat či vysílat. Názvy klient-server mají u multicastu jiný význam než u obecných unicastových aplikací, neboť každý účastník má právo zaslat data všem ostatním účastníkům. Protože je členství dynamické, role se mohou měnit. V praxi se multicast používá například u videopřenosů, kde jeden účastník (videoservert) zasílá na definovanou multicastovou adresu data. Ostatní účastníci (klienti) je pasivně odebírají. Pokud se některý z účastníků rozhodne odhlásit ze skupiny, pošle protokolem IGMP příkaz LEAVE pro odhlášení z multicastové skupiny a ukončení přijímání multicastové komunikace.

Programování multicastových aplikací

Multicast na úrovni IP je podporován pouze schránkami skupiny AF_INET typu SOCK_DGRAM nebo SOCK_RAW. Implicitně jsou multicastové datagramy vytvářeny s TTL=1, což omezuje doručení datagramu mimo lokální síť. Pokud má mít multicastová skupina rozsah větší než jen na lokální síť, je potřeba hodnotu TTL funkcí `setsockopt()` upravit.

Knihovna BSD sockets nabízí možnosti, jak nastavit schránku, aby umožňovala komunikaci typu multicast. K tomu je potřeba upravit vlastnosti schránky funkcí `setsockopt()`. Vlastnosti důležité pro multicastovou komunikaci jsou uvedeny v tabulce 2.9.

Vysílání multicastových dat

Nyní si ukážeme kroky potřebné pro vytvoření schránky, které umí vysílat multicastová data.

1. Vytvoření schránky typu SOCK_DGRAM.

Vlastnost	Popis
IP_MULTICAST_IF	nastaví implicitní rozhraní pro multicast
IP_MULTICAST_TTL	nastaví TTL pro odchozí multicastové pakety
IP_MULTICAST_LOOP	povolí/zakáže multicast na rozhraní loopback
IPV6_MULTICAST_IF	nastaví implicitní rozhraní pro multicast IPv6
IPV6_MULTICAST_TTL	nastaví TTL pro odchozí multicastové pakety Pv6
IPV6_MULTICAST_LOOP	povolení/zakáže multicast IPv6 na rozhraní loopback
IP_ADD_MEMBERSHIP	připojení k multicastové skupině
IP_DROP_MEMBERSHIP	odpojení od multicastové skupiny
IPV6_JOIN_GROUP	připojení k multicastové skupině IPv6
IPV6_LEAVE_GROUP	odpojení od multicastové skupiny IPv6
MCAST_JOIN_GROUP	připojení k multicastové skupině nezávislé na protokolu
MCAST_LEAVE_GROUP	odpojení od multicastové skupiny nezávislé na protokolu

Tabulka 2.9: Vlastnosti schránek pro multicast

2. Nastavení vlastností schránky `IP_MULTICAST_IF`, `IP_MULTICAST_TTL` či `IP_MULTICAST_LOOP`.
3. Zadání multicastové adresy ve struktuře `sockaddr_in` nebo funkci `connect()`.
4. Poslání datagramů funkcí `sendto()`.

Příklad:

```
#include <netinet/in.h>
#include <sys/socket.h>
...
#define PORT 60123
#define GROUP "224.0.0.250"
struct sockaddr_in addr;
unsigned char ttl=4;
int len, fd, cnt;
char message[50];

if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    err();
if ((setsockopt(fd, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl)) < 0)
    err();
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
len = sizeof(addr);
addr.sin_addr.s_addr = inet_addr(GROUP);
...
if ((cnt=sendto(fd,message,sizeof(message),0,(struct sockaddr *)&addr,len)) < 0)
    err();
...
close(fd);
```

Jak vidíme ve výše uvedeném příkladu, není nutné, aby se schránka při vysílání multicastových dat připojila k multicastové skupině příkazem `IP_ADD_MEMBERSHIP`. To je nezbytné pro čtení dat. Pokud není vysílající stanice členem skupiny, které jsou data určena, tak odeslané

datagramy použijí jako adresu odesílatele unicastovou IP adresu rozhraní, ze kterého stanice vysílá.

Příjem multicastových dat

Pro přijímání multicastových dat se používá struktura `ip_mreq`, která obsahuje informace o multicastové skupině, ke které se připojujeme.

```
struct ip_mreq{
    struct in_addr imr_multiaddr;    // adresa třídy D multicastové skupiny
    struct in_addr imr_interface;    // lokální rozhraní pro připojení
}
```

Čtení dat probíhá za pomoci schránky typu `SOCK_DGRAM` a zahrnuje následující kroky:

1. Onicializace rozhraní pro čtení multicastu (adresa, rozhraní) funkcemi `socket()` či `bind()`.
2. Připojení do skupiny nastavením vlastnosti `IP_ADD_MEMBERSHIP`.
3. Čtení dat funkcí `recvfrom()`, zpracování dat.
4. Odpojení od skupiny nastavením vlastnosti `IP_DROP_MEMBERSHIP`.

Příklad:

```
#include <sys/socket.h>
#include <netinet/in.h>
...
#define PORT 60123
#define GROUP "224.0.0.250"
struct ip_mreq mreq;
struct sockaddr_in addr;
int cnt, fd, len;
unsigned int ON=1; // příznak povolující více procesům přistupovat ke schránce
...
if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)
    err();
if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, &ON, sizeof(ON)) < 0)
    memset(&addr, 0, sizeof(addr));
addr.sin_family = AF_INET;
addr.sin_port = htons(PORT);
addr.sin_addr.s_addr = htonl(INADDR_ANY);    // připoj se na implicitní rozhraní
if (bind(fd, (struct sockaddr *)&addr, sizeof(addr)) == -1)
    err();
mreq.imr_multiaddr.s_addr = inet_addr(GROUP);
mreq.imr_interface.s_addr = htonl(INADDR_ANY);
if (setsockopt(fd, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq)) < 0)
    err();
len = sizeof(addr);
while (recvfrom(fd, message, sizeof(message), 0, (struct sockaddr *)&addr, &len) > 0){
    service(message);
    ...
}
close(fd);
```

2.5 Komunikace na úrovni IP a na linkové vrstvě

Doposud jsem se převážně zabývali vytvářením komunikace na transportní vrstvě. Standardní schránky pracují na úrovni TCP či UDP, kde vytvářejí rozhraní pro aplikace používající pro přenos tyto transportní protokoly. Schránky typu TCP a UDP nezasahují do nižších vrstev, pouze na úrovni IP umožňují nastavit IP adresu. Služby nižších vrstev (síťové a linkové) zajišťuje samotná knihovna BSD sockets ve spolupráci s jádrem operačního systému a řadičem síťového rozhraní.

Některé speciální aplikace, které se týkají například správy sítě (směrování, testování provozu, sledování procházejících dat), potřebují přistupovat k nižším datovým jednotkám než jsou transportní či aplikační data. Pro vytváření těchto aplikací můžeme využít vlastnosti schránek typu `raw` z knihovny BSD sockets nebo lze použít další knihovny, například knihovnu `libpcap` pro čtení dat na linkové či síťové vrstvě, nebo knihovnu `libnet` pro zápis těchto dat.

2.5.1 Schránky typu `raw` – komunikace na vrstvě IP

Schránky typu `SOCK_RAW` slouží ke čtení a zápisu dat protokolů ICMP, IGMP a IP. Využívají se například pro implementaci programů `ping` nebo `traceroute`. Pomocí těchto schránek můžeme vytvářet IP datagramy, které nejsou zpracovány jádrem operačního systému. Většina operačních systémů zpracovává IP datagramy typu 1 (ICMP), typu 2 (IGMP), typu 6 (TCP) či typu 17 (UDP). Pro zpracování ostatních protokolů přenášovaných nad IP (např. směrovacího protokolu OSPF, typ 89), musíme přistupovat přímo do těla IP datagramu a tam interpretovat jeho obsah. Schránka `raw` umožňuje například vytvořit vlastní hlavičku IPv4 nastavením vlastnosti schránky `IP_HDRINCL`.

Při práci na síťové úrovni musíme podrobně znát hlavičky zpracovávaných protokolů, například hlavičky IP či ICMP. Vytvoření schránky typu `SOCK_RAW` má také své bezpečnostní omezení – může je vytvářet pouze privilegovaný uživatel `root`.

Dalším omezením při práci se schránkami typu `SOCK_RAW` je vstup dat do těchto schránek ze síťové karty. Při čtení a zpracování došlých IP datagramů se uplatňují následující pravidla:

- Paket TCP či UDP se nikdy nepředávájí do schránky `SOCK_RAW`. Pro jejich čtení musíme pracovat na linkové úrovni (knihovna `libpcap`) nebo využít schránky `SOCK_STREAM` či `SOCK_DGRAM`.
- Většina zpráv ICMP je po zpracování v jádře operačního systému předána schránce. Výjimkou u některých implementací mohou být zprávy typu `echo request`, `timestamp request` či `address mask request`.
- Všechny zprávy IGMP jsou předány schránce typu `raw` až po zpracování v jádře operačního systému.
- Všechny IP datagramy, jejichž typ protokolu jádro nerozpozná, jsou předány do schránky typu `raw`.
- Pokud je IP datagram fragmentován, předá se schránce typu `raw` až celý sestavený IP datagram.

Když jádro operačního systému obdrží IP datagram pro schránku typu `raw`, zkontroluje všechny schránky typu `raw`, které ho očekávají, zda se shodují v informacích v hlavičce da-

tagramu. Teprve pak jim je předána kopie datagramu. Datagram je doručen, pokud platí zároveň následující tři podmínky:

- Číslo protokolu při vytváření schránky funkcí `socket()` se musí shodovat s číslem protokolu zpracovávaného datagramu (např. `IPPROTO_ICMP`).
- Cílová IP adresa datagramu se musí shodovat s adresou zadanou ve funkci `bind()`.
- Cílová IP adresa zadaná ve funkci `connect()` se musí shodovat s adresou odesilatele v hlavičce zpracovávaného IP datagramu.

Jestliže je schránka typu `SOCK_RAW` vytvořena s protokolem číslo 0 (implicitní) a není zavolána funkce `bind()` ani `connect()`, pak se do schránky uloží kopie každého datagramu, který přišel na rozhraní.

Vytvoření schránky typu raw

1. vytvoření schránky – funkce `socket()`

```
int sockfd;
sockfd = socket (AF_INET, SOCK_RAW, protocol);
```

2. nastavení vlastností schránky `IP_HDRINCL`

```
const int ON = 1;
if (setsockopt(sockfd, IPPROTO_IP, IP_HDRINCL, &ON, sizeof(ON)) < 0)
    err()
```

3. nastavení lokální adresy funkcí `bind()` (nepovinné)

Pokud není lokální adresa nastavena, použije se primární IP adresa odchozího síťového rozhraní.

4. nastavení adresy odesilatele funkcí `connect()` (nepovinné)

Použití této funkce nám umožní odesílat data funkcemi `write()` či `send()`, místo `sendto()`.

Příklad použití – komunikace ICMP

```
#include <sys/socket.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
...
struct icmp *icmp_send, *icmp_recv;
// vytvoření schránky
int sockfd;
sockfd = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP)
...
// odeslání ICMP zprávy
icmp_send = (struct icmp *)malloc(8); // struct icmp <netinet/ip_icmp.h>
icmp_send->icmp_type = ICMP_ECHO;
icmp_send->icmp_code = 0;
```

```

icmp_send->icmp_id = pid;                // identifikační pole
seq=0;
icmp_send->icmp_seq = seq++;             // sekvenční číslo datagramu
icmp_send->icmp_cksum = (unsigned int) 0;
icmp_send->icmp_cksum = checksum((unsigned short *)icmp_send, len);
...
if (sendto(sock, (char *)icmp_send, length, 0,
    (struct sockaddr *)&tmp->saddr, sizeof(struct sockaddr_in)) != len)
    error();
...
// čtení IP hlavičky a ICMP dat
if ((length = recvfrom(sock, buffer, BUFSIZE, 0, (struct sockaddr *)&sockaddr_recv,
    (unsigned int *)&size)) > 0)
    ip_recv = (struct ip *) buffer;      // IP datagram <netinet/ip.h>
...
if (ip_recv->ip_p == IPPROTO_ICMP){      // načtení ICMP hlavičky
    icmp_recv = (struct icmp *) (buffer + ip_recv->ip_hl * 4);
    if ((icmp_recv->icmp_id == pid) && (icmp_recv->icmp_type == ICMP_ECHOREPLY)
        && (icmp_recv->icmp_seq == seq-1))
        service(first_host, sockaddr_recv.sin_addr);
    ...
}

```

2.5.2 Knihovna Libnet – zápis dat na linkové vrstvě

Knihovna Libnet [7] poskytuje prostředky pro vytváření paketů a jejich zápis přímo na síťové rozhraní. Její funkce umožňují programátorovi vytvářet jednoduchým způsobem hlavičky IP, UDP a TCP. Zápis na síťové rozhraní je možný buď přes schránku typu `raw` nebo přímým zápisem na linkovou vrstvu funkcí knihovny.

Libnet je knihovna napsaná v jazyce C s implementací pro Linux, FreeBSD, Solaris, Windows NT, MacOS a další operační systémy⁵. Současná verze knihovny 1.1.2 podporuje řadu protokolů na linkové, síťové, transportní a aplikační vrstvě:

- aplikační – např. BGP, RPC, DNS, NTP, BOOTP, DHCP
- transportní – např. TCP, UDP
- síťové – např. IPv4, IPv6, ICMP, IGMP, ARP, RIP, OSPF, GRE, VRRP
- linkové – např. Ethernet II, 802.3, CDP, 802.1x, MPLS

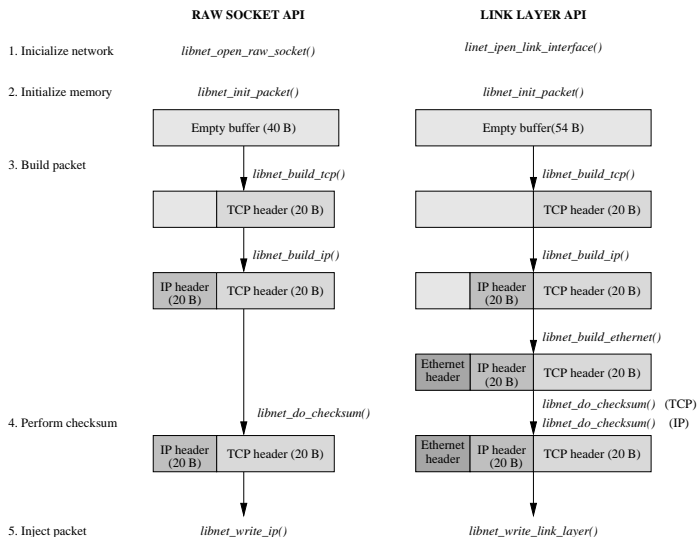
Vytváření datových jednotek PDU a jejich zápis na síťové rozhraní

Libnet umožňuje zapisovat vytvořené datové jednotky na dva typy rozhraní:

- rozhraní `raw socket API` pro zápis IP datagramy
- linkové rozhraní pro zápis ethernetových rámců

Podle typu rozhraní zápisu dat použijeme i příslušné funkce pro vytvoření. Většina aplikací může využít oba přístupy zápisu. Přímý přístup na linkové rozhraní je komplexnější, ale vyžaduje také více programování.

⁵Knihovna je přístupná na URL <http://www.packetfactory.openwall.net/projects/libnet/>



Obrázek 2.19: Vytváření datových rámců v knihovně Libnet

Použití knihovny obsahuje následující kroky (obrázek 2.19):

1. Inicializace sítě

Inicializace sítě využívá schránky `SOCK_RAW`. Knihovna používá postup popsáný dříve v části 2.5.1. U linkového rozhraní se volá funkce `libnet_open_link_interface()`.

2. Inicializace paměti

Inicializace paměti zahrnuje alokaci prostoru pro vytvoření paketu pomocí funkce `libnet_init_packet()`. Jednoduchý paket TCP na úrovni schránky `raw` vyžaduje paměť ro uložení hlavičky IP datagramu, hlavičky TCP a obsahu paketu. Při použití linkového rozhraní, musíme přidat ještě paměť pro uložení hlavičky Ethernetu. Po ukončení je potřeba paměť uvolnit funkcí `libnet_destroy_packet()`.

3. Vytvoření datové jednotky PDU

Datové jednotky se vytváří modulárně. Pro každou protokolovou úroveň se zavolá příslušná funkce knihovny Libnet. Například pro schránku typu `raw` využijeme funkce `libnet_build_ip()` a `libnet_build_tcp()`. Pro linkové rozhraní musíme navíc zavolat funkci `libnet_build_ethernet()`. Při konstrukci PDU musíme dbát na to, abychom správně vložili offset, na který se do vytvářené PDU mají zapisovat data. Pro IP je offset 14 bytů (délka hlavičky Ethernetu), pro TCP je to dalších 20 bytů.

4. Výpočet kontrolního součtu

Protože vytváříme hlavičky PDU ve vlastní režii, musíme také spočítat kontrolní součet hlavičky (checksum). Pro výpočet lze použít funkci `libnet_do_checksum()`, která spočítá kontrolní součet z transportní vrstvy u schránky typu `raw`. U linkového rozhraní je nutné spočítat ještě kontrolní součet hlavičky IP.

5. Zápis PDU na síťové rozhraní (packet injection)

Zápis provádí funkce `libnet_write_ip()` pro schránky `raw` nebo funkce `libnet_write_link_layer()` pro linkové rozhraní. Obě funkce vrátí počet zapsaných bytů nebo hodnotu `-1` v případě chyby.

Funkce knihovny Libnet pro tvorbu ICMP, IP a Ethernetu

```
libnet_build_icmpv4_echo(
    sett->icmp_msg,                /* ICMP message type */
    0,                             /* ICMP code */
    0,                             /* checksum */
    sett->icmp_echo_id,            /* ICMP echo identification */
    sett->icmp_echo_seq,          /* ICMP echo sequence number */
    sett->payload,
    sett->payload_size,
    1,                             /* libnet handle */
    0);                             /* libnet id */

libnet_build_ipv4(
    LIBNET_IPV4_H + header_len +  /* length */
    sett->payload_size,
    sett->ip_tos,                  /* TOS */
    sett->ip_id,                   /* IP ID */
    sett->ip_frag,                 /* IP Frag */
    sett->ip_ttl,                  /* TTL */
    proto,                        /* protocol */
    0,                            /* checksum */
    sett->ip_src,                  /* source IP */
    sett->ip_dst,                  /* destination IP */
    NULL,                         /* payload */
    0,                            /* payload size */
    1,                            /* libnet handle */
    0);                           /* libnet id */

libnet_build_ethernet(
    sett->mac_dst,                 /* ethernet destination */
    sett->mac_src,                 /* ethernet source */
    ETHERTYPE_IP,                 /* protocol type */
    NULL,                         /* payload */
    0,                            /* payload size */
    1,                            /* libnet handle */
    0);                           /* libnet id */
```

Ukázka vytvoření DNS dotazu pomocí knihovny Libnet [9, str. 812-813]

```

#include <libnet.h>
...
static libnet_t *l; /* libnet descriptor */
char qbuf[24];
u_int16_t one;
int packet_size = LIBNET_UDP_H + LIBNET_DNSV4_H + 24;
static libnet_ptag_t ip_tag, udp_tag, dns_tag;
...
if (libnet_init(LIBNET_RAW4, NULL, errbuf) == NULL) /* Initialize libnet */
    err_quit();
...
/* build DNS packet */
dns_tag = libnet_build_dnsv4(
    1234, /* identification */
    0x0100, /* flags: recursion desired */
    1, /* # questions */
    0, /* # answer RRs */
    0, /* # authority RRs */
    0, /* # additional RRs */
    qbuf, /* query */
    24, /* length of query */
    1,
    dns_tag);
/* build UDP header */
udp_tag = libnet_build_udp(
    ((struct sockaddr_in *) local)->sin_port, /* source port */
    ((struct sockaddr_in *) dest)->sin_port, /* dest port */
    packet_size, /* length */
    0, /* checksum */
    NULL, /* payload */
    0, /* payload length */
    1,
    udp_tag);
if (libnet_toggle_checksum(l, udp_tag, LIBNET_OFF) < 0)
    err_quit();
/* build IP header */
ip_tag = libnet_build_ipv4(
    packet_size + LIBNET_IPV4_H, /* len */
    0, /* tos */
    0, /* IP ID */
    0, /* fragment */
    TTL_OUT, /* ttl */
    IPPROTO_UDP, /* protocol */
    0, /* checksum */
    ((struct sockaddr_in *) local)->sin_addr.s_addr, /* source */
    ((struct sockaddr_in *) dest)->sin_addr.s_addr, /* dest */
    NULL, /* payload */
    0, /* payload length */
    1,
    ip_tag);
if (libnet_write(l) < 0)
    err_quit();

```


2.5.3 Knihovna Libpcap – čtení dat na linkové vrstvě

Knihovna Libpcap vytváří API pro čtení a zápis dat na síťové rozhraní. Pomocí knihovny Libpcap můžeme přistupovat ke všem paketům na síti, tj. i k těm, které jsou určeny pro jinou stanici na daném síťovém segmentu (čtení v tzv. promiskuitním režimu). Knihovna dále obsahuje funkce pro zápis binárních dat včetně linkové hlavičky na síťové rozhraní (funkce `pcap_inject()`, `pcap_sendpacket()`) či pro získávání informací o síťovém rozhraní (funkce `pcap_datalink()`, `pcap_list_datalink()`). Narozdíl od knihovny Libnet neobsahuje knihovna Libpcap rutiny pro vytváření paketů TCP, IP datagramů či linkových rámců. Pokud chceme vytvořit tyto pakety, musíme poskládat hlavičky i obsah paketů ve vlastní režii. Proto se knihovna používá zejména pro čtení (odchytávání) dat.

Čtení a analýzu odchycených dat na linkové vrstvě pomocí knihovny Libnet lze rozdělit do několika základních kroků:

1. **Připojení k síťovému rozhraní pomocí funkce `pcap_lookupdev()`**

Funkce `pcap_lookupdev()` vrací ukazatel na síťové zařízení pro odchytávání paketů.

2. **Otevření rozhraní pro čtení funkcemi `pcap_open_live()` či `pcap_open_offline()`**

Funkce `pcap_open_live()` otevře síťové zařízení pro odchytávání paketů. U některých operačních systémů je možné místo ukazatele na síťové zařízení zadat hodnotu `any` nebo `NULL`, což znamená čtení ze všech dostupných síťových zařízení.

Parametry funkce je možné specifikovat maximální velikost přijímaného paketu (doporučuje se 65 535), zda má zařízení pracovat v promiskuitním režimu (tj. odchytávat i pakety určené pro jiné stanice), čas, po který se čeká před načtením paketu, a další. Funkce `pcap_open_offline` umožňuje číst data ze souboru uložená ve formátu `tcpdump`.

3. **Čtení paketů pomocí funkcí `pcap_dispatch()`, `pcap_loop()` a `pcap_next()`**

Pro čtení dat můžeme využít více funkcí. Funkce `pcap_dispatch()` načte a zpracuje zadaný počet paketů. Při on-line zpracování vrací počet paketů ve vstupním bufferu zařízení. Funkce `pcap_loop()` čeká, dokud nedostane požadovaný počet paketů. Funkce `pcap_next()` načítá jeden paket.

4. **Analýza paketu**

Knihovna Libpcap neposkytuje nástroje na analýzu paketů. Načtená data jsou předána jako řetězec aplikaci. Ta pak provádí analýzu a interpretaci dat. Pro čtení dat lze použít funkce knihovny `<netinet/if_ether.h>` pro analýzu ethernetovského rámce, knihovnu `<netinet/ip.h>` pro analýzu IP datagramu, knihovnu `<netinet/ip_icmp.h>` pro protokol ICMP a podobně.

Příklad – načtení a analýza ethernetového rámce pomocí knihovny Libpcap

```
#include <pcap.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netinet/if_ether.h>
...
char *dev;
pcap_t* descr;
```

```

const u_char *packet;
struct pcap_pkthdr hdr;
struct ether_header *ether;
...
dev = pcap_lookupdev(errbuf));          /* get a pointer to a network device */
descr = pcap_open_live(dev,BUFSIZ,0,-1,errbuf) /* open it for sniffing */
packet = pcap_next(descr,&hdr);          /* capture a packet */
ether = (struct ether_header *) packet;   /* analyse the ethernet header */
if (ntohs(ether->ether_type) == ETHERTYPE_IP) /* if type is IP -> analyse IP */
...

```

Shrnutí

Komunikace po síti se modeluje pomocí dvou procesů – klienta a serveru. Klient vytvoří rozhraní pro uživatele, posílá dotazy a zobrazuje odpovědi. Server čeká na požadavky, zpracovává je a posílá odpovědi. Komunikace probíhá podle předem dohodnutých pravidel popsaných protokolem. Protokol je soubor syntaktických a sémantických pravidel pro výměnu informací mezi komunikujícími entitami. Protokol můžeme popsat formálně (stavovými automaty, gramatikami) nebo neformálně (standards RFC).

Při programování komunikace nad TCP/IP se používá knihovna BSD sockets, která implementuje rozhraní zvané schránka (socket). Schránka je programová struktura, do níž komunikující procesy vkládají data a z níž je čtou doručená data. Schránka je specifikována IP adresou a číslem portu odesílatele a příjemce. Schránky se používají pro komunikaci aplikací nad protokoly TCP, UDP a IP. Komunikace TCP zahrnuje ustavení spojení, přenos dat a uzavření spojení. Komunikaci UDP zajišťuje pouze vytvoření a přenos paketů UDP. Výhodou spojení UDP je rychlost a nenáročnost při vytváření spojení.

Aplikační servery, které zpracovávají požadavky od klientů, mohou pracovat v režimu iterativním, kdy vyřizují požadavky postupně, nebo v režimu konkurentní, kdy přijímají a zpracovávají souběžně více požadavků najednou. Při čekání na spojení či příchod paketů může dojít k blokování, kdy se zastaví běh programu a aplikace čeká se na asynchronní I/O událost od jádra operačního systému. Blokování lze řešit pomocí neblokujících schránek, vícenásobného přístupu ke schránkám funkcí `select()` nebo pomocí zaslání signálů.

Schránky knihovny BSD sockets umožňují také vytvářet multicastovou a broadcastovou komunikaci. Oba typy komunikace pracují s přenosem UDP. U schránek je potřeba nastavit podporu broadcastu či multicastu úpravou vlastností schránek funkcí `setsockopt()`. Pro příjem multicastových dat se komunikující aplikaci musí přihlásit do multicastové skupiny nastavením vlastnosti `IP_ADD_MEMBERSHIP`.

Pro komunikaci na nižších vrstvách modelu OSI (linkové a síťové) můžeme využít schránky typu `SOCK_RAW` či externí knihovny. Pro vytváření a posílání dat lze použít knihovnu Libnet, která umožňuje jednoduše vytvářet IP datagramy či ethernetové rámce. Pro čtení dat na nižší úrovni se často používá knihovna Libpcap.

Otázky

1. Jaké jsou výhody spojení UDP oproti TCP? Pro jaké aplikace byste je použili?
2. Uveďte příklady klientů a serverů pro následující služby: elektronická pošta, DNS, systém WWW, adresářové služby, vzdálené přihlašování, přenos souborů.
3. Co se stane, pokud konkurenční server TCP neuzavírá nově vytvořené schránky?
4. Jakou funkcí lze zjistit IP adresu a port přidělené systémem? Uveďte příklad.
5. Vyjmenujte funkce pro konverzi čísel do síťového uspořádání bytů. Kdy byste je použili?
6. Jaký typ záznamu DNS používá funkce `gethostbyname()` a `gethostbyaddr()`?
7. Vysvětlete pojmy iterativní a konkurenční server. Porovnejte jejich činnost.
8. Kdy dochází k blokování při komunikaci? Jak lze tomu zabránit?
9. Jak lze nastavit u schránek BSD komunikaci typu broadcast a multicast?
10. Co musí aplikace udělat, aby mohla přijímat či vysílat multicastová data?
11. Jakým způsobem lze vytvořit aplikaci, která komunikuje na síťové a linkové vrstvě?

Doporučená literatura a standardy

- [1] B.Cain, S.Deering, I.Kouvelas, B.Feener, and A.Thyagarajan. *Internet Group Management Protocol Version 2*. RFC 3376, October 2002.
- [2] D.Meyer. *Administratively Scoped IP Multicast*. RFC 2365, July 1998.
- [3] J.Postel. *User Datagram Protocol*. RFC 768, August 1980.
- [4] J.Postel. *Transmission Control Protocol*. RFC 793, September 1981.
- [5] R. Milner. *A Calculus of Communicating Systems*. Springer Verlag, 1980.
- [6] R.Braden. *Requirements for Internet Hosts – Communication Layers*. RFC 1122, October 1989.
- [7] M. D. Schiffman. *Libnet 101, Part 1: The Primer*. White paper, Security Focus, 2000.
- [8] S.Deering. *Host Extensions for IP Multicasting*. RFC 1112, August 1989.
- [9] W.R. Stevens, B. Fenner, and A. M. Rudoff. *UNIX Network Programming. The Sockets Networking API*. Addison-Wesley, 3rd edition, 2004.
- [10] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960, September 2007.

Kapitola 3

Systém DNS

Adresování a překlad adres je nezbytná součást internetu, bez které se žádná komunikace mezi uživateli neobejde. Že nefunguje překlad doménových jmen, zjistíme během krátkého okamžiku, kdy nejsme schopni načíst jedinou webovou stránku nebo poslat email.

Tato kapitola se zabývá architekturou systému DNS (Domain Name System). Seznámíme se se základními stavebními prvky systému, způsobem uložení dat a přístupu k nim. Popíšeme si také základní typy záznamů, které systém DNS spravuje. Popis systému je rozšířen o některé základní bezpečnostní problémy DNS a ukazuje, jak zajistit DNS proti zneužití. Zmíníme se také o využití DNS pro IP telefonii či lokalizaci a sdílení zdrojů.

3.1 Služba DNS

Základním úkolem služby DNS je mapování (převod) doménových adres (např. pmatousek.fit.vutbr.cz) na IP adresy (147.229.12.101). Doménové adrese se často říká také doménové jméno (domain name). V první kapitole jsme si řekli, že každé síťové rozhraní v internetu obsahuje jednoznačný identifikátor, jímž je IP adresa. Pro uživatele je identifikace počítačů pomocí IP adres nevhodná. Málokterý uživatel Internetu si pamatuje nějakou IP adresu. Na druhou stranu není výhodné, aby síťová zařízení pracovala s textovými řetězci. 32-bitové číslo (či 128-bitové u IP adres verze 6) je velmi vhodný formát pro porovnávání či prefixové vyhledávání.

Už při zavedení IP adres v 70. letech 20. století se začaly používat jmenné ekvivalenty, které se vždy překládaly na IP adresu. Právě systém DNS obsahuje celosvětovou databázi IP adres a jejich srozumitelných ekvivalentů, doménových jmen.

Služba DNS tedy obsahuje databázi všech doménových adres a příslušných IP adres. Definuje také, jak budeme přistupovat k těmto datům. Protože jde o velmi rozsáhlou databázi, je distribuovaná na více počítačů, kde běží speciální servery, kterým se říká *nameservery*, doménové servery (jmenné servery) nebo také servery DNS. IP adresu zjišťujeme z doménového jména dotazem na server DNS. Proces vyhledávání v systému DNS nazýváme rozloučení či rozlišení jména (*name resolution*).

Systém DNS využívá všechny aplikační protokoly (např. HTTP, SMTP, FTP) pro překlad doménových adres na IP adresy. To je vždy první aktivita, kterou aplikace udělá poté, co ji požádáme o připojení na vzdálenou službu a zadáme adresu v podobě doménového jména. Služba si nejprve požádá o překlad na IP adresu. Teprve poté může dojít k navázání spojení.

Mezi základní služby, které poskytuje systém DNS, patří:

- překlad doménových adres na IP adresy (s využitím záznamů typu A, AAAA),
- překlad IP adres na doménové adresy (s využitím záznamů typu PTR),
- překlad aliasů počítačů, překlad na tzv. kanonická jména (s využitím záznamů CNAME),
- určení poštovního serveru pro danou doménu (s využitím záznamů typu MX),
- podpora rozložení zátěže mezi více aplikačních serverů (rotace záznamů),
- sdílení informace v rámci globálního prostoru jmen či
- delegování správy domén na jednotlivé subjekty (pomocí záznamů typu NS).

O jednotlivých typech překladů si podrobněji povíme v kapitole 3.3.

3.1.1 Historie

Pro mapování doménových jmen na IP adresy se původně používal jeden soubor HOSTS.TXT, který spravovala organizace NIC (Network Information Center). Uložená data (mapování doménová adresa – IP adresa) se šířila na další počítače pravidelnými aktualizacemi pomocí služby FTP. S rozšířením lokálních sítí se administrace záznamů přenesla na lokální úroveň. Aby byly změny viditelné i na Internetu, musely se lokálně mapované adresy předávat do NIC. Postupně vzniká koncept distribuované databáze založené na doménových serverech.

Primárním cílem DNS bylo vytvořit konzistentní prostor jmen (name space), který se použije pro odkazování na síťové zdroje. Databáze měla být distribuovaná s možností ukládání lokálních kopií pro zvýšení výkonu systému. Systém DNS byl navržen tak, aby byl přístupný nejen pro IP, ale i pro jiné rodiny protokolů a aplikace. Hlavním úkolem systému DNS je tedy zajistit snadný přístup k informacím uloženým v DNS ze všech počítačů připojených k síti. Následující body shrnují požadavky na databázi DNS:

- Velikost databáze závisí na počtu připojených počítačů, který v budoucnu poroste.
- Většina dat v databázi se nemění často. Systém však musí provádět drobné změny velmi rychle (řádově sekundy a minuty).
- Přístup k datům je důležitější než jejich aktuálnost. Když není možné data aktualizovat, je třeba zajistit alespoň běh služby.
- Originální data se udržují lokálně v takzvaných primárních záznamech (master files). Odtud se šíří na další počítače pomocí systému DNS. Formát záznamů je textový, spravuje je lokální administrátor.

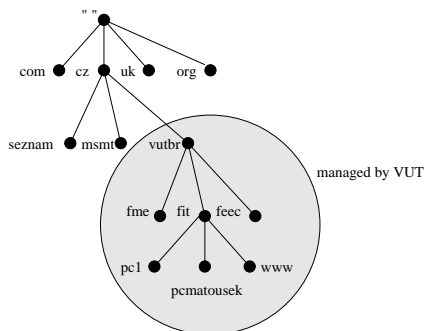
Systém DNS obsahuje seznam všech registrovaných doménových adres a jejich mapování na IP adresy. V prostoru IPv4 to může být až 2^{32} adres, což je fakt hodně. Samozřejmě není potřeba vytvářet pro všechny IP adresy odpovídající doménovou adresu. Například pro IP adresy třídy D a E to nemá smysl. Také pokud je IP adresa použita pouze pro systémové služby, nemá smysl k ní vymýšlet doménové jméno. Naopak, pro některé IP adresy může naopak existovat více doménových jmen, jak si ukážeme v kapitole 3.3.

3.2 Architektura systému DNS

Systém DNS se skládá ze tří hlavních komponent – prostoru doménových jmen, serverů DNS a resolveru [27]. Prostor doménových jmen zahrnuje strukturu, uspořádání a přístup k datům v systému DNS. Servery DNS ukládají tato data ve svých lokálních databázích. Resolver slouží pro přístup k datům v systému DNS. Jednotlivé komponenty systému DNS si nyní podrobněji popíšeme.

3.2.1 Prostor doménových jmen (domain name space)

Jak jsme si už řekli, doménových jmen i IP adres je poměrně hodně. Z důvodu uložení a efektivního vyhledání se logický prostor všech doménových jmen ukládá v systému DNS speciálním způsobem. Systém DNS tvoří databáze hierarchicky uspořádaná jako kořenový strom doménových jmen, viz obrázek 3.1. Tato stromová struktura připomíná uspořádání souborového systému v Unixu. Z pohledu algebry se jedná o acyklický graf – strom, který obsahuje uzly a hrany.



Obrázek 3.1: Hierarchické uspořádání doménových jmen v DNS

Kořen stromu DNS se nazývá **the root**. Uzly stromu jsou pojmenovány textovým řetězcem (bez teček) délky max. 63 znaků. Někdy se uzlům říká domény, ale toto označení je nepřesné, jak si za chvilku ukážeme. Názvy uzlů jsou pouze součástí doménových jmen. Uzly, které mají stejného bezprostředního předchůdce ve stromu, musí mít různý název, aby zaručili rozlišení uzlů (např. `fit.vutbr.cz.`, `feec.vutbr.cz.`, `fme.vutbr.cz.`). Kořen strom má speciální jméno – řetězec nulové délky. Cesta od listu ke kořenu slouží k uložení (a také vyhledávání) doménových adres.

Doména je podstrom v grafu doménových adres. *Doménové jméno* (domain name) je cesta mezi uzlem, který tvoří vrchol domény, a kořenem stromu DNS. Příkladem může být doménové jméno (obvykle se říká pouze doména) `vutbr.cz.`, kterou tvoří všechny uzly v prostoru doménových adres s relativním kořenem v uzlu `vutbr`, který patří do podstromu `cz.` Doména s vrcholem v uzlu ve vzdálenosti jedna od kořene grafu se nazývá doména první úrovně (někdy též TLD, Top Level Domain), doména ve vzdálenosti dva od kořene stromu DNS je doménou (či subdoménou) druhé úrovně a tak dále. Listy stromu označují konkrétní

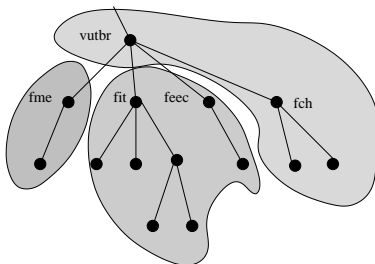
síťová zařízení patřící do dané domény, například servery `pcmatousek`, `www`, `pc1`, v doméně `fit.vutbr.cz..`

Plné (absolutní) doménové jméno (FQDN, Fully Qualified Domain Name) je posloupnost jmen uzlů na cestě až ke kořeni stromu DNS. Tato posloupnost jmen se odděluje tečkami, například `"www.fit.vutbr.cz."`. Všimněte si, že absolutní adresa vždy končí tečkou. Ve skutečnosti je tečka pouze oddělovač a adresa končí jménem kořene DNS stromu, což je řetězec nulové délky. Doménová jména bez tečky se nazývají relativní a jsou interpretována k relativní doméně, ve které se nachází. Když například na fakultním serveru zadáme `ping www`, kde `www` je relativní doménové jméno školního webového serveru, bude se adresa interpretovat v doméně `fit.vutbr.cz..` Ukončující tečku při práci na internetu obvykle nepíšeme, neboť ji při překladu doplňuje aplikace. Při práci s DNS, jako je například konfigurace serveru DNS či klienta DNS, je ukončující tečka nezbytná.

Databáze DNS obsahuje jednotlivá doménové jména uspořádaná do podstromů (domén). Správa domén (přidávání koncových uzlů, rušení uzlů, vytváření subdomén), je decentralizovaná a deleguje se na další organizace. Například doménová jména počítačů `pc1.fit.vutbr.cz`, `pc2.fit.vutbr.cz`, `pc3.fit.vutbr.cz`, `www.fit.vutbr.cz`, `ns.fit.vutbr.cz` atd., patří do domény `fit.vutbr.cz..`, kterou spravuje Fakulta informačních technologií VUT v Brně na svém lokálním serveru DNS `kazi.fit.vutbr.cz`.

Uspořádání prostoru doménových adres

Prozatím jsme se bavili o logickém uspořádání prostoru doménových jmen do hierarchického stromu. Tento strom však není uložen na jednom místě v jedné databázi. DNS je systém hierarchický a decentralizovaný. Jednotlivé části podstromu celého prostoru doménových adres jsou fyzicky uloženy na lokálních serverech DNS, které dohromady tvoří systém DNS. Data o objektech v prostoru DNS (obecně zdrojích, resources) tvoří pouze doménová jména. Záznamy obsahují také informace o primárních a sekundárních serverech DNS, správcích domén, poštovních serverech a podobně. Veškeré tyto informace jsou zapisovány v textovém formátu, který definují RFC 1034 [27] a RFC 1035 [28]. Ukládají se do takzvaných *záznamů DNS* (RR, resource records). Přehled typů záznamů DNS a podrobnější popis jejich formátu je uveden v kapitole 3.3.



Obrázek 3.2: Příklad uspořádání DNS prostoru do zón

Fyzické části prostoru DNS, které jsou pod jednou správou, se nazývají *zóny*. Zóna není

totožná s doménou. Například poskytovatel síťového připojení může spravovat více domén (a subdomén) tvořící jednu zónu. Nebo naopak velká doména (např. `edu`) je rozdělena na více částí a umístěna na více serverech DNS. Například doména `vutbr.cz` může být rozdělena do více zón, které pokrývají různé části doménového prostoru a které spravují různé subjekty, viz obrázek 3.2.

Zatímco doména je část prostoru adres, která má společný suffix (např. doména `vutbr.cz`), zóna je tvořena částmi prostoru uloženého na konkrétním server. Zónu mohou tvořit například subdomény `fit.vutbr.cz` a `feec.vutbr.cz` pod jednou správou. Zóna může obsahovat celou doménu nebo jen část domény. Existují také dva speciální typy zón:

- Zóna **stub** obsahuje pouze informace o tom, které servery subdoménu obsluhují. Sama neobsahuje žádná data.
- Zóna **hint** obsahuje seznam kořenových serverů DNS.

Reverzní mapování adres

Jednou z důležitých funkcí systému DNS je *reverzní (zpětné) mapování* IP adres na doménová jména. Prozatím jsme se bavili o tzv. *přímém mapování*, které je základem DNS a kdy se překládá doménové jméno na příslušnou IP adresu. Reverzní adresování slouží ke zpětnému dohledání doménových adres pro IP adresy, které jsou uloženy například v logovacích souborech. Používá se například při autorizaci počítače, kdy kontrolujeme, zda IP adresa stanice má záznam v systému DNS. Pokud doménové jméno chybí, může to systém vyhodnotit jako použití podvržené IP adresy a komunikaci odmítnout. Toho využívají například poštovní servery v boji proti spamu.

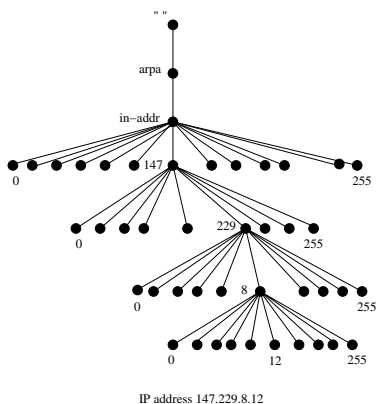
Reverzní mapování DNS adres není tak jednoduché, jak by se na první pohled zdálo. Záznamy v datovém prostoru DNS jsou indexovány podle doménových adres. Pokud známe doménovou adresu, můžeme lehko vyhledat IP adresu. Vyhledání doménové adresy pro známou IP adresy by však znamenalo úplné prohledání celého stromu, kdy v každém uzlu porovnáváme hodnotu uzlu se hledanou IP adresou (tzv. asociativní vyhledávání).

Tento způsob je pro praktické použití nerealizovatelný. Proto existuje v datovém prostoru DNS jedna speciální doména, jejíž uzly jsou pojmenovány čísly reprezentujícími IP adresu ve čtyřbytovém dekadickém formátu odděleným tečkami (například `147.229.8.12`). Doména, ve které jsou tyto IP adresy uloženy, se nazývá **in-addr.arpa**, viz obr. 3.3.

Jak je z obrázku zřejmé, doména **in-addr.arpa** obsahuje 256 subdomén první úrovně, které odpovídají prvnímu bytu IP adresy. Každá z těchto subdomén obsahuje dalších 256 subdomén druhého řádu, pak třetího řádu až do hloubky čtyři. IP adresy ve stromu jsou uspořádány od nejvyššího bytu k nejnižšímu (tj. jak se čtou, zleva doprava), což je opačný postup než u doménového jména. Doménová jména se ukládají od nejvyšší domény, která se píše až na konec.

Například doménová adresa `kazi.fit.vutbr.cz` začíná nejméně obecnou položkou `kazi`, která je ve stromu DNS uložena nejdále od kořene. Pro IP adresu je postup opačný, záznam DNS pro IP adresu `147.229.8.12` je `12.8.229.147.in-addr.arpa`, což je obrácený zápis než vlastní IP adresa.

Toto uspořádání umožňuje lépe delegovat správu subdomény na vlastníka odpovídajícího prostoru IP adres. Například VUT v Brně je odpovědné za doménu `229.147.in-addr.arpa`, neboť má přidělený prostor adres B `147.229.0.0/16`. Tuto reverzní doménu může ovšem dále



Obrázek 3.3: Reverzní mapování v DNS

rozdělit na podsíťi a jednotlivé části podstromu přidělit do správy jejich uživatelů, například fakult.

Registrace a správa domén

Databáze DNS je spravovaná hierarchicky a distribuovaná na velké množství serverů. Koordinaci systému DNS zajišťuje organizace ICANN (Internet Corporation for Assigned Names and Numbers), www.icann.org. Ta je zodpovědná za správu, přidělování a uložení doménových jmen. Každý záznam o doménové adrese musí být jedinečný, aby uživatelé mohli najít odpovídající IP adresy. Organizace ICANN deleguje části prostoru doménových jmen na tzv. akreditované registrátory doménových jmen (ICANN-Accredited Registrar) [15]. ICANN spravuje domény nejvyšší úrovně. Domény nižších úrovní jsou delegovány na další subjekty. Doménou první úrovně mohou být buď národní domény (ccTLD, country code TLD), např. .uk, .cz, nebo generické domény (gTLD, generic top-level domains), viz tabulka 3.1. Seznam registrátorů generických domén první úrovně (TLD, Top-level domains) je na <http://www.internic.net/regist.html>.

doména	použití	doména	použití
com	komerční organizace	aero	letecký průmysl
edu	vzdělávací organizace v USA	biz	obchodní organizace
gov	vládní agentury v USA	coop	družstva
mil	doména US. Military	museum	muzea
net	organizace pro správu sítí	pro	profesionálové (doktoři, právníci)
int	mezinárodní organizace	info	informační místo
org	nevýdělečné organizace	name	jednotlivci

Tabulka 3.1: Generické domény první úrovně

Některé z výše uvedených domén, například **.edu**, **.gov.**, **.int**, **.mil**, mají předem definované použití a mohou je využívat pouze určité subjekty. Domény **aero**, **biz**, **coop**, **museum**, **pro**, **info**, **name** byly přidány v roce 2000, aby se zvýšil počet domén nejvyšší úrovně a také snížil se tlak na doménu **.com**. Od té doby se objevují další domény. Jednou z kategorií jsou tzv. sponzorované domény, které spravuje určitý subjekt pro speciální použití, například **.asia**, **.cat**, **.jobs**, **.mail**, **.mobi**, **.post**, **.tel**, **.travel**, viz <http://www.icann.org/tlds/>. Standard RFC 2606 [10] zavedl také v roce 1999 rezervované zóny pro speciální účely, viz tabulka 3.2.

doména	použití
.test	testovací účely
.example	vytváření dokumentace a příkladů
.invalid	navozování chybových stavů
.localhost	lokální (softwarové) komunikační rozhraní
local	intranet

Tabulka 3.2: Rezervované domény podle RFC 2606

Speciální doménou nejvyšší úrovně TLD je i **.arpa**, která se používá pro účely správy síťové infrastruktury. Doménu **.arpa** spravuje organizace IANA. Použití domény je definováno standardem RFC 3172 [12] a dalšími standardy, které jsou spolu s využitím domény **arpa** uvedeny v tabulce 3.3.

doména	použití	odkaz
e164.arpa	mapování telefonních čísel E.164 na URI	RFC 6116 [4]
in-addr.arpa	mapování IPv4 adres na doménové adresy	RFC 1035 [28]
ip6.arpa	mapování IPv6 adres na doménové adresy	RFC 3596 [35]
iris.arpa	lokalizace služeb Internet Registry Information Services	RFC 4698 [11]
uri.arpa	překlad URI podle systému Dynamic Delegation Discovery System	RFC 3405 [22]
urn.arpa	překlad URN podle systému Dynamic Delegation Discovery System	RFC 3405 [22]

Tabulka 3.3: Speciální domény **.arpa**

Národní domény nejvyšší úrovně (ccTLD) registruje národní správce domény, který také koordinuje přidělování adres v rámci země. Název domény (**.cz**, **.de**, **.uk**) vychází ze standardu ISO-3166 [1]. Správci domén jsou uvedeni v databázi organizace IANA. Např. pro Českou republiku je to organizace **CZ-NIC** - viz [záznam v databázi IANA](http://www.nic.cz/page.php?sid=9). V současné době CZ-NIC udržuje centrální databázi, změny do ní však vkládají oprávnění registrátoři – viz <http://www.nic.cz/page.php?sid=9>.

3.2.2 Server DNS (nameserver)

Další částí systému DNS jsou servery DNS (nameservers). Server DNS je aplikace, která uchovává data z prostoru doménových jmen. Tento prostor je rozdělen do zón a umístěn na jednotlivé servery DNS. Základním úkolem DNS serveru je odpovídat na dotazy směřující na databázi DNS. Server DNS uchovává data ve formě množiny záznamů DNS (resource records). Záznamy jsou uloženy v lokálním souboru nebo si je server načte z jiného serveru DNS pomocí přenosu zón. Informace, které server DNS spravuje a za které je zodpovědný, se

nazývají se autoritativní. Specifikace DNS [27] definuje dva základní typy serverů – primární a sekundární.

- **Primární server DNS** (master, primary nameserver)

Primární server obsahuje úplné záznamy o doménách, které spravuje. Tyto záznamy jsou uloženy lokálně v souboru. Server poskytuje autoritativní (tj. vždy přesné) odpovědi pro tyto domény. Pro každou doménu musí existovat právě jeden primární nameserver.

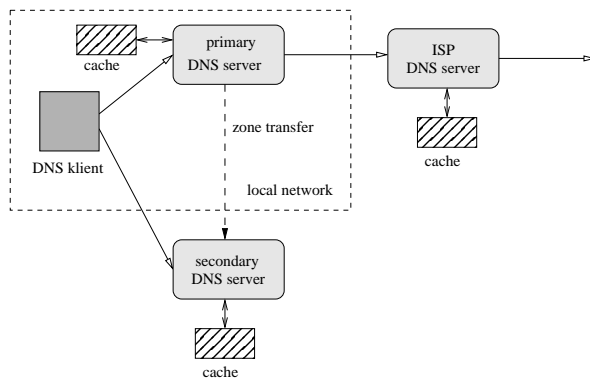
- **Sekundární server DNS** (slave, secondary nameserver)

Sekundární server získává data od primárního serveru. Soubor, který obsahuje databázi konkrétní domény (či subdomény), se nazývá *zónový soubor*. Proces přenosu zónových souborů z primárního serveru na sekundární se nazývá *přenos zón* (zone transfer). Přenos zón je podrobněji popsán v kapitole týkající se komunikace DNS. Sekundární server musí zajistit pravidelný přenos zónových dat a aktuálnost dat. Sekundární server je také autoritativní server pro danou doménu.

- **Záložní server DNS** (caching-only nameserver)

Záložní server pracuje jako proxy server. Přijímá dotazy od klientů a předposílá je dalším serverům DNS. Když záložní server dostane odpověď na svůj dotaz, uchová si ji a použije ji v budoucnosti. Záložní servery poskytují neautoritativní odpovědi, tj. odpovědi, které mohou být neúplné a neaktuální. Zrychlují však proces rezoluce doménového jména.

Zóna DNS je souvislá část jmenného prostoru DNS, pro kterou je daný server DNS autoritativní. Server může obsahovat více zón. Každá zóna obsahuje záznamy DNS pro danou část jmenného prostoru.



Obrázek 3.4: Typy DNS serverů

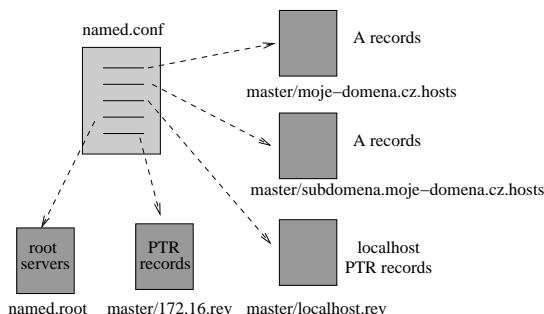
Platnost záznamů DNS na sekundárním a záložním serveru je časově omezena. Hodnota expirace je uvedena u každého záznamu. Pokud dojde k expiraci záznamu, musí server daný záznam smazat ze své databáze, případně si ho znovu načíst z primárního serveru. Pokud

by neprovedl aktualizaci, může dojít k situaci, kdy dva různé servery DNS odpoví na stejný dotaz různým způsobem, což vede ke ztrátě konzistence dat.

Zejména při změně v záznamech DNS (změna IP adresy, přidání nového záznamu apod.) je potřeba počítat s tím, že rozšíření změn trvá v řádu hodin, což je doba, kdy dojde k expiraci původních záznamů a načtení nových. Na druhou stranu by nastavení krátké doby expirace vedlo k častým dotazům na obnovení záznamu a přetížení autoritativních serverů. Protože změny se šíří v síti DNS pomalu, je nutné v případě, kdy potřebujeme zjistit aktuální hodnotu nějakého záznamu v DNS, kontaktovat přímo primární nebo sekundární server DNS.

Konfigurace serveru DNS

V této části si ukážeme příklad konfigurace jednoduchého serveru DNS. Pro demonstraci konfigurace použijeme hojně rozšířenou distribuci serveru DNS BIND (Berkeley Internet Name Domain) a aplikaci `named`. Základní konfigurace serveru DNS je tvořena pěti základními soubory, viz obr. 3.5:



Obrázek 3.5: Příklad konfigurace serveru DNS BIND

named.conf je základní konfigurační soubor serveru DNS. Obsahuje informace o doménách a zónových souborech, které server obsahuje, na jakém rozhraní odpovídá na dotazy apod. Zároveň obsahuje informaci o primárním či sekundárním serveru. Obsahuje též příkazy `acl`, `include`, `key`, `logging`, `options`, `server` či `zone` pro filtrování dotazů, podepisování, logování apod.

```
options {
    directory "/etc/namedb";
    listen-on { 127.0.0.1; 172.16.12.1; };
    forwarders { 147.229.8.12; };
};
zone "." {
    type hint;
    file "named.root";    // odkaz na kořenové nameservy
};
zone "0.0.127.IN-ADDR.ARPA" { // primární server pro loopback
```

```

        type master;           // (reverzní záznam)
        file "localhost.rev";
    };
    zone "moje-domena.cz" {      // primární server pro doménu
        type master;
        file "moje-domena.cz";
    };
    zone "16.172.in-addr.arpa" {  // primární server, reverzní záznamy
        type master;             // pro adresy od 172.16.0.0
        file "172.16";
    };
    zone "moje-domena.cz" {      // sekundární server pro doménu
        type slave;
        file "moje-domena.cz";
        masters {172.16.12.1;};
    };
    zone "16.172.in-addr.arpa" {  // sekundární server, reverzní záznamy
        type slave;             // pro adresy od 172.16.0.0
        file "172.16";
        masters {172.16.12.1;};
    };
};

```

named.root obsahuje seznam kořenových serverů DNS. Tento soubor obsahuje IP adresy kořenových serverů DNS, které používá v případě, že neumí odpovědět na zadaný dotaz.

```

; formerly NS.INTERNIC.NET
;
.                3600000    IN    NS      A.ROOT-SERVERS.NET.
A.ROOT-SERVERS.NET. 3600000    A      198.41.0.4
A.ROOT-SERVERS.NET. 3600000    AAAA    2001:503:BA3E::2:30
;
; formerly NS1.ISI.EDU
;
.                3600000    NS      B.ROOT-SERVERS.NET.
B.ROOT-SERVERS.NET. 3600000    A      192.228.79.201
...

```

named.local, **localhost.rev** jsou slouží pro překlad adresy localhost (loopback).

```

$TTL      3600
@          IN      SOA      ns.moje-domena.cz. root.pci.moje-domena.cz. (
                                20050311; Serial
                                3600   ; Refresh
                                900    ; Retry
                                3600000 ; Expire
                                3600 ) ; Minimum
          IN      NS       ns.moje-domena.cz.
1         IN      PTR      localhost.moje-domena.cz.

```

Přímé zónové soubory obsahují záznamy typu SOA, NS, A, MX, CNAME a další.

```
@ IN SOA ns.moje-domena.cz hostmaster.moje-domena.cz (
    20050809 ; Serial
    28800 ; Refresh 8 hours
    7200 ; Retry 2 hours
    604800 ; Expire 7 days
    86400 ) ; Maximum TTL 1 day
IN NS ns.moje-domena.cz.
IN NS ns.muj-provider.cz.
ns IN A 192.168.10.1
...
```

Reverzní zónové soubory obsahují záznamy typu SOA, NS či PTR.

```
@ IN SOA ns.moje-domena.cz hostmaster.moje-domena.cz (
    20050809 ; Serial
    28800 ; Refresh 8 hours
    7200 ; Retry 2 hours
    604800 ; Expire 7 days
    86400 ) ; Maximum TTL 1 day
IN NS ns.moje-domena.cz.
1 IN PTR ns.moje-domena.cz.
...
```

Při vytváření zóny je potřeba vytvořit minimálně dva záznamy: záznam SOA pro specifikaci správy domény a záznam NS, který ukazuje na autoritativní server dané zóny. Do zóny lze přidávat i další záznamy typu A, CNAME, MX, SRV, PTR a podobně.

Správnou konfiguraci serveru DNS lze ověřit následujícími kroky:

1. Test spuštění DNS serveru: příkaz `"/etc/rc.d/named status"`, informace v souboru `/var /log/messages` či výpis procesů příkazem `"ps ax | grep named"`.
2. Kontrola konfiguračního souboru: `"named-checkconf <filename>"`, kontrola zónového souboru: `"named-checkzone zone <filename>"`.
3. Testování funkčnosti programy `dig`, `host`, `nslookup`, či `rndc` (remote name server control).

Ne všechny informace o doménových jménech se musí nutně načítat ze systému DNS. Z historických důvodů existuje na unixových strojích soubor `/etc/hosts`, kde může být uveden statický překlad doménových jmen na IP adresy. Funkce knihovny BSD sockets určené pro překlad jmen `gethostbyname()` a `gethostbyaddr()` používají oba zdroje. Pořadí zdrojů, kam směřuje dotaz na překlad, specifikuje soubor `/etc/host.conf`, viz následující ukázka konfigurace.

```
order hosts,bind
multi on
```

Tato konfigurace určuje pořadí vyhledávání v systému DNS. Říká, že se nejprve hledají informace v souboru `/etc/hosts` a teprve potom, když nedostaneme požadovanou informaci, se zavolá program `bind`, viz `man host.conf`.

3.2.3 Resolver

Resolver je klientský program, který se dotazuje na data uložená v systému DNS. Uživatelské programy, které potřebují informace z DNS, přistupují k těmto datům pomocí resolveru. Základním úkolem resolveru je

- posílat dotazy na servery DNS,
- interpretovat odpovědi od serveru (přijaté záznamy, chybová hlášení),
- předat informace uživatelskému programu, který o data žádá.

Resolver musí být schopen přistupovat alespoň k jednomu serveru DNS. Od serveru získá přímo hledanou odpověď nebo mu server vrátí odkaz na další server, kde je hledaná informace uložena. Resolver tak může přeposílat dotazy dalším serverům.

Resolver je obvykle implementován jako systémová rutina, která je součástí operačního systému (viz `man resolver`) a ke které přímo přistupují uživatelské programy, například funkce knihovny BSD socket `gethostbyname()`, `gethostbyaddr()` nebo programy `nslookup` či `dig`.

Konfigurace resolveru

Konfigurace resolveru obsahuje informace o primárním a sekundárním doménovém serveru, aktuální doméně, způsobu vyhledávání apod. U unixových systémů je konfigurace uložena v souboru `/etc/resolv.conf`. Soubor obsahuje několik částí:

nameserver obsahuje IP adresu serveru DNS.

Jedná se o IP adresu serveru DNS, který odpovídá na dotazy. Položka může obsahovat více hodnot. Resolver volá servery podle uvedeného pořadí. Pokud neexistuje žádný záznam, dotaz se posílá lokálnímu počítači.

domain definuje lokální doménu.

Položka obsahuje hodnotu implicitní domény. Resolver přidá toto jméno ke každému názvu počítače, který není zadán jako plně kvalifikované. Například k položce `www` přidá jméno `fit.vutbr.cz.`, které je uvedeno v sekci **domain**. Tuto položku také využívá funkce `gethostname()`.

search definuje doménu pro vyhledávání.

Pokud název počítače neobsahuje plně kvalifikované jméno ukončené tečkou, doplní se hodnota v položce **search** nebo záznam **domain**. Obvykle obsahuje pouze jednu položku – název lokální domény. Pokud obsahuje více položek, provádí se vyhledávání ve více doménách (např. `vutbr.cz` a `fit.vutbr.cz`).

sortlist určuje pořadí sítí.

Tato hodnota obsahuje preferované pořadí sítí. Pokud odpověď na DNS obsahuje více IP adres hledaného záznamu, záznamy se přerovnají podle zde uvedeného pořadí. Aplikuje se na hodnoty vrácené funkcí `gethostbyname()`.

options obsahuje další volby.

Toto pole obsahuje další volby, které lze nastavit pro činnost resolveru. Například **timeout**, počet pokusů pro připojení na DNS server **attempts** a další.

Příklad nastavení resolveru:

```
search netlab.fit.vutbr.cz fit.vutbr.cz
nameserver 127.0.0.1
```

3.2.4 Rezoluce dotazů DNS

Rezoluce (resolution, rozlišení) je proces hledání odpovědi v systému DNS. Protože prostor doménových adres je strukturován jako kořenový strom, stačí každému serveru DNS pouze jediná informace, jak vyhledat libovolný uzel ve stromu – adresa kořenového serveru DNS. Server DNS se zeptá kořenového serveru DNS na nejvyšší doménu a pak postupuje po stromové struktuře od kořene až k uzlu, který obsahuje hledanou informaci.

Kořenový server DNS (root nameserver) je autoritativní server DNS pro všechny domény nejvyšší úrovně TLD. Při dotazu na jakékoliv doménové jméno odpoví kořenový DNS server buď přímo hledanou odpovědí nebo vrátí adresu serveru DNS, který informaci obsahuje. Servery DNS pro domény první úrovně obsahují informace o příslušných subdoménách druhé úrovně atd. Činnost kořenového serveru DNS popisuje dokument RFC 2870 [33].

Kořenový server DNS je nezastupitelný pro správný běh celého systému DNS. Z tohoto důvodu existuje třináct kořenových serverů s adresami **[a-m].root-servers.net.**, které jsou rozmístěné po celém světě u různých operátorů, viz obrázek 3.6. Obvykle se nejedná o jediný počítač. Zátěž každého jednoho kořenového serveru je rozložena na více strojů. Přesto každý z nich odpovídá na tisíce dotazů během každé sekundy.



Obrázek 3.6: Umístění kořenových serverů DNS, převzato z www.root-servers.org.

Otázkou je, zda stačí třináct kořenových v případě výpadku. Během posledních let bylo zaznamenáno několik větších útoků na infrastrukturu DNS. Jedním z nich byl útok DDoS

(Distributed Denial of Service) dne 21.10.2002, který byl zaměřen na zahlcení všech třinácti kořenových serverů DNS pakety ICMP, TCP SYN, fragmenty TCP a UDP. Pakety přicházeli na jednotlivé servery rychlostí cca 100 Mb/s. Útok trval cca 75 minut, během něhož byly některé kořenové servery nedostupné. Neobjevili se však žádné dopady na uživatelské sítě, pouze zpoždění v odpovědích od DNS. Jak uvádí oficiální zpráva o útoku¹, útok prokázal dostatečnou robustnost systému DNS i v případě soustředěného útoku.

Druhým větším útokem na infrastrukturu DNS byl útok DDoS ze dne 6.2.2007, který trval dvě a půl hodiny, a poté se opakoval po dalších pět hodin. Při útoku bylo šest ze třinácti kořenových serverů zasaženo, nicméně ostatní kořenové servery pracovali efektivně, takže útok neměl téměř žádný dopad na uživatele Internetu².

Jak vidíme, jsou kořenové servery DNS nezbytné pro činnost systému DNS. Pokud budeme chtít znát například autoritativní servery pro doménu `cz.`, stačí poslat dotaz na jakýkoliv kořenový DNS server. Vráti seznam serverů DNS, které jsou autoritativní pro doménu `cz.`

```
/home/matousp> nslookup -type=NS cz. a.root-servers.net
Server:      a.root-servers.net
Address:     198.41.0.4#53
```

```
Non-authoritative answer:
```

```
*** Can't find cz.: No answer
```

```
cz      nameserver = a.ns.nic.cz.
cz      nameserver = b.ns.nic.cz.
cz      nameserver = c.ns.nic.cz.
cz      nameserver = d.ns.nic.cz.
cz      nameserver = f.ns.nic.cz.
a.ns.nic.cz  has AAAA address 2001:678:f::1
a.ns.nic.cz  internet address = 194.0.12.1
b.ns.nic.cz  has AAAA address 2001:678:10::1
b.ns.nic.cz  internet address = 194.0.13.1
c.ns.nic.cz  has AAAA address 2001:678:11::1
c.ns.nic.cz  internet address = 194.0.14.1
d.ns.nic.cz  has AAAA address 2001:678:1::1
d.ns.nic.cz  internet address = 193.29.206.1
f.ns.nic.cz  has AAAA address 2001:628:453:420::48
f.ns.nic.cz  internet address = 193.171.255.48
```

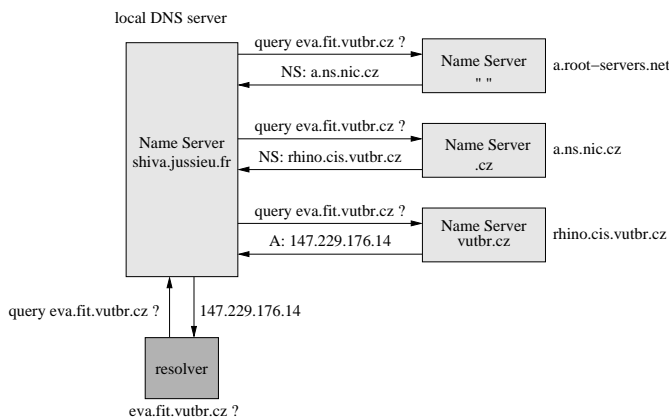
V ukázce vidíme, že kořenový server `a.root-servers.net` sám nezná doménu `.cz.`, ale odkáže nás odkázat na autoritativní servery pro tuto doménu. Jak vidíme, autoritativní servery DNS pro doménu `cz` jsou na pěti serverech `[a-f].ns.nic.cz.`

Jak vypadá v praxi zaslání dotazu na systém DNS (rezoluce) si ukážeme na následujícím příkazu, viz též obrázek 3.7. Uživatelský program, který běží na počítači například někde ve Francii, potřebuje přeložit doménovou adresu `eva.fit.vutbr.cz` na IP adresu. Požadavek předá programu resolveru. Proces vyhodnocování dále probíhá takto:

1. Pokud má resolver vlastní cache paměť, zkusí vyhledat dotaz v ní. Pokud informaci nenajde, přepoše dotaz nejbližšímu (lokálnímu) DNS serveru. V našem případě je to `shiva.jussieu.fr.`

¹P.Vixie, et al: Events of 21-Oct-2002, <http://c.root-servers.org/october21.txt>.

²viz zpráva ICANN, <http://www.icann.org/en/announcements/factsheet-dns-attack-08mar07.pdf>



Obrázek 3.7: Příklad rezoluce rekurzivního DNS dotazu

- DNS server hledá odpověď ve svých záznamech. Pokud ji nenajde (např. jde o dotaz na jinou doménu), musí najít autoritativní server pro danou doménu. Pošle tedy dotaz kořenovému serveru, kde zjišťuje odpověď. Pokud kořenový server sám nezná odpověď, přepośle odkaz na server, který je pro doménu (v našem případě **cz.**) autoritativní.
- Kořenový server tedy odpoví záznamem NS pro doménu **cz.**
- Lokální server DNS postupně kontaktuje autoritativní server domény **.cz.** a požádá o překlad doménového jména.
- Protože ani server **a.ns.nic.cz** nezná doménu **fit.vutbr.cz**, pošle odkaz na server obsahující doménu **vutbr.cz.**
- Lokální server DNS tedy kontaktuje autoritativní server DNS pro doménu **vutbr.cz.**
- Server **rhino.cis.vutbr.cz** odpověď zná a vrátí záznam A obsahující IP adresu hledaného počítače **eva.fit.vutbr.cz.**
- Lokální server DNS poděkuje a přepośle odpověď resolveru. Kopii si uloží do lokální paměti cache a ponechá ji tam, dokud nevyprší platnost záznamu.
- Resolver předá odpověď uživatelskému programu.

Ve výše uvedeném příkladu vidíme, že lokální server DNS se opakovaně dotazuje různých serverů DNS, dokud nedostane požadovanou odpověď. Ostatní servery pouze najdou nejlepší možnou odpověď (obvykle záznam NS), který předají tázajícímu se serveru. Sami se už nesnaží kontaktovat tyto servery DNS a najít autoritativní odpověď. To už je věcí resolveru, aby se opakovaně dotazoval. Pokud by to neudělal resolver, je to na uživateli, aby opakovaně posílal dotazy na jednotlivé servery ve stromu DNS.

Dotaz, který zaslal resolver lokálnímu DNS serveru, byl *rekurzivní*. Servery DNS rozlišují dva typy dotazů – rekurzivní a iterativní.

- **rekurzivní dotaz**

Rekurzivní dotaz je podobný rekurzi, jak ji známe např. z programování. Resolver zaslal dotaz na určitý údaj ve stromu DNS konkrétnímu serveru DNS. Server DNS musí odpovědět na dotaz buď požadovanými daty nebo chybovou hláškou, když například nezná odpověď. Pokud server není autoritativní pro hledaná data, musí se zeptat dalších serverů a najít autoritativní odpověď. Může poslat rekurzivní dotaz na některý z autoritativních serverů a čekat na odpověď. Nebo může poslat iterativní dotaz a získat odkaz na jiný server, který zná odpověď. Zda server podporuje rekurzivní či iterativní dotazování záleží na jeho konfiguraci.

Server DNS, který obdržel rekurzivní dotaz, přeposílá vždy stejný dotaz na další servery. Nikdy se neptá například na záznamy NS pro hledanou doménu. Většina programů (například `nslookup`, `dig`) zasílá rekurzivní dotazy.

- **iterativní dotaz**

Iterativní dotazy šetří práci na straně serveru DNS. Při tomto dotazování vrátí server resolveru nejlepší odpověď, kterou může dát. Více se nedotazuje. Dotazovaný server DNS se podívá do své lokální databáze. Pokud nenajde odpověď, vrátí adresy serverů, které jsou nejbližší hledané adrese.

V příkladu, který jsem výše uvedl, poslal resolver rekurzivní dotaz na lokální server DNS. Tento server pomocí iterativních dotazů vyhledával požadované informace. Odpověď pak předal resolveru.

Ukládání záznamů DNS v paměti cache

Abychom redukovali počet dotazů na systém DNS, používá většina serverů DNS lokální vyrovnávací paměť cache, kde si průběžně ukládá odpovědi od serverů pro budoucí vyhledávání. Některé verze serverů DNS umožňují implementovat také *negativní cache*. V této paměti si server ukládá negativní odpovědi od autoritativních serverů, tj. informace o tom, že hledaný záznam v dané doméně neexistuje. Negativní cache také zrychluje proces rezoluce dotazů DNS. Odpovědi získané z cache paměti nemusí být přesné a označují se jako *neautoritativní*, viz následující příklad:

```
/home/matousp> nslookup www.stanford.edu
Server:          147.229.9.43
Address:         147.229.9.43#53

Non-authoritative answer:
www.stanford.edu canonical name = www-lb.stanford.edu.
Name:   www-lb.stanford.edu
Address: 171.64.13.26
```

V tomto příkladu jsme se dotázali lokálního serveru DNS s adresou 147.229.9.43 (uvedeného v síťové konfiguraci počítače) na IP adresu serveru `www.stanford.edu`. Odpověď, kterou jsme dostali, je neautoritativní, tzn. předává nám ji server, který nespravuje záznamy v zóně

stanford.edu. Pokud bychom chtěli získat autoritativní odpověď, musíme se dotázat primárního či sekundárního serveru pro tuto zónu. Pak odpověď vypadá takto:

```
/home/matousp> nslookup www.stanford.edu Argus.stanford.edu
Server:          Argus.stanford.edu
Address:         171.64.7.115#53

www.stanford.edu      canonical name = www-lb.stanford.edu.
Name:   www-lb.stanford.edu
Address: 171.64.13.26
```

Ukládání lokálních kopií záznamů DNS na neautoritativním serveru DNS přináší kromě zrychlení vyhledávání i jedno nebezpečí – informace nemusí být aktuální. U pozitivních odpovědí se mohl záznam změnit či být zrušen, u negativního ukládání odpovědí se mohlo stát, že chybějící záznam byl přidán. Z tohoto důvodu má každý záznam DNS uvedenu *maximální dobu platnosti záznamu* (TTL, time to live), kterou nastavuje administrátor zóny pro daný zónový soubor.

Hodnota TTL určuje maximální dobu (v sekundách), po kterou je možné data uchovat v lokální paměti cache. To ovšem neznamená, že se data nemohou změnit. Nicméně po uplynutí doby TTL je musí server DNS ze své paměti cache odstranit a načíst si aktuální data z autoritativního serveru. Podobně to platí o zálohování negativních dotazů. I pro negativní dotazy existuje hodnota TTL.

Nastavení hodnoty TTL výrazně ovlivňuje poměr mezi konzistencí databáze na jedné straně a zvýšenou zátěží komunikace na straně druhé. Malé TTL zaručí lepší konzistenci dat, na druhou stranu způsobí větší počet dotazů na vlastní autoritativní servery DNS. Větší TTL může způsobit nekonzistenci dat. RFC 1033 [21] doporučuje jako minimální hodnotu TTL jeden den (86 400 sekund), což je vhodné pro servery, kde dochází k častým změnám. Pro stabilní servery se doporučuje hodnota tři až pět dní. Tyto hodnoty se nastaví v záznamech typu SOA, o kterých si povíme v další kapitole.

3.3 Záznamy DNS (Resource Records)

Pro ukládání informací v datovém prostoru DNS slouží záznamy DNS (resource records, RR). Záznamy jsou uloženy v textové podobě v zónových souborech na serverech DNS. Nejběžnější jsou záznamy typu A, které mapují doménové jméno na IP adresu (tzv. přímé mapování) a záznamy typu PTR pro opačné (reverzní) mapování. V této kapitole si povíme také o záznamech SOA, NS, MX a CNAME. Standardy RFC 1034 [27], RFC 1035 [28], RFC 1183 [7], RFC 3401 [23], RFC 3597 [3] popisují jednotlivé typy záznamů DNS. Některé typy záznamů se běžně používají (AAAA, SIG, NAPTR), jiné jsou spíše experimentální a nerozšířili se (RP, RT, MB). Tabulka 3.4 uvádí přehled standardů pro základní typy záznamů DNS.

3.3.1 Formát záznamů

Všechny typy záznamů DNS mají stejný obecný formát definovaný standardem RFC 1035 [28]. Formát záznamů obsahuje položky NAME, TYPE, CLASS, TTL, RDLLENGTH a RDATA. Každý záznam DNS obsahuje všechny uvedené položky. Položka RDATA se liší podle typu záznamu, kde pro daný typ (např. A či MX) obsahuje odpovídající informace. Struktura záznamu je i s příkladem na obrázku 3.8.

Typ záznamu	Název	Standard
SOA	Start of Authority	RFC 1034
NS	Name Server	RFC 1034
A	Address	RFC 1034
MX	Mail Exchanger	RFC 1034
CNAME	Canonical Name	RFC 1034
PTR	Domain Name Pointer	RFC 1034
NAPTR	Naming Authority Pointer	RFC 2915,3403,3761
TXT	Text	RFC 1034
SRV	Service Record	RFC 2782
LOC	Location	RFC 1876
AAAA	IPv6 Address	RFC 3596
DNSKEY	DNS Key Record	RFC 4034
RRSIG	DNSSEC Signature	RFC 4034
NSEC	Next-Secure Record	RFC 4034
NSEC3	NSEC record version 3	RFC 5155
DS	Delegation Signer	RFC 4034

Tabulka 3.4: Základní typy záznamů DNS a příslušné standardy.

Položka **Name** obsahuje jméno uzlu ve stromu DNS, kde je daný záznam uložen. Položka **Type** určuje typ záznamu (např. CNAME), **Class** definuje třídu záznamu, **TTL** maximální dobu platnosti záznamu. Pokud je hodnota **TTL** nastavena na 0, což je běžné například pro záznamy **SOA**, záznam nesmí být uložen v paměti cache. Pole **RDLENGTH** a **RDATA** obsahují hodnotu záznamu, na kterou se obvykle ptáme.

Resource Records Format	Example
Name (variable length)	www.fit.vutbr.cz
Type (16 bits)	CNAME
Class (16 bits)	IN (0x0001)
TTL (32 bits)	4106 (1 h 8 min 26 s)
RDLENGTH (16 bits)	9
RDATA (variable length)	tereza.fit.vutbr.cz

Obrázek 3.8: Formát DNS záznamu

Na obrázku 3.8 vidíme příklad záznamu typu **CNAME**, který obsahuje kanonické jméno (**CNAME**) pro doménovou adresu **www.fit.vutbr.cz**. Server DNS vrátí na dotaz záznam typu **CNAME** pro uzel **www.fit.vutbr.cz**. Záznam patří do třídy **IN** (Internet), jeho platnost je 1 hodina, 8 minut a 26 sekund, délka dat 9 bytů a obsahuje hodnotu **tereza.fit.vutbr.cz**, což je kanonické jméno pro hledanou doménovou adresu.

Všimavého čtenáře možná napadne, že řetězec **tereza.fit.vutbr.cz** je delší než 9 bytů. Je to tak. DNS používá pro přenos doménových jmen speciální kompresi paketů DNS, kde opakující se části doménových adres nahrazuje odkazy na první výskyt v záznamu. Například

zde vidíme opakující se řetězec `fit.vutbr.cz`, který je možné komprimovat. Více o kompresi paketů je v části 3.4.4.

Kromě třídy IN (Internet), definuje DNS ještě další tři historické třídy – CS (CSNET), CH (CHAOSnet) a HS (Hesiod). CHAOSnet byl protokol pro síť LAN vyvinutý v polovině 70.let na MIT, který se využíval na pracovních stanicích v oblasti umělé inteligence. CHAOSnet měl vlastní adresování s délkou 16 bitů zapisované v osmičkové soustavě, například 315.124. Dnes se už nepoužívá.

Podobně třída Hesiod sloužila v 80.-90. letech pro ukládání textových informací o uživateli, skupinách uživatelů, tiskárnách apod. Například pro uživatele s uživatelským jménem `jim` a skupinou `candlewood` (`jim.candlewood.hesiod`) vrátila UID a GID, například 501.142. Dnes se pro ukládání informací nepoužívá DNS, ale spíše adresářové služby typu LDAP.

Různé typy záznamů DNS se liší v počtu, významu a délce polí, které obsahují v části RDATA. Výše uvedený příklad obsahuje pouze jedno pole s kanonickým jménem. Záznamy typu A obsahují jedno pole s 32-bitovou IP adresou, záznamy typu MX dvě pole s 16-bitovou hodnotou preference a doménovým jménem poštovního serveru. Přesný popis polí u jednotlivých typů záznamů je uveden v dříve zmíněných standardech.

Formát záznamů DNS v zónovém souboru

Výše uvedený formát záznamů DNS je určen pro binární kódování a používá se převážně při komunikaci mezi servery a klienty DNS. Záznamy DNS se v zónových souborech na serverech obvykle zapisují v textové podobě. Obsah odpovídá binární podobě záznamů DNS, formát je odlišný. Pro nás je textové vyjádření důležitější než binární podoba určená pro přenos, protože správce serveru DNS pracuje výhradně s textovými daty. Formát zónového souboru a uložení záznamů v nich je opět definováno standardem RFC 1035 [28]. Vypadá takto:

```
<blank>                                [<comment>]
$ORIGIN <domain-name>                 [<comment>]
$INCLUDE <file-name> [<domain-name>]  [<comment>]
<domain-name> <rr>                     [<comment>]
<blank>                                <rr>      [<comment>]
```

Na začátku zónového souboru jsou definovány dvě řídicí položky `$ORIGIN` a `$INCLUDE`, které nepatří mezi záznamy. Položka `$ORIGIN` slouží ke změně původní adresy uzlu v DNS na jinou hodnotu. Například při nastavení `$ORIGIN fit.vutbr.cz.` na začátku doménového souboru můžeme používat relativní doménová jména (např. `www`, `email`, `kazi`), která se v následujících záznamech zónového souboru interpretují jako absolutní adresy `www.fit.vutbr.cz`, `email.fit.vutbr.cz` či `kazi.fit.vutbr.cz`. Položka `$INCLUDE` vkládá do zónového souboru další soubor např. s definicí dalších záznamů DNS.

Komentáře začínají znakem „;“ (středník). Znak „@“ (zavináč) označuje substituci doménového jména v uzlu (`origin`). Pokud záznam DNS `<rr>` začíná mezerou (`blank`), patří daný záznam k doméně, která byla uvedena u předchozího záznamu. Zónový soubor může obsahovat více záznamů DNS `<rr>`. Každý záznam začíná na novém řádku. Formát DNS záznamu `<rr>` je následující:

```
[<TTL>]  [<class>] <type> <RDATA>
```

Nepovinné hodnoty polí TTL a `class` mohou být vynechány. Pokud chybí, použijí se hodnoty definované v předchozím záznamu, případně se např. pro TTL použije implicitní hodnota daná direktivou `$TTL` na začátku zónového souboru. Příklad zónového souboru vypadá takto:

```
$TTL 3600
@ IN SOA isa.fit.vutbr.cz. root.isa.fit.vutbr.cz.(
    200711090    ; Serial
    3600         ; Refresh
    900          ; Retry
    3600000      ; Expire
    3600 )       ; Minimum
    IN NS      isa.fit.vutbr.cz.
    IN MX 0    eva.fit.vutbr.cz.
isa IN A      10.10.10.1
h01 IN A      10.10.10.101
h02 IN A      10.10.10.102
```

Výše uvedený zónový soubor obsahuje záznamy pro doménu `netlab.fit.vutbr.cz.`, což je definováno v konfiguračním souboru serveru DNS. Zónový soubor obsahuje záznamy pouze této domény. Obsahuje jeden záznam typu SOA, jeden záznam typu NS, jeden záznam typu MX a tři záznamy typu A.

Na začátku zónového souboru je direktivou `$TTL` definována maximální platnost záznamu, která je jednu hodinu (3600 sekund). Znak "@" říká, že doménové jméno uzlu, ve kterém je uložen tento záznam, je shodný s názvem domény uvedeným v konfiguračním souboru serveru DNS `named.conf`. Pokud bychom chtěli vložit záznam jiné domény, potřebujeme napsat celé doménové jména. Místo znaku @ bychom napsali doménu `netlab.fit.vutbr.cz.`. Server DNS `named` totiž při restartu načítá konfigurační soubor `named.conf` a zónové soubory, které se z něho odkazují. Při načítání záznamů se zónových souborů provede substituci znaku "@" za název uvedený v konfiguračním souboru, viz obrázek 3.5.

Dále výše uvedený zónový soubor obsahuje záznam typu SOA, který obsahuje primární soubor domény (tj. jméno primárního DNS serveru), emailová adresa správce (zde se nesmí vyskytovat substituční znak "@", nahrazuje se znakem ".") a položky s hodnotami. Pokud přesahují hodnoty záznamu jeden řádek, je potřeba uzavřít soubor hodnot do kulatých závorek. Po záznamu SOA následuje informace o doméně (záznam typu NS) a záznamy typu A s překlady doménových adres na IP adresy. Podrobnější popis záznamů je v následující kapitole.

3.3.2 Popis nejběžnějších záznamů DNS

Záznam SOA – Start Of Authority

Záznam SOA obsahuje informace týkající se uložení autoritativních dat pro danou zónu. Většinou je to první záznam v zónovém souboru. Každá zóna má právě jeden záznam SOA.

```
$TTL 3600
@ IN SOA isa.fit.vutbr.cz. root.isa.fit.vutbr.cz.(
    2007110901   ; Serial
    3600         ; Refresh
    900          ; Retry
    3600000      ; Expire
    3600 )       ; Minimum
```

Záznam SOA obsahuje jméno primárního serveru DNS pro danou doménu (`isa.fit.vutbr.cz.`). Dále obsahuje kontakt na správce domény – jeho emailovou adresu (`root@isa.fit.vutbr.cz.`). Sériové číslo `serial` slouží k identifikaci změn záznamu. Doporučený formát je `YYYYMMDDnn`,

kde YYYYMMDD označuje rok, měsíc a den, kdy proběhla změna, a hodnota **nn** označuje pořadí změny, například když během jednoho dne došlo k několika změnám.

Důležité je, že při každé změně záznamu se musí sériové číslo změnit. Sériové číslo záznamu SOA totiž používají sekundární servery DNS ke zjišťování změn. Pokud aktualizujeme zónový soubor pro danou doménu a nezměníme sériové číslo záznamu SOA, sekundární DNS server se při aktualizaci podívá na sériové číslo a když zjistí, že se nezměnilo, předpokládá, že se zónový soubor je beze změny a už si nenačte nový obsah zónového souboru.

Hodnota **Refresh** udává čas (v sekundách), jak dlouho má sekundární server čekat, než bude zjišťovat změnu. **Retry** určuje, kdy se sekundární server opět pokusí o přenos zóny v případě neúspěšného připojení a aktualizace dat. Hodnota **Expire** definuje maximální dobu platnosti dat na sekundárním serveru. Pokud server není schopen po uplynutí této doby data aktualizovat, musí je smazat. Hodnota **Minimum** udává implicitní hodnotu TTL záznamů v zónovém souboru. Tyto hodnoty mohou být upřesněny u jednotlivých záznamů v zónovém souboru.

Doporučené hodnoty záznamů pro servery DNS nejvyšší úrovně (TLD) a pro ostatní servery DNS uvádí podle RFC 1537 [26], viz tabulka 3.5.

Položka	Hodnota pro servery TLD	Hodnota pro ostatní servery DNS
Refresh	24 hodin	8 hodin
Retry	2 hodiny	2 hodiny
Expire	30 dní	7 dní
Minimum TTL	4 dny	1 den

Tabulka 3.5: Doporučené hodnoty SOA podle RFC 1537

Záznam NS – Name Server

Záznam typu NS určuje autoritativní server (či servery) pro danou doménu. Pomocí těchto záznamů dochází k budování hierarchické struktury systému DNS. Záznam NS, který je umístěn v daném uzlu stromu DNS, obsahuje ukazatele na níže připojené následovníky uzlu, viz obr. 3.1. **small**

```
fit.vutbr.cz. IN NS gate.feec.vutbr.cz.
              IN NS guta.fit.vutbr.cz.
              IN NS kazi.fit.vutbr.cz.
              IN NS rhino.cis.vutbr.cz.
```

Tento příklad uvádí autoritativní servery, které obsahují platné záznamy DNS o doméně **fit.vutbr.cz**. Pokud bychom chtěli zjistit, který ze serverů je primární a které jsou sekundární, musíme se dotázat záznam SOA a z něho tu informaci vyčíst.

Záznam A – IP Address

Záznam typu A obsahuje přímé mapování doménových adres na IP adresy. Je to jediný záznam, který obsahuje na pravé straně IP adresu. Pokud chceme získat IP adresu například DNS serveru (záznam NS) nebo poštovního serveru (záznam MX), musíme nejdříve získat kanonické jméno obou serverů a teprve poté požádat o A záznam každého z nich. **small**

```
eva.fit.vutbr.cz. IN A 147.229.176.14.
```

Záznam MX – Mail Exchanger

Záznam typu MX nás informuje o poštovním serveru, který pro danou doménu přijímá poštu. Emailová adresa v sobě obsahuje jméno adresáta i místo, kde je jeho poštovní schránka (at, znak"@"). Například emailová adresa `matousp@fit.vutbr.cz`, říká, že elektronická pošta pro uživatele `matousp` se doručuje do emailové schránky uložené v doméně `fit.vutbr.cz`. Mapování domény na adresy poštovního serveru obsahuje záznam MX.

```
fit.vutbr.cz. IN MX 10 kazi.fit.vutbr.cz. # vyšší priorita
                IN MX 20 eva.fit.vutbr.cz
```

Oproti ostatním záznamům obsahuje záznam MX navíc prioritu, která určuje preferenci poštovního serveru. Pokud máme pro danou doménu uvedeno více poštovních serverů, použije se ten s nižší hodnotou. Nižší hodnota označuje vyšší prioritu. V případě, že je prioritní poštovní server nedostupný, použije se pro doručení záložní server podle dalšího záznamu MX. Vlastní hodnota (10 nebo 20) není v záznamech důležitá. Rozhoduje, která hodnota je vyšší a která nižší. Pokud pro danou doménu neexistuje záznam MX, nelze poštu řádně doručit.

Záznam CNAME – Canonical Name

V DNS můžeme pro jeden počítač vytvořit více doménových jmen. Pokud například na počítači `tereza.fit.vutbr.cz` poběží služba WWW, můžeme mu přiřadit snadno zapamatovatelný alias `www.fit.vutbr.cz`. Pokud tam poběží i službaLDAP, přidáme další název `ldap.fit.vutbr.cz`. Záznamy CNAME pro výše uvedený příklad může vypadat takto:

```
www      IN CNAME tereza.fit.vutbr.cz.
ldap     IN CNAME tereza.fit.vutbr.cz.
```

```
tereza IN A 147.229.9.22
```

Při každém dotazu do DNS na počítač `ldap.fit.vutbr.cz` vrátí systém DNS kanonické jméno počítače (tj. `tereza`) a k němu i odpovídající IP adresu uloženou v záznamu typu A. Pomocí aliasů jsme schopni odlišit doménové jméno od IP adresy a v případě potřeby (například přetížení serveru) změnit alias tak, že se odkazuje na záložní server.

Záznam CNAME tedy přiřazuje k aliasu kanonické (oficiální) jméno počítače. Pokaždé, když DNS server při hledání najde záznam CNAME, nahradí doménové jméno kanonickým jménem a pokračuje v hledání.

Alias definovaný záznamem CNAME nesmí nikdy stát na pravé straně záznamu. Všechny ostatní záznamy se musí mapovat na oficiální (kanonické) jméno, nikoliv na alias.

Záznam PTR – Domain Name Pointer

Záznam typu PTR provádí zpětné (reverzní) mapování. To znamená, že převádí číselnou IP adresu na doménové jméno, viz obr. 3.3. Jak jsme uvedli v kapitole 3.2.1, reverzní adresy jsou uloženy ve speciální doméně `in-addr.arpa` v systému DNS.

```
14.176.229.147.in-addr.arpa. IN PTR eva.fit.vutbr.cz.
```

Protože se jedná o jinou doménu, jsou reverzní záznamy uloženy v jiném zónovém souboru než přímé záznamy. Proto většina serverů DNS ukládá ve své konfiguraci jednak zónové soubory pro přímý překlad (například `cz.vutbr.fit`) a jednak zónové soubory pro zpětný překlad

(například 147.229). Jak vidíme, jsou zónové soubory pro přehlednost pojmenovány názvem domény, která je v zónovém souboru uložena. To znamená, že zónový soubor `cz.vutbr.cz` obsahuje záznamy z domény `fit.vutbr.cz`, případně subdomén dané domény. Zónový soubor 147.229 zase obsahuje reverzní záznamy, které obsahují IP adresy z prostoru 147.229.0.0/16. Zónový soubor pro reverzní záznamy musí kromě reverzních záznamů také obsahovat záznam SOA, případně záznamy NS. Vlastní záznamy PTR musí ukazovat pouze na jedno doménové jméno – kanonické jména. Reverzní záznamy nesmí obsahovat ukazatel na aliasy.

Pokud při vytváření záznamů v DNS zapomeneme pro danou doménovou adresu vytvořit také zpětný záznam, obvykle způsobí chybějící záznamy PTR problémy při autentizaci. Týká se to například služeb SMTP, FTP, rsh, ssh a dalších. Důvodem je, že tyto služby často kontrolují domény, ze kterých se uživatelé připojují. Pokud se počítač zkouší připojit z nějaké IP adresy, server nejprve přeloží IP adresu na doménové jméno dotazem na záznam PTR. Pak např. zkontroluje, zda daná doména se nachází v seznamu nebezpečných domén. Pokud chybí záznam PTR, překlad neproběhne a spojení je odmítnuto.

Záznam TXT – Text

Záznam TXT uchovává v DNS textová data týkající se dodatečných informací o doméně, serveru, správci apod. daného uzlu ve stromu DNS, viz následující příklad. Záznamy TXT se v praxi příliš nepoužívají.

```
fi.muni.cz. 1773 IN TXT 'Masaryk University Brno'
fi.muni.cz. 1773 IN TXT 'Botanicka 68a, 602 00, Brno, Czech republic'
fi.muni.cz. 1773 IN TXT 'Faculty of Informatics'
```

Záznam SRV – Service Record

Záznam SRV [2] slouží pro lokalizaci služeb a serverů. Může se také použít pro distribuce zátěže či zálohování služeb, podobně jako je tomu u záznamu MX. Zápis používá speciální notaci pro zápis služby a protokolu, nad kterým služba pracuje. Obecný formát záznamu je

```
_Service._Proto.Name TTL Class SRV Priority Weight Port Target
```

Název služby a protokolu se odděluje od názvu doménového jména podtržítkem, viz následující příklad:

```
_http._tcp.www.hello.edu. IN SRV 0 2 80 www.hello.edu.      # rozdělení zátěže
                               IN SRV 0 1 80 www2.hello.edu.
                               IN SRV 1 1 8000 backup.hello.edu. # prioritá 1 (záloha)
_gopher._tcp.hello.edu     IN SRV 0 0 0 .                  # služba neběží
```

Kromě názvu a adresy obsahuje záznam SRV také prioritu, váhu, port a cílový server, kde běží žádaná služba. Priorita se chová podobně jako u MX záznamu. Váha umožňuje administrátorovi rozložit zátěž na více strojů. Pokud má například první záznam váhu 1 a druhý váhu 2, rozdělí se zátěž v poměru 1:2, tj. druhý stroj dostane dvakrát více požadavků než ten první. Hodnota Port určuje číslo portu, na kterém daná služba běží. Pokud chceme oznámit, že daná služba neběží, jako cíl zadáme tečku.

V dnešní době se záznam SRV používá například u IP telefonie, kdy klient zjišťuje adresu služby, např. SIP serveru, a číslo portu, na kterém služba běží. Následující dotazem může například IP telefon umístění v doméně `vutbr.cz` lokalizovat nejbližší SIP server a port, na

kteře služba SIP běží. Využije se k tomu dotaz na službu SIP nad protokolem UDP, viz ukázka:

```
/home/matousp> nslookup -type=SRV _sip._udp.vutbr.cz
Server:      147.229.9.43
Address:     147.229.9.43#53

_sip._udp.vutbr.cz      service = 0 1 5060 vampire.net.vutbr.cz.
```

Záznam NAPTR – Naming Authority Pointer

Záznam typu NAPTR definovaný RFC 3403 [24] byl navržen pro mapování řetězců na data. Záznam slouží jako podpora pro dynamicky konfigurovatelné systémy DDDS (Dynamic Delegation Discovery Systems). Při vyhledávání dat se nad řetězcí iterativně používají transformační pravidla dokud není dosaženo ukončující podmínky (podobně jako při prepisování nonterminálu v gramatikách).

Záznam NAPTR obsahuje regulární řetězec, který klientský program využije k přepsání řetězce v doménové adrese. Formát záznamu NAPTR obsahuje více hodnot, viz následující ukázka:

NAPTR Order Preference Flags Services Regexp Replacement

První hodnota **Order** určuje, v jakém pořadí se záznamy NAPTR vyhodnocují. Seznam záznamů NAPTR je řazený a zpracovává se od nejmenší hodnoty **Order** k nejvyšší. Pokud mají dva záznamy stejnou hodnotu, tak patří ke stejnému pravidlu a výběr záleží na kombinaci hodnot **Preference** a **Services**.

Hodnota **Preference** určuje, v jakém pořadí budou zpracovávány záznamy NAPTR se stejným pořadím **Order**. Záznam s nižší hodnotou má vyšší prioritu (podobně jako u záznamů MX). Příznaky **Flags** řídí proces přepisování a interpretaci záznamů. Jejich hodnota závisí na aplikaci. Hodnota **Services** popisuje parametry služby. Položka **Regex** obsahuje regulární výraz, který se aplikuje na vstup. Hodnota **Replacement** uvádí kanonické doménové jméno, které se doplní, pokud je součástí regulárního výrazu operace nahrazení. V praxi se parametr **Replacement** příliš nepoužívá.

Následující záznam demonstuje použití záznamu NAPTR pro převod telefonního čísla zapsaného v telekomunikačním standardu E.164 na URI protokolu SIP (Session Initiation Protocol, viz [16]). Tento převod se používá v IP telefonii, kdy uživatel místo adresy SIP vytvoří na IP telefonu telefonní číslo. Pomocí záznamu NAPTR dojde k mapování telefonního čísla ve formátu E.164 na URI. Tento systém mapování se nazývá ENUM (E.164 Number Mapping)[4, 14]. Záznamy NAPTR podobně jako záznamy PTR využívají speciální větev `e164.arpa.` stromu DNS. Následující příklad ukazuje použití záznamu NAPTR pro překlad ENUM:

```
$ORIGIN 4.1.1.4.5.0.2.4.e164.arpa. IN NAPTR 10 50 "u"  
                "E2U+sip" "!~\\+(.*)$!sip:\\1@vutbr.cz!" .
```

Tento záznam slouží k převodu telefonní čísla VUT s předčíslem +42054114 podle standardu E.164 na adresu SIP. Telefonní čísla jsou registrována v doméně 4.1.1.4.5.0.2.4.e164.arpa. Pro překlad všech telefonních čísel z daným předčíslym nám stačí jediný záznam v

DNS. Je to díky tomu, že se využívá regulární výraz. Díky tomu nemusíme pro každou klapku přidávat záznam v daném reverzním podstromu.

Výše uvedený záznam NAPTR obsahuje příznak "u", který říká, že pravidlo je koncové (tj. dále se nepřekládá) a jeho výsledkem je URI. Znak "!" v regulárním výrazu odděluje část, která se použije pro porovnání od části, která je výsledkem substituce. Tento regulární výraz odstraní z telefonního čísla ve formát E.164 znak "+". Všechny ostatní znaky až do konce vstupního řetězce (vyjádřené zápisem "(.*)") se vloží na místo ve výsledném řetězci odpovídající parametru "\1". Parametr tohoto typu se nazývá zpětná reference a odkazuje se na první závorku substituovaného výrazu. S tímto zápisem se můžeme setkat např. v programovacím jazyku Perl. Význam použití regulárních výrazů je podrobně popsán například v RFC 2915 [25].

V našem případě se například číslo +420541141108 zapsané ve stromě DNS jako 8.0.1.1.4.1.1.4.5.0.2.4.e164.arpa převede na adresu sip:420541141008@vutbr.cz. Aplikaci regulárního výrazu neprovádí serveru DNS, ale klient SIP, který žádá o záznam NAPTR. To ukazuje i následující dotaz na záznam NAPTR:

```
nslookup -type=naptr 8.0.1.1.4.1.1.4.5.0.2.4.e164.arpa
```

```
8.0.1.1.4.1.1.4.5.0.2.4.e164.arpa naptr = 200 50 "u" "E2U+h323"
"!~\\+(.*)" $!h323:\\10h323.vutbr.cz!" .
8.0.1.1.4.1.1.4.5.0.2.4.e164.arpa naptr = 100 50 "u" "E2U+sip"
"!~\\+(.*)" $!sip:\\1@vutbr.cz!" .
```

Výsledkem dotazu je celý záznam NAPTR obsahující regulární výraz, který klient sám aplikuje na telefonní číslo. Za zmínku stojí i tečka na konci výrazu. Tato tečka označuje prázdné pole typu Replacement. Použití záznamu NAPTR pro IP telefonii se budeme věnovat v kapitole 7.

Záznam LOC – Location

Záznam typu LOC definovaný standardem RFC 1876 [6] umožňuje administrátorovi zapsat do DNS údaje o umístění počítače, sítě či dalších zdrojů. Popis lokace zahrnuje informace o zeměpisné šířce (latitude), délce (longitude) a výšce (altitude). Zeměpisná šířka a délka mohou být vyjádřeny ve stupních (minutách a sekundách), výška se udává v metrech. Následující ukázka ukazuje použití záznamu LOC pro serveru kazi.fit.vutbr.cz:

```
kazi.fit.vutbr.cz. 14400 IN LOC 49 13 35.000 N 16 35 45.000 E 195.00m 1m 10000m 10m
```

Výše uvedený příklad ukazuje záznam LOC pro server SW1A2AA.find.me.uk, jehož umístění je zadáno jako souřadnice 49°13'35.000" severní šířky a 16°35'45.000" východní délky, 195 metrů nad hladinou moře. Dále záznam obsahuje nepovinné údaje o velikosti entity, tj. délka 1 m (což pro server asi stačí) a přesnosti polohy, tj. 10 km metrů horizontálně a 10 metrů vertikálně.

Otázkou je, k čemu tyto informace slouží. Na jednu stranu mohou udávat fyzickou cestu paketů, napojení na GPS apod. Na druhou stranu je tu bezpečnostní riziko a také fakt, že poloha serverů se mění. V současné době najdeme záznamy typu LOC v DNS spíše výjimečně.

AAAA – IPv6 Address

Posledním záznamem, o kterém se zde podrobněji zmíníme, je záznam typu AAAA. Tento záznam definovaný standardem RFC 3596 [35] mapuje doménové adresy na IP adresy verze 6 (IPv6). Zápis je vcelku jednoduchý, podobně jako u záznamu A. Následující ukázka obsahuje příklady dvou záznamů typu AAAA a také ukázkou reverzního záznamu pro IPv6 adresu serveru www.cesnet.cz.

```
isa.fit.vutbr.cz. 14400 IN AAAA 2001:67c:1220:8b0::93e5:b012
www.cesnet.cz. 85443 IN AAAA 2001:718:1:101::4

4.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.1.0.1.0.0.0.8.1.7.0.1.0.0.2.ip6.arpa.
86400 IN PTR www.cesnet.cz.
```

Složitější je vytváření reverzních záznamů pro adresy IPv6. Tyto adresy patří do speciální domény **ip6.arpa.**, která obsahuje reverzní záznamy adresového prostoru IPv6. Protože je každá subdoména stromu **ip6.arpa.** čtyřbitová, tvoří reverzní doménovou adresu IPv6 celkem 32 šestnáctikových číslic oddělených tečkou. Zároveň se zde nemůže použít notace zkracování adres IPv6 pomocí znaků ":" a ":::". Pro to se pro generování reverzních adres IPv6 používají automatizované nástroje (např. **sipcalc**), které pro danou adresu IPv6 vygenerují její reverzní záznam.

Pro síť s IPv6 a IPv4 může mít jedno síťové rozhraní adresu IPv4 i IPv6. V tom případě jsou pak v DNS oba záznamy – A i AAAA.

3.3.3 Přehled záznamů DNS

Jak jsme si ukázali v předchozích odstavcích, datový prostor doménových adres obsahuje nejen překlady doménových adres na IP adresu (záznamy typu A), ale také spoustu dalších informací týkající se domén, služeb, regulárních výrazů a podobě. Každý překlad DNS lze jednoduše popsat jako mapování (funkci), která převádí jeden typ informace na jiný. Pro správné využití systému DNS je nezbytné správně rozlišit jednotlivé typy mapování. V tabulce 3.6 uvádíme přehled základních typů typů DNS a jejich použití.

Záznam	Mapování	Příklad
A	doménové jméno → IP adresa	tereza.fit.vutbr.cz → 147.229.9.22
PTR	IP adresa → doménové jméno	22.9.229.147.in-addr.arpa. → tereza.fit.vutbr.cz.
NS	doména → doménový server	fit.vutbr.cz. → gate.fee.vutbr.cz.
MX	doména → poštovní server	fit.vutbr.cz. → kazi.fit.vutbr.cz.
SOA	doména → identifikace správce	fit.vutbr.cz. → boco.fee.vutbr.cz. michal.fit.vutbr.cz. 200710032 10800 3600 604800 86400
CNAME	doménové jméno → doménové jméno	www.fit.vutbr.cz. → tereza.fit.vutbr.cz.
AAAA	doménové jméno → IPv6 adresa	www.cesnet.cz. → 2001:718:1:101:204:23ff:fe52:221a
PTR	IPv6 adresa → doménové jméno	
	a.1.2.2.5.e.f.f.f.3.2.4.0.2.0.1.0.1.0.0.0.8.1.7.0.1.0.0.2.ip6.arpa. → www.cesnet.cz.	

Tabulka 3.6: Přehled základních typů překladů DNS

Z tabulky je například vidět, že pouze záznam typu A překládá doménovou adresu na IP adresu. Pokud definujeme poštovní server pomocí záznamu MX, resp. doménový server pomocí záznamu NS, překládá se doména na doménovou adresu poštovního serveru, resp.

DNS serveru, nikoliv na jejich IP adresu. Překlad na IP adresu zajistí až vyhledání záznamu typu A či AAAA.

3.3.4 Rozdělování zátěže pomocí rotace záznamů

Novější verze serverů DNS umožňují rozdělit zátěž na více serverů pomocí tzv. záznamů s posunutými adresami (shuffle address records). Znamená to, že na jeden dotaz odpovídá DNS server různými IP adresami, které pravidelně rotuje. Nejedná se o *vyvažování zátěže* (load balancing), které řeší například záznamy SRV. Zde se dotazy rozdělují deterministicky a rovnoměrně, nikoliv podle skutečného vytížení serverů. Mluvíme tedy o *rozdělování zátěže* (load distribution). U některých serverů DNS (např. BIND 9) je možné pomocí volby `rrset-order` v konfiguračním souboru nastavit způsob rotace záznamů, například fixní, cyklický či náhodný.

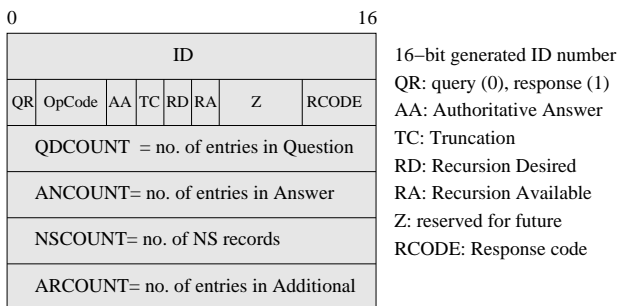
Pro implementaci rotace záznamů se využívá více záznamů typu A, které ukazují na stejné jméno počítače, ale mají různou IP adresu. Pokud je v zóně více záznamů typu A pro doménové jméno, server automaticky přiděluje různé IP adresy (rovnoměrně rotuje záznamy, tzv. round-robin load distribution), viz následující příklad.

```
foo.bar.baz. 60 IN A 192.168.1.1
foo.bar.baz. 60 IN A 192.168.1.2
foo.bar.baz. 60 IN A 192.168.1.3
```

Tento přístup je vhodný, pokud máme více ekvivalentních síťových zdrojů, například servery FTP či WWW se stejným obsahem (mirroring). U záznamů je vhodné nastavit menší hodnotu TTL, aby záložní servery, které nepodporují vícenásobné záznamy, tyto záznamy po krátké době vymazaly z paměti cache.

3.4 Přenos dat a komunikace v DNS

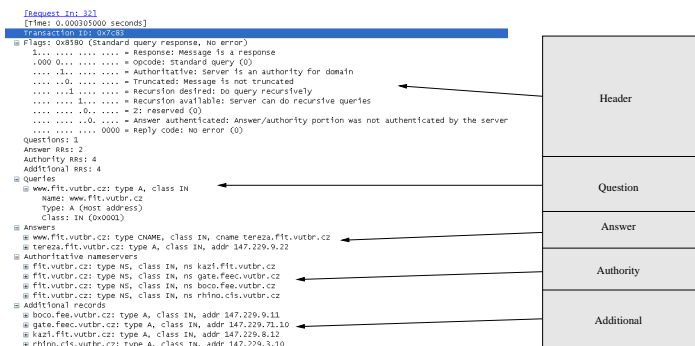
Protokol DNS a veškerá komunikace v systému DNS je popsána standardem RFC 1035 [28]. Protokol DNS používá pro posílání dotazů transportní protokol UDP s číslem portu 53. Velikost paketů UDP je standardem DNS omezena na 512 bytů. Delší zprávy je nutné rozdělit do více paketů přenášených nad UDP pomocí bitu TC (TrunCation) v hlavičce protokolu DNS (obr. 3.9). Protože jde o nespolehlivou komunikaci, musí aplikace – v případě ztráty paketu



Obrázek 3.9: Hlavička paketu DNS podle RFC 1035

– poslat dotaz na systém DNS opakovaně. Pakety UDP také mohou přijít v jiném pořadí, než klient žádal. Pro identifikaci odpovědi slouží 16-bitové číslo v hlavičce paketu UDP. Pro přenos zón se nepoužívá transportní protokol UDP ale TCP.

Paket DNS tvoří pět základních položek (obrázek č. 3.10): hlavičku (header), částu obsahu-



Obrázek 3.10: Formát paketu DNS

jící dotaz (question), část s odpovědí (answer), záznam obsahující informace o záznamech NS (authority) a sekce obsahující doplňující informace (additional). Hlavička je povinná položka paketu DNS. Tvoří ji pole pevné délky, které obsahuje informace o zbývajících částech paketu – počet záznamů v dalších sekcích paketu, viz obrázek 3.9, typ paketu (dotaz/odpověď), příznaky pro rekurzivní vyhledávání (RD, RA), rozdělení odpovědi do více paketů UDP (TC) či typ odpovědi (RCODE).

3.4.1 Aktualizace zónových dat

Vytvoření, přidávání, změnu či rušení dat v zónovém souboru můžeme provádět ručně editací textového souboru nebo pomocí automatizovaných nástrojů. Důležité je si uvědomit, že tyto změny lze provádět pouze na primárním serveru DNS. Pokud provedeme změnu například na sekundárním serveru, přepíše se tato změna při nejbližší synchronizaci. Aktualizace zónového souboru tvoří následující kroky:

1. Aktualizace sériového čísla v SOA záznamů příslušného zónového souboru.

```
@ IN SOA isa.fit.vutbr.cz. root.isa.fit.vutbr.cz. (
    2008073002 ; Serial
    3600 ; Refresh
    900 ; Retry
    3600000 ; Expire
    3600 ) ; Minimum
```

2. Přidání, změna či odstranění záznamů typu A, CNAME, MX, apod. v přímém zónovém souboru, např. `cz.vutbr.fit.netlab`.


```
pc12 IN A 10.1.1.12
www IN CNAME pc12
```

3. Přidání, změna či odstranění záznamu typu PTR v reverzním zónovém souboru, např. 10.1.1.

```
12 IN PTR pc12.fit.vutbr.cz.
```

4. Restart primárního serveru DNS, kdy si aplikace načte aktualizovaná data do paměti.

3.4.2 Dynamické změny v DNS

Prostředí internetu lze považovat za dynamické prostředí, kde se neustále mění topologie, uživatelé se připojují a odpojují od sítě. Toto chování má vliv na změny v databázi DNS. Jde například o uživatele, kteří získávají IP adresy přes DHCP pomocí vytáčeného spojení či kabelového modemu. Také tyto dynamicky přidělené IP adresy je potřeba mapovat na doménové adresy. Z pohledu systému DNS to vyžaduje přidání nového záznamu typu A. Protože není možné tyto změny provádět ručně, vznikl v roce 1997 standard RFC 2136 [31], který popisuje *dynamické aktualizace v DNS* a příkaz `UPDATE`.

Dynamické aktualizace umožňují přidávat, měnit a rušit záznamy DNS v rámci dané domény. Většina dynamických aktualizací je využívána servery DHCP, které přidělí automaticky počítači IP adresu a zároveň zaregistrují převod IP adresy na doménovou adresu v DNS. Využívá se přitom funkce resolveru `ns_update()`. Existuje také program `nsupdate`, který spustí změny z příkazové řádky. Následující příklad přidá do databáze DNS nový záznam typu A v doméně `netlab.fit.vutbr.cz` uvedené v příkazu `prereq`.

```
% nsupdate
> prereq nxdomain netlab.fit.vutbr.cz
> update add pc3.netlab.fit.vutbr.cz. A 10.10.10.134
>
```

Jakmile server provede dynamickou aktualizaci, musí inkrementovat sériové číslo zónového souboru, které oznamuje změnu sekundárním serverům. Toto se také děje automaticky. Primární server však nemusí z hlediska efektivity měnit sériové číslo záznamu SOA při každé dynamické změně. Některé servery (např. BIND 8) provedou změnu sériového čísla až po stu aktualizacích nebo po pěti minutách (podle toho, co nastane dříve). Souvisí to se zónovým přenosem, o kterém se budeme bavit v další kapitole. Po každé změně sériového čísla může primární server poslat zprávu DNS NOTIFY a vyvolat zónový přenos mezi primárním serverem a sekundárními servery, což s sebou přináší zvýšený přenos a zpracování aktualizací.

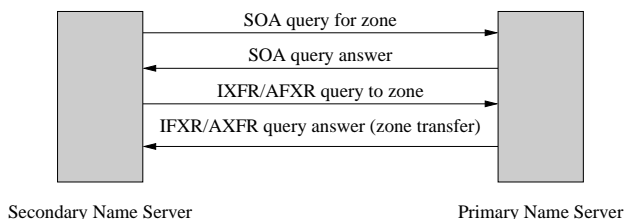
Důležitou částí dynamické aktualizace je také autentizace resolveru, který aktualizaci provádí. Zabezpečení dynamické aktualizace popisuje podrobněji RFC 3007[5] a využívá mechanismy TSIG a DNSSEC (viz též kapitola 3.5).

3.4.3 Zónový přenos

Většina komunikace v systému DNS probíhá mezi resolvery a servery. Resolver pošle serveru DNS jednoduchý dotaz na získání informace ze jmenného prostoru DNS. Server informaci vyhledá a pošle požadovaná data resolveru. Pokud informaci nezná, může se dotázat dalších serverů.

Jiným způsobem komunikace v systému DNS je *zónový přenos*, kdy dochází k přenosu zónových souborů mezi primárním serverem (master) a sekundárním serverem (slave). Tradičně používají sekundární servery DNS schéma vyzývání (polling). Interval vyzývání odpovídá intervalu aktualizací (refresh interval) uvedený v záznamu SOA. Aktualizace nastává tehdy, když expiruje doba platnosti záznamů. Teprve v tomto okamžiku sekundární server zjišťuje případné změny v záznamech na primárním serveru DNS a tyto změny nahrává i do své databáze.

Tento přístup může způsobit dočasnou nekonzistenci dat. Bylo by vhodnější, kdyby primární server informoval podřízené sekundární servery o tom, že došlo ke změně v zónovém souboru, změnila. Standard RFC 1996[29] přináší mechanismus zasílání upozornění o změnách formou dotazu DNS NOTIFY.



Obrázek 3.11: Schéma zónového přenosu

Když primární server zjistí podle sériového čísla, že se změnila zóna, pošle speciální oznámení všem sekundárním serverům pro tuto zónu. Jejich adresy jsou uloženy v záznamu NS pro danou zónu. Jakmile sekundární server obdrží zprávu NOTIFY, potvrdí ji. Primární server poté přestane zprávu vysílat. Sekundární server poté jedná, jako kdyby došlo k uplynutí času aktualizace (refresh time) a požádá o přenos celé zóny (příkaz AXFR).

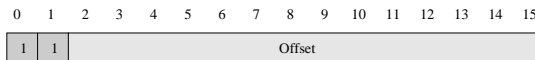
Pokud je zóna příliš obsáhlá, není vhodné při aktualizaci jednoho záznamu přenášet všechny informace. K tomu se používá tzv. přírůstkový přenos zón (incremental zone transfer, IXFR). Při tomto přenosu řekne podřízený sekundární server primárnímu serveru, kterou verzi zóny má a požádáho o zaslání změn. Žádost IXFR v sobě obsahuje záznam SOA. Když server obdrží žádost o přírůstkový přenos, projde si lokální záznamy změn ve své databázi, kde je popsáno, čím se liší verze sekundárního a primárního serveru. Veškeré nalezené změny přepoše primární server sekundárnímu serveru nazpět. Pokud není nalezen záznam o změnách, pošle primární server celou zónu, jako kdyby obdržel příkaz AXFR.

Přenos celého zónového souboru je citlivá otázka z hlediska bezpečnosti. Při jednoduchých dotazech od resolveru odpovídá server DNS pouze na konkrétní dotazy. Uživatel neví, jaké záznamy existují v dané doméně a jaké IP adresy jsou zaregistrovány v DNS. Při načítání zónového souboru získává server informace o celé doméně (či subdoméně). Z tohoto důvodu je v konfiguraci primárního serveru DNS obvykle nastavena IP adresa sekundárního serveru, který může požádat o zónový přenos, kolik serverů se může současně přihlásit a podobně. Pro přenos zón se také doporučuje vytvořit zabezpečené spojení mezi oběma servery.

3.4.4 Komprese paketů DNS

Když se podíváme síťovým analyzátořem na komunikaci DNS, zjistíme, že přenášená data jsou daleko menší, než obsah, který vypíše například program `nslookup`. Je to proto, že při vytváření paketů DNS dochází ke komprimování textových dat. Tato činnost je vhodná, protože některé řetězce se v paketu DNS opakují, například jména doména apod. Proto se místo výskytu jména používá ukazatel na první výskyt řetězce.

Každé doménové jméno se skládá z posloupnosti řetězců obsahující jména domén (například "eva", "fit", "vutbr", "cz"). Jak vidíme, jména domén mají proměnlivou délku. Narozdíl například od jazyka C, není řetězec v paketu DNS ukončen znakem 0x00, ale zapisuje se v paketech DNS jinak. Doménové jméno se ukládá jako posloupnost dvojic délka/hodnota. Každá dvojice obsahuje délku domény a název domény. Tato posloupnost dvojic je ukončena nulovým oktetem. Například doménové jméno "eva.fit.vutbr.cz" se v paketu DNS zapisuje jako řetězec znaků "3 eva 3 fit 5 vutbr 2 cz 0" (v zápisu ASCII)³, kde první byte označuje délku řetězce, který následuje. Posloupnost je ukončena nulovým řetězcem, což je vlastně název kořenového uzlu ve stromu jmen DNS.



Obrázek 3.12: Formát ukazatele při kompresi paketů DNS

Toto je jeden typ uložení doménového jména. Protože se zejména suffixy doménových jmen opakují, využívá se při ukládání doménových jmen ukazatelů na předchozí výskyt jména (či jeho části). Zda se jedná o řetězec obsahující doménové jméno nebo o ukazatel, poznáme podle prvních dvou bitů délky, které mají speciální význam. Pokud jsou první dva bity tohoto bytu nulové, zbývajících šest bitů obsahuje délku řetězce, který následuje počínaje dalším oktetem. Pokud jsou první dva bity jedničkové (např. 0xC0), pak následující bity nevyjadřují délku paketu, ale ukazatel (pointer) na předchozí výskyt řetězce v paketu. Tento ukazatel je vlastně offset od začátku paketu DNS, kde se zmíněný řetězec vyskytuje. Offset tvoří jednak posledních šest bitů bytu s délkou, který začíná posloupností 11 (binárně), a také následujícím bytem. Dohromady tvoří ukazatel 14 bitů, viz obrázek 3.12.

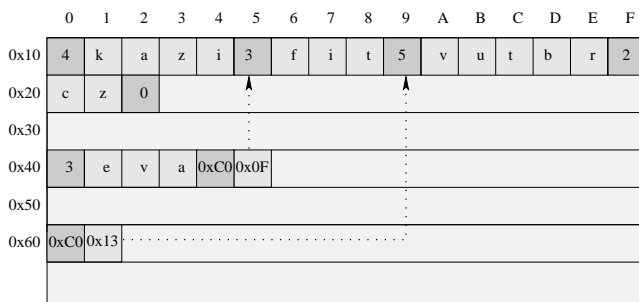
Použití ukazatelů vede k významné redukci velikosti paketu. Opakující se výskyty části doménového jména v paketu se nahradí ukazatelem na první výskyt. Pole **Offset** určuje vzdálenost od začátku paketu DNS, tj. od prvního oktetu DNS. Příklad komprese doménových jmen `kazi.fit.vutbr.cz`, `eva.fit.vutbr.cz`, `vutbr.cz` je naznačen na obrázku 3.13.

Protože první dva bity délky se používají k rozlišení řetězce od ukazatele, pro délku se použije pouze následujících šest bitů. Proto může mít řetězec obsahující doménové jméno délku maximálně $2^6 - 1$, tj. 63 bytů. Standard také uvádí, že maximální délka doménového jména (konkatenace všech domén) je 255 bytů.

3.4.5 Programování komunikace v systému DNS

Přestože systém DNS existuje desítky let a můžeme nalézt spoustu aplikací typu server i klient, občas je zapotřebí vytvořit vlastní program, který se dotazuje na informace v DNS.

³V bytovém zápisu to bude posloupnost bytů "03 65 76 61 03 66 69 74 05 76 75 74 62 72 02 63 7a 00"



Obrázek 3.13: Příklad komprese doménových jmen v paketu DNS

Pro programování lze použít schránky BSD, o kterých jsme mluvili v předchozí kapitole. Vytvoření zprávy DNS je z pohledu programátora složitější, neboť hlavička obsahuje spoustu parametrů a voleb, pro každý typ záznamu se trošku jiný formát sekci dotaz a odpověď, pakety využívají komprimaci textový řetězců doménových jmen a další.

Proto se doporučuje využití knihoven `<resolv.h>` (man resolver) a `<arpa/nameser.h>`, které obsahují funkce pro posílání dotazů DNS a zpracování odpovědí. Každý program, který využívá tyto funkce, musí obsahovat následující hlavičkové soubory:

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>
```

V této části si představíme funkce pro vytváření a zasílání požadavků na DNS, procedury pro zpracování odpovědí od serveru a ukážeme si příklad aplikace – klienta DNS.

Funkce pro vytváření a zasílání požadavků

Následující funkce slouží k vytvoření požadavků na DNS a jejich odeslání na server. Funkce patří mezi systémové funkce operačního systému. Pro komunikaci není nutné explicitně vytvářet spojení UDP. Toto zajišťují samotné funkce.

res_search() – příprava dotazu na DNS

```
int res_search(const char *dname, int class, int type, u_char *answer, int len);
```

Tato funkce pracuje na nejvyšší úrovni. Připravuje odeslání dotazu na DNS. Při volání ji používá např. funkce `gethostbyname()`. Funkce dostane doménové jméno `dname`, pomocí seznamu v resolveru tam doplní doménu a takto připravený dotaz předá funkci `res_query()` pro odeslání dotazu. Dotaz na doménové jméno je obsažen v prvním parametru `dname`, dále následují parametry dotazu a řetězec pro odpověď `answer` o velikosti `len`. Pokud není doménové jméno absolutní (FQDN), doplní se doména podle informací v proměnné `HOSTALIASES`.

res_query() – vytvoření dotazu na DNS ů `int res_query(const char *dname, int class, int type, u_char *answer, int len);`

Tato funkce vytváří dotaz na záznamy v DNS. Pomocí funkce **res_mkquery()** vytvoří požadavek v binární podobě, zašle jej funkcí **res_send()** serveru a vyhodnotí, zda v pořádku došel. Má podobné parametry jako **res_search()**. Liší se tím, že nedoplňuje doménové jméno.

res_mkquery() – vytváří zprávu do DNS

`int res_mkquery(int op, const char *dname, int class, int type, const u_char *data, int datalen, const u_char *newrr, u_char *buf, int buflen);`

Funkce tvoří paket pro DNS. Vyplňuje políčka v hlavičce protokolu DNS, provádí kompresi doménových jmen, doplňuje část dotazu v paketu DNS. Vytvoří dotaz na záznam typu `op` v DNS. Vytvořený paket umístí do bufferu `buf`.

res_send() – zaslání dotazu na server

`int res_send(const char *msg, int msglen, u_char *answer, int anslen);`

Funkce pošle naformátovanou DNS zprávu `msg` a počká na odpověď, kterou uloží do proměnné `answer`. Data posílá pomocí protokolu UDP, může použít i TCP.

res_init() – inicializace struktury `_res`

`int res_init(void);` Funkce **res_init()** načte konfiguraci ze souboru `resolv.conf` a inicializuje strukturu `_res`. Tato struktura je používána dalšími funkcemi **res_xxx()**. Obsahuje adresy serverů DNS, maximální dobu pro čtení dat, implicitní domény a další nastavení získaná z lokální konfigurace **resolveru**.

Funkce pro zpracování odpovědi

Pro zpracování odpovědi od serveru se používají funkce z knihovny `<arpa/nameser.h>`:

ns_initparse() – inicializace a vytvoření datové struktury `ns_msg` pro zpracování odpovědi dalšími funkcemi `ns_xxx()`

ns_msg_base(), **ns_msg_end()**, **ns_msg_size()** – vrátí ukazatel na začátek a konec paketu včetně délky paketu.

ns_msg_count() – vrátí počet záznamů v jednotlivých sekcích paketu DNS, např. question, zone, answer, nameserver, update, additional a další.

ns_rr_name(), **ns_rr_type()**, **ns_rr_class()**, **ns_rr_rdata()**, **ns_rr_xxx()** – vrátí obsah příslušného pole ze zprávy DNS, např. pole `name`, `type`, `class`, `rdata` a dalších.

Příklad aplikace – zaslání dotazu na DNS a zpracování odpovědi

```
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <arpa/inet.h>
#include <resolv.h>
#include <netdb.h>
```

```

int resolve(const char *dname){
    in_addr_t addr4;
    register int i;
    int nRet, ipAddr[4] = {0, 0, 0, 0};
    char buf[lndname];
    if ((addr4 = inet_network(dname)) != -1){
        for (i = 0; addr4; ){
            ipAddr[i++] = addr4 & 0xFF;
            addr4 >>= 8;
        }
        sprintf(buf, "%u.%u.%u.%u.in-addr.arpa", ipAddr[i % 4], ipAddr[(i+1) % 4],
            ipAddr[(i+2) % 4], ipAddr[(i+3) % 4]);
        nRet = runDnsQuery(buf, ns_t_ptr, TRUE);          // zjišťuje záznam PTR
    }
    else if ((nRet = runDnsQuery(dname, ns_t_a, FALSE))) // zjišťuje záznam typu A
        nRet = runDnsQuery(dname, ns_t_mx, FALSE);      // zjišťuje záznam typu MX
    return nRet;
}

int runDnsQuery(const char *dname, int nType, int bNoDataError){
    u_char pResAnswer[lanswer], Uncompressed[lndname];
    int nResAnswerLen, i, j;
    ns_msg hMsg;
    ns_rr rr;
    const u_char *p;
    // zaslání dotazu a načtení odpovědi
    if (nResAnswerLen = (res_search(dname, ns_c_in, nType, pResAnswer, lanswer)) < 0)
        err();
    if (ns_initparse(pResAnswer, nResAnswerLen, &hMsg)) err(); // zpracování odpovědi
    for (i = 0; i < ns_msg_count(hMsg, ns_s_an); i++){          // zpracování záznamu
        if (ns_parserr(&hMsg, ns_s_an, i, &rr) < 0) err();
        switch (ns_rr_type(rr)){
            case ns_t_cname:                                     // záznam typu CNAME
                if (nType == ns_t_mx) break;
                if (ns_name_uncompress(ns_msg_base(hMsg), ns_msg_end(hMsg), ns_rr_rdata(rr),
                    Uncompressed, lndname) < 0) err();
                printf("%s is a nickname for %s\n", ns_rr_name(rr), szUncompressed);
            case ns_t_a:                                         // záznam typu A
                printf("%s has address ", ns_rr_name(rr));
                p = ns_rr_rdata(rr);
            case ns_t_mx:                                         // záznam typu MX
                p = ns_rr_rdata(rr);
            case ns_t_ptr:                                       // záznam typu PTR
                if (ns_name_uncompress(ns_msg_base(hMsg), ...) < 0)
                    break;
        }
    }
    return TRUE;
}

int main(int argc, char *argv[]){
    for (int i = 1; i < argc; i++)
        resolve(argv[i]);
    return 0;
}

```

3.4.6 Zjišťování informací v systému DNS

Většina aplikačních programů používá pro překlad doménových adres a dotazování se na záznamy v DNS služby resolveru, který je součástí operačního systému. Pro uživatele jsou tyto služby transparentní a většinou o nich ani neví.

Pokud chceme získat nějaké informace ze systému DNS, můžeme využít speciální programy, které z DNS serveru komunikují. V následující části popíšeme tři základní programy pro komunikaci v DNS – `nslookup`, `host` a `dig`.

Program nslookup

Program `nslookup` slouží jako klient DNS, který posílá dotazy DNS serveru. Pracuje ve dvou režimech – interaktivním a řádkovém. Pokud spustíme program bez parametrů, přepneme se do interaktivního režimu. Většinou se používá `nslookup` v neinteraktivním řádkovém režimu. Po spuštění můžeme specifikovat, ke kterému serveru se přihlašujeme a jaké typy záznamů požadujeme. Pokud ne zadáme typ záznamu, program vyhledává záznamy typu A a PTR.

V následujícím příkladu zjišťujeme mailový server pro doménu `stud.fit.vutbr.cz`. Ptáme se tedy serveru DNS `kazi.fit.vutbr.cz` na záznamy typu MX pro danou doménu. Server vrátí dva servery: primární poštovní server (`eva.fit.vutbr.cz`) a sekundární poštovní server (`kazi.fit.vutbr.cz`).

```
/home/matousp> nslookup -type=MX stud.fit.vutbr.cz kazi.fit.vutbr.cz
Server:          kazi.fit.vutbr.cz
Address:         147.229.8.12#53

stud.fit.vutbr.cz      mail exchanger = 10 eva.fit.vutbr.cz.
stud.fit.vutbr.cz      mail exchanger = 20 kazi.fit.vutbr.cz.
```

Narozdíl od resolveru komunikuje program `nslookup` pouze s jedním serverem, tj. neposílá rekurzivní dotazy. Pokud nenajde odpověď, musíme sami zkusit vyhledat doménový server, který obsahuje hledaná data. Program nepoužívá funkce knihovny `resolv.h`, ale vlastní funkce. Některé implementace programu podporují i přenos zón. Pokud program vrátí neautoritativní odpověď, je potřeba zjistit autoritativní server pro danou doménu a poslat dotaz na tento server. V následujícím serveru hledáme autoritativní odpověď na překlad doménového jména `www.cesnet.cz`.

```
1) /home/matousp> nslookup www.cesnet.cz // hledání odpovědi na lokální serveru DNS
Server:          147.229.9.43
Address:         147.229.9.43#53
```

```
Non-authoritative answer:
Name:   www.cesnet.cz
Address: 195.113.144.230
```

```
2) /home/matousp> nslookup -type=NS cesnet.cz // hledání autoritativního DNS serveru
Server:          147.229.9.43
Address:         147.229.9.43#53
```

```
Non-authoritative answer:
cesnet.cz      nameserver = ns.cesnet.cz.
cesnet.cz      nameserver = ns.cesnet.
cesnet.cz      nameserver = decsys.vsb.cz.
```

Authoritative answers can be found from:

```

3) nslookup www.cesnet.cz ns.cesnet.cz      // hledání autoritativní odpovědi
Server:      ns.cesnet.cz
Address:     195.113.144.194#53

Name:   www.cesnet.cz
Address: 195.113.144.230

```

Program host

Dalším užitečným programem pro posílání dotazů na DNS je program **host**. Používá se pro překlad doménových adres na IP adresy a zpět. Při použití parametrů je možné vyhledávat i jiné typů záznamů či žádat o přenos zón. Následující příklad ukazuje možnosti použití programu **host**.

```

/home/matousp> host www.cesnet.cz
www.cesnet.cz has address 195.113.144.230
www.cesnet.cz has IPv6 address 2001:718:1:101::4
www.cesnet.cz mail is handled by 100 rs.cesnet.cz.
www.cesnet.cz mail is handled by 10 mail.cesnet.cz.

```

```

/home/matousp> host -t mx stud.fit.vutbr.cz

stud.fit.vutbr.cz mail is handled by 20 kazi.fit.vutbr.cz.
stud.fit.vutbr.cz mail is handled by 10 eva.fit.vutbr.cz.

```

Program dig

Program **dig** (Domain Information Groper) je nástroj pro zaslání dotazů na server DNS. Program zobrazuje odpovědi tak, jak je vrací. Z tohoto důvodu jsou výpisy dlouhé. Pro základní dotazování jsou vhodnější nástroje **nslookup** nebo **host**.

Základní formát je "**dig @server name type**", kde **server** je jméno dotazovaného serveru DNS, **name** je doménové jméno záznamu, který hledáme a **type** určuje typ záznamu, například ANY, A, MX a podobně.

Dig implicitně vyhledává záznamy typu A. Výpis programu **dig** je uspořádán podobně jako formát paketu DNS – obsahuje část **HEADER**, sekce **QUERY**, **ANSWER**, **AUTHORITY** a **ADDITIONAL**, viz následující ukázka. Program umí také přenášet zóny.

```

/home/matousp> dig www.cesnet.cz
; <<>> DiG 9.6.2-P1 <<>> www.cesnet.cz
;; global options: +cmd
;; Got answer:
;; ->HEADER<- opcode: QUERY, status: NOERROR, id: 61045
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 3, ADDITIONAL: 2

;; QUESTION SECTION:
;www.cesnet.cz.                IN      A

;; ANSWER SECTION:
www.cesnet.cz.                63351   IN      A      195.113.144.230

;; AUTHORITY SECTION:

```



```

cesnet.cz.      85903  IN      NS      decsys.vsb.cz.
cesnet.cz.      85903  IN      NS      ns.cesnet.cz.
cesnet.cz.      85903  IN      NS      ns.ces.net.

;; ADDITIONAL SECTION:
ns.cesnet.cz.   85956  IN      A       195.113.144.194
ns.cesnet.cz.   85956  IN      AAAA    2001:718:1:1::2

;; Query time: 1 msec
;; SERVER: 147.229.9.43#53(147.229.9.43)
;; WHEN: Tue Oct 25 18:22:04 2011
;; MSG SIZE rcvd: 157

```

Podobně jako i další programy umožňuje **dig** nastavit způsob přenosu dat (UDP či TCP), možnost rekurzivního či nerekurzivního zpracování, cílový port a další. Pokud chceme vyhledávat v reverzním doménovém stromě, musíme zadat volbu **-x**.

3.5 Zabezpečení DNS

Služba DNS je veřejná. Využívá ji každý uživatel komunikující po internetu. Odpovědi systému DNS jsou většinou přijímány jako důvěryhodné. DNS však komunikuje přes nezabezpečený datových kanál, kde je možné odpovědi odchytit, změnit či podvrhnout. Slabým místem jsou také paměti cache záložních serverů. Pokud "vnutíme" serveru DNS informaci do paměti cache, tak ji dále neověřuje a na dotazy odpovídá uloženou hodnotou, aniž by se dotazoval autoritativního serveru. Útoky typu *cache poisoning* (otrávení paměti cache) vedly k vytvoření standardů na podepisování DNS zpráv – TSIG [30] a DNSSEC [32]. O typech útoků, bezpečnostních standardech a způsobu zabezpečení si povíme v této kapitole.

3.5.1 Bezpečnostní rizika v DNS

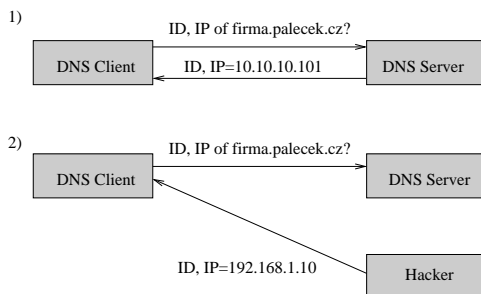
Dokument RFC 3833 [8] z roku 2004 se zabývá možnými způsoby ohrožení systému DNS a ukazuje, jak ochránit je třeba DNS chránit při zachování základního charakteru služby, tj. že jde o veřejnou službu, která je všeobecně dostupná kterémukoliv uživateli internetu. Mezi dva důležité úkoly, které je potřeba z pohledu bezpečnosti DNS zajistit, patří:

- integrita dat (tj. data nejsou změněna během přenosu)
- autentizace zdroje dat (tj. odesilatel dat mohou důvěřovat)

Základní mechanismus, který se používá pro zajištění integrity dat a autentizace zdroje dat, je systém veřejných klíčů a podepisování záznamů DNSSEC. Dále je to technika TSIG pro podepisování záznamů soukromým klíčem. Mechanismy DNSSEC a TSIG budou podrobně popsány v další kapitole. Mezi základní třídy útoků na systém DNS patří:

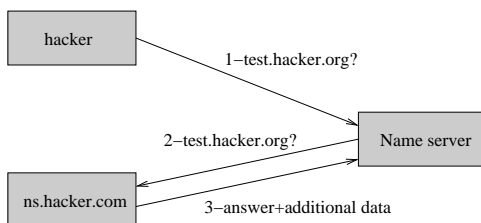
Odoslech paketů. Při tomto útoku útočník sleduje komunikaci a v případě dotazu na DNS vrátí nesprávnou odpověď, případně v odpovědi pozmnění nějaké informace. Tím se mu podaří například přesměrovat provoz na server s jinou IP adresou, posílání elektronické pošty apod. Řešením je zajištění integrity paketů DNS pomocí podepisování záznamů DNSSEC. Pro zónový přenos je vhodná například symetrická autentizace pomocí TSIG.

Hádání ID paketu a predikce odpovědi. Identifikační číslo paketu DNS je pouze 16-bitové. Číslo cílového portu serveru je známé (53), číslo portu klienta je také 16-bitová hodnota. Existuje tedy pouze 2^{32} možných kombinací těchto hodnot, což je proveditelné pro útok hrubou silou. Sledováním síťového provozu můžeme tyto hodnoty předvídat nebo hádat. Při odchycení dotazu je resolver velmi náchylný k přijetí falešné odpovědi. Resolver, který zkontroluje podpis DNSSEC či TSIG, pozná neautorizovanou odpověď. Na obrázku 3.14 je znázorněn případ, kdy útočník podvrhne odpověď a vnutí klientovi nesprávnou IP adresu serveru. K útoku potřebuje znát ID původního dotazu.



Obrázek 3.14: Podvržení odpovědi útočníkem

Zřetězení jmen (otrávení paměti cache). Zřetězení jmen je podmnžinou útoků typu otrávení vyrovnávací paměti cache. Základem útoku je vložení nesprávné informace do paměti cache. To lze dosáhnout například změnou informací v poli RDATA odpovědi, zejména v záznamech CNAME, NS a DNAME. Klient vyšle dotaz, na nějž mu útočník vnutí odpověď, která je pozměněná. V části RDATA jsou přidány do sekce **Additional** jiná data např. nesprávné IP adresy autoritativních serverů DNS apod, viz obr. 3.15.



Obrázek 3.15: Vložení nesprávných informací do paměti cache (otrávení)

Takový útok se odehrál na InterNIC v červenci 1997, kdy adresa www.internic.net byla přesměrována na IP adresu www.alternic.net. Hrozbou tohoto útoku je, že útočník může do odpovědi libovolné doménové adresy a podrobnější informace o záznamech, která útočník prohlásí za související se zaslaným dotazem. Zjištění takového útoku je velmi obtížné.

Většinu těchto útoků lze odvrátit kontrolou podpisů prostřednictvím DNSSEC, neboť resolver může ověřit, zda odesílatel zná tajný klíč, jehož odpovídající veřejný klíč je i s ověřením přístupný na veřejném místě v DNS.

Znemožnění služby (Denial of Service). Systém DNS je zranitelný útokem DoS, jak jsme si popisovali v kapitole 3.2.4. Mechanismy podepisování a autentizace DNSSEC a TSIG tyto útoky neřeší. Spíše je zhoršují, neboť podepisování je časově i výpočetně náročná operace. Řešení útoků DoS je na úrovni konfigurace DNS serveru (omezení počtu dotazů) či vlastní síti (kontrola počtu navázaných spojení z jedné IP adresy).

Odmítnutí domény. Odmítnutí domény souvisí s otázkou, zda kontrolovat (ověřovat) neexistenci domény. Problém je, zda by měl být resolver schopný detekovat zrušení dat útočником v odpovědi DNS. Tato otázka se stále dost diskutuje. Je jasné, že je potřeba ověřit neexistenci neúplně zapsaných (wildcards) záznamů. DNSSEC obsahuje mechanismus, který umožňuje určit, které autoritativní názvy existují v zóně a který typy autoritativních záznamů existují pro dané názvy pomocí záznamů NSEC a NSEC3.

3.5.2 TSIG – podepisování transakcí

Mechanismus TSIG (Transaction Signatures) popisuje autentizaci transakcí DNS pomocí symetrické kryptografie. TSIG je popsán ve standardu RFC 2845 [30] z roku 2000, v roce 2003 byl popsán aktualizovaný přístup nazvaný GSS-TSIG, viz RFC 3645 [34]. Mechanismus využívá autentizace se sdíleným tajným klíčem a jednocestnou hashovací funkci HMAC-MD5 [13]. TSIG, narozdíl od DNSSEC, nezajišťuje autentizaci a integritu samotných dat, ale pouze odesílatele a příjemce, tedy datových přenosů (transakcí) mezi nimi.

Podepisování TSIG využívá nový typ záznamu TSIG (250), který obsahuje autorizační kód (hash) celého záznamu DNS (bez pole TSIG) spolu s názvem algoritmu, času podpisu a dalších údajů. Záznam TSIG nesmí být uložen v paměti cache a neobjeví se ani v zónových souborech. Je jedinečný pro každou odesílanou zprávu, proto ho není nutné ukládat.

TSIG zajišťuje autentizaci zdroje (serveru DNS) a integritu dat pomocí autentizačního kódu (kryptografického součtu) HMAC-MD5. Záznam TSIG se vkládá do sekce **Additional**. Jakmile se sestaví paket DNS, spočítá se kontrolní součet a vloží záznam TSIG. Po přijetí zprávy odesílatel zkontroluje typ hašovacího algoritmu, název klíče a platnost autentizačního kódu. Spočítá se autentizační kód záznamu a porovná s hodnotou ve zprávě. Pokud se shodují, je záznam přijat. Klíče jsou definovány v konfiguračním souboru serveru DNS `named.conf`, viz následující ukázka pro zónu `palecek.cz`:

```
server ns1:                                server ns2:
  key ns1-ns2.palecek.cz. {                key ns1-ns2.palecek.cz. {
    algorithm hmac-md5;                     algorithm hmac-md5;
    secret 'RCMEsfegvM0VnuA==';             secret 'RCMEsfegvM0VnuA==';
  };                                         };
                                           server 192.249.249.3 {
                                           keys {ns1-ns2.palecek.cz.};
                                           }
  zone 'palecek.cz' {                       zone 'palecek.cz' {
    type master;                             type slave;
    file 'db.palecek.cz';                     masters {192.249.249.3};
    allow-transfer { key ns1-ns2.palecke.cz. }; file 'db2.palecek.cz';
  };                                         };
                                           };
```

Tento příklad ukazuje konfiguraci primárního serveru **ns1** a sekundárního serveru **ns2**. Oba servery sdílejí stejný klíč a stejný zabezpečovací algoritmus HMAC-MD5. Primární server přijímá pouze podepsané žádosti na přenos zóny **allow-transfer**. Položka **server** u **ns2** vytváří asociaci mezi serverem a klíčem. Tato položka říká, že veškeré žádosti na server 192.249.249.3 se budou podepisovat klíčem **ns1=ns2.palecek.cz**.

Klíč lze vytvořit příkazem **dnssec-keygen**, který vygeneruje záznam DNS (soubor **.key**) a soukromý klíč **.private**, viz následující ukázka.

```
>dnssec-keygen -a hmac-md5 -b 256 -n HOST palecek.cz

Kpalecek.cz.+157+22718.key:
  palecek.cz. IN KEY 512 3 157 6Vl4IroYv42NjHaJRybpQsXVvLDVEG4otheLKH1/io=

Kpalecek.cz.+157+22718.private:
  Private-key-format: v1.2
  Algorithm: 157 (HMAC_MD5)
  Key: 6Vl4IroYv42NjHaJRybpQsXVvLDVEG4otheLKH1/io=
  Bits: AAA=
```

TSIG předpokládá použití klíče pro každou dvojici serverů DNS. Toto je nevhodné pro servery komunikující s velkým množstvím jiných serverů, neboť počet klíčů roste exponenciálně s počtem serverů. Problémem je také v distribuci klíčů, neboť se jedná o soukromý klíč, který je potřeba bezpečným způsobem předat druhé straně.

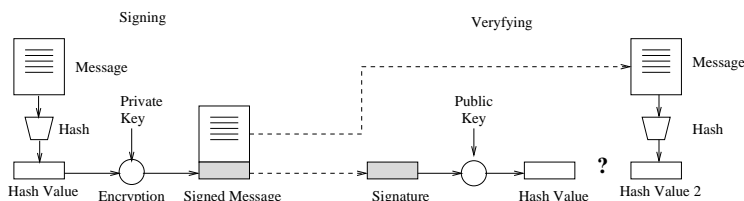
Mechanismu TSIG je možné použít i pro resolvers, kteří se dotazují jednoho serveru či několika málo serverů. Stejně tak je tento mechanismus vhodný pro záložní servery, které se odkazují na jeden konkrétní server. Například programy **dig** nebo **nsupdate** umí načíst klíč ze souborů **.key** a použít ho k zaslání podepsaných dynamických aktualizací. Použití pro resolvers na běžných pracovních stanicích je obtížné z hlediska distribuce klíčů. TSIG neumožňuje vytvářet bezpečné spojení se všemi servery DNS, ale pouze s těmi, s kterými se dohodl na klíčích. Pro autentizaci komunikace s veřejnými servery se používá DNSSEC.

3.5.3 DNSSEC – podepisování záznamů

Standard DNSSEC (DNS Security Extension)[32] z roku 2005 definuje rozšíření protokolu DNS pro zabezpečení přenosu dat v systému DNS pomocí asymetrické kryptografie s použitím veřejného a soukromého klíče. Narozdíl od symetrické kryptografie a mechanismu TSIG, existují u DNSSEC dva klíče – soukromý pro podepisování a veřejný pro ověření podpisu. Princip podepisování a ověřování podpisu je na obrázku 3.16.

DNSSEC používá nové typy záznamů v DNS – záznam **DNSKEY** pro uložení veřejného klíče, záznam **RRSIG** obsahující podpis konkrétního záznamu, dále záznamy **NS** pro sekvenci uspořádání záznamů v doméně a záznam **DS** pro ověření podpisu záznamu **DNSKEY** pomocí vyšší autority.

Všechny tyto záznamy se používají pro *podepisování zón* pomocí DNSSEC. Podepsaná zóna znamená, že zónový soubor obsahuje kromě všech záznamů zóny (typu **A**, **MX**, **CNAME**, **PTR** apod.) také elektronický podpis ke každému z těchto záznamů. Elektronický podpis – přesněji řečeno podepsaný kontrolní součet (hash) záznamu – se uloží do záznamu **RRSIG**. Použije se k ověření integrity záznamu a autentizaci vlastníka. Podpis se ověřuje oproti veřejnému klíči, který je přístupný v záznamu **DNSKEY**.



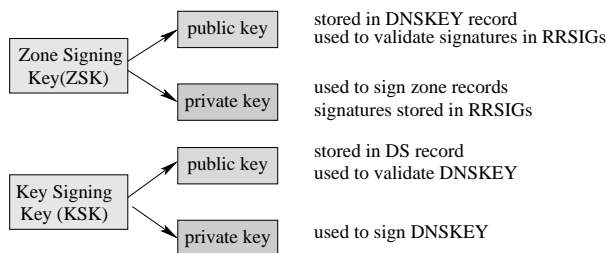
Obrázek 3.16: Generování a ověření elektronického podpisu.

Pro vytvoření podpisu a následné ověření pomocí asymetrické kryptografie potřebujeme vygenerovat klíč. Protože se jedná o asymetrickou kryptografii, používá se dvojice klíčů – soukromý a veřejný klíč.

- Soukromý klíč slouží k podepisování. Klíč se uchovává na bezpečném místě.
- Veřejný klíč slouží k ověření podpisu. Je volně dostupný. Pro kontrolu pravosti veřejného klíče se používají například certifikáty X.509. U DNSSEC se pravost veřejného klíče ověřuje podepsáním vyšší autoritou pomocí KSK (viz dále).

Oba klíče jsou algoritmicke závislé, tzn. k danému soukromému klíči přísluší konkrétní veřejný klíč. Oba se vygenerují současně. Více se o principech asymetrické kryptografie můžete dočíst v [20].

Pomocí dvojice soukromý a veřejný klíč jsme schopni podepsat záznamy v DNS a také zkontrolovat, zda je daný podpis platný. Co však chybí, je potvrzení, že můžeme daným klíčům a podpisům důvěřovat. Co se stane, když si útočník sám vytvoří podvržené záznamy, které podepíše svým klíčem a nabídne nám k ověření svůj platný veřejný klíč? Pak ověříme pravost záznamů a jsme spokojeni. Nicméně jsem získal nepravdivé (neautorizované) údaje.



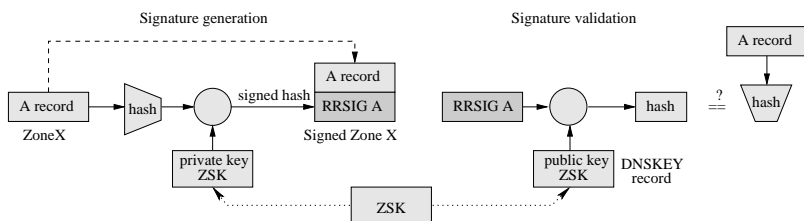
Obrázek 3.17: Klíč pro podpis zóny (ZSK) a klíč pro podepisování klíčů (KSK)

Z tohoto důvodu nám nestačí jenom klíč pro podpis zóny ZSK (Zone Signing Key). Je potřeba ještě klíč pro ověření těchto klíčů, takzvaný KSK (Key Signing Key). Pro podepisování klíčů opět použijeme asymetrickou kryptografii. Máme tedy dva páry klíčů ZSK a KSK, které se použijí pro vybudování důvěry mezi servery DNS. Jejich vztah a uložení v DNS ukazuje

obrázek 3.17. Tyto dva páry klíčů, ZSK a KSK, tvoří základ systému zabezpečení DNSSEC. Používají se k podepisování a validaci zón a k podepisování a validaci klíčů pro podpis zón. Klíče KSK vytváří (přesněji řečeno veřejný klíč KSK) vytváří tzv. *důvěryhodný vstupní bod SEP* (Security Entry Point), viz [17]. Společně vytváří propojení KSK a ZSK tzv. *řetězec důvěry* (chain of trust).

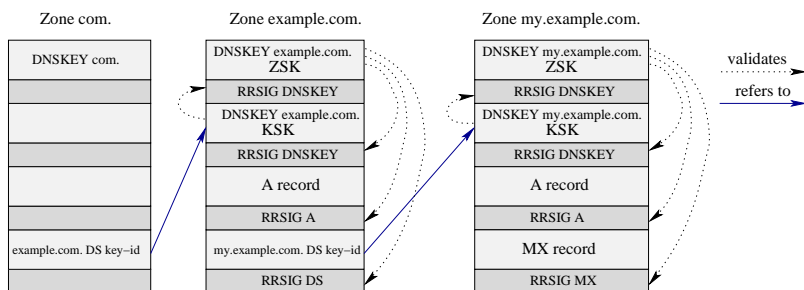
3.5.4 Řetězec důvěry

Podepsaná zóna obsahuje veřejný DNS (uložený v záznamu DNSKEY), podpisy záznamů (uložené v záznamech RRSIG), odkazy na další záznamy (záznamy NSEC), případně záznam DS ověřující klíč zóny v DNSKEY. Pokud zóna neobsahuje tyto záznamy, jedná se o nepodepsanou zónu. Příklad podepsání a ověření záznam v DNS je na obrázku 3.18.



Obrázek 3.18: Podepsání záznamu typu A a ověření podpisu

Jak jsme si již řekli, záznam typu DNSKEY obsahuje veřejný klíč pro podepisování záznamů v dané zóně. Tento klíč se používá k ověření podpisu záznamů uložených v RRSIG. Podpisy se připojují k záznamům typu A, CNAME, MX, NS a podobně v dané zóně. Podepsaný je i záznam DS. DNSSEC určuje i umístění záznamů. Záznam DS by měl být umístěn v nadřazené zóně, viz obrázek 3.19.



Obrázek 3.19: Řetězec důvěry pomocí záznamů DS a DNSKEY

Záznamy DNSKEY a DS vytváří posloupnost (řetězec) podepsaných záznamů navzájem potvrzující pravost podpisů, tzv. *řetězec důvěry* (chain of trust). Pravost záznamu DS ověřuje podpis v záznamu RRSIG, který se kontroluje pomocí veřejného klíče uloženého v záznamu

DNSKEY dané zóny. Záznam DS obsahuje kontrolní součet (hash) jiného klíče DNSKEY. Tento klíč je tedy autentizován informací v DS. Záznam DS se obvykle nachází v nadřazené zóně, která se nazývá *bodem delegace* (delegation point). Dvojici souvisejících záznamů DNSKEY a DS nazýváme *důvěryhodný pevný bod* (trust anchor).

Může se stát, že k dané zóně A neexistuje bod delegace, to jest neexistuje záznam DS v nadřazené zóně, který by obsahoval kontrolní součet záznamu DNSKEY v zóně A. Taková zóna je podepsaná, netvoří však řetězec důvěry. Nazývá se *ostrůvkem bezpečnosti* (island of security) a podepsané záznamy této zóny bychom měli používat opatrně.

Z tohoto popisu vyplývá, že je nutné, aby existoval nějaký počáteční bod řetězce důvěry. Protože je řetězec důvěry budován nad stromem DNS, je vhodné, aby jeho počátečním bodem byla kořenová zóna. V červenci 2010 došlo tedy k podepsání kořenové zóny, čím se vytvořil základní důvěryhodný bod pro vytváření řetězce důvěry (tzv. root trust anchor). Problematiku podepisování zón, delegace a rotace klíčů řeší RFC 4641 [18]. V následující části se podrobněji podíváme na jednotlivé záznamy DNS, které se používají v rozšíření DNSSEC.

3.5.5 Záznamy DNS pro DNSSEC

Rozšíření DNSSEC se vyvíjelo od roku 1999. Původní standard RFC 2535 [9] používal pro podepisování dat v DNSSEC záznamy typu KEY, SIG, NXT, které se dnes už nepoužívají. Místo nich byly roku 2005 standardizovány záznamy DNSKEY, RRSIG a NSEC. V roce 2008 došlo k aktualizaci a místo záznamů NSEC se doporučuje používat NSEC3, viz RFC 5155 [19]. Přehled nejdůležitějších záznamů DNS pro rozšíření DNSSEC je uveden v tabulce 3.7.

Záznam	Význam	Standard
DNSKEY (DNS Key Record)	veřejný klíč pro ověření podpisu	RFC 4034
RRSIG (Resource Record Signature)	podpis pro daný záznam	RFC 4034
DS (Delegation Signer)	potvrzení pravosti klíče v DNSKEY	RFC 4034
NSEC (Next-Secure Record)	odkaz na další záznam v doméně	RFC 4034
NSEC3 (NSEC version 3)	viz NSEC bez procházení zóny	RFC 5155
NSEC3PARAM (NSEC3 parameters)	parametry pro NSEC4	RFC 5155

Tabulka 3.7: Záznamy DNS použité při zabezpečení DNSSEC

Jejich význam si více popíšeme v následující části.

Záznam DNSKEY – DNS Key Record

DNSSEC používá pro podepisování a autentizaci záznamů DNS kryptografií s veřejným klíčem. Veřejný klíč se ukládá do záznamu DNSKEY. Jednotlivé záznamy v zóně se podepisují privátním klíčem ZSK, pro ověření se používá odpovídající veřejný klíč v záznamu DNSKEY. Součástí záznamu je kromě klíče také typ klíče, algoritmus pro ověřování (např. RSA/MD5, Diffie-Hellman, DSA/SHA-1 či RSA/SHA-1) a další, viz následující ukázka.

```
> dig +multi fit.vutbr.cz dnskey
fit.vutbr.cz. 7943 IN DNSKEY 257 3 5 (
  AwEAAcgo0p3tkJBYisSHYRdrkrKT0+tk2mtm3uypUIY
  feFAGyQSZtwzj3wGKkBRWpdsShxXP0xUjSruYTPeX1j0
  D4HjIx/s7YIRqWYdCw5CUB3JXLk5ygLcW+MI/+MBR+RW
  tKHwBi73/+ZrZMBR6na3V5yT1bn4VsM p0h53f8TMzU=
  ) ; key id = 59533
```

```
fit.vutbr.cz. 7943 IN DNSKEY 256 3 5 (
    AwEAAcyccLNg09xZedVqpf/ODkSqLwngQXS1e9Fe0dV6
    D4hcr1jJzHML sD5l7UCdcXdneyJnS6UZwHlV/12U4v7j
    ) ; key id = 35421
```

Výše uvedené záznamy obsahují veřejné klíče KSK a ZSK pro doménu `fit.vutbr.cz.` Jedná se o veřejný klíč KSK (hodnota 257), který se použije pro podpis ZSK a veřejný klíč ZSK (hodnota 256). Oba klíče používající k autentizaci algoritmus RSA-SHA-1 (hodnota 5). Záznamy také obsahují identifikátor klíče (key id), který se používá v podpisu (v záznamu RRSIG) jako odkaz na klíč, podle kterého je možné zkontrolovat podpis. Délka klíče v ukázce je zkrácena.

Oba klíče ZSK a KSK se umísťují do zóny, pro kterou jsou vygenerovány. Klíč ZSK (ID 35421) se použije pro podepsání všech záznamů v zóně včetně záznamu DNSKEY obsahující ZSK. Je jediná výjimka – záznam DNSKEY obsahující ZSK se podepisuje KSK. Otisk klíče KSK se pak umístí v záznamu DS nadřazené domény k vytvoření řetězce důvěry. Podpisy klíčů v zónovém souboru `fit.vutbr.cz` vypadají následovně:

```
fit.vutbr.cz. 3750 IN RRSIG DNSKEY 5 3 14400 20111201081631 (
    2011101081631 35421 fit.vutbr.cz.
    rYwrvsFfErREkbt4WddUWZMKBUiHJ35h29AU4hAywFDZ
    Uaa+Z4/iIpY/86bNobbZppT3mcd0exwMunDEnac= )
fit.vutbr.cz. 3750 IN RRSIG DNSKEY 5 3 14400 20111201081631 (
    2011101081631 59533 fit.vutbr.cz.
    BKAywfPKJsZC+BKKGgd0Ip4VG5rAkmWApY1p2FitsTh
    3SsUovKET159WUv1lr4hpD1REH610lgOG6leAea//68d
    AOfKsyer1WWbCT3ylcfGFsMV5RFdDt15rA== )
```

Z výpisu je vidět, že první podpis je proveden klíčem s ID 35421, tj. ZSK. Jedná se tedy o podpis záznamu DNSKEY obsahující KSK. Druhý podpis je proveden KSK (id 59533), což znamená, že je to podpis záznamu DNSKEY obsahující ZSK provedeného KSK (typ 257). Je také vidět, že podpis je delší než podpis předchozího záznamu. Je to proto, že klíč KSK se generuje s větší délkou a obměňuje se méně často. Protože má větší délku, naroste i délka podpisu. Více o formátu a použití záznamu RRSIG pro podpis je v další části.

Záznam RRSIG – Resource Record Signature

Záznamy RRSIG obsahují elektronický podpis jednotlivých záznamů v podepsané zóně. Přesněji je říci, že podpis RRSIG se nevztahuje k jednotlivým záznamům, ale k množině záznamů, které mají stejné doménové jméno (vlastníka), typ, třídu a TTL. Z těchto údajů a obsahu záznamů se vypočítá podpis. Formálně může podpis zapsat následujícím způsobem, kde znak „—“ označuje konkatenaci [32]:

```
signature = sign (RRSIG_RDAT | RR(1) | RR(2) ...)
RR(i) = owner | type | class | TTL | RDATA length | RDATA
```

Záznam RRSIG obsahuje také dobu platnosti podpisu, použitý algoritmus, jméno podepisující autority a příznak, který se odkazuje na veřejný klíč pro ověření podpisu.

```
> dig +dnssec +multi fit.vutbr.cz mx
fit.vutbr.cz. 13905 IN MX 20 eva.fit.vutbr.cz.
fit.vutbr.cz. 13905 IN MX 10 kazi.fit.vutbr.cz.
```



```
fit.vutbr.cz. 13905 IN RRSIG MX 5 3 14400 20111201081631 (
                20111101081631 35421 fit.vutbr.cz.
                s+6p3B1IzpYpf/7PVN3Cd8aouew/OnIucu52mdEIBv+X
                oRK271HnpR5JY+UaPkwp1po7gwKddg/FvyjghoM= )
```

Výše uvedený výpis obsahuje záznamy MX pro doménu `fit.vutbr.cz` a elektronický podpis těchto dvou záznamů. Podpis obsahuje dobu platnosti podpisu (TTL podpisu musí odpovídat TTL příslušného záznamu), dále typ algoritmu RSA-SHA-1 (hodnota 5), počet řetězců v názvu záznamu (hodnota 3, tj. `fit.vutbr.cz`), dobu platnosti záznamu MX a dobu expirace (hodnota 20111201081631 znamená 1.12.2011 v 8:16.31). Hodnota 35421 je odkaz (key tag) na veřejný klíč ZSK v záznamu DNSKEY, který se použije pro ověření podpisu. Název `fit.vutbr.cz` je doménové jméno podepisující entity, to jest vlastníka záznamu DN-SKEY. Zbytek záznamu RRSIG tvoří vlastní podpis uložený v kódování Base64.

Záznam NSEC – Next-Secure Record

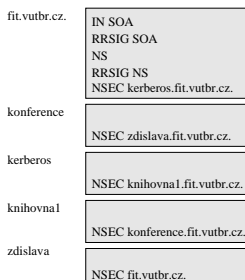
Záznam NSEC obsahuje dvě důležité informace: další doménové jméno záznamu v doméně a typ záznamů DNS, které se vztahují k tomuto doménovému jménu.

Záznamy NSEC se používají k autoritativnímu potvrzení neexistence záznamu v doméně. Tím, že záznam NSEC ukazuje na další doménové jméno, nám vlastně říká, že mezi současným doménovým jménem a dalším jménem, není žádná další položka. Bez záznamů NSEC by nešla autoritativně ověřit neexistence záznamů, protože neexistující záznam nemůže mít podpis. Pak by mohl útočník na dotaz na neexistující doménové jméno poslat podvrženou odpověď obsahující požadovaný záznam.

Použití záznamů NSEC vyžaduje seřazení jednotlivých záznamů DNS podle doménových jmen. Standard RFC 4034 [32] popisuje řazení kanonických doménových jmen. Kanonické řazení doménových jmen porovnává doménová jména postupně podle domén zprava doleva. Domény na stejné úrovni se porovnávají jako dva řetězce podle ASCII hodnot a bez ohledu na velká či malá písmena. Pokud se nejpravější doménová jména shodují, porovnávají se domény na druhé úrovni a tak dále. Kanonické řazení demonstruje následující příklad:

```
example
a.example
ylkjkljk.a.example
Z.a.example
zABC.a.EXAMPLE
z.example
\001.z.example
*.z.example
\200.z.example
```

Řazení je logické, tj. nesouvisí s aktuálním umístěním záznamů v zónovém souboru. Poslední záznam seřazení podle řazení kanonických doménových jmen se odkazuje na první záznam seřazeného seznamu doménových jmen. Příklad řazení je na obrázku 3.20. Tento obrázek ukazuje řazení záznamů DNS podle doménových jmen v následujícím pořadí: `fit.vutbr.cz`, `kerberos`, `knihovna1`, `konference` a `zdislava`. Poslední záznam ukazuje opět na první záznam, tj. doménové jméno `fit.vutbr.cz`. Na tomto příkladu můžete také vidět, že postupným doptáváním se na záznamy NSEC jsme schopni vyčíst všechny záznamy dané domény, tj. zjistit, jaká doménová jména se používají pro zařízení v dané doméně. Z pohledu bezpečnosti se jedná o závažnou otázku bezpečnosti sítě. Proto byl roku 2008 navržen alternativní způsob



Obrázek 3.20: Příklad řazení záznamů v podepsaném zónovém souboru `fit.vutbr.cz.`

pro autentizovaný způsob potvrzení neexistence doménového jména zvaný NSEC verze 3 [19]. O tom, jak funguje, si povíme později.

Záznam NSEC se umísťuje zpravidla nakonec záznamů k doménovému jménu. Za ním následuje ještě podpis v záznamu RRSIG. Následující příklad ukazuje použití záznamu NSEC.

```

fit.vutbr.cz. 86398 IN NSEC kerberos.fit.vutbr.cz. NS SOA MX TXT RRSIG NSEC DNSKEY
fit.vutbr.cz. 86398 IN RRSIG NSEC 5 3 86400 20111102094352 (
    20111102094352 35421 fit.vutbr.cz.
    nmUzpEnG4vIZN0uVRo5ja/cSEZG0Jec8fMPLAAiU2c2X
    1PmpGYjLB6AFNCTEYM9+ZnEAACWY7xsYEFWxYoc= )

```

Jedná se o záznam NSEC u doménového jména `fit.vutbr.cz.`, který ukazuje na další záznam (NEXT) v zóně následujícím za jménem `fit.vutbr.cz.` podle kanonického řazení. Zároveň obsahuje seznam záznamů všech záznamů u jména `fit.vutbr.cz.`, což jsou záznamy NS, SOA, MX, TXT, RRSIG, NSEC a DNSKEY. K záznamu NSEC je opět připojený klíč s podpisem záznamu.

Záznam DS – Delegation Signer

Záznam DS se odkazuje na záznam DNSKEY a používá se k autentizaci klíč v tomto záznamu. Záznam DS obsahuje odkaz na klíč DNSKEY, typ algoritmu (např. SHA-1) a otisk klíče (digest) v DNSKEY. Otisk klíče stačí pro identifikaci a autentizaci veřejného klíče uloženého v záznamu DNSKEY. Pomocí záznamu DS může resolver ověřit klíč v záznamu DNSKEY, na který se záznam DS odkazuje.

Záznamy DS a DNSKEY mají stejného vlastníka (tj. obsahují stejné doménové jméno), ale jsou uloženy na různých místech. Záznam DS se uchovává v rodičovské zóně. Například záznam DS pro doménu `fit.vutbr.cz.` je uložen v zónovém souboru `vutbr.cz.` Naopak, odpovídající klíč DNSKEY je uložen v zóně potomka, tj. `fit.vutbr.cz.`, viz obrázek 3.19. Příklad záznamů DNSKEY a DS je v následující ukázce.

```

>dig +dnssec +multi fit.vutbr.cz DNSKEY
fit.vutbr.cz. 12346 IN DNSKEY 257 3 5 (
    AwEAAcgo0p3tkJBYisSHYRdrkrTKT0+tk2mtm3uypUIY
    feFAgyQSZtwzj3wGKkBRWpdsShxXP0xUjSruYTPeX1j0
    WBi73/+ZrZMBR6na3V5yTlbn4VsMp0h53f8TMzU=
    ) ; key id = 59533

```

```

>dig +dnssec +multi fit.vutbr.cz DS
fit.vutbr.cz. 84249 IN DS 59533 5 1 (
    7814F57DE4BB7FD48CF3DD952549D6CFF162B21F )
fit.vutbr.cz. 83951 IN RRSIG DS 5 3 86400 20111120120731 (
    20111021120731 39756 vutbr.cz.
    Z018YyedzSvAwuUiw1FI275zupL+BwC5RgIvCDUZ21+C
    NqxDAzbd0NrMZverbGP6PCsMR691U7mCqkj+YMVtq1JM
    Dn1a3XQpmWqrZ4KIL1J+LX291aA6BD/f8H3x+IY= )

```

Zde vidíme KSK pro doménu `fit.vutbr.cz`, který je uložený v záznamu DNSKEY v zónovém souboru `fit.vutbr.cz`. Tento klíč s ID 59533 se použije pro podepsání zónového klíče ZSK. Odkaz na tento klíč je umístěn v záznamu DS, který je umístěn v rodičovské zóně `vutbr.cz`. Zároveň je podepsán klíčem ZSK rodičovské zóny `vutbr.cz` s ID 39756.

Záznam NSEC3 – Next-Secure Record version 3

Nedílnou součástí zabezpečení DNSSEC jsou záznamy NSEC, které slouží pro potvrzení neexistence záznamu. Jak jsme si ukázali výše, nevýhodou použití záznamů NSEC je to, že umožňují načtení celé zóny pomocí průchodu seznamu NSEC. Postupným vyčítáním záznamů NSEC (zone enumeration) jsme schopni získat všechna doménová jména dané domény včetně typů záznamů, které jsou pro dané jméno registrované v DNS. Přestože spousta serverů DNS filtruje přenos zón pouze na důvěryhodné servery DNS, pomocí procházením řazeného seznamu kanonických jmen v doméně jsme schopni tyto informace získat bez jakéhokoliv omezení.

Toto je hlavní motivace k návrhu nového typu záznamů NSEC3 [19]. Tyto záznam zajišťují stejnou funkci jako NSEC, tj. potvrzení neexistence záznamu. Proto potřebují podobně jako NSEC vytvořit řazenou posloupnost všech registrovaných doménových jmen, aby mohly detekovat neexistující doménové jméno.

NSEC3 však nepoužívá pro řazení kanonická jména, ale číselnou hodnotu hašovací funkce, kterou aplikuje na původní jméno. Výsledné číselné hodnoty numericky řadí do seznamu (tzv. *hash order*). Toto pořadí je stejné, jako u řazení kanonických jmen dle RFC 4034 [32]. Příklady výstupů hašovací funkce obsahuje následující ukázka:

```

H(example)      = 0p9mhaveqvm6t7vb15lop2u3t2rpb3tom
H(a.example)    = 35mthgpgcu1qg68fab165klnsnk3dpv1
H(ai.example)   = gjeqe526p1bf1g8mk1p59enfd789njgi
H(ns1.example)  = 2t7b4g4vsa5smi47k61mv5bv1a22bojr
H(ns2.example)  = q04jkcevqvmu85r014c7dkba38o0ji5r
H(w.example)    = k8udempv1j2f7eg6jebps17vp3n8i58h
H(*.w.example)  = r53bq7cc2uvmubfu5ocmm6pers9tk9en
H(x.w.example)  = b4um86eghhds6nea196smvml04ors995
H(y.w.example)  = j16neoaepv8b5o6k4ev33abha8ht9fgc
H(x.y.w.example) = 2vptu5timamqttgl14luu9kg21e0aor3s
H(xx.example)   = t644ebqk9b1bcna874givr6joj62mlhv
H(2t7b4g4vsa5smi47k61mv5bv1a22bojr.example)
    = kohar7mbb8dc2ce8a9qv18hon4k53uhi

```

Záznam NSEC3 obsahuje informaci o tom, jaká hašovací funkce je použita (hash algorithm), jaká je inicializační sekvence (tzv. *salt*), kolik iterací hašovací funkce se provede nad původním doménovým jménem (iterations), ukazatel na další hašované jméno a bitová mapa, která obsahuje seznam typů DNS, které jsou vytvořeny pro dané doménové jméno. Příklad podepsané zóny s použitím záznamů NSEC3 je v následující ukázce:

```

example. 3600 IN SOA ns1.example. bugs.x.w.example. 1 3600 300 (
    3600000 3600 )
RRSIG SOA 7 1 3600 20150420235959 20051021000000 (
    40430 example.
    Hu25UIyNPmvPIVBrdN+9Mlp9Zql39qaUd8i
    VI2LmKusbZsT0Q== )
NS ns1.example.
NS ns2.example.
RRSIG NS 7 1 3600 20150420235959 20051021000000 (
    40430 example.
    PV0gtMK1HHeSTau+HwDWC8Ts+6C8qtqd4pQJ
    CnMXjtz6SyObxA== )
DNSKEY 257 3 7 AwEAAcUlFV1vhmqx6NSOU0q2R/dsR7Xm3upJ (
    AbsUdblMFin8CVF3n4s= )
RRSIG DNSKEY 7 1 3600 20150420235959 (
    20051021000000 12708 example.
    AuU4juU9RaxescSmStrQks3Gh9Fb1GB1VU31
    MGQZf3bH+QsCtg== )
NSEC3PARAM 1 0 12 aabbccdd
RRSIG NSEC3PARAM 7 1 3600 20150420235959 (
    20051021000000 40430 example.
    rN05XSA3PqOU3+4VvGWYwUMffl0dxqnXHWJ
    TLQsjlkyhG6Gc== )
Op9mhaveqvm6t7vbl5lop2u3t2rp3tom.example. NSEC3 1 1 12 aabbccdd (
    2t7b4g4vsa5smi47k61mv5bv1a22bojr MX DNSKEY NS
    SOA NSEC3PARAM RRSIG )
RRSIG NSEC3 7 2 3600 20150420235959 20051021000000 (
    40430 example.
    IBHYH6bLRxK9rC0bMJPwQ4mLIuw85H2EY762
    BOCXJZMnpwhpA== )
2t7b4g4vsa5smi47k61mv5bv1a22bojr.example. A 192.0.2.127
RRSIG A 7 2 3600 20150420235959 20051021000000 (
    40430 example.
    K+iDP4eY8IOkSiKaCjg3tC1SQkeloMeub2GW
    k8p6xHMPZumXlw== )
NSEC3 1 1 12 aabbccdd (
    2vptu5timamqttgl4luu9kg21e0aor3s A RRSIG )
RRSIG NSEC3 7 2 3600 20150420235959 20051021000000 (
    40430 example.
    4TFoNxZuP03gAXEI634Yw0c4YBNITrj413iq
    NI6mRk/r1d0SUw== )

```

Záznam NSEC3PARAM obsahuje parametry NSEC3 (hašovací algoritmus, příznaky, ite-race, inicializační sekvenci), kterou jsou potřebné pro autoritativní servery, aby mohli vypo-čítat hašovací funkci pro doménové jméno.

Nevýhodou návrhu NSEC3 je, že není zpětně kompatibilní s dřívějšími standardy DNSSEC. Pokud například resolver nerozumí NSEC3, nebude schopen validovat odpovědi. Kompatibi-lita například vyžaduje, aby se pro generování podpisů DNSKEY používali algoritmy DSA-NSEC3-SHA1 nebo RSASHA1-NSEC3-SHA1. V současné době není mnoho zabezpečených domén, které by používaly záznamy NSEC3.


```

                                3600 ) ; Minimum
IN      NS      palecek.cz.
IN      MX      0 mail.palecek.cz.
pc1     IN      A      10.10.10.102
palecek.cz. IN DNSKEY 257 3 5 AwEAAfr5lFc/Ke2IcS6L/eZ+psaHUaisGj69lxxkLfxTPVJ70G
NFHw9m1NjhXsyXlE9A/HTcSba7oEAUHN1i2iYQlLFiQVcEWu0v0AE8zLXhsdaRmbtVoZ
palecek.cz. IN DNSKEY 256 3 5 AwEAAbqZoS2n2S/ma3WG9qhgegL6toPjwhcxCOsaVdGXqgWjP
41 EiZSlU6MTwjE/HNs3CS7UGKL/j0ypLzP7wen7MAbMQHQW8L4xdw7SGQN XorEPuN/zWvNU8+v9VII

```

4. Podepsání zóny soukromým klíčem.

```

>> dnssec-signzone -o palecek.cz db.palecek.cz
db.palecek.cz.signed: // zónový soubor pro doménu palecek.cz
palecek.cz.      3600   IN SOA  palecek.cz. root.palecek.cz. (
                                20050322 ; serial
                                3600     ; refresh (1 hour)
                                900      ; retry (15 minutes)
                                3600000  ; expire (5 weeks 6 days 16 hours)
                                3600     ; minimum (1 hour)
                                )
3600   RRSIG  SOA 5 2 3600 20111208135222 (
20111108135222 35205 palecek.cz.
So0a/tXxW3zB5FRDE+LG21xKpbfu/pNcdRhC
vYB2XjIKWKAha7Wy4qV0h0C0RwQ= )
3600   NS      palecek.cz.
3600   RRSIG  NS 5 2 3600 20111208135222 (
20111108135222 35205 palecek.cz.
hN5HeB7zUgu10qexZ/ATF1l1HEOKiXcRrmBS
Jy01T0pCSVJoUbZBXDUJ8vy9lFM= )
3600   MX      0 mail.palecek.cz.
3600   RRSIG  MX 5 2 3600 20111208135222 (
20111108135222 35205 palecek.cz.
yG8UAe51tKSGc+X8Y71zhtHESIov0rCOUM+R
ns811M2yMewuN3H0V8jzwRg3mq4= )
3600   NSEC    firma.palecek.cz. NS SOA MX RRSIG NSEC DNSKEY
3600   RRSIG  NSEC 5 2 3600 20111208135222 (
20111108135222 35205 palecek.cz.
o14yMIow+tcALntZml050A3wv9fHkZWuyS3I
gNH2Ws4gWP97jzixkmR41rtmxhQ= )
3600   DNSKEY  256 3 5 (
AwEAAbqZoS2n2S/ma3WG9qhgegL6toPjwhcx
COsaVdGXqgWjPCd+en41
) ; key id = 35205
3600   IN DNSKEY 257 3 5 (
AwEAAfr5lFc/Ke2IcS6L/eZ+psaHUaisGj69
lxxkLfxTPVJ70G/okNFHw9m1NjhXsyXlE9A/
HTcSba7oEAUHN1i2iYQlLFiQVcEWu0v0AE8z
) ; key id = 12094
3600   RRSIG  DNSKEY 5 2 3600 20111208135222 (
20111108135222 12094 palecek.cz.
sdd0lmjckou62MQRbhY7MhjpJsLU6irzqp+q
t0v71KwflG28FfRUe2L0KGMh1E6u/SZF4C4= )
3600   RRSIG  DNSKEY 5 2 3600 20111208135222 (
20111108135222 35205 palecek.cz.
GfFFFaG5o51wR3LXR8CLubS+3pFQFGDi6DiC5

```

```

pc1.palecek.cz. 3600 IN A Ej9su0lzkdqrX+aDwuA65XySiejfEUPMgq/n
                  3600 RRSIG 10.10.10.102
                        A 5 3 3600 20111208135222 (
                        20111108135222 35205 palecek.cz.
                        HiZ+9q/yxikbKzuUPFBKi0zHwLcts34yD0/f
                        UAMxTKFnpmuovWwDq5gA1102yY= )
                  3600 NSEC pc2.palecek.cz. A RRSIG NSEC
...

```

Ve výše uvedeném výpisu můžeme vidět, že pro každý záznam existuje podpis (záznam RRSIG). Dále jsou v zónovém souboru záznam s klíčem (DNSKEY) a jeho podpis. Záznam NSEC definují řazení, tj. určují, jaký další záznam následuje po předchozím. Tyto záznam slouží ke kontrole negativních odpovědí.

- Umístění záznamu DS pro ověření podpisu zónového klíče do nadřazené domény.

```

dsset-palecek.cz:
palecek.cz.      IN DS 12094 5 1 51121AA0B28D46698518A2A843C55614940BD812

```

- Nastavení zónového souboru v konfiguraci DNS serveru.

```

zone 'palecek.cz' {
    type master;
    file 'db.palecek.cz.signed';
};

```

Při každé změně zónového souboru musí změněný souboru znovu podepsán. Budování řetězů důvěry a hierarchické podepisování klíčů pomocí záznamů DS je popsáno ve standardu RFC 4034 [32]. Další důležitou věcí, kterou musí administrátor zajistit, je rotace klíčů po expiraci jejich platnosti a znovupodepisování záznamů. Na to už dnes existují automatizované prostředky, které tuto výměnu klíčů zajišťují.

Mechanismus DNSSEC tvoří komplexní hierarchický systém pro zabezpečení informací v DNS proti podvržení a upravení. Pomocí principů veřejné kryptografie je možné vybudovat důvěryhodný řetěz DNS serverů. Podepisování DNSSEC záznamů se začíná ve světě více prosazovat a postupně se podepisují jednotlivé zóny a budují se řetězce důvěry pro ověření podepsaných zón. Díky podepsání kořenového serveru DNS lze jednotlivé podepsané domény propojit do jednoho stromu a ověřit pomocí důvěryhodné autority.

Shrnutí

Překlad doménových adres DNS je důležitou a nezbytnou součástí pro fungování komunikace v Internetu. Systém DNS obsahuje databázi doménových jmen a dalších informací, které jsou uloženy v záznamech DNS. Službu DNS tvoří soustava serverů DNS, které tuto databázi spravují, a dále resolversy, které k databázi přistupují.

Doménových prostor DNS je hierarchicky uspořádán a jeho správa je rozdělena na nižší subjekty. Koordinaci DNS a správu nejvyšších domén řídí organizace ICANN, která spravuje národní domény a generické domény první úrovně. Domény nižší úrovně (subdomény) spravují vlastníci domén. Prostor doménových adres vytváří kořenový strom, kde jednotlivé uzly jsou domény nižší úrovně a cesta ve stromu tvoří doménovou adresu. Pokud cesta zahrnuje i

kořen, jedná se o absolutní doménové jméno (fully qualified domain name). Fyzicky jsou části doménového prostoru rozděleny do zón a uloženy na doménových serverech.

Základním úkolem systému DNS je překlad (mapování) doménových adres na IP adresy. Pro zpětný (reverzní) překlad existuje speciální doména `.in-addr.arpa.`, která obsahuje prostor číselných IP adres rozdělených hierarchicky do stromu podle jednotlivých bytů adresy.

Konkrétní informace v systému DNS jsou uloženy v datových strukturách, které se nazývají záznamy DNS (resource records). Existuje velké množství záznamů, které uchovávají různé typy dat. Mezi nejpoužívanější záznamy patří záznamy typu A, které mapují doménovou adresu na IP adresu, záznamy typu PTR pro reverzní překlad, záznamy NS pro mapování domény na autoritativní doménový server. Záznamy typu MX obsahují adresu poštovního serveru pro danou doménu, záznamy SOA uchovávají informace o platnosti záznamů a správě domény. Pro potřeby IP telefonie se využívá například záznamů NAPTR a SRV. Zabezpečení DNS využívá záznamy DNSKEY, RRSIG, NSEC či DS. IP adresy protokolu IPv6 jsou uloženy v záznamech AAAA, reverzní mapování v PTR.

Komunikaci v DNS zajišťuje aplikační protokol DNS, který zpravidla běží nad transportním protokolem UDP. Pro delší přenosy (nad 512 bytů), což jsou zejména zónové přenosy, se využívá transportní protokol TCP, port 53. PDU protokolu DNS obsahuje hlavičku s informacemi o dotazu/odpovědi, sekci dotazu, odpovědi a dodatečné (upřesňující) informace.

Většina komunikace probíhá mezi resolversy a servery DNS. Jedná se o jednoduché dotazy na vyhledávání. Dalším typem komunikace je synchronizaci mezi primárním a sekundárním serverem DNS, kterému se říká zónový přenos. K zónovému přenosu dochází buď po uplynutí platnosti záznamů v DNS na sekundárních či záložních serverech, nebo zasláním informace DNS NOTIFY. Sekundární server žádá buď aktualizaci celé zóny nebo pouze dat, která se změnila od poslední aktualizace.

Resolver je většinou součástí operačního systému. Obsahuje funkce pro vytváření, posílání a dekodování požadavků na systém DNS. Funkce resolveru lze využít k napsání vlastních programů pro vyhledávání informací v DNS. Pro vyhledávání informací v DNS lze využít programy `nslookup`, `host` a `dig`.

Informace v DNS jsou nezbytné pro správnou činnost komunikace po Internetu. V případě změny, poškození či nedostupnosti může dojít k omezení a výpadkům spojení. Existují případy vnucení nesprávných údajů do paměti `cache` sekundárních a záložních serverů. Z tohoto důvodu se snaží DNS zajistit integritu a autentizaci dat. Privátnost (šifrování) není potřeba zajišťovat, neboť systém DNS je ze své podstaty navržený jako veřejný systém.

Existují dva způsoby zajištění bezpečnosti přenosu dat v DNS – systém TSIG a DNSSEC. TSIG využívá podepisování transakcí (nikoliv záznamů) mezi komunikujícími stranami pomocí tajného klíče. Toto omezuje jeho činnost pouze na servery pod jednou administrativní správou. Mechanismus DNSSEC naopak vytváří strukturu vzájemně důvěryhodných serverů pomocí veřejných klíčů ověřovaných vyšší autoritou. Jednotlivé záznamy v DNS jsou podepsány soukromým klíčem. Protože jsou podpisy i veřejné klíče pro ověření uloženy ve speciálních záznamech, dochází k enormnímu nárůstu objemu dat. Také zpracování podpisů a ověřování je výpočetně náročné. Přesto dochází k postupnému nasazování rozšíření DNSSEC.

Otázky

1. Popište činnost a architekturu systému DNS.
2. Co je to autoritativní server? Jakým způsobem zjistím, které servery jsou pro danou doménu autoritativní a který doménový server je primární?
3. Co znamená neautoritativní odpověď? Jak získat autoritativní?
4. Uveďte význam a příklad záznamů SOA, A, PTR, NS, MX, CNAME.
5. K čemu lze použití záznamy NAPTR, SRV či TXT?
6. Jak lze nastavit distribuci zátěže pomocí DNS?
7. Co je to zóna a jak souvisí s doménou?
8. Co je to přenos zón? K čemu slouží příkaz DNS notify?
9. Co se stane, když
 - chybí pro daný počítač záznam typu PTR?
 - pro jeden počítač existují na více serverech primární DNS záznamy?
 - administrátor upraví záznam na sekundárním DNS serveru?
10. Jaké sekce tvoří paket DNS a co tyto části obsahují?
11. Jak lze aplikací dig/nslookup získat záznamy A, PTR, NS, MX?
12. Co je to iterativní DNS server a jak pracuje?
13. Jaká jsou bezpečnostní rizika systému DNS?
14. Na jakém principu pracuje rozšíření DNSSEC?
15. Uveďte příklad vyhledání následujících informací:
 - zjištění MX záznamů dané domény
 - kanonického jméno počítače
16. Vytvořte DNS záznamy pro počítače firmy ttt.cz se sítí 81.19.10.0/24
 - WWW server (www.ttt.cz), DNS server (ns.ttt.cz),
 - poštovní server (mail.ttt.cz), pracovní stanice p1-p10
 - záložní DNS a poštovní server – zaloha.isp.cz (IP adresa 81.0.233.80)
17. Vytvořte příklad přímého a reverzního zónové souboru pro firmu s 10 počítači, kde je jeden poštovní server a jeden WWW server s kanonickým jménem.
18. Čím se liší zabezpečení TSIG a DNSSEC? Proti jakým útokům zabezpečují data v DNS?
19. Popište činnost při přenosu zón.
20. Které záznamy používá DNSSEC? Vysvětlete jejich význam.

Doporučená literatura a standardy

- [1] *ISO 3166 Maintenance Agency – ISO's focal point for country codes*. International standard, ISO, 1974.
- [2] A.Guldbrandsen, P.Vixie, and L.Esibov. *A DNS RR for specifying the location of services (DNS SRV)*. RFC 2782, February 2000.
- [3] A.Gustafsson. *Handling of Unknown DNS Resource Record (RR) Types*. RFC 3597, September 2003.
- [4] S. Bradner, L. Conroy, and K. Fujiwara. *The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)*. RFC 6116, March 2011.
- [5] B.Wellington. *Secure Domain Name System (DNS) Dynamic Update*. RFC 3007, November 2000.
- [6] C.Davis, P.Vixie, T.Goodwin, and I.Dickinson. *A Means for Expressing Location Information in the Domain Name System*. RFC 1876, January 1996.
- [7] C.Everhart, L.Mamakos, R.Ullmann, and P.Mockapetris. *New DNS RR Definitions*. RFC 1183, October 1990.
- [8] D.Atkins and R.Austein. *Threat Analysis of the Domain Name System (DNS)*. RFC 3833, August 2004.
- [9] D.Eastlake. *Domain Name System Security Extensions*. RFC 2535, March 1999.
- [10] D.Eastlake and A.Panitz. *Reserved Top Level DNS Names*. RFC 2606, June 1999.
- [11] E.Gunduz, A.Newton, and S.Kerr. *IRIS: An Address Registry (areg) Type for the Internet Registry Information Service*. RFC 4698, October 2006.
- [12] G.Huston. *Management Guidelines & Operational Requirements for the Address and Routing Parameter Area Domain ("arpa")*. RFC 3172, September 2001.
- [13] H.Krawczyk, M.Bellare, and R.Canneti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104, February 1997.
- [14] B. Hoeneisen, A. Mayrhofer, and J. Livingood. *IANA Registration of Enumservices: Guide, Template, and IANA Considerations*. RFC 6117, March 2011.
- [15] J.Postel. *Domain Name System Structure and Delegation*. RFC 1591, March 1994.
- [16] J.Rosenberg, H.Schulzrinne, G.Camarillo, A.Johnston, J.Peterson, R.Sparks, M.Handley, and E.Schooler. *SIP: Session Initiation Protocol*. RFC 3261, June 2002.
- [17] O. Kolkman, J. Schlyter, and E. Lewis. *Domain Name System KEY (DNSKEY) Resource Record (RR) Secure Entry Point (SEP) Flag*. RFC 3757, April 2004.
- [18] O. Kolman and R. Gieben. *DNSSEC Operational Practices*. RFC 4641, September 2006.

- [19] B. Laurie, G. Sisson, R. Arends, and D. Blacka. *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence*. RFC 5155, March 2008.
- [20] Wanbo Mao. *Modern Cryptography. Theory and Practice*. Prentice Hall, 2004.
- [21] M.Lottor. *DOMAIN ADMINISTRATORS OPERATIONS GUIDE*. RFC 1033, November 1987.
- [22] M.Mealling. *Dynamic Delegation Discovery System (DDDS) Part Five: URIARPA Assignment Procedures*. RFC 3405, October 2002.
- [23] M.Mealling. *Dynamic Delegation Discovery System (DDDS), Part One: The Comprehensive DDDS*. RFC 3401, October 2002.
- [24] M.Mealling. *Dynamic Delegation Discovery System (DDDS), Part Three: The Domain Name System (DNS) Database*. RFC 3403, October 2002.
- [25] M.Mealling and R.Daniel. *The Naming Authority Pointer (NAPTR) DNS Resource Record*. RFC 2915, September 2000.
- [26] P.Beertema. *Common DNS Data File Configuration Errors*. RFC 1537, October 1993.
- [27] P.Mockapetris. *Domain Names — Concepts and Facilities*. RFC 1034, November 1987.
- [28] P.Mockapetris. *Domain Names — Implementation and Specification*. RFC 1035, November 1987.
- [29] P.Vixie. *A Mechanism for Prompt Notification of Zone Changes (DNS NOTIFY)*. RFC 1996, August 1996.
- [30] P.Vixie, O.Gudmundsson, D.Eastlake, and B.Wellington. *Secret Key Transaction Authentication for DNS (TSIG)*. RFC 2845, May 2000.
- [31] P.Vixie, S.Thomson, Y.Rekhter, and J.Bound. *Dynamic Updates in the Domain Name System (DNS UPDATE)*. RFC 2136, April 1997.
- [32] R.Arends, R.Austein, M.Larson, D.Massay, and S.Rose. *Resource Records for the DNS Security Extensions*. RFC 4034, May 2005.
- [33] R.Bush, D.Karrenberg, M.Kosters, and R.Plzak. *Root Name Server Operational Requirements*. RFC 2870, June 2000.
- [34] S.Kwan, P.Garg, J.Gilroy, L.Esibov, J.Westhead, and R.Hall. *Generic Security Service Algorithm for Secret Key Transaction Authentication for DNS (GSS-TSIG)*. RFC 3645, October 2003.
- [35] S.Thomson, C.Huitema, V.Ksinant, and M.Soussi. *DNS Extensions to Support IP Version 6*. RFC 3596, October 2003.

Kapitola 4

Poštovní služby

Elektronická pošta, konference, diskuzní fóra – to jsou služby, které shrnujeme pod názvem poštovní služby. Poštovní služby, jak už název napovídá, slouží k přenosu zpráv mezi dvěma či více uživateli. Nejrozšířenější a nejznámější službou je bezesporu elektronická pošta, která se používá od vzniku Internetu v 80. letech 20. století. K poštovním službám se řadí i elektronické konference (mailing lists) pro zasilání zpráv většímu počtu účastníků, či diskuzní skupiny (NetNews, Usenet) pro výměnu informací a diskuzi velkého množství uživatelů v rámci určité zájmové skupiny. Přestože architektura systému diskuzních skupin Usenet je jiná než u elektronické pošty, cíl služby – výměna zpráv – je podobná, a proto tuto službu řadíme také k poštovním službám. Pokud bychom hledali analogii elektronických poštovních služeb s klasickými poštovními službami, tak elektronické poště odpovídá přenos a doručování dopisů, elektronickou konferenci můžeme přirovnat k rozšiřování oběžníků ve firmě a diskuzní fóra odpovídají odborným časopisům a novinám, kde probíhá diskuze na vybranými články. Výhodou elektronických poštovních služeb je jednak bezplatné použití, snadná dostupnost a především rychlost doručení. Nesmíme také zapomenout na spolehlivost, možnost zálohování a vyhledávání v archívech doručených zpráv. Mezi nevýhody elektronických služeb patří jejich neosobní charakter, možnost snadného zneužití a určitá inflace zpráv vyplývající ze snadného a bezplatného použití. Také tam patří nedostatečná autentizace uživatele a jednodušší možnosti podvrhu zpráv.

V této kapitole se podíváme na architekturu jednotlivých elektronických poštovních služeb. Představíme systém elektronické pošty, způsob vytváření zpráv, popíšeme si přenosový protokol SMTP (Simple Mail Transfer Protocol), protokoly pro čtení pošty POP (Post Office Protocol) a IMAP (Internet Mail Access Protocol). Budeme se zabývat zabezpečením poštovních služeb, zejména autentizací a šifrování zpráv pomocí technik PGP a S/MIME. U elektronických konferencí (mailing list) ukážeme dva přístupy jejich realizace pomocí jednoduchých seznamů typu "aliases" nebo pomocí specializovaných aplikací typu listserver. Ukážeme si možnosti nastavení a použití těchto konferencí. V poslední části této kapitoly se podíváme na architekturu systému diskuzních skupin Usenet, přenosový protokol NNTP (Network News Transfer Protocol) a způsobu vytváření, čtení a ukládání diskuzních příspěvků na diskuzním serveru.

4.1 Elektronická pošta

Elektronická pošta (electronic mail, e-mail) je jedna z nejpoužívanějších služeb Internetu. Výměna dat mezi uživateli elektronickou formou byla také jednou z motivací vzniku Internetu. Návrh formátu a přenosového protokolu emailových zpráv pochází z počátku 80. let, viz standardy RFC 821 a RFC 822 [1, 2]. V 90. letech se použití rozšířilo i mimo akademickou sféru. Dnes si bez emailů téměř neumíme představit život. Narozdíl od ostatních komunikačních prostředků jakou jsou pošta, telefon, či SMS je použití elektronické pošty zcela zdarma. Styl emailové komunikace se liší od jiných komunikačních prostředků. Svým způsobem jde o demokratický komunikační prostředek, kde není neobvyklé posílat emaily i respektovaným a vysoce postaveným osobám.

4.1.1 Architektura elektronické pošty

Původní emailový systém přenášel jednoduché textové zprávy, kde na prvním řádku byla adresa příjemce. Přenos probíhal pomocí kopírování souborů službou FTP. Později byl návrh rozšířen. V roce 1982 byl definován protokol SMTP (Simple Mail Transfer Protocol) pro přenos emailových zpráv [1, 3] a standard pro tvorbu emailů [1, 3]. V roce 1984 organizace CCITT vytvořila standard X.400 pro elektronickou poštu. Tento komplexní a robustní systém se neujal a dnes se již téměř nepoužívá.

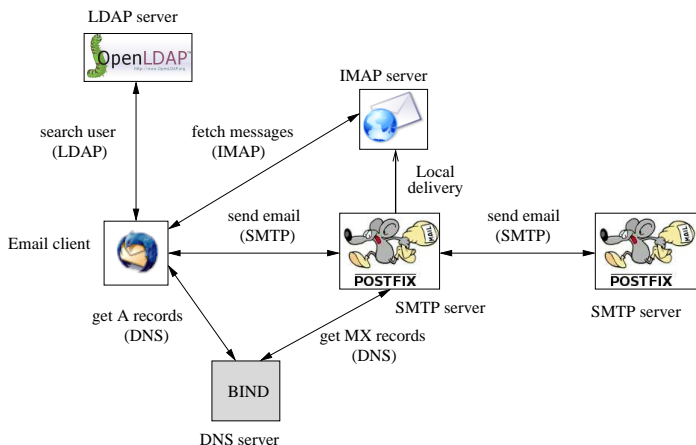
Základní architekturu systému elektronické pošty tvoří dvě entity – *uživatelský agent UA* (User Agent) a *agent pro přenos zpráv MTA* (Message Transfer Agent). Uživatelský agent je klientská aplikace, která slouží k vytváření, odesílání a čtení elektronické pošty. Agent pro přenos zpráv předává zprávu mezi jednotlivými uzly až na místo doručení. Agentem pro přenos zpráv může být poštovní server (mail server), např. sendmail, postfix, MS Exchange.

Systém elektronické pošty nepracuje pouze s přenosovým protokolem SMTP. Tvoří ho přístupové protokoly pro čtení doručených zpráv ze schránek (IMAP a POP3). Pro správné doručení pošty jsou také nezbytné služby DNS, konkrétně záznamy typu MX. Tyto záznamy k dané doméně (např. fit.vutbr.cz) přiřazují poštovní servery, které pro tuto doménu přijímají elektronickou poštu. Pokud tyto záznamy chybí, odmítne SMTP server poštu doručit a vrátí ji adresátovi. Podrobnosti o záznamech MX lze najít v části 3.3.

Systém elektronické pošty rozšířený o další potřebné služby je zobrazena na obrázku 4.1. Na obrázku je zobrazen emailový klient, který slouží k vytváření a odesílání elektronické pošty. Při vytváření pošty lze pro vyhledání adresy příjemce použít buď lokální adresář, který bývá součástí emailového klienta, nebo adresářovou službu LDAP. Ta umí podle vlastního jména osoby vyhledat emailovou adresu příslušného uživatele. Více o adresářových službách se dočtete v kapitole 5.

Vytvořený email předá poštovní klient protokolem SMTP nejbližšímu SMTP serveru (tzv. server pro odchozí poštu, outgoing SMTP server). SMTP server zprávu přijme a zkontroluje podle emailové adresy, zda je určena pro lokálního uživatele. Pokud ano, uloží ji do schránky příslušného příjemce. Pokud je email určen pro jinou než lokální doménu, zeptá se SMTP server DNS serveru, kdo má být cílovým poštovním serverem (dotaz na záznam typu MX). Poté se přes protokol SMTP připojí s cílovým SMTP serverem a předá mu danou zprávu.

Doručené zprávy se ukládají do schránek s doručenou poštou (tzv. Inbox), do které má přístup pouze uživatel, na jehož adresu byl email poslán. Příchozí poštu spravují servery pro příchozí poštu. Jejich obsah lze získat buď prohlížením souboru z lokálního systému, což není příliš efektivní a praktické, nebo pomocí poštovního klienta přes přístupové protokoly IMAP



Obrázek 4.1: Systém elektronické pošty a další služby

a POP3.

Na obrázku 4.2 vidíme základní architekturu systému elektronické pošty definované standardem RFC 821 [1, 3], která je tvořena uživatelskými agenty a agenty pro přenos zpráv. Základním přenosovým protokolem je SMTP, případně jeho rozšířená verze ESMTP (Extended SMTP). Uživatelský agent (poštovní klient) vytváří uživatelské rozhraní pro vytváření a čtení emailů. Zároveň se používá pro manipulaci se schránkami, do kterých zprávy ukládáme. Obvykle podporuje protokoly SMTP, POP3 a IMAP.

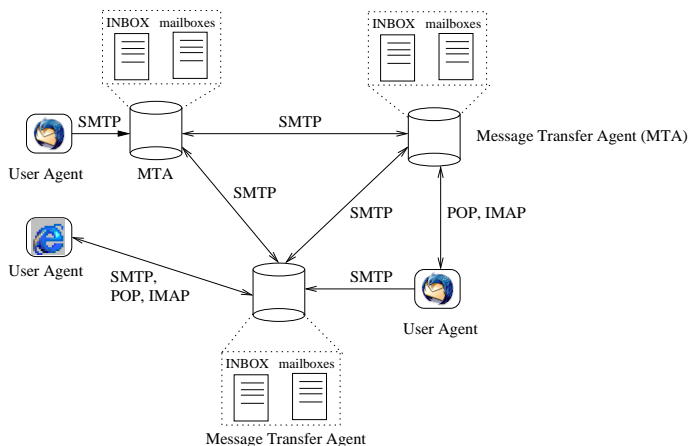
Pro přenos zpráv slouží MTA. Před vlastním přenosem vytváří odesílající MTA tzv. obálku zprávy (envelope), která nese informace o vysílající aplikaci a počítači. Příjímající MTA zprávu přijme, odstraní obálku a uloží dopis do schránky příchozí pošty daného uživatele. MTA jsou servery (např. sendmail), které běží na pozadí a čekají na portu 25 na příchozí poštu. Součástí MTA může být i proces IMAP, který spravuje na straně serveru uživatelské schránky a umožňuje přesun zpráv ze schránky pro příchozí poštu do dalších schránek.

4.1.2 Formát zpráv elektronické pošty

Formát zpráv přenášených systémem elektronické pošty definuje standard RFC 822 [2, 4]. Původní formát – sedmibitový text – už dnes nestačí pro přenos obrazových či multimediálních dat. Proto bylo v roce 1995 definováno rozšíření MIME (Multipurpose Internet Mail Extension) [5, 6], které strukturovat emailové zprávy tak, aby mohly obsahovat i netextová data. Struktura zprávy v porovnání s klasickým dopisem je zobrazena na obr. 4.3. Každá emailová zpráva se skládá ze dvou základních částí – obálky (envelope) a zprávy (message), která je tvořena hlavičkou zprávy (header) a tělem (body) oddělených prázdným řádkem.

Hlavička zprávy (message header) obsahuje řádky ve formátu:

```
field_name: field_body CRLF
```



Obrázek 4.2: Architektura elektronické pošty: klient (UA), server (MTA)

Pořadí polí (řádků) v hlavičce není garantováno. Při přeposílání zpráv nesmí dojít ke změně pořadí původních polí, zejména u cesty zprávy. Povinná pole ve zprávě jsou datum a čas napsání zprávy (Date:) a adresa odesílatele (pole From:). Přehled důležitých polí ve zprávě je uveden v tabulce 4.1.

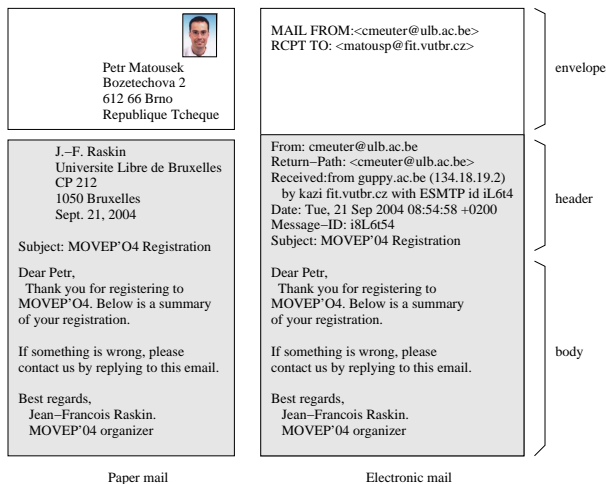
Identifikátor zprávy (Message-Id) je globálně jedinečná hodnota pro identifikaci emailu. Pro zajištění jednoznačnosti se často přidává IP adresa odesílajícího počítače, případně čas a datum vytvoření. Dalšími údaji je např. číslo zpracovávaného procesu. Norma doporučuje identifikátor rozdělit na část identifikace zprávy na počítači a část identifikace domény, oddělené znakem . Formát identifikátoru zprávy je:

Message-Id: <time/process-id @ domain>

Příkladem identifikátoru Message-ID je například hodnota 20080911084427.10026b2117b1wwkc@email.fit.vutbr.cz z následující zprávy. Z této ukázky můžeme vyčíst, že byla vytvořena 11.9.2008 v 8:44:27 našeho času (CEST, Central European Summer Time). Odesílatel xskurla@stud.fit.vutbr.cz ji vytvořil v prohlížeči Mozilla přes webovou emailovou bránu Horde na počítači email.fit.vutbr.cz (Message-ID). Odesílatel byl připojen přes ADSL připojení slovenského poskytovatele T-COM, ADSL server 78.99.125.198. Zpráva pak byla předána na SMTP server eva.fit.vutbr.cz a uložena na server kazi.fit.vutbr.cz pro uživatele matousp@fit.vutbr.cz.

Z emailu dále můžeme zjistit, že zpráva byla psána s diakritikou v kódování ISO 8859-2. Pro přenos SMTP, který je obecně sedmibitový, odesílající program osmibitovou zprávu s diakritikou převědl na sedm bitů transformací `quoted-printable` (viz dále). Zpráva byla před doručením testována antispamovým programem MIMEDefang, dokonce na dvou serverech.

From xskurl00@stud.fit.vutbr.cz Thu Sep 11 08:44:28 2008



Obrázek 4.3: Struktura zprávy

```
Return-Path: <xskurl00@stud.fit.vutbr.cz>
Received: from eva.fit.vutbr.cz (eva.fit.vutbr.cz [147.229.176.14]) by kazi.fit.vutbr.cz
(envelope-from xskurl00@stud.fit.vutbr.cz) (8.14.3/8.14.1) with ESMTP id m8B6iRXn063855
(version=TLSv1/SSLv3 cipher=DHE-RSA--SHA bits=256 verify=OK) for <matousp@fit.vutbr.cz>;
Thu, 11 Sep 2008 08:44:27 +0200 (CEST)
Received: from fit.vutbr.cz (hermina.fit.vutbr.cz [147.229.9.15]) by eva.fit.vutbr.cz
(envelope-from xskurl00@stud.fit.vutbr.cz) (8.14.3/8.14.1) with ESMTP id m8B6iRAi083028
for <matousp@fit.vutbr.cz>; Thu, 11 Sep 2008 08:44:27 +0200 (CEST)
Received: from adsl-dyn198.78-99-125.t-com.sk (adsl-dyn198.78-99-125.t-com.sk [78.99.125.198])
by email.fit.vutbr.cz (Horde Framework) with HTTP; Thu, 11 Sep 2008 08:44:27 +0200
Message-ID: <20080911084427.10026b2117blwkc@email.fit.vutbr.cz>
Date: Thu, 11 Sep 2008 08:44:27 +0200
From: "=?iso-8859-2?b?qWt1cmxhIA==?=" =?iso-8859-2?b?SuFu?=" <xskurl00@stud.fit.vutbr.cz>
To: matousp@fit.vutbr.cz
Subject: ISA
MIME-Version: 1.0
Content-Type: text/plain; charset=ISO-8859-2; DelSp="Yes"; format="flowed"
Content-Disposition: inline
Content-Transfer-Encoding: quoted-printable
User-Agent: Internet Messaging Program (IMP) H3 (4.2)
X-Originating-IP: 78.99.125.198
X-Remote-Browser: Mozilla/5.0 (Windows NT 6.0; cs; rv:1.9.0.1) Gecko/2008070208 Firefox/3.0.1
X-Spam-Score: 0.516 () AWL,RDNS_NONE
X-Scanned-By: MIMEDefang 2.64 on 147.229.8.12
X-Scanned-By: MIMEDefang 2.64 on 147.229.176.14
X-IMAPbase: 1221119473 90 NonJunk
Status: RO
X-Status: A
X-Keywords:
```

Skupina	Pole	Poznámka
Datum odeslání	Date:	lokální čas a datum, povinné pole, pouze jednou
Identifikace odesílatele	From: Sender Reply-To:	autor zpráva, povinné pole, pouze jednou kdo skutečně zprávu odeslal (např. sekretářka) na kterou adresu se má odpovědět
Identifikace adresáta	To: Cc: Bcc:	prvotní adresáti zprávy (carbon copy, přes kopírák) informace pro další adresáty, jimž není zpráva přímo poslána (blind Cc) adresáti, jejichž adresu neuvidí ostatní adresáti
Identifikace zprávy	Message-ID: In-Reply-To: References:	jednoznačný identifikátor zprávy, generován počítačem při odeslání, identifikace se během cesty nemění v odpovědi nesou identifikaci původní zprávy odkaz na původní zprávy, na kterou odpo- ovídá; používá se často v diskuzních příspěvcích (Usenet)
Informační pole	Subject: Comments: Keywords:	krátký řetězec vyjadřující obsah zprávy, v od- povědích se předmět změnil na Re: (res, lat.) další komentář těla zprávy klíčová slova oddělená čárkou
Přeposílání zprávy	Resent-Date: Resent-From: Resent-To: Resent-Message-ID:	původní datum zprávy identifikace původního odesílatele identifikace původních adresátů identifikace původní zprávy
Záznamy o cestě zprávy	Return-Path: Received:	

Tabulka 4.1: Některá pole hlavičky zprávy podle RFC 2822

X-UID: 1

Dobry den,

chcel by som sa sptat, ci sa budu dat uznat body z projektu z predmetu ISA.

J=E1n =A9kurla, xskurl00@stud.fit.vutbr.cz

Standard MIME – Multipurpose Internet Mail Extension

V době vzniku Internetu byly zprávy elektronické pošty výhradně textové a používalo sedmi-bitové kódování ASCII. Nároky na přenos se postupně zvěšovaly. Uživatelé chtěli používat diakritiku, ne-latinské abecedy (ruština, hebrejštiny), východní jazyky bez abecedy (čínština, japonština) či binární obrázky. Toto řeší rozšíření MIME [5, 6], které rozděluje tělo zprávy na nezávislé části podle typu. Každou část MIME kóduje různým způsobem. Umožňuje tak poslat zprávu tvořenou textovými daty, binárním digitálním podpisem a s přílohou dokumentu z Wordu. Každý typ obsahu je zakódován odlišným kódováním (např. plain text, base64 apod.).

V hlavičce zprávy je uveden typ kódování zprávy a odkaz na první část ve formátu MIME. Každá část obsahuje textový typ, případně podtyp obsahu (například text/plain), dále pak znakovou sadu (iso-8859-2) a typ kódování, který je pro přenos použit (Content-Transfer-

Hlavička	Význam	Příklad
MIME-Version:	verze MIME	MIME-Version: 1.0
Content-Description:	typ zprávy	S/MIME Encrypted Message
Content-Id:	identifikace přiřazená UA	< 005dc9d6ca8c0@JIRKA >
Content-Transfer-Encoding:	kódování obsahu zprávy	quoted-printable
Content-Type:	typ a formát obsahu zprávy	text/plain; charset=ISO-8859-2
Content-Length:	počet bytů zprávy	729

Tabulka 4.2: Hlavičky přidávané MIME

Encoding). Základní hlavičky MIME s příkladem jsou uvedeny v tabulce 4.2.

Mezi nejčastěji používaná schemata kódování obsahu patří mechanismus **base64** používaný pro netextová binární data a schema **quoted-printable** doporučené pro textové osmibitové zprávy. Další způsoby kódování jsou uvedeny v tabulce 4.3. Kódování **base64**

Typ	Popis
7-bit	7-bitové ASCII znaky s řádky max. 1000 znaků
8-bit	8-bitové ASCII, porušuje definici SMTP (7-bitů)
binary	porušuje definici SMTP, není zajištěno správné doručení
base64	kódování 8-bitových znaků do 7-bitů (binárních data)
quoted-printable	kódování 8-bitových ASCII znaků do 7-bitů (textová data)

Tabulka 4.3: Schemata kódování zprávy pro přenos přes SMTP

převádí tři osmibitové znaky na čtyři šestibitové znaky. Každý šestibitový znak je převeden podle speciální tabulky na jeden znal ASCII, který kóduje tuto hodnotu. Například slovu “čau” odpovídá podle ISO 8859-2 kód “E8 61 75” (hex), což je binárně 1110 1000 0110 0001 0111 0101. Pokud rozdělíme těchto 24 bitů na čtyři šestice, dostaneme řetězec 111010 000110 000101 110101, čili “3A 06 05 35” hexadecimálně. Každou šestici převedeme pomocí převodní tabulky **base64** [5] na ASCII znaky a dostaneme výsledný kódovací řetězec “6GF1”, který odpovídá kódovanému slovu “čau”.

Kódování **quoted-printable** je jednodušší na vytváření. Kódování se týká pouze osmibitových znaků, sedmibitové znaky se reprezentují původní ASCII hodnotou. Každý osmibitový znak se převede na tři sedmibitové znaky. Prvním znakem je uvozující znak “=”, následuje pak reprezentace osmibitového znaku v hexadecimální podobě. Například slovo “čau” by se reprezentovalo jako “=E8au”.

Typ obsahu u standardu MIME se uvádí v hlavičce zprávy ve tvaru **Content-type: type/sub type**. Pokud je zpráva tvořena více typy obsahu, použije se speciální typ **Multipart**, který rozdělí zprávu na samostatně kódované části. Jednotlivé části lze zobrazovat paralelně, vybrat pouze jeden typ obsahu (například zpráva přenášená v kódování HTML a textu ASCII) apod. Pro přenos textových zpráv se používá kód **Plain** pro neformátovaný text či **Enriched** pro jednoduché formátování. Pokud zpráva obsahuje přílohy, je v typu **Application** uveden typ přílohy, např. **msword**. Přehled základních typů a podtypů MIME je uveden v tabulce 4.4. Následující zpráva obsahuje dvě části kódované standardem MIME. Podtyp **mixed** nám říká, že jde o nezávislé části. První část je textová a obsahuje znaky z množiny ISO 8859-2. Tato část byla zakódována mechanismem **quoted-printable**. Druhou částí je aplikace vytvořená v MS Wordu, konkrétně soubor “prosinec 2004.doc”. Tato zpráva je k dopisu připojena jako

Typ	Podtyp	Popis
Text:	Plain	Neformátovaný text
	Enriched	Text obsahující jednoduché formátovací příkazy
Image:	Gif, Jpeg	obrázek formátu GIF, JPEG
Audio:	Basic	zvuk
Video:	Mpeg	film ve formátu MPEG
Application:	Octet-stream	neinterpretovaná sekvence bytů
	Postscript	dokument ve formátu Postscript
Message:	Rfc822	zpráva MIME RFC 822 (pro přeměrování)
	Partial	zpráva byla rozdělena při přenosu
	External-body	zpráva se musí načíst přes síť
Multipart:	Mixed	nezávislé zprávy v určeném pořadí
	Alternative	stejná zpráva v jiném formátu
	Parallel	části je potřeba zobrazit souběžně
	Digest	každá část je kompletní zpráva podle RFC 822

Tabulka 4.4: Typy a podtypy standardu MIME, RFC 2045 [5]

příloha (Content-Disposition: attachment) a zakódována technikou base64.

```

From: svanov@fit.vutbr.cz Wed Nov 3 16:10:19 2004
Return-Path: <fitinfo-bounces@fit.vutbr.cz>
Message-Id: <200411031424.iA3E09nT002868@kazi.fit.vutbr.cz>
From: =?iso-8859-2?Q?Eva_Sv=E1novsk=E1?= <svanov@fit.vutbr.cz>
To: "FITINFO" <fitinfo@fit.vutbr.cz>, "studinfo" <studinfo@fit.vutbr.cz>
Date: Wed, 3 Nov 2004 15:24:09 +0100
MIME-Version: 1.0
Content-Type: multipart/mixed;
    boundary="-----_NextPart_000_0035_01C4C1B9.2B9F0990"
X-Mailer: Microsoft Office Outlook, Build 11.0.6353
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.2180
Sender: fitinfo-bounces@fit.vutbr.cz
Errors-To: fitinfo-bounces@fit.vutbr.cz

This is a multi-part message in MIME format.
-----_NextPart_000_0035_01C4C1B9.2B9F0990
Content-Type: text/plain;
    charset="iso-8859-2"
Content-Transfer-Encoding: quoted-printable

P=F8epos=ED1=E1m nab=EDdku koncert=F9 v prosinci 2004, viz p=F8=EDloha.
=20
Eva Sv=E1novsk=E1
-----_NextPart_000_0035_01C4C1B9.2B9F0990
Content-Type: application/msword;
    name="prosinec 2004.doc"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
    filename="prosinec 2004.doc"

OM8R4KGxGuEAAAAAAAAAAAAAAAAAAAAAPgADAP7/CQAGAAAAAAAAAAAAAAAACAAAAAowAAAAAAAA
EAAApQAAAAEAAAD+////AAAAKEAAACiAAAA////////////////////////////////////////
AAAA
-----_NextPart_000_0035_01C4C1B9.2B9F0990

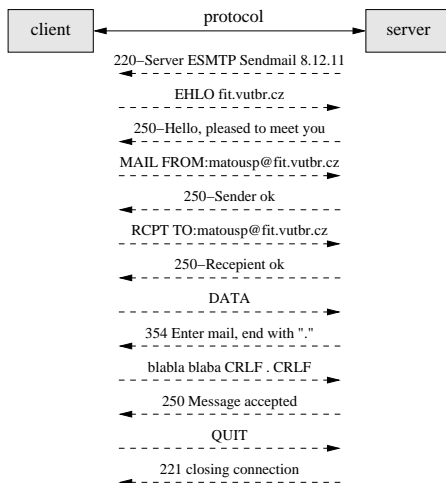
```

4.1.3 Přenos zpráv – protokol SMTP

Když jsem se bavili o architektuře systému elektronické pošty, zmínili jsme se o přenosovém protokolu SMTP (Simple Mail Transfer Protocol). Tento aplikační protokol je základem systému elektronické pošty. Byl definován v roce 1982 ve standardu [1]. Dnes se používá i jeho rozšířená verze ESMTP (Extended SMTP) [3].

Protokol SMTP pracuje nad transportním protokolem TCP a využívá port 25. Je to jednoduchý textový protokol pro přenos zpráv. V původním standardu byl definován pro přenos sedmibitových zpráv a i dnes se doporučuje přenášet zprávy v sedmibitovém tvaru¹. Poštovní server poslouchá na portu 25 a přijímá protokolem SMTP příchozí poštu. Pokud není zpráva určena pro lokální systém, předá ho SMTP server dalšímu SMTP serveru pro cílové doručení. Znamená to, že poštovní server se při přijímání zpráv chová jako SMTP server a při předávání zpráv jako SMTP klient. Toto chování není neobvyklé – využívá se zejména u proxy serverů. My se s ním můžeme také setkat u služby LDAP při synchronizaci dat mezi primárním a sekundárním LDAP serverem.

Vlastní komunikace přes SMTP je velmi jednoduchá – viz obrázek 4.4. Po navázání spojení



Obrázek 4.4: Komunikace pomocí protokolu SMTP

na transportní vrstvě pošle server klientovi uvítací zprávu, ve které oznamuje verzi protokolu, kterou používá (SMTP či ESMTP), případně další nastavení, která podporuje (například 8BITMIME pro osmibitový přenos). Následuje identifikace klienta příkazy HELO (SMTP) či EHLO (Extended HELO, ESMTP). Při přenosu zprávy je potřeba zadat odesílatele (MAIL FROM), adresáta (RCPT TO) a vlastní tělo dopisu (DATA). Po ukončení vkládání dat je zpráva předána druhé straně k odeslání. Spojení se ukončuje příkazem QUIT. Seznam příkazů protokolu SMTP je uveden v tabulce 4.5. Komunikace SMTP podle RFC 2821 probíhá vždy

¹Pokud obsahují binární či jiná osmibitová data, lze využít rozšíření MIME

Příkaz	Význam
EHLO, HELO	identifikace klient (plně kvalifikované jméno domény)
AUTH	zahájení autentizace odesilatele pomocí SASL
MAIL FROM	adresa odesilatele (mailbox)
RCPT TO	identifikace adresáta
DATA	7-bitová textová data ukončená <CRLF>.<CRLF>
VERFY	klient otestuje identifikaci adresáta
EXPN	zobrazí seznam adres mailing listu typu alias
HELP	vypíše pomocné informace klientovi
NOOP	prázdný příkaz
QUIT	uzavření spojení

Tabulka 4.5: Přehled příkazů protokolu SMTP

anonymně – protokol SMTP ani ESMTP nepodporuje autentizaci odesilatele. Je to proto, že uživatel může využít při komunikaci různém SMTP server podle místa, kde je připojen. Sdílení hesla by v tomto případě bylo hodně obtížné. Nicméně rozšíření RFC 4954 z roku 2007 [7] definuje rozšíření SMTP (příkaz AUTH) pro autentizaci odesilatele pomocí SASL. Pokud server podporuje autentizaci klienta, proběhne autentizace a je možné ustanovit bezpečný přenos pro komunikaci SMTP. Pokud SMTP server autentizaci nepodporuje, odmítne příkaz AUTH a komunikace probíhá podle standardního scénáře.

Příklad ověření uživatele pomocí SASL ukazuje následující výpis. V tomto příkladu server oznámí podporu autentizace SASL i bezpečného přenosu TLS. Klient zahájí inicializaci bezpečné vrstvy TLS příkazem STARTTLS a následuje autentizace uživatele příkazem AUTH. Ukázka z výpisu komunikace SMTP:

```
S: 220-smtp.example.com ESMTP Server
C: EHLO client.example.com
S: 250-smtp.example.com Hello client.example.com
S: 250-AUTH GSSAPI DIGEST-MD5
S: 250-ENHANCEDSTATUSCODES
S: 250 STARTTLS
C: STARTTLS
S: 220 Ready to start TLS
... TLS negotiation proceeds, further commands
   protected by TLS layer ...
C: EHLO client.example.com
S: 250-smtp.example.com Hello client.example.com
S: 250 AUTH GSSAPI DIGEST-MD5 PLAIN
C: AUTH PLAIN dGVzdABOZXNOADeYmZQ=
S: 235 2.7.0 Authentication successful
```

Chybějící či na serveru nepodporovaná autentizace s sebou přináší palčivý problém spamů a podvržených emailů. SMTP server by měl u každého příchozího spojení určit, zda je autorizované pro preposílání emailů. Pokud SMTP server přijme každé spojení bez ohledu na odesilatele (tzv. open-relay server), může být zneužitý jako brána pro spammy. Obvykle poskytovatelé internetového spojení umožní připojení k SMTP serveru pouze svým zákazníkům, např. na základě IP adresy ze svého prostoru IP adres. Pokud je spojení z jiné adresy, bude žadatel odmítnut s tím, aby využíval služby svého vlastního poskytovatele. Pokud je adresátem zprávy lokální uživatel daného SMTP serveru, pak je odesílatel autorizován předat

serveru zprávu bez ohledu na svou IP adresu.

Kromě přijímání a přeposílání emailových zpráv vykonává SMTP server ještě další služby. Pokud například zpráva nemůže být doručena (další server je nedostupný, neznámá adresa, neznámý uživatel apod.), je zpráva odmítnuta a server vygeneruje chybovou odpověď, viz následující ukázka:

```
The original message was received at Tue, 17 Apr 2007 09:06:21 +0200 (CEST)
from pcmatousek.fit.vutbr.cz [147.229.12.91]
```

```
----- The following addresses had permanent fatal errors -----
<libor.vejpushka@autocont.cz> (reason: 550 No such user in the domain)
```

```
----- Transcript of session follows -----
... while talking to ares.autocont.cz.:
>>> >>> RCPT To:<libor.vejpushka@autocont.cz>
<<< 550 No such user in the domain
550 5.1.1 <libor.vejpushka@autocont.cz>... User unknown
```

```
Reporting-MTA: dns; kazi.fit.vutbr.cz
Received-From-MTA: DNS; pcmatousek.fit.vutbr.cz
Arrival-Date: Tue, 17 Apr 2007 09:06:21 +0200 (CEST)
```

```
Final-Recipient: RFC822; libor.vejpushka@autocont.cz
Action: failed
Status: 5.1.1
Remote-MTA: DNS; ares.autocont.cz
Diagnostic-Code: SMTP; 550 No such user in the domain
Last-Attempt-Date: Tue, 17 Apr 2007 09:06:21 +0200 (CEST)
```

```
Subject: cenaov nabidka pocitacu pro FIT VUT v Brne
From:Petr Matousek <matousp@fit.vutbr.cz>
Date: Tue, 17 Apr 2007 09:06:20 +0200
To:libor.vejpushka@autocont.cz
```

Dobry den, ...

Výše uvedený výpis byl vygenerován na poštovním serveru, který je primární pro přijímání pošty pro doménu `autocont.cz` (viz záznamy MX domény `autocont.cz`). Pošta určená pro uživatele `libor.vejpushka@autocont.cz` nebyla doručena, protože daný uživatel na tomto serveru (a této doméně) nemá uživatelský účet. Jde pravděpodobně o překlep v emailové adrese. Poštovní server MTA vrátí odpověď odeslateli s udáním chyby a hlavičkou emailu. Zároveň se kopie dopisu přepoše správci lokálního poštovního systému, který by měl sledovat nedoručené dopisy a reagovat na případné chyby v nastavení. Pokud odesílatel ve zprávě chybně uvedl svou emailovou adresu, nelze chybu o nedoručení zprávy poslat a odesílatel se nedozví, že zpráva nebyla korektně doručena.

Pokud je server pouze přechodně nedostupný, zpráva na procházejícím SMTP server se uloží do fronty nedoručené pošty a doručení se opakuje. Podle konfigurace serveru se ji pokusí SMTP server doručit opakovaně během další tří dní. Pokud se mu to nepodaří, vygeneruje chybovou zprávu, odešli ji na emailovou adresu odesílatele a vyřadí zprávy z fronty. Odesílatel je tedy vždy informován, že zpráva nebyla doručena. Pokud uvede nesprávnou zpáteční adresu, nelze tuto chybovou zprávu doručit.

Pokud je daný SMTP server koncovým server, tzn. zpráva je určena pro lokálního uživatele, uloží server příslušnou zprávu do zvláštního souboru – poštovní schránky pro příchozí poštu (tzv.

INBOX). Na unixových systémech je tato schránka umístěna v souboru `/var/mail/login`. Uživatel si ji odtud může vyzvednout buď přímo procházením souboru, nebo pomocí služeb pro čtení doručených zpráv – protokolů IMAP či POP3.

4.1.4 Čtení zpráv – služby POP3, IMAP

Doručená emailová zpráva je uložena na cílovém poštovním serveru, jehož adresa je pro danou doménu definována záznamy MX v DNS. Zprávy jsou uloženy v textovém souboru. Pro čtení emailových zpráv slouží dvě služby využívající protokol POPv3 (Post Office Protocol) [8] nebo protokol IMAPv4 (Internet Message Access Protocol) [9]. Je možné také využít přístup přes Webové služby (—Webmail). Zde se jedná pouze o HTTP proxy, které zprostředkovává přístup k poštovní schránce. Proto se zde službě Webmail nebudeme více věnovat.

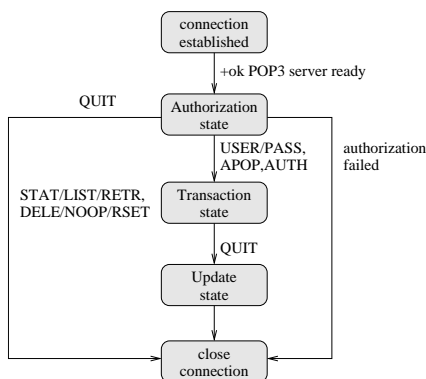
Protokoly POP3 a IMAP neslouží pouze ke čtení doručení zpráv. Protože není vhodné mít všechny doručené zprávy v jednom souboru, umožňují služby POP3 a IMAP vytvořit další schránky (soubory) a přichodí zprávy například tematicky rozdělovat. POP3 a IMAP umožňují vytvářet další schránky, přesouvat a rušit v nich zprávy, vyhledávat ve zprávách a podobně. Obě služby pracují na principy klient-server. Klientem je obvykle poštovní program (např. Thunderbird, Outlook, pine), který kromě odesílání zpráv protokolem SMTP umí také načítat došlé zprávy protokolem POP3 či IMAP. POP3 či IMAP server je aplikace, která běží na serveru, kde se nacházejí schránky přijaté elektronické pošty. POP3/IMAP server přijme přicházející spojení, autentizuje klienta a pak provádí operace nad jeho poštovní schránkou.

Protokol POPv3 - Post Office Protokol

Protokol POPv3 [8] je jednodušší protokol z hlediska činnosti i implementace než protokolem IMAP. Jedná se o aplikační protokol využívající transportní protokol TCP, port 110. POP3 načítá data z poštovního serveru a kopíruje je na lokální počítač. Umí pracovat pouze s jedinou schránkou na straně serveru – schránkou pro přichodí poštu INBOX. Při připojení a autentizaci uživatele umístí POP3 na schránku zámek tak, aby ji mohl využívat pouze jeden klient zároveň. Zprávy přenáší protokol na stranu klienta, kde se zobrazují off-line. Manipulace se zprávami ve schránce (přesuny, mazání) neprobíhají on-line, ale aplikují se až na konci před ukončením komunikace, kdy dojde ke synchronizaci. Nevýhodou tohoto přístupu je, že v případě přerušení spojení se operace nezapišou správně. Navíc se může stát, že při nekorektním ukončení spojení zůstane na schránce umístěn zámek, který časem expiruje nebo ho musí ručně odstranit správce systému.

Činnost protokolu je popsána stavovým automatem na obrázku 4.5. Po navázání spojení se dostává server do autorizačního stavu, kde probíhá ověření totožnosti uživatele (příkazy USER, PASS). Protože je protokol textový a uživatelské jméno i heslo jsou předávány v otevřené podobě, je potřeba zajistit důvěrnost spojení jiným způsobem, například tunelování transportního spojení pomocí TLS/SSL (Transport Layer Security/Secure Sockets Layer) nebo SASL (Simple Authentication Security Layer) [10]. Otevřená komunikace přes POP3 je náchylná pro odposlech hesla, proto se nedoporučuje. Zabezpečení přenosu POP3 přes SASL popisuje standard RFC 5034 [11].

Po přihlášení se uživatel dostává do stavu Transaction, kde provádí operace nad schránkou – načítání zpráv, rušení, vyhledávání apod. Seznam příkazů protokolu POP3 je uveden v tabulce 4.6. Po ukončení komunikace příkazem QUIT se server dostane do stavu aktualizace (Update), kde odstraní zprávy určené pro zrušení.



Obrázek 4.5: Chování protokolu POP3

Příklad komunikace protokolu POP3 je uveden v následující ukázce. Uživatel *mrose* se přihlásí pomocí mechanismu APOP ke své schránce, která obsahuje dvě zprávy o délce 320 bytů. Příkazem LIST vypíše pořadí a délku obou zpráv. Příkazem RETR si načte zprávu číslo 1 na lokální systém a pak ji příkazem DELE zruší na serveru.

```

S:      +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:      APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:      +OK mrose's maildrop has 2 messages (320 octets)
C:      STAT
S:      +OK 2 320
C:      LIST
S:      +OK 2 messages (320 octets)
S:      1 120
S:      2 200
S:      .
C:      RETR 1
S:      +OK 120 octets
S:      <the POP3 server sends message 1>
S:      .
C:      DELE 1
S:      +OK message 1 deleted
C:      QUIT
S:      +OK dewey POP3 server signing off (maildrop empty)

```

Z této komunikace je vidět jedna důležitá vlastnost protokolu POP3 – protokol neumí pracovat s více schránkami na straně serveru. Neexistuje příkaz na změnu schránky. Přesto umožňují poštovní programy vytvoření více schránek a kopírování příchozích zpráv do nich. Tyto schránky nejsou umístěny na serveru, ale na lokálním systému, kde běží poštovní klient. Pokud uživatel si načte zprávu v práci a uloží na pracovním počítači, nedostane se k ní například z domácího počítače. Toto je hlavní nevýhoda protokolu POP3 a rozdíl oproti protokolu IMAP, který umožňuje uložení a zpracování došlé pošty na vzdáleném serveru.

Příkaz	Význam
USER	jméno identifikující poštovní schránku
PASS	heslo pro přístup ke schránce/serveru
APOP	bezpečné přihlášení ke schránce pomocí MD5
CAPA	server pošle podporované algoritmy pro zabezpečení SASL
AUTH	zahájení SASL autentizace
STAT	počet zpráv ve schránce + jejich velikost
LIST	seznam zpráv ve schránce a jejich velikost
RETR	server pošle požadovanou zprávu klientovi
DELE	nastaví u zprávy příznak "deleted"
RSET	smaže příznak "deleted" u zpráv označených pro zrušení
TOP	server pošle klientovi hlavičku a zadaný počet řádek zprávy
UIDL	server vypíše jednoznačný identifikátor zprávy (MsgId)
NOOP	prázdný příkaz
QUIT	uzavření spojení, smaže zprávy označené "deleted"

Tabulka 4.6: Přehled příkazů protokolu POP3

Bezpečné připojení přes SASL [11] rozšiřuje množinu příkazů o příkaz **CAPA**, kterým si klient vyžádá množinu podporovaných mechanismů SASL na straně serveru, a dále o příkaz **AUTH** pro autentizaci uživatele, viz následující příklad.

```
S: +OK pop.example.com BlurdyBlurp POP3 server ready
C: CAPA
S: +OK List of capabilities follows
S: SASL PLAIN DIGEST-MD5 GSSAPI ANONYMOUS
S: STLS
S: IMPLEMENTATION BlurdyBlurp POP3 server
S: .
C: AUTH PLAIN dGVzdABOZXNOAHRlc3Q=
S: +OK Maildrop locked and ready
```

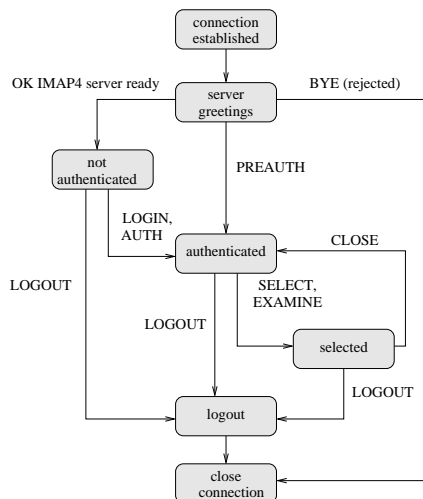
Ve výše uvedeném případě posílá server podporované mechanismy autentizace SASL – **PLAIN**, **DIGEST-MD5**, **GSSAPI**, **ANONYMOUS**. Klient si příkazem **AUTH** vybral metodu **PLAIN** [12] s posílá inicializační řetězec (challenge) kódovaný algoritmem Base64.

Protokol IMAP – Internet Message Access Protocol

Protokol IMAP ve své aktuální verzi 4 je definován v RFC 3501 [9]. Podobně jako protokol POP3 se jedná o aplikační protokol určený ke čtení došlé pošty. Využívá transport TCP, pracuje na portu 143. Protokol IMAP umí pracovat s více schránkami na vzdáleném poštovním serveru. Narozdíl od POP3 umožňuje současnou práci více klientů se schránkami. Pokud se další klient přihlásí ke schránce, označí se jako "read-only" a pokud chce některý z nich zapisovat do schránky, musí ji znovu otevřít. IMAP umí pracovat nejen z celou zprávou, ale z jednotlivými položkami z hlavičky dopisu (např. vyhledávání, řazení), čtení části zpráv apod.

Po připojení IMAP klienta k serveru nenačítá klient všechny zprávy v dané schránce jako POP3, ale pouze hlavičky. Teprve když chce ke zprávě přistoupit uživatel, načte i její tělo. Výrazným způsobem tak IMAP minimalizuje datové přenosy mezi poštovním klientem a

serverem. Pro off-line zpracování umožňují poštovní klienti načíst vybrané schránky, odpojit se od serveru, pracovat off-line a po dalším připojení synchronizovat schránky.



Obrázek 4.6: Stavy protokolu IMAPv4

Pro zprávy uložené na serveru si udržuje IMAP soubor atributů pro jednotlivé zprávy. Označuje jimi zprávy, které jsou nové, které byly přeposlány apod. Jak je vidět, IMAP server poskytuje mnohem komplexnější služby než POP3 server. Přináší to však značné nároky na implementaci serveru i výpočetní nároky. Z tohoto důvodu poskytovatelé internetového spojení raději pro své zákazníky poskytují POP3 spojení než IMAP.

Činnost IMAP serveru je zakreslena na obrázku 4.6. Po navázání spojení se dostává klient do autentizovaného stavu. Po úspěšné autentizaci přejde do autentizovaného stavu, kde může zadávat příkazy pro manipulaci se schránkou. Před začátkem práce si musí klient příkazy **SELECT** nebo **EXAMINE** vybrat schránku, se kterou bude pracovat. Po ukončení práce s touto schránkou příkazem **CLOSE** přechází zpět do autentizovaného stavu. V tomto stavu může vytvářet, přejmenovávat či rušit schránky (**CREATE**, **RENAME**, **DELETE**), označit schránku pro sledování (**SUBSCRIBE**, **UNSUBSCRIBE**) apod. Ve stavu **Selected** lze provádět operace nad zprávami dané schránky – můžeme je načítat (**FETCH**), prohledávat (**SEARCH**), rušit (**EXPUNGE**) apod. Po ukončení práce se schránkami se klient příkazem **LOGOUT** odhlásí a uzavře spojení. Přehled příkazů protokolu IMAP lze najít v tabulce 4.7.

IMAP server umožňuje ukládat ke zprávám atributy, které definují operace vlastnosti dané zprávy. Mezi základní atributy patří **Seen** (přečtená zpráva), **Answered** (odpověď zaslána), **Flagged** (nastavení příznaků), **Deleted** (označená zpráva ke zrušení), **Draft** (návrh zprávy), **Recent** (nová zpráva) apod. Poštovní klienti umí tyto atributy načíst a zobrazit uživateli, což usnadňuje práci se zprávami.

Následující příklad komunikace klienta a serveru ukazuje použití protokolu IMAP při manipulaci se schránkou. Na počátku se klient přihlásí k serveru a autentizuje (login mrc,

Příkaz	Význam
CAPABILITY	seznam podporovaných činností serveru (např. typ autentizace)
NOOP	vrací počet zpráv (zjišťování nových zpráv)
LOGOUT	uzavření spojení
STARTTLS	nastavení zabezpečení TLS pro komunikaci
AUTHENTICATE	zahájení autentizace SASL
LOGIN	přihlášení uživatele
SELECT	vybere požadovanou schránku, zobrazí stav zpráv
EXAMINE	stejně jako SELECT, schránka pouze ke čtení
CREATE	vytvoří schránku, i vnořenou schránku
DELETE	zruší schránku
RENAME	přejmenuje schránku
SUBSCRIBE	zařadí schránku mezi aktivní schránky (LSUB)
UNSUBSCRIBE	vyřadí schránku z množiny aktivních schránek
LIST	seznam schránek
LSUB	seznam aktivních schránek
STATUS	stav požadované schránky
APPEND	přidá zprávu na konec uvedené schránky
CLOSE	smaže zprávy s příznakem Delete a přejde do stavu autorizace
EXPUNGE	smaže zprávy s příznakem Delete
SEARCH	prohledává zprávy ve schránce
FETCH	vrátí vyhledaná data obsažená ve zprávách (velikost, odesílatel)
STORE	změní zadaná data na novou hodnotu ve zprávách (nastaví Delete)
COPY	kopíruje zprávy do nové schránky

Tabulka 4.7: Přehled příkazů protokolu IMAPv4

heslo secret). Opět vidíme, že i IMAP přenáší uživatelské jména a heslo v otevřené podobě. Podobně jako u protokolu POP3 se doporučuje využít přístup SSL/TLS či SASL pro čtení zpráv z poštovního serveru.

```

S:  * OK IMAP4rev1 Service Ready
C:  a001 login mrc secret
S:  a001 OK LOGIN completed
C:  a002 select inbox
S:  * 18 EXISTS
S:  * FLAGS (\Answered \Flagged \Deleted \Seen \Draft)
S:  * 2 RECENT
S:  * OK [UNSEEN 17] Message 17 is the first unseen message
S:  * OK [UIDVALIDITY 3857529045] UIDs valid
S:  a002 OK [READ-WRITE] SELECT completed
C:  a004 fetch 12 body[header]
S:  * 12 FETCH (BODY[HEADER] {342}
S:  Date: Wed, 17 Jul 1996 02:23:25 -0700 (PDT)
S:  From: Terry Gray <gray@cac.washington.edu>
S:  Subject: IMAP4rev1 WG mtg summary and minutes
S:  To: imap@cac.washington.edu
S:  cc: minutes@CNRI.Reston.VA.US, John Klensin <KLENSIN@MIT.EDU>
S:  Message-Id: <B27397-0100000@cac.washington.edu>
S:  MIME-Version: 1.0
S:  Content-Type: TEXT/PLAIN; CHARSET=US-ASCII

```

Po úspěšné autentizaci se uživatel připojí na schránku příchozí pošty (select inbox), kde je 18 zpráv. Uživatel si načte zprávu číslo 12, konkrétně hlavičku zprávy. Z výpisu je zajímavé si všimnout řetězce "a00x", který uvozuje každou datovou výměnu klienta. Tento řetězec je nutný k rozlišení více transakcí jednoho uživatele, např. prokládanými operacemi nad různými schránkami.

Z aplikací používaných pro přenos elektronické pošty lze uvést například řádkové poštovní klienty (UA) `elm`, `pine` či grafické aplikace `Thunderbird`, `Outlook`. Mezi nejběžnější poštovní servery (MTA) patří program `sendmail`, `qmail` a `postfix`. Čtení doručených zpráv zajišťují aplikace `popper` pro službu POP3 a `imapd` pro IMAP.

4.2 Zabezpečení poštovních služeb

Při přenosu elektronické pošty prochází zpráva přes mnoho uzlů, které jsou pod různou správou. Internet je založen na principu přepínané paketové sítě, přesněji řečeno samostatně směrovaných IP datagramů. Z pohledu odesílatele či příjemce nelze ovlivnit, kudy zpráva půjde. Např. při změně topologie sítě v důsledku přerušení některé linky na cestě se pakety pošlou jinou cestou k cíli. To je princip přenosu "best delivery", který zajišťuje protokol IP. Na druhé straně to přináší nebezpečí odposlechu přenášených zpráv, změnění či podvržení zprávy. Každý uživatel elektronické pošty si musí být vědom této skutečnosti a přiměřeně na ni reagovat. Pro důležité zprávy by měly být zabezpečeny.

Podobně to je s použitím služeb SMTP, POP a IMAP, které používají nešifrovaný přenos. To je kritické zejména pro služby POP a IMAP, kdy se uživatel připojuje ke své schránce. Součástí připojení je autorizace uživatelským heslem a jménem, které protokoly přenášejí v otevřené textové podobě, což může být zdrojem útoků. U služby SMTP, která není autorizovaná, se objevuje jako velký problém zneužití služby pro zaslání nevyžádaných či komerčních dopisů (spamů).

V této kapitole definujeme požadavky na bezpečnost přenosu elektronické pošty a ukážeme si některé prostředky, které mohou uživatelé i správci použít pro zabezpečení elektronické pošty. Ukážeme si princip systému klíčů PGP a S/MIME, možnosti vytváření tunelového spojení a filtrování spamových zpráv. V této části nebudeme vysvětlovat princip používaných šifer, což je nad rámec tohoto textu. Zájemce může najít podrobnosti například v [13].

4.2.1 Požadavky na bezpečnost

Pokud chceme nějakým způsobem chránit nějaká aktiva (např. elektronickou poštu), musíme si nejprve stanovit, na jaké hrozby máme reagovat. Posléze hledáme prostředky, jak hrozby snížit či úplně eliminovat. Mezi základní požadavky na zabezpečení přenosu patří:

- **Důvěrnost (confidentiality)**

Zabezpečení proti čtení cizí osobou, například při odposlechu na síti. Odposlech lze provádět pomocí paketového analyzátoru např. `tcpdump` či `Ethereal`. Přestože přepínané sítě snižují nebezpečí odposlechu cizí stanicí, lze zahltit přepínač zprávami z podvržených MAC adres a způsobit přeplnění CAM paměti, ve které se nachází přepínací tabulka. Pokud je tabulka přeplněna, chová se přepínač jako rozbočovač, tzn. kopíruje vstupní data na veškeré výstupy. Útočník pak má možnost odposlechnout i přenosy, které by do jeho segmentu sítě nepatřily. Důvěryhodnost zprávy zajišťujeme šifrováním.

Protože je nemyslitelné, že by si každý pár komunikujících účastníků vyměňoval šifrovací klíče, využívá se techniky asymetrické kryptografie s veřejným a tajným klíčem. Odesílatel, který zašifruje zprávu veřejným klíčem příjemce má jistotu, že ji nemůže číst nikdo jiný, než právě vlastník odpovídajícího tajného klíče, v tomto případě příjemce.

- **Autentizace (authentication)**

Požadavek autentizace vychází z možnosti snadně vygenerovat emailovou zprávu a poslat ji pod cizí identitou. To lze jednoduchým připojením na službu SMTP a znalostí čtyř příkazů SMTP. Protože je protokol SMTP textový a komunikuje přes TCP, lze k tomu využít připojením pomocí služby telnet na port 25. Pozorný adresát může z hlavičky emailu zjistit nekonzistenci mezi adresou odesílatele vloženou do hlavičky emailové zprávy a skutečnou adresou serveru, na kterém byla zpráva vygenerována a která je součástí obálky. Pro zajištění autentizace se používá mechanismus asymetrické kryptografie, konkrétně elektronický (digitální) podpis.

- **Integrita dat (integrity)**

Šifrování zpráv je časově náročné a ne vždy je zpráva natolik důvěrná, aby její obsah nemohla vidět třetí osoba. Přestože nese správa ověřený digitální podpis odesílatele, mohla být zpráva během přenosu upravena. Pro zajištění integrity dat se používají jednocestné hašovací funkce, které vygenerují ke každé zprávě řetězec fixní velikosti, který určitým způsobem závisí na všech znacích (a bitech) zabezpečované zprávy. Tomuto řetězci se říká *charakteristika zprávy* (message digest). Pro generování se používají algoritmy typu MD4, MD5, SHA-1 apod. Příjemce obdrží zprávu, spočítá její charakteristiku a porovná vypočtenou hodnotu z připojenou hodnotou. Pokud se liší, zpráva byla během přenosu modifikována.

- **Neodmítnutelnost (nonrepudiation)**

Neodmítnutelnost zprávy je důležitá zejména pro finanční transakce. Mohu například poslat šifrovaný a charakteristikou zabezpečený příkaz bance. Příkazem žádám například o převedení určité sumy z mého účtu na jiný účet. Po měsíci se zastavím v bance a s upřímným překvapením jim sdělím, že jsem o žádný převod nežádal a že udělali chybu. Mám pravdu – z doručené zprávy nelze zajistit, že jsem ji poslal já a ne nikdo jiný, kdo získal veřejný klíč banky a použil můj elektronický podpis z jiné zprávy. Neodmítnutelnost tedy znamená ověření odesílatele tak, že nikdo jiný nemohl provést danou operaci. Využívá se k tomu opět asymetrická kryptografie, konkrétně tajný klíč, kterým odesílatel podepíše (zašifruje) nějakou zprávu, která mu byla předtím doručena v podobě zašifrované jeho veřejným klíčem.

- **Dostupnost, kontrola přístupu**

Posledním typem zabezpečení, o kterém se zde zmíníme, je zajištění dostupnosti služby a kontrole přístup ke službě tak, aby ji mohli používat legitimní uživatelé. Některé způsoby zajištění kontroly přístupu se odehrávají na úrovni znalosti hesla pro přístup ke schránce elektronické pošty. U diskuzních skupin Usenet, kde není podpora autentizace uživatele, lze omezit dostupnost diskuzních skupin na určitý rozsah IP adres, ze kterých mohou uživatelé číst příspěvky.

Systém by měl také reagovat na pokusy o zneužití služby typu DoS (Denial of Service). Tento typ útoku se snaží neúměrným zatížením služby falešnými požadavky zahltnit

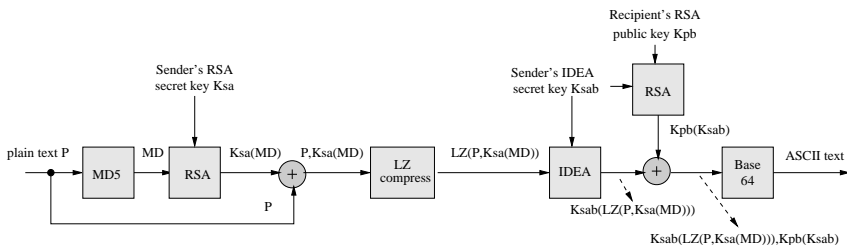
službu a zamezit legitimním uživatelům, aby ji používali. Příkladem může být generování nesmyslných zpráv na SMTP server. Jinými typy útoků, které spadají do této kategorie, jsou například spamy (nevyžádané emaily), podvodné emaily (hoax) a další.

Pro řešení se používají různé antispamové techniky, například systém kontroly hlaviček emailu MIMEDefang, různé databáze zneužívaných domén (tzv. blacklisty), kontrola reverzních záznamů odesílajících SMTP serverů apod.

4.2.2 PGP – Pretty Good Privacy

Jednou z technik pro zajištění bezpečnosti emailových zpráv je schéma PGP vytvořené roku 1995 Philem Zimmermannem. PGP umožňuje nejenom zprávy šifrovat, ale zajišťuje také autentizaci odesílatele, integritu zprávy i neodmítnutelnost. PGP je soubor programů, které slouží pro šifrování, podepisování, autentizaci zpráv a pro kompresi dat. Systém PGP umožňuje také uživateli spravovat šifrovací klíče.

PGP využívá pro šifrování symetrický šifrovací algoritmus IDEA (International Data Encryption Algorithm) s klíčem o délce 128 bitů. Princip činnosti PGP je ukázán na obrázku 4.7. Uživatel A chce odeslat zprávu příjemci B. Zprávu chce podepsat a zajistit proti odposlechu. Systém PGP nejprve vygeneruje charakteristiku zprávy pro nezašifrovanou zprávu P .



Obrázek 4.7: Zabezpečení emailové zprávy pomocí PGP

Aby ji nikdo nemohl znovu vypočítat pro upravenou zprávu, charakteristiku zprávy se podepíše tajným klíčem K_{SA} odesílatele. Podepsaná charakteristika se připojí k nezašifrované zprávě. Zpráva je pak komprimovaná algoritmem LZ (Lempel-Ziv). V tomto okamžiku je zpráva stále nezašifrovaná. Dále systém PGP vygeneruje náhodný řetězec K_{SAB} , který bude použit jako tajný klíč pro symetrické šifrování zprávy. Tímto klíčem je zpráva zašifrována algoritmem IDEA. Vznikne zpráva $K_{SAB}(LZ(P, K_{SA}(MD)))$, která obsahuje zašifrovanou a podepsanou původní zprávu. Ke zprávě je nutné ještě připojit klíč K_{SAB} na dešifrování. Pro zajištění důvěry pro přenos je tento klíč zašifrován asymetrickým algoritmem RSA za použití veřejného klíče příjemce K_{PB} . Pouze příjemce vlastní tajný klíč K_{SB} , kterým řetězec dešifrovat a získat symetrický klíč K_{SAB} pro dešifrování emailové zprávy. Výsledný zašifrovaná a podepsaná zpráva s klíčem je transformována algoritmem Base64 do 7-bitového tvaru, neboť protokol SMTP podporuje pouze 7-bitový přenos (viz [1] a kapitola 4.3). Výsledek je předán adresátovi. Adresát (příjemce) opačným postupem získá klíč, kterým je zašifrována zpráva, dešifruje její obsah a ověří integritu.

Pro zabezpečení potřebuje PGP veřejný a tajný klíč odesílatele pro ověření přijatých zpráv a podepisování odesílaných zpráv. Dále potřebuje veřejné klíče příjemců pro šifrování zpráv. Fakticky to znamená získat od každého příjemce jeho veřejný klíč. Veřejné klíče mohou být umístěny na některém veřejném serveru PGP, např. wwwkeys.cz.pgpg.net, který ručí za jejich platnost. Další možností je bezpečně předat (nejlépe osobně) veřejný klíč příjemce na lokální systém odesílatele. Důvěryhodnost veřejného klíče je Achillovou patou systému. Pokud útočník zamění veřejný klíč příjemce svým veřejným klíčem, stačí mu odchytnout zprávu a je schopen ji dešifrovat. Z tohoto důvodu je potvrzení pravosti veřejného klíče jedním z nejdůležitějších částí systému bezpečné komunikace. Využívají se k tomu například certifikáty.

Akce	Příkaz
Načtení klíče ze serveru	<code>gpg --keyserver wwwkeys.cz.pgpg.net --recv-keys ID</code>
Vygenerování klíčů	<code>gpg --gen-keys</code>
Výpis klíčů	<code>gpg --list-keys</code>
Export veřejného klíče	<code>gpg --output FILE --export ID (binární forma)</code> <code>gpg --armor --output FILE --export ID (text)</code>
Načtení veřejného klíče	<code>gpg --import FILE</code>
Ověření pravosti klíče	<code>gpg --edit-key ID -fpr/sign</code>
Šifrování veřejným klíčem	<code>gpg --output FILE.gpg --encrypt --recipient ID FILE</code>
Dešifrování soukromým klíčem	<code>gpg --output FILE --decrypt FILE.gpg</code>

Tabulka 4.8: Použití zabezpečení GNU Privacy Guard (GnuPG)

Systém PGP vytváří na uživatelském počítači speciální datové struktury určené pro ukládání tajného klíče a veřejných klíčů, tzv. *klíčenky* (key rings). Při podepisování, šifrování či dešifrování zpráv vybere systém PGP z klíčenky potřebný klíč. Tajný klíč je uložen v klíčence v zašifrované podobě a jeho použití vyžaduje heslo. To zabraňuje zneužití digitálního podpisu v případě odcizení souboru s tajným klíčem. Tabulka 4.8 ukazuje možnosti použití PGP v systému Unix.

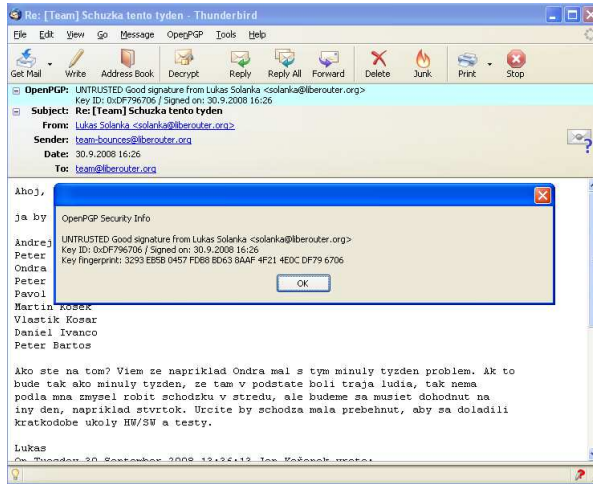
Systém PGP je integrován do běžných poštovních klientů (např. pine, Mozilla (enigmail), Outlook) a jeho použití je při psaní a čtení emailových zpráv transparentní. Protože distribuce původního balíku PGP byla z hlediska použití patentovaných algoritmů a technologií problematická, začali se používat volně šiřitelné implementace PGP, například OpenPGP [14]. Příklad ukázky kontroly podepsané emailové zprávy je na obrázku 4.2.2

4.2.3 S/MIME – Secure MIME

Další standardem pro zabezpečení emailových zpráv je rozšíření Secure MIME (S/MIME) definovaný ve standardech RFC 3850 [15] a RFC 3851 [16]. Podobně jako PGP poskytuje autentizaci, integritu dat, důvěrnost a neodmítnutelnost. Je to flexibilní technika, která podporuje různé typy kryptografických algoritmů. Využívá přitom standard MIME k zabezpečení různých typů emailových zpráv.

Princip je vzájemné důvěry zajišťovaný certifikáty je podobný standardu PEM (Privacy Enhanced Mail), který se neujal. Standard PEM totiž vyžadoval certifikaci klíčů podle doporučení X.509 a pevnou strukturu certifikačních autorit s jedním kořenem. Toto v praxi nebylo možné zajistit. Standard S/MIME také využívá certifikáty k potvrzení pravosti klíčů, nicméně validitu certifikátu může potvrzovat více subjektů, jejichž pravost lze jednotlivě ověřovat.

Zpráva zabezpečená technikou S/MIME využívá standard MIME pro definování různých částí zprávy. Přidanou sekci MIME je elektronický podpis, který se stane při odeslání (a



Obrázek 4.8: Kontrola podepsané zprávy klíčem PGP v programu Thunderbird

podepsání emailu) součástí emailové zprávy. Zabezpečení S/MIME je tedy součástí hlavičky emailové zprávy. Tím se přístup S/MIME liší od PGP, neboť PGP je záležitost aplikační, tzn. podpis i šifrování je součástí těla (obsahu) emailové zprávy. S/MIME se používá na úrovni hlavičky a podpis je připojen jako další část dopisu, viz následující ukázka:

```
From matousp@kazi.fit.vutbr.cz Fri Aug 22 09:33:29 2008 +0200
Message-ID: <48AE6BAB.20404@fit.vutbr.cz>
Date: Fri, 22 Aug 2008 09:32:59 +0200
From: Petr Matousek <matousp@fit.vutbr.cz>
...
Content-Type: multipart/signed; protocol="application/x-pkcs7-signature"; micalg=sha1
; boundary="-----ms090207020705030504000904"
```

This is a cryptographically signed message in MIME format.

```
-----ms090207020705030504000904
Content-Type: text/plain; charset=ISO-8859-2
Content-Transfer-Encoding: 7bit
```

```
Toto je testovací email ... Petr
-----ms090207020705030504000904
Content-Type: application/x-pkcs7-signature; name="smime.p7s"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7s"
Content-Description: S/MIME Cryptographic Signature
```

```
MIAGCSqGSib3DQEHAQCAMIACAQExCzAJBgUrDgMCGGUAMIAGCSqGSib3DQEHAQAAoIIJqjCC
...
JFFfb0Aa033GnDVpY88Y71KFNhKwu02EzceAAAAA
-----ms090207020705030504000904--
```

V ukázce vidíme, že je připojena část typu `application/pkcs7-signature`, která obsahuje elektronický podpis. Parametr `micalg` definuje jednocestný algoritmus pro verifikaci podpisu. V naší ukázce je to algoritmus SHA-1, další možné jsou MD5, SHA-256, SHA-384, SHA-512 a další. Vlastní podpis je zakódován do 7-bitového formátu metodou `base64`. Příklad šifrovaného a podepsaného dopisu standardem S/MIME můžeme vidět v následující ukázce:

```
From matousp@fit.vutbr.cz Mon Oct 13 21:17:59 2008 +0200
Return-Path: <matousp@fit.vutbr.cz>
Received: from nbmatousek.fit.vutbr.cz (ip4-83-240-62-147.cust.nbox.cz [83.240.6
Message-ID: <48F39EE6.2030204@fit.vutbr.cz>
Date: Mon, 13 Oct 2008 21:17:58 +0200
From: Petr Matousek <matousp@fit.vutbr.cz>
MIME-Version: 1.0
To: "matousp >> Petr Matousek" <matousp@fit.vutbr.cz>
Subject: test S/MIME
Content-Type: application/x-pkcs7-mime; name="smime.p7m"
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename="smime.p7m"
Content-Description: S/MIME Encrypted Message

MIAGCSqGSIB3DQEHA6CAMIACAQAQxggFnMIIBYwIBADBLMEMxEjAQBgoJkiaJk/IsZAEZFgJj
ejEZMBcGCgmSJomT8ixkARkWCWNlc25ldC1jYTESMBAGA1UEAxMJOVQVTTkVUIENBAGRCS1Qe
...
NsW/n3arSVXuD9SIh2t4HgIZhiPNxAws4ZFLIywC1LjdsMDfwr03NpWBgB1ZXIT/vhIGml9Sq
nzKzZi20B0NuqZypwqibTcwGkqz3zXlyjHoWg4gFiZ5oe0XBAhD1JUrWLEeOAAAAAAAAAAAA
AAA=
```

Z hlavičky je vidět, že se jedná o zprávu šifrovanou standardem S/MIME. Pro šifrování se použil veřejný klíč adresáta, aby nikdo kromě něj nemohl číst zprávu. Zpráva je také podepsaná, což však není vidět, neboť podpis je součástí šifrované zprávy.

S/MIME oproti PGP rozšiřuje zpracování hlaviček MIME, které je součástí většiny emailových klientů. PGP vyžaduje nadstavbu nad emailovým klientem (modul, plug-in), který zpracovává obsah přijatých emailů a vyhledává v něm informace o šifrování, podpisu apod.

```
From xkobie00@stud.fit.vutbr.cz Fri Jul 6 11:10:51 2007
Return-Path: <xkobie00@stud.fit.vutbr.cz>
MIME-Version: 1.0
...
Subject: Detekce typu protokolu
X-Enigmail-Version: 0.94.0.0
OpenPGP: id=AB1892C2
Content-Type: text/plain; charset=ISO-8859-2
Content-Transfer-Encoding: 7bit
-----BEGIN PGP SIGNED MESSAGE-----
Hash: SHA1
Ahoj,
na seminari si povidal ze nejaký kluk delal diplomku na detekci
protokolu pomoci neuronovych siti. Mohl bys me ji prosimte poslat ?
Diky moc

Petr K.
-----BEGIN PGP SIGNATURE-----
Version: GnuPG v1.4.7 (GNU/Linux)
Comment: Using GnuPG with Fedora - http://enigmail.mozdev.org
iD8DBQFGjiM6r8I+9asYksIRAjTPAJOUmYmZf7FxFkJlxo49A20MpUNfB4wCfZhqz
u5gm+JNa8LWj8Qnu4Dz4aWg=
=NuCK
-----END PGP SIGNATURE-----
```

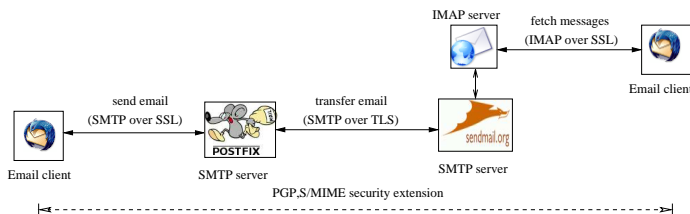
Výše uvedený email ukazuje použití PGP v praxi. Vytvořený email je vložen do speciální sekce, za níž následuje vlastní PGP podpis. Přestože to připomíná sekce MIME, jedná se o aplikační záležitost. Pokud příjemce nemá klienta, který by podporoval zpracování PGP, uvidí v těle dopisu přesně to, co je zobrazeno výše. Pro některé laické uživatele to může být překvapením.

4.2.4 Zabezpečení přenosu na úrovni transportních protokolů

Výše uvedené techniky PGP a S/MIME vytvářejí zabezpečený přenos mezi uživatelskými aplikacemi, tj. emailovým klientem odesílatele a klientem příjemce. Obě pracují na úrovni aplikačního protokolu SMTP a jsou součástí buď hlavičky či těle SMTP. Obě techniky podporují jak autentizaci, tak šifrování, zajištění integrity dat či neodmítnutelnost. Oba pracují na principu asymetrické kryptografie.

Pokud si odesílatel a příjemce nevymění veřejné klíče, není možné šifrovat mezi odesílatelem a příjemcem zprávu. Jestliže příjemce neumí ověřit digitální podpis odesílatele, ztrácí možnost autentizace odesílatele a verifikace integrity dat.

Existují však i další techniky, které se používají pro snížení rizika odposlechu přenášených dat a které jsou založeny na principech šifrování transportního spojení pomocí SSL/TLS nebo ssh. Obě tyto techniky zajišťují bezpečný přenos mezi koncovými body transportního spojení, viz obrázky 4.9, a nelze jimi ověřovat identitu uživatele. Přesto mají důležité místo v zabezpečení počítačové komunikace.



Obrázek 4.9: Zabezpečení emailové komunikace pomocí SSL/TLS.

Použití aplikace stunnel

Aplikace **stunnel** slouží k vytvoření bezpečného kanálu mezi dvěma počítači prostřednictvím bezpečných socket TLS/SSL [17, 18]. Slouží například pro vytvoření bezpečného spojení mezi poštovním klientem a SMTP serverem přes veřejnou síť. Toto spojení lze zabezpečit pomocí SSL přímo, tzn. je potřeba na obou stranách nakonfigurovat SSL podporu. Většina poštovních klientů podporu SSL obsahuje, u serverů je to vzhledem k obtížnosti implementace nevhodné. Místo toho se na straně serveru používá **stunnel**, což je speciální skript (angl. wrapper), který otevřenou komunikaci (např. SMTP) “zabaluje” do vytvořeného šifrovaného tunelu. Tunel se vytvoří externími funkcemi knihovny OpenSSL, takže server nemusí sám o sobě podporovat SSL.

Nejčastěji se **stunnel** používá na straně serveru. Na straně klienta se konfiguruje použití SSL. Klient zahájí komunikaci např. na portu 993 (IMAP přes SSL). Na druhé straně očekává server, který přijímá šifrovanou komunikaci. Ve skutečnosti tam běží **stunnel**, který vytvoří

bezpečnou asociaci s klientem, přijímá a dešifruje aplikační data. Ta pak předává například IMAP server, který je zpracovává a stejným kanálem posílá odpovědi. **Stunnel** se nejčastěji používá pro čtení pošty protokoly IMAP a POP3. Příklad nastavení je v následující ukázce:

```
stunnel -d 995 -r /usr/sbin/ipop3d -- ipop3d
stunnel -d 993 -r /usr/sbin/imapd -- imapd
```

Použití aplikace ssh

Podobný tunel lze vytvořit i použitím aplikace **ssh**, která vytváří bezpečné šifrované spojení mezi dvěma počítači. Pokud například náš poštovní klient nepodporuje TLS/SSL spojení, lze pomocí **ssh** vytvořit šifrovaný SSL kanál, do kterého klient zapisuje nešifrovaná aplikační data a na druhé straně kanálu je server aplikace **ssh** dešifruje a předává cílové aplikaci. Příklad vytvoření tunelového spojení pro protokol SMTP demonstruje následující ukázka:

```
% ssh -2 -N -f -L 5025:localhost:25 user@mailserver.example.com
user@mailserver.example.com's password: *****
% telnet localhost 5025
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 mailserver.example.com ESMTTP
```

V tomto případě vytvoří uživatel bezpečný tunel z lokálního počítače a lokálního portu 5025 na vzdálený počítač `mailserver.example.com` a vzdálený port 25. Aby tento **ssh** tunel mohl vzniknout, musí mít uživatel **user** přístup na vzdálený počítač. Pokud ho tam má, přihlásí se tam přes **ssh** a zároveň vytvoří tunel. Nyní veškerá lokální komunikace na port 5025 (viz ukázka) je předána do zabezpečeného tunelu a předána na cílový počítač a cílový port 25. Podobným způsobem je možné nastavit i bezpečné čtení elektronické pošty protokolem POP3, viz další příklad:

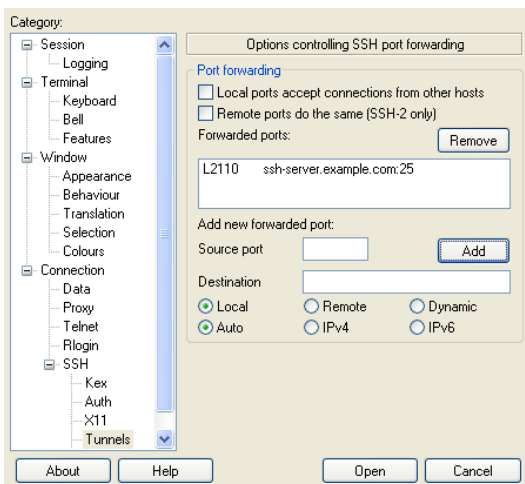
```
% ssh -2 -N -f -L 2110:mail.example.com:110 user@ssh-server.example.com
user@ssh-server.example.com's password: *****
```

Výše uvedené ukázky demonstrují SSH tunel v prostředí Unix. Podobně lze tunel vytvořit i pod MS Windows, například v aplikaci Putty, viz obrázek 4.10.

4.2.5 Spamy, hoaxy, viry

Jedním ze současných velkých problémů elektronické pošty je její zneužívání k rozesílání reklamních zpráv, nepravdivých informací, podvodných emailů či k šíření škodlivého kódu, např. virů, malware, adware a dalších. Není to zase tak neočekávané, když uvažíme záplavy reklamních letáků, které nacházíme ve schránkách klasické pošty. Komerční emaily nic nestojí a lze jimi během několika minut zaplavit spoustu emailových schránek. Jak se proti takové záplavě bránit? To je jeden z úkolů, které řeší současné služby elektronické pošty.

Než si představíme možná řešení, podíváme se na typy škodlivých emailů, se kterými se můžeme setkat. Prvním typem jsou nevyžádané zprávy (unsolicited emails), které se označují slovem spam. Jde převážně o komerční reklamní emaily. Spamy jsou rozesílány hromadně na emailové adresy, které odesílatel získal z různých databází, webů, internetových seznamů apod. Na Internetu pracují automatické programy, tzv. roboti, které procházejí webové stránky a hledají řetězce, které připomínají emailové adresy. Tyto adresy ukládají do databáze, které jsou zdrojem pro rozesílání spamů.



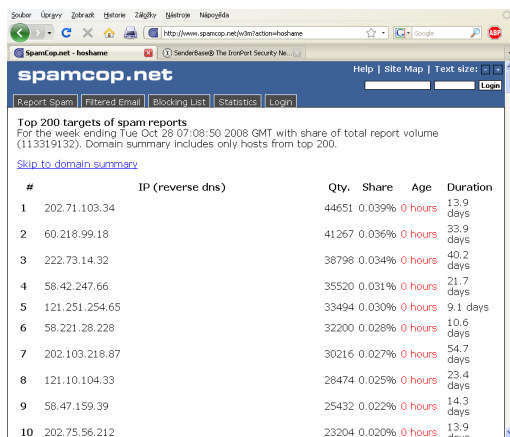
Obrázek 4.10: Vytváření ssh tunelu v programu Putty

Spam může být i nevyžádaná reklamní zpráva od organizace, kde jsem se z nějakého důvodu zaregistroval a z důvodu přihlášení uvedl svou emailovou adresu, např. elektronický obchod, diskuzní skupina, zpravodajský server, apod. Pokud tato organizace použije mou emailovou adresu k rozesílání reklamních nabídek, aniž bych s tím vyslovil souhlas, dopouští se zneužití emailové adresy a její emaily lze považovat za spamy.

Obrana proti spamům probíhá na různých úrovních. Spamy jsou obecně považovány za nedovolenou emailovou komunikaci a v mnohých státech existují legislativní normy, které produkci spamů postihují. V české republice je to zákon o č. 480/2004 Sb. “o některých službách informační společnosti a o změně některých zákonů” a zákon č. 101/200 Sb. v účinném znění “o ochraně osobních údajů a o změně některých zákonů”. Tyto zákony stanoví povinnosti fyzické i právnické osoby při zacházení s osobními údaji a zajišťování služeb elektronickými prostředky. Stanoví i postihy při porušení a zneužití získaných informací. V případě, že máme podezření, že nějaká organizace, již jsme poskytly údaje o elektronické poště zneužila tyto údaje pro reklamní účely a ani po vyzvání neodstranila naši emailovou adresu ze své databáze pro rozesílání komerčních nabídek, lze podat stížnost na Úřad pro ochranu osobních údajů a požádat o zahájení správního řízení. Podnět lze zadat i elektronicky přes Webové stránky úřadu na <http://www.uoou.cz>.

Na úrovni technické lze bojovat proti spamům různými prostředky. Jedním z nich je zamezit spammerům rozesílat spamy. Prakticky to znamená, že poskytovatel internetového spojení (ISP, Internet Service Provider) má sledovat aktivitu svých uživatelů. V případě, že zjistí podezření na rozesílání spamů, je povinen zasáhnout, tzn. znemožnit spammerovi zneužit jeho připojení. Odesílatel spamu totiž ke své činnosti potřebuje poštovní server (SMTP server), přes který posílá emaily do Internetu. Buď může využít SMTP server svého poskytovatele

nebo svůj vlastní SMTP server. Ten však musí být připojen k nějakému SMTP serveru připojenému do Internetu, tj. na straně ISP. ISP proto povolují připojení ke svému SMTP serveru pouze svým registrovaným uživatelům a v případě zneužití služby napomenout svého zákazníka, případně mu zabránit v přístupu k této službě.



#	IP (reverse dns)	Qty.	Share	Age	Duration
1	202.71.103.34	44651	0.039%	0 hours	13.9 days
2	60.218.99.18	41267	0.036%	0 hours	33.9 days
3	222.73.14.32	38796	0.034%	0 hours	40.2 days
4	58.42.247.66	35520	0.031%	0 hours	21.7 days
5	121.251.254.65	33494	0.030%	0 hours	9.1 days
6	58.221.28.228	32200	0.028%	0 hours	10.6 days
7	202.103.218.87	30216	0.027%	0 hours	54.7 days
8	121.10.104.33	28474	0.025%	0 hours	23.4 days
9	58.47.159.39	25432	0.022%	0 hours	14.3 days
10	202.75.56.212	23204	0.020%	0 hours	13.9 days

Obrázek 4.11: Seznam blokovaných adres SpamCop

Často se stává, že útočník využije pro rozesílání spamů špatně nakonfigurovaný SMTP server, který neautorizuje (např. podle IP adresy) klienty, kterým dovoluje posílat emaily. Takový server s otevřeným spojením (tzv. open-relay server) je nebezpečím pro ostatní SMTP servery, neboť jim často přeposílá spamy. Z tohoto důvodu existují organizace, které aktivně vyhledávají takto nezabezpečené servery tím, že se přes ně zkusí připojit. Pokud se jim to povede, označí server za otevřený a pošlou příslušnému správci výzvu k nápravě. Pokud nezajistí bezpečnost daného serveru, umístí IP adresu serveru na seznam zdrojů spamů, tzv. *blacklist*. Ostatní SMTP servery si udržují kopie tohoto seznamu a blokují komunikaci se servery na blacklistu. Mezi seznamy zdrojů spamu patří například databáze SBL (Spamhaus Block List, www.spamhaus.org) či SCBL (SpamCop Blocking List, www.spamcop.net), viz obrázek 4.11.

Další technikou, která se používá pro filtrování emailů, je tzv. *graylisting*. Princip graylistingu vychází z teze, že spamy posílá automat na velké množství adres, které mohou být zastaralé a neplatné. Proto se tento automat při neúspěšném doručení nepokouší zprávu doručit opakovaně. A toho využívá poštovní server, který při prvním připojení odmítne email od neznámého odesílatele. Odmítnutí je dočasné, SMTP vrátí chybu *temporarily eject*. Zároveň si uloží adresu odesílatele. Pokud email posílá řádný SMTP server, uloží odmítnutou zprávu do fronty nedoručené pošty a po nějaké době se ji pokusí znovu doručit. Při druhé doručení ji cílový SMTP server přijme. Pokud email posílá automat, o doručení se pravděpodobně už nepokouší.

Třetí technikou, kterou zde popíšeme, je kontrola reverzních záznamů v DNS, tj. ověřování, že připojený poštovní klient, který posílá poštu, má v DNS ke své IP adrese přidělenou platnou doménovou adresu. Pokud je však adresa registrovaná v DNS, znamená to, musí

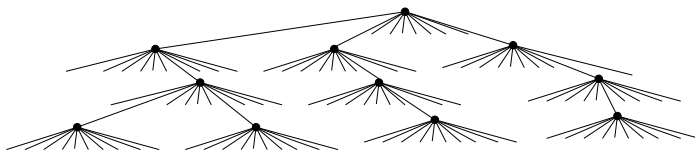
existovat majitel, který každý rok za doménu platí a na jehož totožnost při registraci ověřovala registrační autorita. Spammeri často využívají k odesílání hromadných emailů internetové kavárny a další veřejně přístupná místa, která nemají IP adresy registrované v DNS.

Typy útoku	Způsob ochrany
podvržené dopisy, spamy, hoaxy	kontrola reverzních záznamů, blacklisting, graylisting SpamAssasin, autentizace uživatele u SMTP
viry	heuristická analýza obsahu – MIMEDefang, antiviry
podvržení identity	elektronický podpis PGP či S/MIME
odposlech přenášených zpráv	šifrování PGP či S/MIME, přenos nad SSL/TLS
odposlech hesel	šifrovaný přenos SSL/TLS či autentizace SASL, MD5 u POP3, IMAP

Tabulka 4.9: Ochrana poštovních služeb

Blacklisting, graylisting a kontrola reverzních záznamů jsou techniky, které používají organizace, aby ochránili své zaměstnance před záplavou spamů. Tyto techniky kontrolují IP adresy a odmítnout SMTP spojení, pokud pochází z podezřelé domény. K dokončení přenosu zprávy protokolem SMTP vůbec nedojde, spojení je přerušeno na základě údajů v obálce zprávy (příkazy SMTP). Oproti tomu existují techniky, které na straně poštovního serveru přijmou email, nicméně pak ho zkoumají, zda neobsahuje příznaky spamu, zda neobsahuje v příloze virus, zda nejde o podvodný email snažící se vylákat z uživatele přihlašovací údaje např. k bankovnímu účtu (phishing) apod.

Tato analýza je výpočetně mnohem náročnější. Programy jako SpamAssasin (spamassassin.apache.org/), Clamav (www.clamav.net) či MIMEDefang (www.mimedefang.org) pracují na principu vyhledávání řetězců, kontrole polí v hlavičce emailů či používají statistické metody. Jednou z nich je i bayesovská analýza, která prochází tělo emailu a počítá frekvence výskytů různých znaků. Na základě Bayesovi teorie pravděpodobnosti a databáze "normální" frekvence znaků a slov počítá pravděpodobnost, že email je spam. Samozřejmě, že výpočetní náročnost i čas podobných analýz je větší než u běžného vyhledávání řetězců.



Obrázek 4.12: Exponenciální šíření hoaxů

Kromě spamů a virů se můžeme často setkat také s poplašnými a falešnými řetězovými zprávami, které se řadí do kategorie *hoax*. Jejich hlavní charakteristikou je, že vyzývají příjemce, aby je rozeslal na dalších n adres. Zdůvodňují to obecným prospěchem, vidinou nenačalého štěstí, záchranou lidského života či varováním před nebezpečím. Často čtou adresáti tyto zprávy zrychleně a pokládají je za pravdivé nebo přinejmenším neškodné. V dobré víře je rozešlou na pět, deset či patnáct adres, viz 4.12. A to je přesně to, o co jde autorům – záplavové šíření informací. Je to vlastně jako pyramidová hra rostoucí exponenciální řadou

n^k , kde n je počet kopií, které každý účastník řetězové zprávy pošle a k je pořadí účastníků v řetězu, tj. hloubka stromu. Při počtu deseti kopií je třetími účastníky v řadě rozesláno $10^3 = 1000$ zpráv a pátými účastníky už sto tisíc zpráv, což rozhodně není málo.

Na co nezapomenout při psaní emailů

Elektronická pošta patří i přes svou textovou podobu mezi základní komunikační prostředky a jako taková slouží k budování mezilidských vztahů. Její použití je snadné, rychlé a nic nestojí. To může často vést k naivitě či lehkomyšlnosti při použití této služby. Přestože důsledky nemusíme vidět, nevhodně napsaný či přeposlaný email může způsobit velké problémy. Na závěr kapitoly o emailch bych chtěl shrnout pár bodů, nad kterými by měl každý uživatel popřemýšlet, viz tab. 4.10.

Elektronická pošta není určena k přenášení fotek z dovolené. Můžete snadno přeplnit schránku adresáta. Na začátku emailu je vhodné pozdravit a na konci se podepsat. Dvakrát přemýšlejte, než odeslete email. Je to nevratná událost. Neočekávejte, že si adresát přečte váš email tentýž den. Nevkládejte grafiku do emailů. Adresát může číst emaily pomocí textového klienta. V zahraničí vaši přátelé příliš neocení českou diakritiku. Neodpovídejte na nevyžádanou poštu, pokud nechcete povzbudit spammera k ještě větší aktivitě. Pokud dostanete email, dejte to vědět odesílateli. Většinou nepozná, jestli jste email nedostali a nebo s ním jen nemluvíte. Pokud vám pošta nechodí, oznamte to administrátorovi. Raději ústně. Jemu pošta pravděpodobně chodí a o vašich problémech možná nic netuší. Pokud chcete někomu emailem vynadat, počkejte s odesláním na druhý den. Druhý den ten email smažte. Zdvouřlost v emailch není směšná a určitě potěší.

Tabulka 4.10: Jak psát emailové zprávy.

4.3 Elektronické konference (mailing lists)

Elektronická pošta se používá nejen pro přenos zpráv mezi dvěma komunikujícími účastníky, ale často je zapotřebí rozeslat zprávu více příjemcům. Pokud jde o pracovní skupinu, je nevhodné, aby si seznam kontaktů udržoval každý člen této skupiny a prováděl veškeré aktualizace. Z tohoto důvodu vznikly techniky pro vytváření elektronických konferencí. Podobně jako elektronická pošta využívají elektronické konference protokol SMTP. Při posílání zprávy na adresu konference jsou kopie zprávy rozeslány všem účastníkům konference. Přenos je pro účastníka konference naprosto transparentní, tzn. nemusí speciálně upravovat odesílané emaily či využívat jiné služby.

Centrální rozesílání elektronické pošty více uživatelům, tj. elektronickou konferencí, lze zajistit buď konfigurací aliasů na SMTP serveru (mailing list) nebo použitím aplikace, která řídí rozesílání zpráv přihlášeným účastníkům konference – tzv. listserver.

Jednodušší variant je použití aliasů na SMTP serveru. Alias je vlastně speciální emailová adresa, která odkazuje na emailové adresy účastníků konference. Příkladem může být třeba alias `postmaster@netlab.fit.vutbr.cz`. Zpráva odeslaná na tuto adresu se přepoše na všechny adresy v seznamu `aliases` – viz následující příklad.

```
root: cejkar
postmaster: matousp, root, novotny@seznam.cz
```


Emailová zpráva směřovaná na adresu `postmaster@netlab.fit.vutbr.cz` se uloží do lokálních schránek `matousp` a `cejkar`. Jedna kopie se také přepoše na adresu `novotny@seznam.cz`. Všimněte si možnosti rekurzivního zpracování aliasu "root" v položce "postmaster".

Použití souboru `aliases` na SMTP serveru je vhodné pro krátké seznamy, které se příliš často nemění. Seznam `aliases` ručně vytvoří správce serveru. Když odesílatel pošle zprávu na adresu aliasu, zpráva se na SMTP serveru zkopíruje a přepoše na emailové adresy odpovídající aliasu. Nevýhodou použití aliasů pro elektronické konference je nutnost manuálního vytvoření seznamu a jeho aktualizace. Navíc může být alias zneužit k posílání spamů, protože je tato adresa přístupná všem. Proto je vhodnější najít aplikaci, která nebude jenom přeposílat emailové zprávy všem adresátům ve skupině, ale zároveň kontrolovat, kdo zprávu posílá. K tomu se používají **elektronické konference typu listserver**.

Elektronická konference typu listserver pracuje, podobně jako soubor `aliases`, na principy rozesílání zpráv poslaných na speciální emailovou adresu skupiny všem účastníkům skupiny (konference). Využívá přenosový protokol SMTP. Narozdíl od klasické elektronické pošty či seznamů typu `aliases` rozšiřuje hlavičku v těle zprávy o pole, která označují název konference a další informace o příspěvku do konference, viz tabulka 4.11.

Hlavička	Význam – příklad
List-Id	identifikace konference "Informace FIT <fitinfo.fit.vutbr.cz>"
List-Help	nápověda ke konferenci (URL) "<mailto:fitinfo-request@fit.vutbr.cz?subject=help>"
List-Subscribe	způsob registrace do konference (URL) "<mailto:fitinfo-request@fit.vutbr.cz?subject=subscribe>"
List-Unsubscribe	zrušení členství v konferenci (URL) "<mailto:fitinfo-request@fit.vutbr.cz?subject=unsubscribe>"
List-Post	metoda pro posílání příspěvků (URL) "<mailto:fitinfo@fit.vutbr.cz>"
List-Owner	kontakt na administrátora (URL)
List-Archive	přístup k archivovaným příspěvkům (URL) "<http://www.fit.vutbr.cz/mailman/private/fitinfo>"

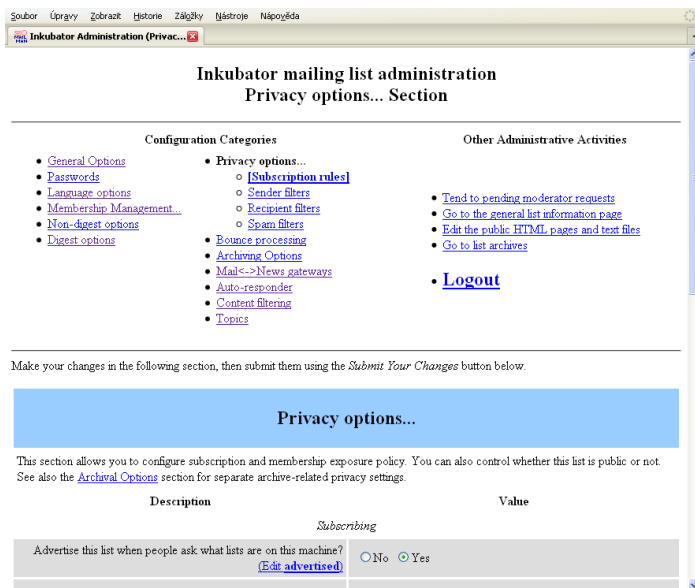
Tabulka 4.11: Hlavičky příspěvků elektronické konference podle RFC 2369 a 2919 [19, 20]

Tyto hlavičky přidává server pro práci s elektronickými konferencemi. Aplikace umožňuje vytvářet, rušit a modifikovat elektronické konference, zapisovat tam uživatele, přiřazovat jim práva, nastavit informace o konferenci, správci apod. Přihlášení do konference je možné příkazem `SUBSCRIBE` (`SUB`) poslaným v předmětu emailové zprávy na adresu konference. Server si uloží adresu účastníka podle jeho zpáteční adresy v hlavičce emailu. Zrušení členství je možné příkazem `UNSUBSCRIBE` (`UNSUB`).

Konference mohou být veřejné (kdokoliv se může přihlásit do konference) či uzavřené (pouze pro určitou skupinu účastníků). Registraci účastníků obvykle potvrzuje administrátor (správce) konference. Zaslání příspěvků do konference opět může být podle nastavení konference umožněno pouze zapsaným účastníkům. Může být však povoleno i pro libovolného uživatele. Existují konference moderované, kdy rozeslání příspěvku kontroluje a odsouhlasuje správce konference, či nemoderované, kdy se příspěvek automaticky rozešle všem zaregistrovaným účastníkům.

Administraci konference vykonává správce obvykle prostřednictvím webového rozhraní

konference – viz obrázek 4.13. Mezi používané aplikace pro vytváření elektronických konferencí typu listserver slouží například mailman či starší aplikace majordomo a listserver.



Obrázek 4.13: Konfigurace elektronické konference – mailman

Příklad hlavičky zprávy, která byla odeslána do elektronické konference a potom přeposlána aplikací účastníkům, můžeme vidět na následujícím výpisu:

```
From eysselt@fit.vutbr.cz Tue Oct 18 14:36:07 2005
Return-Path: <fitinfo-bounces@fit.vutbr.cz>
Received: from tereza.fit.vutbr.cz (tereza.fit.vutbr.cz [147.229.9.22]) by
kazi.fit.vutbr.cz (envelope-from fitinfo-bounces@fit.vutbr.cz) (8.13.5/8.13.5)
with ESMTp id j9ICZwTo91461;
...
Message-ID: <01f401c5d3e0$7c6c8930$340ce593@PCEYSSLT>
From: "Milos Eysselt" <eysselt@fit.vutbr.cz>
To: <fitinfo@fit.vutbr.cz>
Date: Tue, 18 Oct 2005 14:35:56 +0200
MIME-Version: 1.0
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2900.2180
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.2180
X-Scanned-By: MIMEDefang 2.49 on 147.229.8.12
X-Scanned-By: MIMEDefang 2.49 on 147.229.8.12
```

```

Subject: [Fitinfo] zadani DIP(+SEP) pro MIT k obhajobe v cervnu 2007 (1)
X-BeenThere: fitinfo@fit.vutbr.cz
X-Mailman-Version: 2.1.6
Precedence: list
List-Id: Informace FIT <fitinfo.fit.vutbr.cz>
List-Unsubscribe: <http://www.fit.vutbr.cz/mailman/listinfo/fitinfo>,
  <mailto:fitinfo-request@fit.vutbr.cz?subject=unsubscribe>
List-Archive: <http://www.fit.vutbr.cz/mailman/private/fitinfo>
List-Post: <mailto:fitinfo@fit.vutbr.cz>
List-Help: <mailto:fitinfo-request@fit.vutbr.cz?subject=help>
List-Subscribe: <http://www.fit.vutbr.cz/mailman/listinfo/fitinfo>,
  <mailto:fitinfo-request@fit.vutbr.cz?subject=subscribe>
Content-Type: text/plain; charset="iso-8859-2"
Sender: fitinfo-bounces@fit.vutbr.cz
Errors-To: fitinfo-bounces@fit.vutbr.cz
X-Spam-Score: 0.328 ( ) AWL
Content-Transfer-Encoding: 8bit
X-MIME-Autoconverted: from quoted-printable to 8bit by kazi.fit.vutbr.cz

Ve Ct 2005-10-20 konci registrace/prihlasovani se na PI3+DPI v
EI-MGR-5. ...

```

Z hlavičky je možné vyčíst, že se jedná o příspěvek do konference FITINFO s emailovou adresou `fitinfo@fit.vutbr.cz`. Podle políčka **Precedence: list** můžeme určit, že jde o konference a nikoliv o obvyklý email. Následují pak položky identifikující konferenci, způsob přihlášení a odhlášení, posílání příspěvků apod. [19, 20]. Všimněte si, že většina hodnot těchto položek je uvedena ve formátu URL.

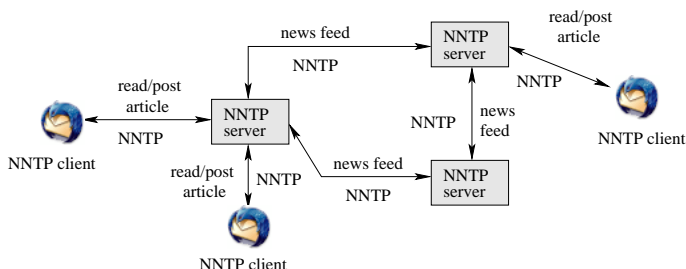
4.4 Diskuzní skupiny Usenet (NetNews)

Diskuzní skupiny Usenet, známé též pod názvem News nebo NetNews, patří také do oblasti poštovních aplikací. I když se použitím i způsobem komunikace podobají elektronické poště, jedná se o zcela odlišný způsob přenosu a zpracování dat. Místo pojmu "zpráva" se pro přenášené datové jednotky používá název *příspěvek* (article). Příspěvky jsou umístěny na serveru, odkud si je klienti pomocí protokolu NNTP (Network News Transfer Protocol) [21, 22]. Příkladem serveru je aplikace INN (InterNetNews). Klientem může být speciální aplikace pro čtení diskuzí (např. `tin`, `rtin`) nebo běžný poštovní klient jako např. `Thunderbird`, `Outlook Express`. Na serveru mohou být umístěny různé diskuzní skupiny. Čtenář, který chce sledovat danou diskuzní skupinu, se do ní přihlásí, načítá si uložené příspěvky, případně zasílá nové příspěvky. Diskuzní skupiny je obecně otevřena všem uživatelům. Příspěvky se neposílají konkrétnímu účastníkovi diskuze, ale jsou adresovány přímo na diskuzní skupinu. Adresy diskuzních skupin jsou hierarchicky uspořádány a připomínají adresy v DNS. Příkladem diskuzní skupiny může být například diskuzní skupina uložená na serveru `news.fit.vutbr.cz` k předmětu ISA na Fakultě informačních technologií VUT s adresou `vutbr.fit.courses.isa`.

4.4.1 Architektura systém Usenet

Na obrázku 4.14 je naznačena architektura systému Usenet. Tvoří ji Usenet servery (též NNTP servery), které uchovávají diskuzní skupiny včetně jednotlivých příspěvků. Diskuzní skupiny

nemusí být lokální pouze na určitém serveru. Pokud jde o veřejnou diskuzní skupiny, lze ji sdílet s ostatními skupinami. Příspěvky jsou mezi servery přenášeny protokolem NNTP².



Obrázek 4.14: Architektura systému Usenet

Pokud se chce uživatel připojit do diskuze, přihlásí se protokolem NNTP ke konkrétní diskuzní skupině (příkaz GROUP). Zde může číst jednotlivé příspěvky (příkaz ARTICLE), reagovat na ně, případně zasílat vlastní příspěvky (příkaz POST). Příspěvek je uložen na lokálním serveru Usenet, ke kterému je uživatel připojen. Pokud je daná diskuzní skupina rozšířena na více serverů, probíhá mezi servery synchronizace. Při synchronizaci jsou přenášeny z jednoho serveru nové články na druhý server (tzv. news feed). Toto je rozdíl oproti elektronické poště, kdy se zpráva směřuje mezi SMTP servery podle záznamů MX k cíli. V systému USENET se příspěvky ukládají lokálně. Pokud je chtějí číst uživatelé na jiném server, musí se dohodnout správci obou serverů a synchronizovat požadované diskuzní skupiny. Další rozdíly mezi elektronickou poštou a systémem Usenet jsou shrnuty v tab. 4.12.

	Elektronická pošta	Systém Usenet
Zasílání příspěvků ¹	protokol SMTP	protokol NNTP
Čtení příspěvků	protokoly POP3, IMAP	protokol NNTP
Adresát příspěvků	uživatel	diskuzní skupina
Adresování	emailová adresa	adresa diskuzní skupiny
Uložení příspěvků	poštovní schránka adresáta	uložení na serveru
Platnost příspěvků	zprávu musí zrušit adresát	automatická expirace na serveru
Přenos příspěvků	zasílání mezi SMTP servery	peer-to-peer spojení mezi servery Usenet
Použití	osobní korespondence	zájmová sdružení
Hromadná korespondence	kopie všem uživatelům	pouze jeden příspěvek do konference

¹ u elektronické pošty se používá termín zpráva (message), v Usenetu termín příspěvek (article)

Tabulka 4.12: Porovnání elektronické pošty a diskuzních skupin Usenet

Jak vidíme z výše uvedené tabulky, použití i architektura systému Usenet je odlišná od klasické elektronické pošty. Zaslané příspěvky se ukládají na serveru Usenet a mohou se šířit i na další servery v síti Usenet. Z globálního pohledu se jedná o distribuovanou síť serverů s diskuzními skupinami, které si mezi sebou vyměňují zaslané příspěvky. Protože denní přírůs-

²Dříve se pro přenos používal protokol UUCP (Unix to Unix copy) [23]. Dnes je spíše historií.

tek příspěvků všech diskuzních skupiny se pohybuje v desítkách gigabytech, není vhodné šířit všechny diskuzní skupiny na všechny servery Usenet. Obvykle se sdílení diskuzních skupin řídí požadavky zákazníků/uživatelů. Pokud chtějí uživatelé sledovat určitou globální diskuzní skupiny, požádají správce lokálního serveru Usenet o zpřístupnění. Lokální server Usenet bude odebírat požadovanou diskuzní skupiny ze vzdáleného serveru. Jinou možností je, že se uživatel připojí prostřednictvím svého klienta přímo na vzdálený server Usenet a tam bude sledovat danou diskuzní skupinu. Podobně jako u elektronických konferencí mohou být diskuzní skupiny moderované či nemoderované, lokální či veřejné. Protože původní standard protokolu NNTP neumožňoval autentizaci uživatelů, byl přístup k lokálním (neveřejným) diskuzním skupinám kontrolován jednoduchými způsoby, např. přes IP adresu připojeného uživatele. Rozšíření RFC 4643 [22] přidalo možnost autentizovat uživatele příkazem AUTHINFO, a tak povolit přístup k serveru Usenet pouze lokálním uživatelům.

4.4.2 Formát příspěvků a jejich organizace

Formát příspěvků Usenet vychází ze standardu zpráv elektronické pošty RFC 822 [2] a je definován standardem RFC 1036 [24]. Standardní příspěvek v diskuzích Usenet se skládá z několika řádek hlavičky, které odděluje od těla příspěvku prázdný řádek. Řádky hlavičky mají formát <field_name>: <value>. Mezi povinné položky v hlavičce patří From, Date, Newsgroup, Subject, Message-ID a Path. Volitelné položky v hlavičce jsou například Follow up-To, Sender, References, Distribution, Keywords, Summary, Approved či Xref, viz následující ukázka.

```
Path: news.fit.vutbr.cz!not-for-mail
From: Cejka Rudolf <cejkar@fit.vutbr.cz>
Newsgroups: vutbr.fit.courses.isa
Subject: Re: Projekt housenka - Otazka ohladom zadanych parametrov.
Date: Wed, 15 Oct 2008 09:51:39 +0000 (UTC)
Organization: Brno University of Technology, FIT
Lines: 47
Distribution: local
Message-ID: <gd4efb$1g8c$1@fit.vutbr.cz>
References: <gco1o7$2t2p$5@fit.vutbr.cz> <gcv3rp$2dv3$1@fit.vutbr.cz>
           <op.ui00usx23bi37z@tomas-notebook.kn.vutbr.cz>
NNTP-Posting-Host: kazi.fit.vutbr.cz
X-Trace: hodka.fit.vutbr.cz 1224064299 49420 147.229.8.12 (15
        Oct 2008 09:51:39 GMT)
X-Complaints-To: postmaster@fit.vutbr.cz
NNTP-Posting-Date: Wed, 15 Oct 2008 09:51:39 +0000 (UTC)
User-Agent: tin/1.8.3-20070201 ("Scotasay") (UNIX) (FreeBSD/7.0-STABLE (i386))
Xref: news.fit.vutbr.cz vutbr.fit.courses.isa:1464

Tomas Dzurnak <xdzurn00@stud.fit.vutbr.cz> wrote:
> 1. nie je mi jasne ako moze ktorykolvek clanok husenky
vyuzit parameter
> LH ?
```

Výše uvedený příspěvek byl zaslán z počítače kazi.fit.vutbr.cz, programem tin na systému FreeBSD. Odesílatelem je uživatel s adresou cejkar@fit.vutbr.cz, který patří k organizaci "Brno University of Technology, FIT". Příspěvek byl zaslán do diskuzní skupiny vutbr.fit.courses.isa, což je lokální skupina umístěná na serveru news.fit.vutbr.cz. Ze zprávy můžeme dále vyčíst pořadí článku v diskuzi na serveru news.fit.vutbr.cz (1464) a

identifikaci příspěvku (gd4efb\$1g8c\$1 @fit.vutbr.cz). Článek se odkazuje na další tři články uvedené v položce **References**. Přehled hlaviček příspěvků NNTP je uveden v tabulce 4.13.

Název hlavičky	Popis
From:	emailová adresa odesílatele příspěvku
Date:	datum a čas doručení příspěvku na serveru Usenet
Newsgroups:	název diskuzní skupiny (skupin), kam patří příspěvek
Subject:	předmět příspěvku
Message-ID:	identifikace příspěvku ve formátu <code>uniqueID@domain</code>
Path:	seznam Usenet serverů, přes které byl příspěvek poslán
References:	seznam Message-ID příspěvků, na něž článek reaguje
Distribution:	rozsah šíření příspěvku
Organization:	krátký popis organizace, kam náleží odesílatel
Approved:	adresa administrátora moderované skupiny
Lines:	počet řádků obsahu (těla) příspěvku
XRef:	lokální informace o diskuzní skupině a pořadí příspěvku

Tabulka 4.13: Příklady hlaviček příspěvků Usenet podle RFC 1036 [24]

4.4.3 Přenosový protokol NNTP (Network News Transfer Protocol)

Pro přenos příspěvků v systému Usenet, čtení a zaslání zpráv od uživatelů slouží protokol NNTP. Protokol NNTP byl definován v roce 1986 standardem RFC 977 [25], dnes se používá aktualizace RFC 3977 [21]. Jedná se o aplikační protokol využívající služby transportního protokolu TCP. NNTP pracuje na portu 119. Slouží k distribuci příspěvků, vyhledávání, čtení a zaslání do diskuzních skupin Usenet. Diskuzní skupiny, které obsahují jednotlivé příspěvky, jsou umístěny na serverech Usenet (NNTP serverech). Příspěvky jsou v rámci skupin indexovány pro vzájemné odkazování a výměnu mezi dalšími servery v systému Usenet. Každý článek má také svou dobu životnosti (expirace) danou nastavením lokálního serveru Usenet. Po uplynutí této doby je článek vymazán.

NNTP se používá ve dvou režimech. První se nazývá *režim čtení* (reader mode), v kterém klient stahuje příspěvky z databáze serveru uživateli, případně zasílá články od uživatele. Druhý režim činnosti se nazývá *režim přenosu* (transit mode, peering). V tomto režimu dochází k přenosu velkého množství příspěvků mezi servery. Přestože oba režimy využívají stejný přenosový protokol, někdy se tyto dva odlišné typy komunikujících serverů implementují zvlášť. K přepnutí do režimu přenosu slouží příkazy **IHAVE**, zpět příkaz **MODE READER**, viz tabulka 4.14.

Pokud chce uživatel přečíst nějaký příspěvek, připojí se pomocí klient NNTP k serveru a přeneše si k sobě daný příspěvek. Narozdíl od elektronické pošty, která je uložena do poštovních schránky uživatele po svém doručení, zde se musí uživatel sám přihlásit do diskuzní skupiny a aktivně si načíst příspěvky ze serveru. Je tu patrná analogie s klasickými novinami (News). Mezi základní operace NNTP serveru patří:

- správa a řízení spojení (příkazy **CAPABILITIES**, **MODE READER**, **QUIT**)
- autentizace uživatele dle RFC 4643 [22] (**AUTHINFO**)
- výběr diskuzní skupiny a příspěvku (**GROUP**, **LISTGROUP**, **LIST**, **LAST**, **NEXT**)

- načtení příspěvku (ARTICLE, HEAD, BODY, STAT)
- zaslání příspěvku (POST, IHAVE)
- informační příkazy (DATE, HELP, NEWSGROUPS, NEWNEWS)

Následující příklad ukazujeme komunikaci pomocí protokolu NNTP. V příkladu se uživatel připojí anonymně k serveru Usenet `news.fit.vutbr.cz`, příkazem `GROUP` se připojí k diskuzní skupině `vutbr.fit.courses.isa`, která obsahuje 60 příspěvků s pořadovými čísly 132 až 191. Uživatel si vybere poslední příspěvek a příkazem `ARTICLE` ho načte ze serveru.

```
telnet news.fit.vutbr.cz 119
Trying 147.229.9.11...
Connected to boco.fee.vutbr.cz.
Escape character is '^]'.
200 news.fit.vutbr.cz InterNetNews NNRP server INN 2.2.2 13-Dec-1999 ready (posting ok).
GROUP vutbr.fit.courses.isa
211 60 132 191 vutbr.fit.courses.isa
STAT 132
223 132 <dhg04o$1sak$1@boco.fee.vutbr.cz> status
ARTICLE 191
220 191 <dj3bpk$1nfr$1@boco.fee.vutbr.cz> article
Path: news.fit.vutbr.cz!not-for-mail
From: =?ISO-8859-2?Q?Jarom=EDr_Smr=E8ek?= <xsmrce00@stud.fit.vutbr.cz>
Newsgroups: vutbr.fit.courses.isa
Subject: Re: =?ISO-8859-2?Q?Probl=E9m_se_sign=Eily?=
...
```

Ano norma zadána nebyla, takže gcc bere implicitně normu C90, což je poměrně restriktivní...

V další části uživatel zapíše na server příkazem `POST` nový příspěvek. Musí samozřejmě vložit hlavičku i tělo příspěvku oddělené prázdným řádkem. Po úspěšném zaslání se uživatel odhlásí z diskuzní skupiny.

```
POST
340 Ok
From: matousep@fit.vutbr.cz
Subject: testik news
Newsgroups: vutbr.fit.courses.isa
Lines: 5
```

```
Toto je kratky prispevek vytvoreny prikazy NNTP.
Petr Matousek
.
240 Article posted
quit
205 .
```

Ve výše uvedených příkladech jsme viděli příkazy pro komunikaci mezi klientem a serverem. Servery Usenet mohou také komunikovat mezi sebou a vyměňovat si příspěvky pomocí příkazů (`NEWSGROUPS`, `NEWNEWS`, `IHAVE`). V tabulce 4.14 je seznam základních příkazů protokolu NNTP.

Přenos dat mezi servery Usenet pomocí příkazu `IHAVE` není vhodný. Pomocí příkazu `IHAVE` nabídne klient (první server) druhému serveru nový příspěvek. Protože nelze současně zaslat

Příkaz	Popis
CAPABILITIES	žádost klienta o nastavení serveru
MODE READER	přepnutí do režimu READER (čtení příspěvků)
QUIT	ukončení spojení
AUTHINFO	autentizace uživatele
GROUP, LISTGROUP	výběr diskuzní skupiny Usenet
LIST	zaslání informací o skupinách klientovi
LAST	posun ukazatele aktuálního příspěvku zpět
NEXT	posun ukazatele aktuálního příspěvku dopředu
ARTICLE	vybere a načte zadaný příspěvek
BODY	vybere a načte obsah zadaného příspěvku
HEAD	vybere a načte hlavičku zadaného příspěvku
STAT	vypíše informace o aktuálním příspěvku
POST	zaslání jednotlivého příspěvku do diskuzní skupiny
IHAVE, CHECK, TAKETHIS	přenos příspěvků mezi servery
NEWSGROUP	seznam nových skupin vytvořených na daném serveru
NEWNEWS	seznam nových příspěvků vytvořených na daném serveru
HELP	seznam příkazů implementovaných na serveru

Tabulka 4.14: Přehled příkazů protokolu NNTP

více příkazů **IHAVE**, musí klient vyčkat na odpověď od serveru, zda má o příspěvek zájem. teprve poté pošle tento příspěvek. Standard RFC 4644 [26] navrhuje efektivnější způsob přenosu – asynchronní přenos (streaming). Příkaz **IHAVE** je nahrazen dvěma příkazy **CHECK** a **TAKETHIS**, které jsou nezávislé a dovolují souběžné zpracování. Klient prostřednictvím paralelních příkazů **CHECK** pošle serveru seznam článků, které má k dispozici. Pokud o ně server projeví zájem, klient je odešle příkazem **TAKETHIS**.

Shrnutí

Poštovní služby, jak jsme si ukázali v této kapitole, tvoří důležitou oblast elektronické komunikace. Zahrnují elektronickou poštu, elektronické konference a diskuzní skupiny Usenet. Architektura elektronické pošty zahrnuje klienty MUA (Message User Agents) a servery MTA (Message Transfer Agents), které přenášejí zprávy. Přenosový protokol SMTP byl původně sedmibitový a i dnes můžeme najít servery, které podporují pouze sedmibitová data. Osmibitová data je nutné v emailové zprávě speciálně označit a zakódovat do sedmi bitů. Používá se proto standard MIME a kódování "Quoted-printable" či "base64".

Ke směrování elektronické pošty se používají záznamy typu MX v DNS, které pro danou doménu definují poštovní serveru, kam se má pošta doručit. Pro přístup k elektronické poště se používají protokoly POP3 a IMAP, které kromě načtení doručených zpráv umožňují také pracovat s uloženými dopisy, přesouvat je do dalších schránek, vyhledávat v obsahu a podobně. Pro bezpečnou práci s emailem je potřeba zajistit autentizaci odesílatele a zabezpečit obsah před odposlechem. Mezi techniky autentizace emailových zpráv patří PGP či S/MIME, které využívají asymetrickou kryptografii s veřejným klíčem. Tyto techniky umožňují zprávy také šifrovat. Pro šifrovaný přenos lze použít i rozšíření přenosových protokolů SMTP, POP3 a IMAP o podporu SASL. Pokud nejsou tyto techniky k dispozici, lze pomocí speciálních aplikací (stunnel, ssh) vytvořit bezpečný tunel, kterým přenášíme nezabezpečené zprávy.

Elektronické konference slouží k rozesílání emailových zpráv více uživatelům. Mohou být implementovány jako seznam emailových adres na poštovním serveru, podle kterého se kopíruje došlý email pro účastníky konference. Pro uzavřené či moderované konference je vhodné použít aplikace typu listserver. Elektronické konference pracují podobně jako elektronická pošta nad protokolem SMTP, pro své potřeby pouze rozšiřují hlavičky zpráv o identifikaci konference, kontakt na administrátora a další.

Diskuzní skupiny Usenet jsou poštovní službou, která je nezávislá na elektronické poště, poštovních serverech a protokolu SMTP. Usenet tvoří síť vzájemně komunikujících serverů, které si mezi sebou pomocí protokolu NNTP vyměňují příspěvky zaslané do diskuzních skupin. Diskuzní skupiny mohou být lokální či veřejné, podobně jako u elektronické konference lze skupiny moderovat či je nechat otevřené. Formát příspěvku vychází z formátu zprávy elektronické pošty, pole v hlavičce jsou však odlišná. Pro čtení či zaslání příspěvků se používá protokol NNTP. Obvykle můžeme k diskuzním skupinám přistupovat stejným programem (klientem) jako k elektronické poště, například programy Outlook, Thunderbird, Pine a podobně.

Přestože je elektronické poštovní služby konkurují tradičním poštovním službám, určité je nenahradí. Elektronické služby mají přednost v rychlé, bezplatné a snadné komunikaci. Jsou však vhodnější pro menší zásilky informačního charakteru. Pro úřední či zdvořilostní komunikaci budou klasické poštovní služby stále nezastupitelné.

Otázky

1. Popište doručování elektronické pošty.
2. Proč se používá MIME standard? Uveďte základní typy a podtypy.
3. Popište činnost protokolu SMTP. Jaké činnosti SMTP server vykonává?
4. Porovnejte protokoly POP3 a IMAP4. Který je vhodnější pro off-line zpracování?
5. Uveďte příklad klienta a serveru pro poštovní služby.
6. Jakým zajistit, aby poštovní schránka byla přístupná z více počítačů?
7. Popište architekturu a použití diskuzních skupin Usenet.
8. V čem se systém elektronické pošty a systém Usenet?
9. Umět přečíst a porozumět hlavičce zprávy podle RFC 822.
10. Jaké jsou obvyklé útoky na poštovní služby? Jak lze služby zabezpečit?
11. Porovnejte PGP a S/MIME. Jak můžeme šifrovat zprávu pomocí PGP?

Doporučená literatura a standardy

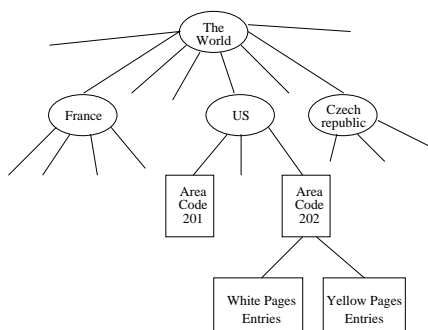
- [1] J.B.Postel. *Simple Mail Transfer Protocol*. RFC 821, August 1982.
- [2] D.H.Crocker. *Standard for the format of ARPA Internet text messages*. RFC 822, August 1982.
- [3] J.Klensin. *Simple Mail Transfer Protocol*. RFC 2821, April 2001.
- [4] P.Resnick. *Internet Message Format*. RFC 2822, April 2001.
- [5] N.Freed and N.Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*. RFC 2045, November 1996.
- [6] N.Freed and N.Borenstein. *Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types*. RFC 2046, November 1996.
- [7] R.Siemkowski and A.Melnikov. *SMTP Service Extension for Authentication*. RFC 4954, July 2007.
- [8] J.Myers and M.Rose. *Post Office Protocol — Version 3*. RFC 1939, May 1996.
- [9] M.Crispin. *Internet Message Access Protocol — Version 4rev1*. RFC 3501, March 2003.
- [10] A.Melnikov and K.Zeilenga. *Simple Authentication and Security Layer (SASL)*. RFC 4422, June 2006.
- [11] R.Siemkowski and A.Menon-Sen. *The Post Office Protocol (POP3) Simple Authentication and Security Layer (SASL) Authentication Mechanism*. RFC 5034, July 2007.
- [12] K.Zeilenga. *The PLAIN Simple Authentication and Security Layer (SASL) Mechanism*. RFC 4616, August 2006.
- [13] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.
- [14] J.Callas, L.Donnerhacke, H.Finney, and R.Thayer. *OpenPGP Message Format*. RFC 2440, November 1998.
- [15] B.Ramdsdell. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling*. RFC 3850, July 2004.
- [16] B.Ramdsdell. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification*. RFC 3851, July 2004.
- [17] S.Blake-Wilson, M.Nystrom, D.Hopwood, J.Mikkelsen, and T.Wright. *Transport Layer Security (TLS) Extensions*. RFC 4366, April 2006.
- [18] T.Dierks and E.Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*. RFC 4346, April 2006.
- [19] G.Neufeld and J.Baer. *The Use of URLs as Meta-Syntax for Core Mail List Commands and their Transport through Message Header Fields*. RFC 2369, July 1998.

- [20] R.Chandhok and G.Wenger. *List-Id: A Structured Field and Namespace for the Identification of Mailing Lists*. RFC 2919, March 2001.
- [21] C.Feather. *Network News Transfer Protocol (NNTP)*. RFC 3977, October 2006.
- [22] J.Vinocur and K.Murchison. *Network News Transfer Protocol (NNTP) Extension for Authentication*. RFC 4643, October 2006.
- [23] M.R.Horton. *UUCP mail interchange format standard*. RFC 976, February 1986.
- [24] M.Horton and R.Adams. *Standard for Interchange of USENET Messages*. RFC 1036, December 1987.
- [25] B.Kantor and P.Lapsley. *Network News Transfer Protocol. A Proposed Standard for the Stream-Based Transmission of News*. RFC 977, February 1986.
- [26] J.Vinocur and K.Murchison. *Network News Transfer Protocol (NNTP). Extension for Streaming Feeds*. RFC 4644, October 2006.

Kapitola 5

Adresářové služby

Adresářové služby (Directory Services) zajišťují přístup k datům typu adresáře (telefonního seznamu). Slovo adresář zde má jiný význam než v kontextu souborového systému a diskových svazků. Adresář zde znamená specializovanou databázi osobních jmen, telefonních čísel, emailových adres a dalších informací. Historicky se adresáře či telefonní seznamy vydávaly v tištěné podobě jako několikadílné svazky. Některé adresáře byly určeny pro vyhledání klasických telefonních čísel osob (White Pages), jiné se specializovali na seznamy firem (Yellow Pages). S rozvojem počítačů v 80. letech 20. století se začaly objevovat první síťové služby, které využívaly Internet k vytváření globálního telefonního seznamu, viz obr. 5.1. Tyto po-



Obrázek 5.1: Struktura globálního adresáře (telefonního seznamu)

kusy byly postaveny na standardech ITU-T X.500 či jiných typech databází, např. CCSO nameserver se jménem `ph`. Na konci 90. let 20. století byl vytvořen nový standard IETF Lightweight Directory Access Protocol (LDAP), který zcela nahradil systém X.500. Mezi další rozšířené adresářové služby patří Active Directory, IBM Directory Service, Novell eDirectory, Oracle Internet Directory a další. V této kapitole se podívám na původní standard X.500 a pak podrobněji na dnes běžně používanou službu LDAP.

Architektura LDAP, která vychází z definice X.500, byla vytvořena v roce 1997 [1]. Standard byl postupně aktualizován, dnes jsou aktuální doporučení RFC 4510 [2] a RFC 4511 [3].

Základním cílem adresářové služby LDAP bylo vytvořit distribuovanou adresářovou službu na Internetu, která bude lehkou rozšiřitelná. Protože na mnoha místech už existovali lokální služby typu adresář, sloužil standard LDAP k propojení těchto lokálních adresářů a vytvoření globálního adresáře pomocí protokolu LDAP. Podobný cíl si kladl i standard X.500, nicméně kvůli náročnosti implementace nebyla adresářová služba X.500 ve větším měřítku realizována. Podobně jako DNS i systém LDAP si kladl za cíl umožnit uživatelům rovný přístup k datům, tzn. neupřednostňovat žádnou skupinu. Pro identifikaci dat a rychlý přístup k záznamům byl použit hierarchický přístup podobný modelu DNS či unixovému souborovému systému.

5.1 Architektura adresářových služeb

Standard X.500 definuje adresář jako "soubor otevřených systémů, které spolupracují za účelem uchování logické databáze informací obsahující množinu objektů v reálném světě" [4]. Uživatelé adresáře (lidé či aplikace) mohou číst a modifikovat informace nebo jejich části podle definovaných přístupových práv.

Informace obsažená v adresáři se souborně nazývá *Directory Information Base (DIB)*. Skládá se ze záznamů (entries), které obsahují množinu informací o objektu (attributes). Každý atribut má typ a hodnotu. Typy atributů konkrétního záznamu závisí na třídě objektu, pro kterou byl záznam vytvořen a která popisuje jeho strukturu.

Záznamy v databázi DIB jsou uspořádány do kořenového stromu *Directory Information Tree (DIT)*, v němž vrcholy reprezentují záznamy. Záznamy ve vyšší části stromu (blíže ke kořeni), reprezentují objekty jako jsou například země či organizace, záznamy v nižší části popisují osoby či aplikační procesy. Každý záznam má jednoznačný identifikátor *distinguished name DN* (význačné jméno), které identifikuje záznam. DN záznamu se skládá z DN nadřazeného záznamu a hodnoty atributu záznamu. DIB obsahuje dvě třídy záznamů – (1) informace o uživateli (model uživatelských informací) a (2) administrativní a operační informace (model administr. a operačních informací).

Adresářové služby se liší od klasických (např. relačních) databází v přístupu k datům a v požadavcích na funkčnost. To je dáno zejména charakterem uložených dat (stabilní data, velký počet záznamů) a nejčastějšími operacemi nad daty (vyhledávání).

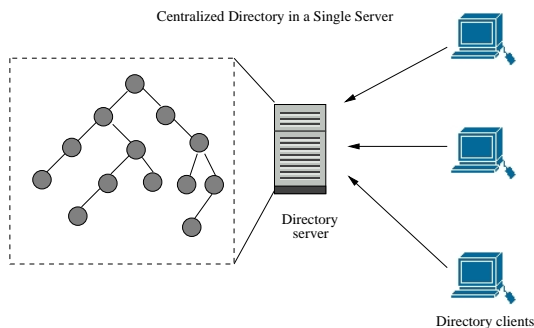
Mezi požadavky na adresářové služby ze strany uživatele patří jednoduchý přístup z běžných aplikací (např. poštovní klient, WWW prohlížeč), zobrazení aktuálních informací, krátký čas odezvy a přehlednost zobrazení dat. Z pohledu administrátora je kladen důraz na spolupráci s ostatními aplikacemi, jednoduchá správa dat a možnost řízení přístupu k různým skupinám dat pro různé kategorie uživatelů.

Z pohledu uložení dat lze adresář považovat za speciální typ (podtřidu) databáze, která je optimalizována pro čtení. Data v adresáři jsou stabilní, tzn. nemění se příliš často. Adresářové služby používají jednoduché transakce nad daty, většinou nad jedním záznamem. Nevyžadují atomické zpracování transakcí či zotavení po chybě (rollback). Pro práci s databází nejsou zapotřebí operace pro spojování dat (join) či vyhledávání nad více záznamy. Není nutné implementovat složité prohledávání pomocí jazyka SQL. Samotná data v databázi nevyžadují striktní konzistenci a mohou být mírně neaktuální, např. telefonní číslo v replikovaném záznamu.

5.1.1 Jmenný prostor (Name space)

Jmenný prostor adresáře popisuje způsob uložení a uspořádání dat v databázi. U většiny globálních databází na Internetu (systém DNS, uložení zpráv elektronických konferencí News, databáze MIB a další) se používá hierarchický způsob uložení dat ve formě kořenového stromu. Ne jinak je tomu i u adresářových služeb. Záznamy jsou uloženy ve stromové struktuře DIT, která umožňuje jednoduché dělení do skupin, podporuje distribuovanou správu záznamů či přidávání nových informací bez vlivu na další součásti systému.

Data mohou být fyzicky uložena na jednom adresářovém serveru (Directory Server), jak ukazuje obrázek 5.2.



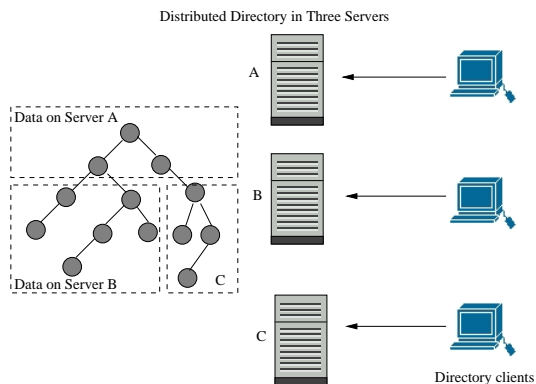
Obrázek 5.2: Centralizovaný adresář na jednom serveru

Jinou možností je rozdělení adresáře na více serverů, viz obrázek 5.3, jak jsme to viděli například u systému DNS. U DNS se rozděljuje jmenný prostor doménových jmen na zóny, které spravují jednotlivé autority. I u adresáře může dojít k rozdělení adresářového stromu na více částí pod různou správou. Při dotazování klienta na informaci v adresáři musí klient znát přesnou báзовou adresu vrcholu, odkud se začne strom procházet.

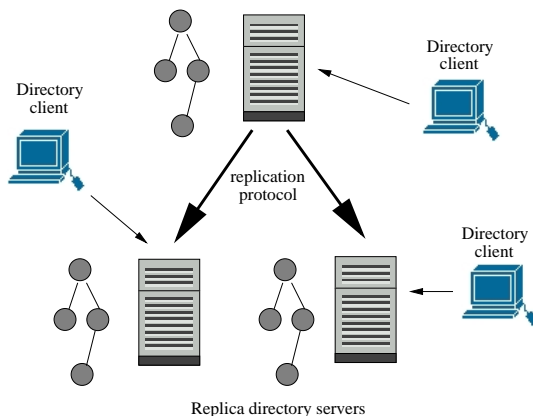
Z hlediska provozu je často vhodné rozložit nejčastěji používaná data do více lokalit a distribuovat tak zátěž na adresářovou službu. Podobně jako u služby DNS existují primární a sekundární servery, tak i u adresářových služeb se počítá s vytvořením kopií dat, tzv. replikací. Příklad architektury využívající replikaci je na obrázku 5.4.

5.1.2 Adresář X.500

Standard adresářových služby X.500 pochází z roku 1988. Standard definuje protokoly a informační model pro globální adresářovou službu, která je nezávislá na výpočetním prostředí, síťové architektuře apod. Jmenný model X.500, který popisuje způsob uspořádání dat, používá koncept jednoho virtuálního adresáře zahrnujícího všechny státy na světě, podobně jako je tomu u služby DNS. Komunikační část architektury využívá několik protokolů (viz obr. ??): protokol Directory Access Protocol (DAP) pro přístup do databáze adresářového serveru, protokol Directory System Protocol (DSP) pro výměnu operačních informací a protokol DISP (Directory Information Shadowing Protocol) pro sdílení dat mezi servery. Vlastní databáze



Obrázek 5.3: Distribuovaný adresář na více serverech

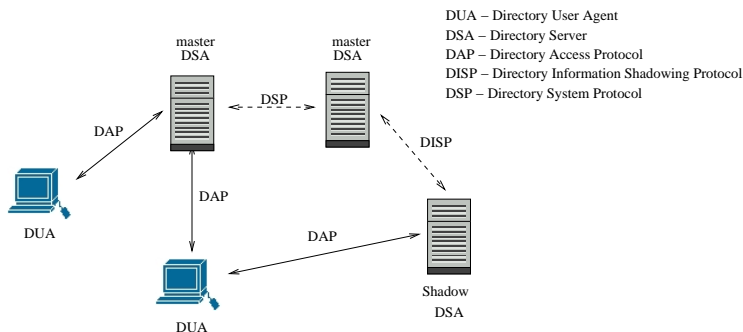


Obrázek 5.4: Rozložení zátěže na více serverů

X.500 je uložena v množině adresářových serverů DSA (Directory Server), z nichž každý obsahuje část adresáře, DIB (Directory Information Base).

Standard X.500 mimo jiné popisuje reprezentaci dat v záznamech (třídy objektů a atributy, information model), organizaci dat a jejich jmen v adresářovém strom DIT (name model), model přístupu k datům (access control), způsob správy a údržby dat (management model) a komunikaci mezi jednotlivými částmi systému pomocí protokolů DSP, DOP, DAP či DISP (funkční model).

Protože architektura X.500 byla příliš náročná na implementaci a běh klienta i serveru,



Obrázek 5.5: Architektura X.500 – funkční model

vznikla posléze odlehčená verze (lightweight) nazvaná LDAP. Tento zjednodušuje architekturu celého systému a tím i minimalizuje požadavky na klienta a server. Pro komunikaci používá pouze jediný protokol LDAP.

5.2 Adresářová služba LDAP

Lightweight Directory Access Protocol (LDAP) je internetový protokol pro přístup k adresářovým službám. Název LDAP označuje celou adresářovou službu, nejen přenosový protokol. V této části textu se podíváme, jak se adresář LDAP vytváří, z jakých částí se skládá, jaké informace může obsahovat a jak jsou tyto informace uspořádané. Také se podíváme na komunikaci mezi klientem a serverem služby LDAP a na způsob zabezpečení dat.

Formálně se celý popis adresáře LDAP včetně komunikace, uložení dat a zabezpečení popisuje pomocí tzv. modelů. Každý model standardizuje jednu z hlavních částí (pohledů) na adresářovou službu. Definice modelů LDAP vychází z obecného popisu adresářových služeb zavedeného standardem ITU-T X.500 [4].

Systém LDAP je definován pomocí množiny čtyř modelů:

- *Informační model* popisuje uložení informací v adresáři. Definuje typy dat, které můžeme vkládat, způsob jejich uložení a operace, které nad daty můžeme provádět (porovnání, řazení, vyhledání podřetězce). Informační model popisuje záznamy (např. záznam o studentovi na škole), atributy záznamu (např. jméno, příjmení, ročník) a hodnoty (konkrétní položky). Tyto informace pro konkrétní adresář tvoří tzv. adresářové schéma.
- *Jmenný model* se dívá na uložená data a definuje, jak jsou hierarchicky uspořádána a jak se na ně odkazovat. Popisuje strukturu adresáře, kterou budujeme z jednotlivých záznamů. Při pojmenování používá jednoznačné jméno nazývané *Distinguished Name*, *DN* (význačné jméno), které svou strukturou připomíná DNS adresu.
- *Funkční model* se zabývá přístupem k datům. Popisuje operace, které se provádějí prostřednictvím příkazů LDAP protokolu. Základní operací nad adresářem je vyhledávání

(search). Funkční model například definuje, v jaké části adresáře vyhledáváme (celý strom, podstrom, plochá struktura), kde začínáme hledat, podle jakých atributů a další.

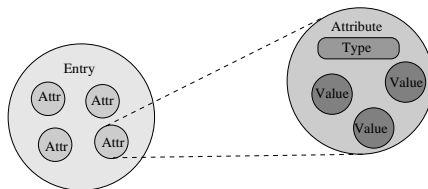
- *Bezpečnostní model*, jak už název napovídá, se zabývá zabezpečením informací uložených v adresáři. Definuje způsob přístupu k datům (autorizovaný či neautorizovaný přístup), práva k vkládání či modifikaci dat a autentizační metody (MD5 SASL, TLS).

Při návrhu a správě adresářové služby se musí administrátor zabývat každou z výše uvedených oblastí. Pečlivý návrh zejména informačního a jmenného modelu může ušetřit spoustu práce a starostí při rozšiřování či změně stávajícího modelu. V následujících podkapitolách se podrobněji podíváme na jednotlivé modely adresáře LDAP.

5.2.1 Informační model

Informační model adresáře LDAP definuje typy dat a základní jednotky pro uložení informací, tzv. *záznamy* (entries). Záznamy obsahují informace o objektech reálného světa, které jsou uloženy v adresáři. Záznam je tvořen množinou atributů, které obsahují informaci o objektu, který záznam reprezentuje. Záznam může mít více atributů. Každý atribut má typ a hodnotu. Některé atributy mohou obsahovat více hodnot, jiné pouze jedinou, viz obr. 5.6.

Příkladem záznamem může být záznam o konkrétním uživateli třídy *person* a může obsahovat atributy typu *sn* (surname, příjmení), *cn* (commonName, celé jméno) či *userPassword* (heslo) s hodnotami například *sn="Michálek"*, *cn="Martin Michálek"*, *userPassword="Patrik06"*. Tyto hodnoty mohou tvořit záznam v adresářovém stromu DIT, který je umístěn v uzlu s hodnotou *"uid=michalek, ou=ostrava, dc=kvz, dc=cz"*.

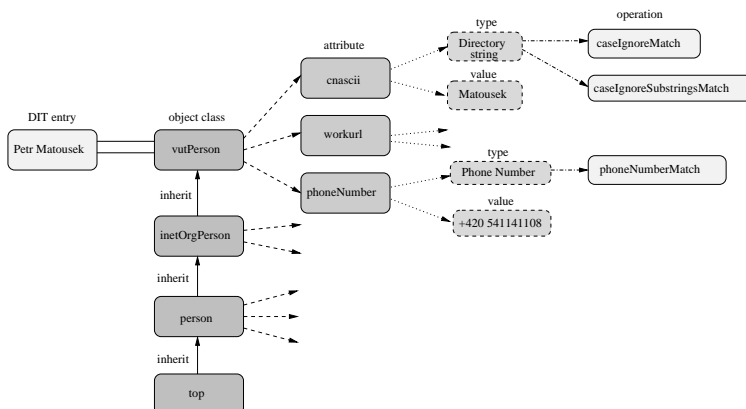


Obrázek 5.6: Složení záznamu adresářového stromu DIT

Informační model obecně slouží k popisu ukládaných dat v adresáři, zejména k definici struktury dat (množina atributů) a datového typu, do kterého se informace ukládají (typy atributů). Strukturu dat, přesněji řečeno třídu objektů, si může administrátor adresářové služby sám nadefinovat, včetně výběru datových typů, kam se informace uloží. S výběrem datového typu (řetězec, celé číslo, obrázek jpg, binární data, apod.) souvisí i operace, které lze nad těmi typy vykonávat – například bitové porovnání, vyhledání podřetězců, booleovské operace apod. Všechny tyto operace lze jednoduše definovat v konfiguraci služby LDAP pomocí tzv. adresářového schematu, který popisuje informační model adresáře.

Jednoduchý příklad je obrázku 5.7. Ukazuje jeden záznam v adresáři, který obsahuje informace o uživateli se jménem Petr Matoušek. Záznam patří do třídy objektů *utPerson*. Tato třída obsahuje atributy zděděné z vyšších tříd i nové atributy, například *cnascii* či *phoneNumber*. Každý atribut obsahuje typ a hodnotu. Hodnota atributu *cnascii* je typu

Directory String. Nad tímto typem atributu lze provádět operace porovnání přesné shody bez ohledu na velká písmena (`caseIgnoreMatch`) či vyhledání podřetězce (`caseIgnoreSubstringsMatch`). Atributy typu telefonní číslo lze pouze porovnávat operací **phoneNumberMatch**.



Obrázek 5.7: Příklad informačního modelu LDAP (adresářové schéma)

V následujícím textu se podíváme, jak se obecně definují atributy a třídy objektů v LDAP adresáři, jaká pravidla pro porovnání můžeme použít a jak z daných objektů a atributů vytvořit adresářové schéma.

5.2.1.1 Třída objektu (object class)

Třída objektů (Object Class) je množina objektů, které sdílejí stejnou charakteristiku [5]. Následující výčet uvádí základní vlastnost třídy objektů.

- Třída popisuje objekty a záznamy, které k objektům náleží.
- Kontroluje operace nad adresářem (způsob porovnání).
- Upravuje umístění objektu ve stromu DIT (hierarchii).
- Definuje, které atributy může či musí záznam obsahovat.
- Přiřazuje pro skupiny záznamů způsob přístupu (kontrola přístupu).

Podobně jako v objektovém programování i v popisu objektů adresáře existuje dědičnost. Třída popisující typ objektu může být odvozena z přímo nadřazené třídy (nebo více tříd) nebo z generické třídy. Nejvyšší generická třída se jmenuje "top". Při odvození třídy z více nadřazených tříd mluvíme o tzv. vícenásobné dědičnosti.

Každá třída definuje množinu povinných atributů a množinu volitelných atributů, které mohou záznamy objektu obsahovat. Podtřída může některé zděděné atributy, které byly původně nepovinné, označit jako povinné. V LDAP existují tři kategorie tříd objektů:

- **abstraktní třída**

Abstraktní třídy vytváří základní charakteristiku pro odvozené třídy. Nelze vytvořit záznam patřící přímo do abstraktní třídy. Záznamy musí patřit do strukturální nebo pomocné třídy. Všechny strukturální třídy jsou odvozeny (přímo či nepřímo) z abstraktní třídy "top".

```
( 2.5.6.0 NAME 'top' ABSTRACT MUST objectClass )
```

- **strukturální třída**

Strukturální třídy se používají při vytváření struktury jmen objektů. Záznamy objektů strukturální třídy obsahují informace o objektech reálného světa popisovanými danou třídou. Strukturální třída se používá pro určení pozice záznamu v DIT a pro definici obsahu záznamu. Strukturální třídu záznamu nelze měnit. Každá strukturální třída je podtřídou (přímo či nepřímo) abstraktní třídy "top".

- **pomocná (auxiliary) třída**

Pomocná třída se používá pro rozšíření charakteristik záznamů, například rozšířením množiny povinných a volitelných atributů.

Formální definice třídy zapsaná v notaci ABNF (rozšířená notace BNF[6]) vypadá takto:

```
ObjectClassDescription = LPAREN WSP
    numericoid                ; object identifier
    [ SP "NAME" SP qdescrs ]   ; short names (descriptors)
    [ SP "DESC" SP qdstring ]   ; description
    [ SP "OBSOLETE" ]          ; not active
    [ SP "SUP" SP oids ]        ; superior object classes
    [ SP kind ]                ; kind of class
    [ SP "MUST" SP oids ]       ; attribute types
    [ SP "MAY" SP oids ]        ; attribute types
    extensions WSP RPAREN

kind = "ABSTRACT" / "STRUCTURAL" / "AUXILIARY"
```

Jak vidíme, definice třídy obsahuje jedinou povinnou položku – identifikátor OID. Další položky jsou volitelné: NAME (jméno třídy), DESC (slovní popis třídy), OBSOLETE (neaktivní třída, zastaralá), SUP (nadřízená třída), typ třídy (ABSTRACT, STRUCTURAL, AUXILIARY). Na konci definice může třída obsahovat seznam povinných (MUST) a volitelných (MAY) atributů.

Příkladem třídy může být například třída **person** (viz níže), která popisuje objekty reprezentující osoby.

```
( 2.5.6.6 NAME 'person'
    SUP top
    STRUCTURAL
    MUST ( sn $ cn )
    MAY ( userPassword $ telephoneNumber $ seeAlso $ description ) )
```

OID třídy **person** je 2.5.6.6. Je to strukturální třída, která patří přímo pod abstraktní třídu "top" (není odvozena z jiné třídy). Musí obsahovat atributy **sn** (surname, příjmení), a **cn** (CommonName, celé jméno). Může také obsahovat atributy **userPassword**, **telephoneNumber**, **seeAlso** a **description**. Odvozenou třídou z třídy **person** je například **organizationalPerson**, která rozšiřuje její vlastnosti (atributy):

```
( 2.5.6.7 NAME 'organizationalPerson'
  SUP person
  STRUCTURAL
  MAY ( title $ x121Address $ registeredAddress $ destinationIndicator $
    preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $
    telephoneNumber $ internationalISDNNumber $ facsimileTelephoneNumber $
    street $ postOfficeBox $ postalCode $ postalAddress $
    physicalDeliveryOfficeName $ ou $ st $ l ) )
```

V tabulce 5.1 můžeme vidět přehled standardizovaných tříd adresáře LDAP podle RFC 4519 [7]. Tyto třídy objektů se používají při vytváření adresářových schémat LDAP.

Třída	OID	SUP	Atributy (povinné; volitelné)
applicationProcess	2.5.6.11	top	cn; seeAlso,ou,l,description
country	2.5.6.2	top	c; searchGuide,description
dcObject		top	dc;
device	2.5.6.14	top	cn; serialNumber,seeAlso,owner,ou,o,l,desc
groupOfNames	2.5.6.9	top	member,cn; businessCategory,seeAlso,owner,ou,o,desc
locality	2.5.6.3	top	; street,seeAlso,searchGuide,st,l,desc
organization	2.5.6.4	top	o; userPassword, searchGuide, businessCategory, ...
organizationalPerson	2.5.6.7	person	; title, telephoneNumber, postalAddress, ...
organizationalRole	2.5.6.8	top	cn; telephoneNumber, street, postalAddress, ...
organizationalUnit	2.5.6.5	top	ou; businessCategory, l, postalAddress, ...
person	2.5.6.6	top	sn, cn; userPassword, telephoneNumber, seeAlso, desc
uidObject	1.3.6.1.1.3.1	top	uid;

Tabulka 5.1: Příklad tříd objektů v adresáři LDAP (položka SUP označuje nadřazenou třídu)

Speciálním záznamem, se kterým se můžeme v adresáři setkat, je záznam typu *alias*. Alias je alternativní jméno objektu nebo záznamu, které umožňuje odkazovat se na jiný objekt či záznam. Významem odpovídá například symbolickému linku v unixovém souborového systému. Příklad objektu je například následující záznam se jménem DN "cn=bar,dc=example,dc=com", který se odkazuje na jiný záznam v jiném adresářovém stromu "cn=foo,dc=widget,dc=com".

```
dn: cn=bar,dc=example,dc=com
objectClass: top
objectClass: alias
objectClass: extensibleObject
cn: bar
aliasedObjectName: cn=foo,dc=widget,dc=com
```

Záznamy typu *alias* musí patřit do třídy *alias*, která obsahuje povinný atribut *aliasObjectName*. Konverze aliasu na jméno objektu se nazývá dereference a znamená systematické nahrazení jména aliasu odpovídající hodnotou v atributu.

```
( 2.5.6.1 NAME 'alias'                // definice třídy alias
    SUP top STRUCTURAL
    MUST aliasedObjectName )
( 2.5.4.1 NAME 'aliasedObjectName'    // definice atributu aliasObjectName
    EQUALITY distinguishedNameMatch
    SYNTAX 1.3.6.1.4.1.1466.115.121.1.12
    SINGLE-VALUE )
```

Obecně se doporučuje vyhnout záznamům třídy **alias**, zejména kvůli náročném zpracování, například při vyhledávání. Pokud se odkazovaný záznam nachází na jiném serveru, musí server se záznamem **alias** kontaktovat odkazovaný server pokaždé, když se se záznamem pracuje. Problém může nastat také při zrušení objektu na vzdáleném serveru, na který zůstává aktivní **alias**.

Místo použití aliasů se doporučuje používat u adresářových služeb jiné způsoby odkazů, například odkazy typu **referrals** či LDAP URL, o kterých se budeme bavit v dalších kapitolách.

5.2.1.2 Atributy

Atributy popisují vlastnosti objektu. Atribut tvoří typ atributu a hodnota. Typ tvoří více položek, které např. popisují, zda může mít atribut více hodnot, jakým způsobem se zapisují hodnoty (syntax hodnoty atributu), jak se hodnoty porovnávají, zda je nad nimi definováno uspořádání a další. Hodnoty atributů musí splňovat syntax definovanou typem atributu. Některé atributy popisují informace o uživateli a nazýváme je uživatelské atributy (user attributes). Jiné záznamy popisují operační a administrativní informace, ty se nazývají operační atributy (operational attributes).

Operační atributy (operational attributes) se používají při správě záznamů v DIT, obvykle se generují automaticky a jsou určeny pouze pro čtení. Mezi operační atributy patří **creatorsName** (DN osoby, která vložila atribut do schematu), **createTime** (čas vytvoření), **modifiersName** (DN osoby, která atribut poslední modifikovala), **modifierTimestamp** (čas poslední modifikace), a další.

Uživatelské atributy (user attributes) jsou atributy, které obsahují specifické informace vztahované k danému objektu, které uživatel může číst, modifikovat, rušit apod. Příkladem může být například **name** (jméno osoby), **mail** (emailová adresa) a podobně.

Podobně jako u tříd lze i u atributů provádět odvození podtypu z nadřazeného typu (dědičnost). Odvozený podtyp musí mít stejné použití jako nadřazený typ a také syntaktický zápis hodnot musí být stejný (nebo podrobnější). Pokud není definována operace ekvivalence pro daný typ atributu, nelze atribut použít pro pojmenování či pro vyhledávání. Každý atribut je jednoznačně určen identifikátorem OID (Object Identifier).

Specifikaci typů atributů a objektů, které se často využívají v LDAP adresářích, lze najít např. v RFC 4519 [7]. Formálně zapisujeme typ atributu LDAP podle RFC 4512 [8] takto:

```
AttributeTypeDescription = LPAREN WSP
    numericoid                ; object identifier
    [ SP "NAME" SP qdescrs ]  ; short names (descriptors)
    [ SP "DESC" SP qdstring ] ; description
    [ SP "OBSOLETE" ]         ; not active
    [ SP "SUP" SP oid ]       ; supertype
```

```

[ SP "EQUALITY" SP oid ]      ; equality matching rule
[ SP "ORDERING" SP oid ]     ; ordering matching rule
[ SP "SUBSTR" SP oid ]       ; substrings matching rule
[ SP "SYNTAX" SP noidlen ]   ; value syntax
[ SP "SINGLE-VALUE" ]         ; single-value
[ SP "COLLECTIVE" ]          ; collective
[ SP "NO-USER-MODIFICATION" ] ; not user modifiable
[ SP "USAGE" SP usage ]      ; usage
extensions WSP RPAREN        ; extensions

usage = "userApplications"    / ; user
       "directoryOperation"   / ; directory operational
       "distributedOperation" / ; DSA-shared operational
       "dSAOperation"         ; DSA-specific operational

```

Tato na první pohled nečitelná definice nám říká, že každý atribut musí mít svůj numerický identifikátor a může obsahovat následující volitelné položky – NAME (jméno atributu), DESC (slovní popis atributu), OBSOLETE (atribut není aktivní), SUP (ukazatel na nadtyp), EQUALITY (způsob porovnání), ORDERING (způsob řazení), SUBSTR (způsob vyhledávání podřetězce), SYNTAX (ukazatel na definici zápisu hodnoty), SINGLE-VALUE (omezení na jednu hodnotu), COLLECTIVE, NO-USER-MODIFICATION a USAGE (typ použití, tj. uživatelský či operační atribut). Pokud je uvedeno pole SUP, pak pravidla pro porovnání, řazení a hledání podřetězce, pokud nejsou explicitně uvedena, se dědí od nadřazeného typu. Příkladem může být například atribut `cn` (`CommonName`), který je odvozen od nadtypu `name`.

```

( 2.5.4.3 NAME 'cn'
  SUP name )
( 2.5.4.41 NAME 'name'
  EQUALITY caseIgnoreMatch
  SUBSTR caseIgnoreSubstringsMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15 )

```

Atribut `cn` má OID 2.5.4.3. Hodnotou atributu `cn` může být například "Martin Michálek". Jak je vidět z příkladu, atribut `name` s identifikátorem 2.5.4.41 definuje pravidlo na porovnávání řetězců (EQUALITY) s typem `caseIgnoreMatch`. Základní atributy a třídy objektů adresáře LDAP jsou standardizovány, např. v RFC 4519 [7], viz tabulka 5.2.

Každý záznam může obsahovat různé atributy. Pro všechny záznamy v adresářovém stromu je povinný atribut `objectClass`, který specifikuje, do jaké třídy daný záznam patří. Jinak řečeno, každý záznam musí patřit do alespoň jedné třídy a příslušnost k dané třídě je zapsána v atributu `objectClass` – viz následující definice:

```

( 2.5.4.0 NAME 'objectClass'
  EQUALITY objectIdentifierMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.38 )

```

Tento atribut slouží mimo jiné ke kontrole povolených atributů záznamu. Servery podle standardu nesmí dovolit změnu tohoto atributu u záznamu, například nelze změnit záznam

¹domainComponent (dc) popisuje komponentu DNS adresy, např. vutbr

Název atributu	OID	Typ hodnoty	Příklad hodnoty atributu
businessCategory	2.5.4.15	Directory String	"transportation"
c	2.5.4.6	Country String	"de", "fr"
cn	2.5.4.3	Directory String	"Martin Michálek"
dc (domainComponent) ¹	2.5.4.11	IA5 String	"fit", "vutbr"
description	2.5.4.13	Directory String	"barevná tiskárna"
distinguishedName	2.5.4.49	DN	"uid=matousp,dc=fit,dc=vutbr,dc=cz"
givenName	2.5.4.42	Directory String	"Petr"
l (localityName)	2.5.4.7	Directory String	"Geneva"
o (organizationName)	2.5.4.10	Directory String	"FIT VUT v Brně"
ou (organizationalUnitName)	2.5.4.11	Directory String	"FIT UIFS"
sn (surname)	2.5.4.4	Directory String	"Michálek"
uid	2.5.4.3	Directory String	"xmatou01"
userPassword	2.5.4.35	Octet String	

Tabulka 5.2: Příklad atributů adresáře LDAP

třídy **person** na záznam třídy **country**. Při vytváření záznamu a specifikaci třídy záznamu se implicitně záznamu přidávají také všechny nadřazené třídy, ze kterých je daná třída odvozena.

Všimněte si, že definice atributu **name** v prvním příkladu neobsahuje přímo pravidlo na porovnání (matching rules), pouze odkaz na něj (identifikátor **caseIgnoreMatch**). Pravidla na porovnání se uplatní při práci s daty, např. vyhledání, nastavení či změně či hodnot. *Porovnávací pravidla* formálně zapisujeme v následujícím formátu (notace ABNF):

```
MatchingRuleDescription = LPAREN WSP
    numericoid           ; object identifier
    [ SP "NAME" SP qdescrs ] ; short names (descriptors)
    [ SP "DESC" SP qdstring ] ; description
    [ SP "OBSOLETE" ]      ; not active
    SP "SYNTAX" SP numericoid ; assertion syntax
    extensions WSP RPAREN  ; extensions
```

Definice porovnávacích pravidel obsahuje identifikátor objektu (pravidla) OID, název pravidla (NAME), slovní popis (DESC), informaci, zda je pravidlo aktuální (OBSOLETE) a ukazatel na definici syntaktického zápisu povolených hodnot, nad kterými se porovnání může provádět (SYNTAX). Implementace pravidel není součástí standardu a záleží na konkrétním adresářovém serveru. Některá pravidla pro porovnání jsou uvedena v tabulce 5.3 spolu s typy hodnot atributů, nad kterými mohou být daná pravidla použita. Příkladem porovnávacího pravidla může být pravidlo na porovnání řetězců **caseIgnoreMatch**, které při porovnání ignoruje velikost písmen:

```
( 2.5.13.2 NAME 'caseIgnoreMatch' SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)
```

Hodnota OID 2.5.13.2 jednoznačně identifikuje pravidlo, položka syntax se odkazuje na typ hodnoty, která se v atributu může vyskytovat. Identifikátor "1.3.6.1.4.1.1466.115.121.1.15" (typ Directory String) popisuje jeden či více znaků ze znakové sady UCS (Universal Character Set) [9], např. "Martin Michálek":

```
( 1.3.6.1.4.1.1466.115.121.1.15 DESC 'Directory String' )
```

V tabulce 5.4 je seznam některých typů hodnot atributů používané v adresáři LDAP. Plný výčet je uveden v RFC 4517 [10].

Název pravidla	OID	Objekt porovnání (typ hodnoty)
bitStringMatch	2.5.13.16	BitString
booleanMatch	2.4.13.13	Boolean
caseExactIA5Match	2.5.13.13	IA5 String
caseExactMatch	2.5.13.5	Directory String
caseExactOrderingMatch	2.5.13.6	Directory String
caseExactSubstringsMatch	2.5.13.7	Substring Assertion
caseIgnoreIA5Match	1.3.6.1.4.1.1466.109.114.2	IA5 String
caseIgnoreListMatch	2.5.13.11	Postal Address
caseIgnoreMatch	2.5.13.2	Directory String
distinguishNameMatch	2.5.13.1	DN
generalizedTimeMatch	2.5.13.27	Generalized Time
integerMatch	2.5.13.14	Integer
objectIdentifierMarch	2.5.13.0	OID
telephoneNumberMatch	2.5.13.20	Telephone Number

Tabulka 5.3: Porovnávací pravidla nad atributy LDAP

5.2.1.3 Jednoznačné jméno (Distinguished Name)

Každý záznam má své relativní jméno, které se vztahuje k svému bezprostřednímu předchůdci (rodiči) ve stromové struktuře DIT. Náзву se říká *relativní jednoznačné jméno*, (Relative Distinguished Name, RDN). RDN je složeno z neuspořádané množiny jednoho či více atributů záznamu (např. ou=FIT UIFS či cn=Petr Matoušek+ou=FIT UIFS). Plně kvalifikované jméno DN se skládá z RDN a DN svého bezprostředního předchůdce, např. uid=matousp, ou=FIT UIFS, dc=FIT,dc=CZ. Toto jméno jednoznačně odkazuje na záznam ve stromu DIT.

5.2.1.4 Adresářové schéma

Adresářové schéma (Directory Schema) je množina pravidel, které určují, co bude uloženo v adresářové službě a jak mají klient a server zacházet s informacemi při adresářových operacích, např. při vyhledávání. Každý záznam je před vložením do adresáře zkontrolován, zda vyhovuje pravidlům adresářového schéma. Schéma definuje omezení nad uloženými informacemi, např. velikost, rozsah, formát datových hodnot apod.

Každý záznam v adresáři obsahuje množinu povinných (required) atributů a množinu volitelných (optional) atributů. Seznam těchto atributů je popsán v adresářovém schématu. Schéma nedefinuje vztahy mezi záznamy, jak je tomu například v relační databázi. Vztahy jsou popsány ve jmenném modelu adresáře.

5.2.2 Jmenný model

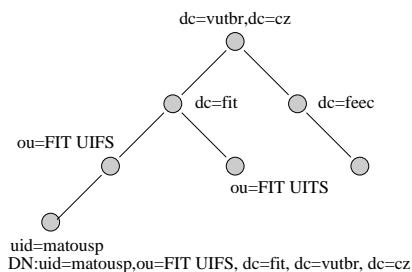
Zatímco informační model popisoval data, která se do adresáře budou vkládat, jejich formát a datové typy, jmenný model popisuje způsob uspořádání dat a vztahy mezi záznamy. Jmenný model tedy pracuje pouze se záznamy. Ukládá je do stromové struktury, která se nazývá DIT (Directory Information Tree).

DIT je stromová struktura, kde vrcholy tvoří záznamy adresáře. Hrana mezi vrcholy definuje vztahy mezi záznamy. Pokud existuje hrana z bodu X do bodu Y, pak je záznam v X

Typ hodnoty atributu	OID	Příklad
BitString	1.3.6.1.4.1.1466.115.121.1.6	'0101111101'B
Boolean	1.3.6.1.4.1.1466.115.121.1.7	TRUE
CountryString	1.3.6.1.4.1.1466.115.121.1.17	"US"
DirectoryString	1.3.6.1.4.1.1466.115.121.1.15	"This is a value of #!%#"
DN	1.3.6.1.4.1.1466.115.121.1.12	uid=jsmith,dc=example,dc=net
Enhanced Guide (search filter)	1.3.6.1.4.1.1466.115.121.1.21	person#(sn\$EQ)#oneLevel
GeneralizedTime	1.3.6.1.4.1.1466.115.121.1.24	199412161032-0500
IA5 String	1.3.6.1.4.1.1466.115.121.1.26	"vutbr"
Integer	1.3.6.1.4.1.1466.115.121.1.27	1321
JPEG	1.3.6.1.4.1.1466.115.121.1.28	
NumericString	1.3.6.1.4.1.1466.115.121.1.36	"15 079 672 281"
OctetString	1.3.6.1.4.1.1466.115.121.1.40	
OID	1.3.6.1.4.1.1466.115.121.1.38	1.2.3.4, cn
PostalAddress	1.3.6.1.4.1.1466.115.121.1.41	11209 Casey Rd.\$Tampa, FL 33618\$USA
Telephone Number	1.3.6.1.4.1.1466.115.121.1.50	+1 512 315 0280

Tabulka 5.4: Typy hodnot atributů v adresáři LDAP

bezprostředním předchůdce (angl. immediate superior, přímý nadřízený) záznamu v Y. Záznam Y lze popisovat jako následníka (angl. subordinate, podřízený) vzhledem k X. Místo termínů předchůdce a následník se můžeme setkat i s pojmy rodič (parent) a potomek (child). Stromový model adresáře lze vytvořit např. podle organizačního nebo podle geografického hlediska, viz příklad 5.8. Hierarchická stromová struktura je podobná struktuře unixového



Obrázek 5.8: Příklad stromu DIT

souborového systému. Kořenový záznam je speciální záznam o konfiguraci stromu s následující strukturou:

```

dn: dc=fit,dc=vutbr,dc=cz          # kořenový záznam
dc: fit
o: VUT v Brne
objectClass: organization
objectClass: dcObject
objectClass: top

```

Uzel stromu může obsahovat data i potomky. K identifikaci uzlu slouží Distinguished Name (DN), které tvoří posloupnost relativních DN od uzlu ke kořeni stromu, např. dn: uid=matousp,dc=fit,dc=vutbr,dc=cz.

Existuje také speciální záznam typu alias, který tvoří buď alternativní jméno záznamu, nebo se odkazuje na jiný uzel (jinou část stromu). Jeho význam je podobný symbolickým linků v Unixu. Například záznam s DN `cn=bar,dc=example,dc=com`, který se odkazuje na `cn=foo,dc=widget,dc=com` má následující strukturu:

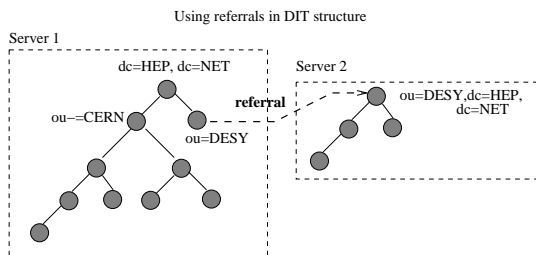
```
dn: cn=bar,dc=example,dc=com
  objectClass: top
  objectClass: alias
  objectClass: extensibleObject
  cn: bar
  aliasedObjectName: cn=foo,dc=widget,dc=com
```

Vzhledem k náročným operacím přístupu (vyhledávání) a vyhodnocování záznamů typu alias, je vhodné se jejich použitím raději vyhnout. Kromě speciálního záznamu typu alias se můžeme setkat s objekty typu odkaz (referral). Jedná se vlastně o třídu `referral` s atributem `ref`:

```
(2.16.840.1.113730.3.2.6 NAME 'referral'          # třída referral
  DESC 'namedref: named subordinate referral'
  SUP top
  STRUCTURAL
  MUST ref)
(2.16.840.1.113730.3.1.34 NAME 'ref'              # atribut ref
  DESC 'namedref: subordinate referral URL'
  EQUALITY caseExactMatch
  SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
  USAGE distributedOperation)
```

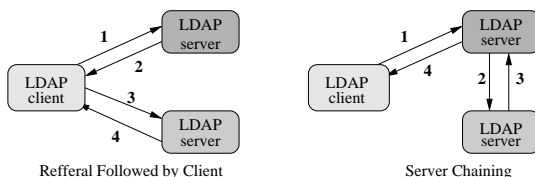
Atribut `ref` třídy obsahuje URL ukazující na jiný LDAP server, kde může být informace uložena. Příkladem může být např. odkaz na jiný LDAP server, viz obr. 5.9:

```
dn: ou=DESY,dc=HEP,dc=NET
ref: ldap://ldap.desy.de/ou=DESY,dc=HEP,dc=NET
```



Obrázek 5.9: Použití odkazu typu referral v DIT

S použitím odkazů souvisí jedna důležitá věc a to zpracování odkazů. Existují dva základní přístupy, jak zpracovat odkaz – zpracování na straně klienta (referral followed by client) nebo zpracování na straně serveru, tzv. zřetězení (server chaining). Z hlediska možnosti záplavového šíření dotazů při zřetězení, existuje v protokolu LDAP od verze 2 pouze první přístup, tzn.



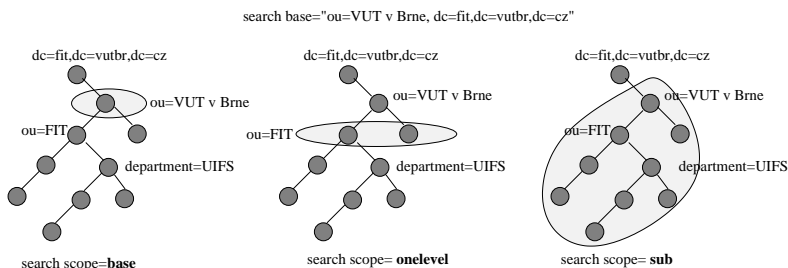
Obrázek 5.10: Zpracování odkazů typu referral v LDAP

zpracování odkazů na straně klienta. Rozdíl obou přístupů je znázorněn na obrázku 5.10. Zpracování odkazů na straně klienta může být buď (1) automatické – klient po obdržení odpovědi typu referral snaží spojit s odkazovaným serverem a získat potřebnou informaci – nebo (2) manuální, kdy klient vrátí uživateli odpověď s odkazem a uživatel musí sám iniciovat spojení s dalším serverem.

5.2.3 Funkční model

Funkční model popisuje operace, které lze provádět nad adresářem, zejména dotazy (prohledávání adresáře), změny dat (přidávání, aktualizace, rušení záznamu) a řízení přístupu k datům (identifikace klienta).

Funkční model popisuje jednak příkazy protokolu LDAP [3], dále pak způsob a rozsahy vyhledávání informací v adresářovém stromě. Při prohledávání zejména rozsáhlých databází je vhodné omezit podstrom vyhledávacích informací. LDAP definuje tři typy úrovně vyhledávání v adresáři: vyhledávání pouze v zadaném bázevém objektu (base), vyhledávání pouze v bezprostředních následnících zadaného objektu (one-level) nebo vyhledávání v celém podstromu adresáře (subtree), viz obrázek 5.11. Podle typu dat lze při vyhledávání zvolit různé způsoby



Obrázek 5.11: Rozsahy vyhledávání v adresáři LDAP

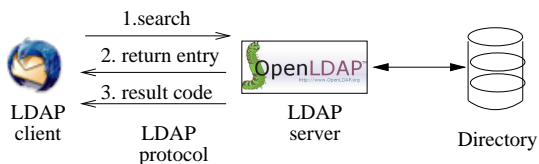
porovnání dat. Ne pro všechny typy dat jsou definovány veškeré typy porovnání. Pokud je nad daným typem definováno řazení (např. jména, čísla, apod.), lze použít při porovnání relace typu menší, větší, apod., případně vyhledávání v řetězci na rovnost, podřetězec či přibližnou shodu. Některé atributy nelze číst, lze však zjišťovat přítomnost hodnoty daného atributu.

Příklad vyhledávacích filtrů je uveden v tabulce 5.5.

Filtr	Formát	Příklad	Poznámka
equality	(attr=value)	(sn=matousek)	všechna příjmení
substring	(attr=[leading]*[any]*[trailing])	(sn=*matous*)	
approximate	(attr~value)	(sn~=matousek)	
greater than	(attr>=value)	(sn>=matousek)	
less than	(attr<=value)	(sn<=matousek)	
presence	(attr=*)	(sn=*)	bez atributu mail
AND	(&(filter1)(filter2))	(&(sn=matousek)(ou=UIFS*))	
OR	((filter1)(filter2))	((sn~=matousek)(sn=matousek))	
NOT	(!(filter1)(filter2))	(!(mail=*))	

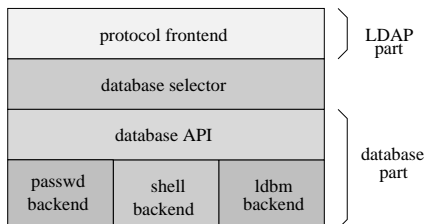
Tabulka 5.5: Vyhledávací filtry adresáře LDAP

Adresář LDAP je klasická síťová služba, která je postavená na výměně zpráv mezi LDAP klientem a LDAP serverem. LDAP klient vytvoří zprávu (dotaz), kterou pošle protokolem LDAP serveru. LDAP server provede akci nad adresářem a vrátí odpověď, která může obsahovat jeden či více záznamů. Veškerá komunikace protokolem LDAP je implementována nad transportním protokolem TCP, port 389. Příklad komunikace je na obrázku 5.12. Protokol



Obrázek 5.12: Klient a server služby LDAP

LDAP definuje pouze komunikaci a přístup k datům, nikoliv formu uložení dat. Pro uložení dat LDAP lze použít například standardní unixový soubor `/etc/passwd`, příkazové rozhraní shell či speciální databázový formát `ldbm` (indexované b-stromy), viz obr. 5.13. Mezi základní



Obrázek 5.13: Způsob uložení dat v LDAP

operace protokolu LDAP patří následující příkazy:

Bind – slouží přihlášení do adresáře, dohodnutí metody autentizace a identifikace klienta (DN uživatele). Bind umožňuje jednoduché připojení s posláním hesla v otevřené podobě či připojení pomocí SASL s autentizací pomocí kontrolního součtu MD5

Unbind – slouží k ukončení spojení

Search – základní příkaz na vyhledávání. Příkaz umožňuje definovat začátek vyhledávání (bázové DN), rozsah vyhledávání (bázová adresa, přímý potomci, celý podstrom), masku vyhledávání (přítomnost atributu, shoda hodnoty v záznamu, podřetězec, přibližné porovnávání), dále pak seznam atributů, které se klient požaduje a limit pro vyhledávání (např. maximální počet nalezených záznamů, časový limit).

Odpověď operaci Search zahrnuje vyhledané záznamy, vyhledané odkazy URL ukazující na další servery a informace o stavu vyhledávání.

Compare – slouží k porovnání hodnoty atributu u zadaných záznamů. Tato operace vrací hodnoty `false` či `true`.

Abandon – zrušení předchozí operace

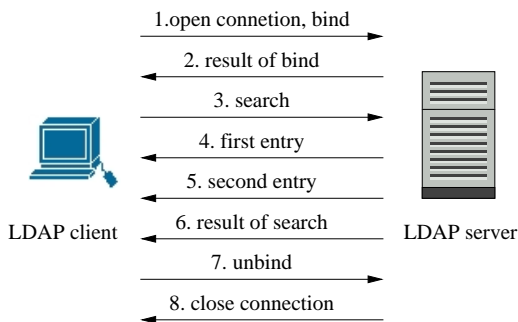
Modify – změna konkrétního záznamu (přidání hodnoty do atributu, vytvoření nového atributu, zrušení hodnoty atributu, nahrazení hodnoty)

Add – přidání nového záznamu

Delete – zrušení záznamu (v listu)

Modify Distinguish Name – změna nejlevější komponenty v DN záznamu

Příklad komunikace je na obrázku 5.14.



Obrázek 5.14: Komunikace protokole LDAP

5.2.4 Bezpečnostní model

Poslední model, o kterém si něco krátce řekneme, je bezpečnostní model. Tento model zajišťuje ochranu dat v adresáři proti neautorizovanému přístupu. Jak jsme již uvedli v části komunikace, LDAPv3 server musí implementovat autentizaci pomocí SASL. Z hlediska zabezpečení můžeme LDAP servery rozdělit do tří skupin:

- veřejné servery s daty pouze pro čtení - mohou povolit anonymní přístup
- servery podporující autentizaci heslem - musí implementovat MD5 SASL (RFC 2831)
- servery podporující kryptování a autentizaci - musí implementovat operace pro TLS (RFC 2830), autentizace pomocí veřejných klíčů/certifikátů

Vlastní definice kontroly přístupu řízená pomocí seznamů ACL (access control lists) se nastává v konfiguraci serveru. Příklad takového zabezpečení je uveden v následujícím výpisu:

```
access to attrs=userPassword
    by self write
    by * compare

access to attrs=commonName,givenName,surname,telephoneNumber,mail,otherMailbox,
businessCategory,homeInstitute,labeledURI,description,division
    by * read

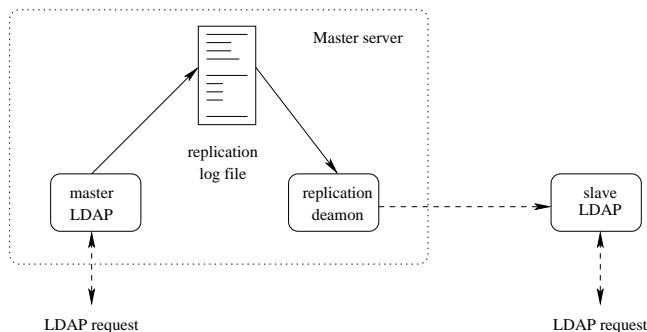
access to attrs=account,group,section,userid,ccid,gg,gidNumber,uuu,roomNumber
    by domain=localhost read
    by domain=*.cern.ch read
    by * none

access to *
    by domain=*.in2p3.fr read
    by domain=*.rl.ac.uk read
    by domain=*.desy.de read
    by domain=*.fnal.gov read
    by addr=137.138.* read
    by addr=127.0.0.1 read
    by * search
```

5.2.5 Šíření dat pomocí LDAP - replikace

Adresář LDAP definuje způsob sdílení části jmenného prostoru mezi více serverů. Proces přenosu dat se nazývá replikace. Replikace je přenos dat z primárního LDAP serveru (master) na sekundární LDAP server (replika, slave server) v rámci stejného jmenného kontextu (name context). Jedná se o princip podobný přenosu zón u DNS, distribuci konferencí Usenet (NNTP), kopie souborů v distribuovaném souborovém systému AFS (Andrew File System) apod. Z pohledu efektivity se replikace doporučuje pouze pro servery, které jsou stabilní, tzn. není tam mnoho změn. Princip replikace je znázorněn na obrázku 5.15.

Vlastní přenos dat zajišťuje speciální proces, replikační démon. Před prvním provedením replikace se inicializuje databáze sekundárního serveru tak, že se načte obsah databáze primárního serveru. Další změny se pak přidávají formou replikace. Během replikace posílá replikační démon požadavky na změnu dat ve formě příkazů LDAP (ldap_add, ldap_modify). Vystupuje



Obrázek 5.15: Replikace dat v LDAP

zde ve formě klienta, který se autentizuje na serveru a pak volá operace na aktualizaci dat. Předávané informace pro replikaci jsou uloženy v textovém formátu LDIF (Lightweight Directory Interchange Format), viz následující ukázka, která u zadaného záznamu mění atribut `ou` na hodnotu `EP/ED/SD`:

```

replica: listbox3.cern.ch:389
time: 904838414
dn: cn=Angelos Agoritsas id=17,ou=People,o=CERN,o=HEP,c=NET
changetype: modify
replace: ou
ou: EP/ED/SD

```

Replikace probíhá tak, že primární server ukládá veškeré změny v databázi do replikačního záznamu (replication log). Replikační proces přečte informace z replikačního záznamu a pošle je na sekundární server. Při navazování spojení se vždy vyžaduje autentizace replikačního procesu. Podle typu přístupových práv přijímá sekundární server požadavky na změnu v databázi. Sekundární server zpracuje data od replikačního procesu stejně jako lokální požadavky na změnu dat v databázi.

Replikační proces obsahuje informace o tom, jaká část adresářového stromu se bude replikovat (DN podstromu, objekty, filter), jaké atributy se budou replikovat, dále je potřeba zadat jména sekundárních LDAP serverů obsahujících kopii stromu a typ synchronizace. LDAP používá tři typy synchronizace:

- okamžitou synchronizaci (immediate), kde ihned po změně v databázi primárního serveru dojde k replikaci,
- odloženou synchronizaci (delayed), kdy se veškeré změny na primárním serveru ukládají do replikačního záznamu, který replikační server načítá při svém spuštění. Replikace se provádí podle časového plánu (podobně jako cron).
- manuální synchronizaci (one-shot), při které se modifikace taktéž ukládají, nicméně replikační server je spuštěn jednorázově administrátorem.

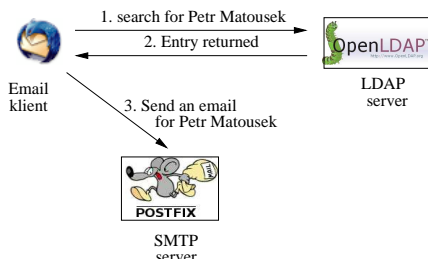
Při replikaci je nutné dodržet základní podmínku, že sdílená data musí mít stejný formát. Při replikaci můžeme přenášet pouze vybraná data, nikoliv celé záznamy.

5.3 Použití

V této kapitole si ukážeme některé možnosti použití adresářové služby LDAP v dalších síťových službách.

5.3.1 Vyhledání emailové adresy adresáta

Základním použitím adresářové služby LDAP je integrace LDAP klienta do poštovních programů, obr. 5.16. Poštovní aplikace se při psaní emailové zprávy dotáže LDAP serveru na přesnou emailovou adresu adresáta. V případě, že je definováno v adresáři více uživatelů se stejným jménem, vrátí všechny nalezené, z nichž si uživatel vybere.



Obrázek 5.16: Vyhledání adresáta elektronické pošty v adresáři LDAP

V konfiguraci poštovního klienta je potřeba nadefinovat adresu LDAP serveru, bázovou adresu (kontext), od které se vyhledává a způsob autentizace.

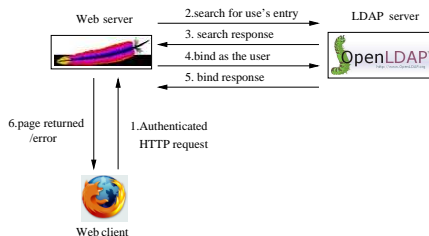
5.3.2 Vyhledání volajícího účastníka v IP telefonii

Podobným způsobem jako vyhledání emailové adresy slouží i vyhledání telefonního čísla volaného účastníka v IP telefonii. VoIP telefony (hardwarové či softwarové dnes běžně obsahují LDAP klienta, která se spojí se zadaným LDAP serverem a vyhledá atribut typu **Telephone Number** v daných záznamech.

5.3.3 Verifikace autentizačních údajů

Služba LDAP se používá také k autentizaci uživatelů jiných služeb, například webové služby, viz obrázek 5.17. Webový klient, který se přihlašuje na server, zadá své autentizační údaje (login a heslo). Server je ověří vůči LDAP serveru tak, že požádá o připojení (operace **bind**) se autentizací. Do příkazu zadá uživatelské jméno a heslo. Pokud příkaz **bind** správně projde, Web server se odpojí a považuje uživatele za korektně autentizovaného. Pokud se operace nezdaří, vrátí chybu autentizace.

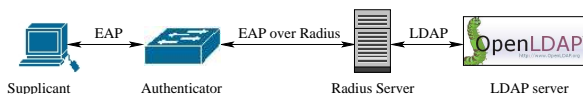
Podobně jako pro službu WWW se tento princip autentizace používá také pro unixové přihlašování a ověřování identity uživatelů. Místo autentizace přes soubor `/etc/passwd` se definuje mechanismus autentizace přes LDAP server. Princip ověřování je stejný jako ve výše uvedené autentizaci služby WWW.



Obrázek 5.17: Ověření uživatele webové služby

5.3.4 Použití adresářových služeb při autentizaci 802.1x

Standard IEEE 802.1x definuje ověřování zařízení na úrovni linkové vrstvy modelu OSI. Při připojení zařízení do sítě (přes přepínač či WiFi AP), zašle aktivní prvek klientovi (suplikant) pomocí protokol EAP (Extensible Authentication Protocol)[11] výzvu uživateli k autentizaci. Po přijetí odpovědi přeposílá aktivní prvek autentizační data na Radius či TACACS server. Obvykle však tento server neobsahuje jména a hesla uživatelů, pouze autorizační informace pro uživatele (např. VLAN síť). Tento Radius server pomocí protokolu LDAP požádá LDAP server o ověření hesla, viz obrázek 5.18. Pokud se ověření zdařilo, umožní aktivní pr-



Obrázek 5.18: Použití adresářové služby LDAP při autentizaci 802.1x

vek připojení klientského zařízení do sítě LAN. V případě neúspěšné autentizace je přístupový port na úrovni L2 zablokován.

5.3.5 Autentizace veřejného klíče pomocí certifikátů X.509

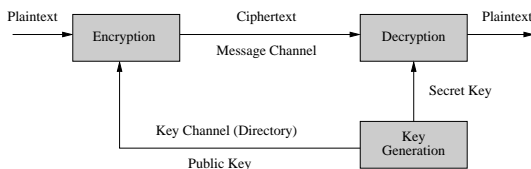
V tomto příkladu použití adresářových služeb nejde přímo o službu LDAP, ale o použití mechanismu autentizace definovaného adresářovou službou X.500, konkrétně ověřování veřejného klíče podle standardu X.509.

Pro zabezpečení komunikací a autentizaci uživatele či aplikace pomocí veřejné kryptografie (kombinace veřejného a soukromého klíče), je nutné, aby si komunikující strany vyměnily veřejné klíče, které se používají k šifrování či k autentizaci pomocí digitálního podpisu. Přímá výměna veřejného klíče mezi komunikujícími entitami je možná pouze v případě malého počtu účastníků. V případě většího počtu účastníků je tento systém nevhodný. Z tohoto důvodu se využívá třetích stran, které uchovávají veřejné klíče, potvrzují jejich pravost. Těmito třetími stranami se říká certifikační autority a potvrzení, které prokazuje platnost veřejného klíče, se nazývá certifikát.

5.3.5.1 Certifikát

Certifikát veřejného klíče je strukturovaný datový záznam, který mimo jiné obsahuje položky jako je jednoznačná identifikace držitele certifikátu (entity, která vlastní veřejný klíč či jinou potvrzovanou informaci) a parametry veřejného klíče. Certifikát je elektronicky podepsán vydávající certifikační autoritou (CA). Podpis CA vytváří kryptografická vazba mezi identitou držitele a jeho veřejným klíčem. Partner, který používá certifikát držitele, důvěřuje platnosti této vazby, pokud důvěřuje CA, která certifikát vydala.

Pro přenos veřejného klíče se používá bezpečný kanál, kterému se také říká adresářový (directory-based channel), viz obr. 5.19 [12].



Obrázek 5.19: Přenos veřejného klíče

Datová struktura certifikátu definovaná v normě ITU-T X.509 [13] a IETF RFC 5280 [14] obsahuje identifikaci držitele certifikátu, veřejný klíč držitele, identifikace vydavatele (certifikační autority), číslo certifikátu, platnost certifikátu a další údaje týkající se použití certifikátu:

```

Certificate ::= SEQUENCE {
    tbsCertificate TBSCertificate,
    signatureAlgorithm AlgorithmIdentifier,
    signatureValue BIT STRING}

TBSCertificate ::= SEQUENCE {
    version                [0] EXPLICIT Version DEFAULT v1,
    serial Number           CertificationSerialNumber,
    signature               AlgorithmIdentifier,
    issuer                  Name,                // sequence of RDNs, i.e. Directory String
    validity                Validity,
    subject                 Name,
    subjectPublicKeyInfo    SubjectPublicKeyInfo,
    issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,
                        -- If present, version MUST be v2 or v3
    subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,
                        -- If present, version MUST be v2 or v3
    extensions [3] EXPLICIT Extensions OPTIONAL
                        -- If present, version MUST be v3
}

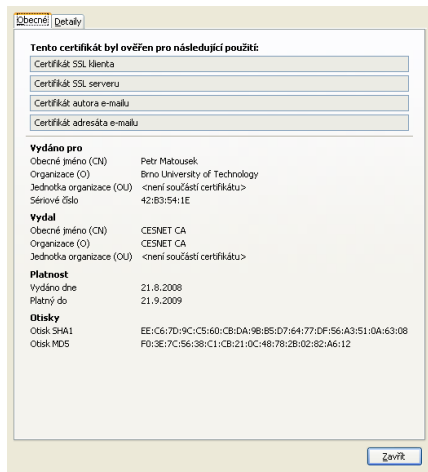
Validity ::= SEQUENCE {
    notBefore Time,
    notAfter Time
}
  
```

```

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm      AlgorithmIdentifier,
    subjectPublicKey BIT STRING }
}

```

Příklad certifikátu je na obrázku 5.20. Standard ITU-T X.509 popisuje rámec pro au-



Obrázek 5.20: Příklad certifikátu

tentizaci pomocí adresáře (The Directory: Authentication framework). Tento rámec definuje infrastrukturu certifikace ve formě stromu DIT (Directory Information Tree). Každý nelistový uzel tohoto stromu reprezentuje entitu (certifikační autoritu), jejíž certifikát k veřejnému klíči byl vystaven nejbližším rodičovským uzlem. Listy stromu jsou jsou koncové entity (uživatelé, aplikace). CA mohou být na různé úrovni či v určité doméně zájmu – např. národní CA, akademické CA (CESNET), školní CA (CA VUT) apod. Kořenová certifikační autorita ověřuje svůj vlastní veřejný klíč (self-signed certificate). Pro služby Internetu se používá definice architektury systému výměny veřejných klíčů PKI (Public Key Infrastructure) definovanou v RFC 5280.

Obecně existuje více způsobů, jak pomocí adresáře ověřovat platnost veřejných klíčů:

- X.509 certifikáty (Public Key Infrastructure PKI)
- Síť důvěry PGP (PGP Web of trust)
- SPKI (Simple PKI)

5.3.6 Řádkové příkazy pro přístup k adresáři LDAP

Součástí většiny unixových systémů je knihovna OpenLDAP obsahující server i klienta pro práci s adresářem LDAP. Knihovna obsahuje příkazy `ldapsearch`, `ldapmodify` a další pro

práci na adresářem. Příklad použití je uveden v následujících ukázkách činnosti:

- Vyhledání emailové adresy uživatele

```
ldapsearch -h ldap.fit.vutbr.cz -s sub -b "dc=vutbr,dc=cz" "(cnascii=Matousek*)" mail

dn: uid=matousf,dc=fit,dc=vutbr,dc=cz

dn: uid=xmatou13,dc=fit,dc=vutbr,dc=cz
mail: xmatou13@stud.fit.vutbr.cz

dn: uid=dummy2076,dc=fit,dc=vutbr,dc=cz
mail: matousp@dcse.fee.vutbr.cz

dn: uid=matousp,dc=fit,dc=vutbr,dc=cz
mail: matousp@fit.vutbr.cz

search: 2
result: 0 Success
```

- Vyhledání informací o LDAP serveru

```
ldapsearch -h ldap.fit.vutbr.cz -s sub -b "dc=vutbr,dc=cz" "(objectclass=*)"

# fit.vutbr.cz
dn: dc=fit,dc=vutbr,dc=cz
dc: fit
o: VUT v Brne
o: VUT
objectClass: organization
objectClass: dcObject
objectClass: top

# Manager, fit.vutbr.cz
dn: cn=Manager,dc=fit,dc=vutbr,dc=cz
cn: Manager
sn: Manager
objectClass: inetOrgPerson
postalAddress: Bozetechova 2 612 66 Brno
mail: root@fee.vutbr.cz
```

Kromě řádkových příkazů je knihovna LDAP implementována i v řadě programovacích jazyků, například LDAP C SDK, LDAP Java SDK, Net::LDAP Perl, Python-LDAP apod. Podobně je definován i přístup k LDAP adresáři přes službu HTTP. Zde se využívá adresování záznamů v adresáři speciální URI typu `ldap` s následující syntaxí:

```
ldap://<host>:<port>/<dn>[?<attrs>] [<scope>] [<filter>] [<extension>]
např.
ldap://ldap.fit.vutbr.cz/dc=vutbr,dc=cz?cn,mail,phoneNumber?cn=*matousek*
```

5.4 Shrnutí

Adresářové služby zajišťují přístup k informacím v elektronické databázi. Tato databáze obsahuje zejména informace o osobách, jejich telefonních číslech, emailových adresách a další veřejné údaje. Adresář může obsahovat i záznamy o zařízeních, službách a dalších objektech.

Narozdíl od klasických databází má adresář speciální vlastnosti. Data v adresáři jsou stabilní, struktura databáze je optimalizovaná pro rychlé vyhledávání. Adresář umožňuje distribuci dat pomocí replikace či odkazů na jiné servery, propojování s aplikacemi apod.

Architektura služby LDAP vychází ze standardu ITU-T X.500 a definuje způsob uspořádání záznamů do stromové struktury (informační model), strukturu záznamů a operace nad nimi (jmenný model), komunikaci mezi klientem a serveru a způsoby vyhledávání (funkční model) a zabezpečení informací (bezpečnostní model).

Adresářové služby se nepoužívají pouze ke zjišťování informací o osobách, i když je to jejich základní funkce. Velkou roli hraje například ověřování totožnosti uživatele (autentizace), při níž se aplikace (např. autentizační démon, WWW server, či Radius server) žádají LDAP server o validaci autentizačních údajů zadaných uživatelem dané služby.

5.5 Otázky

1. Co jsou to adresářové služby a k čemu slouží?
2. Jak jsou uspořádána data v adresáři? Jak se popisují?
3. Jaký je vztah mezi záznamy, atributy a hodnotami v LDAP? Uveďte příklad.
4. Co je to X.500? Čím se liší od LDAP?
5. Jaké operace používá LDAP? Uveďte příklad dotazu na vyhledání osoby na zadaném serveru.
6. Co je to replikace, k čemu slouží a jak pracuje?
7. Jak lze sdílet data v adresáři LDAP?
8. Popište propojení LDAP a emailového klienta. Uveďte na příkladu klienta nastavení LDAP serveru.
9. Navrhněte adresářové schéma pro FIT VUT. Použijte organizační rozdělení – struktura fakulty, ústavy apod.
10. Navrhněte atributy (včetně typů) pro objekt `ucitelFIT`, `studentFIT`.
11. Vytvořte DIT pro VUT.
12. Co je to adresářové schéma? Vytvořte adresářové schéma pro záznam `student`
13. Navrhněte adresářové schéma pro FIT VUT.
14. Popište architekturu LDAP systému z pohledu komunikace.
15. Jaká porovnávací pravidla lze použít nad atributy LDAP adresáře? Uveďte příklad.

16. Jak lze použít LDAP pro autentizace uživatelů Webových služeb?
17. Uveďte, jak lze využít adresářové služby v IP telefonii či v autentizaci 802.1x.

Doporučená literatura a standardy

- [1] M.Wahl, T.Howes, and S.Kille. *Lightweight Directory Access Protocol (v3)*. RFC 2251, December 1997.
- [2] K.Zeilenga. *Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map*. RFC 4510, June 2006.
- [3] J.Sermersheim. *Lightweight Directory Access Protocol (LDAP): The Protocol*. RFC 4511, June 2006.
- [4] ITU-T. *The Directory – Overview of concepts, models and services*. X.500 (2005), August 2005.
- [5] ITU-T. *The Directory: Models*. X.501 (2005), August 2005.
- [6] D.Crocker and P.Overell. *Augmented BNF for Syntax Specifications: ABNF*. RFC 4234, October 2005.
- [7] A.Sciberras. *Lightweight Directory Access Protocol (LDAP): Schema for User Applications*. RFC 4519, June 2006.
- [8] K.Zeilenga. *Lightweight Directory Access Protocol (LDAP): Directory Information Models*. RFC 4512, June 2006.
- [9] ISO. *Universal Multiple-Octet Coded Character Set (UCS)*. ISO/IEC 10646-1:2003, June 2006.
- [10] S.Legg. *Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules*. RFC 4517, June 2006.
- [11] B.Aboba, D.Simon, and P. Eronen. *Extensible Authentication Protocol (EAP) Key Management Framework*. RFC 5247, August 2008.
- [12] Wanbo Mao. *Modern Cryptography. Theory and Practice*. Prentice Hall, 2004.
- [13] ITU-T. *The Directory: Public-key and attribute certificate frameworks*. X.509 (2005), August 2005.
- [14] D.Cooper, S.Santesson, S.Farrell, S.Boeyen, R.Housley, and W.Polk. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*. RFC 5280, May 2008.
- [15] C. Hunt. *TCP/IP Network Administration*. O'Reilly, 3rd edition, 2002.
- [16] M. Lucas. *Síťový operační systém FreeBSD*. ComputerPress, 2003.
- [17] M.Wahl, T.Howes, and S.Kille. *Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*. RFC 2253, December 1997.

- [18] T.Howes. *The String Representation of LDAP Search Filters*. RFC 2254, December 1997.
- [19] T.Howes and M.Smith. *The LDAP URL Format*. RFC 2255, December 1997.
- [20] M.Wahl, H.Alvestrand, J.Hodges, and R.Morgan. *Authentication Methods for LDAP*. RFC 2829, May 2000.
- [21] J.Hodges and R.Morgan. *Lightweight Directory Access Protocol (v3): Technical Specification*. RFC 3377, September 2002.

Kapitola 7

Hlasové služby

Hlasové služby patří dneska mezi velice populární služby na Internetu. Zahrnují nejenom přenos hlasu po Internetu (Voice over IP, VoIP), ale i další služby, jako je on-line textová komunikace (Instant Messaging, IM), zjišťování přítomnosti osoby (Presence Services), vytváření konferenčních hovorů, přenos obrazu (videotelefonní služby) a další. Pro hlasovou komunikaci lze využít buď specializované aplikace typu Skype, NetMeeting či plnohodnotné služby pro podporu IP telefonie, kam patří různá řešení postavené na protokolech SIP či H.323. V následujícím textu si povíme něco o architektuře IP telefonie a jejímu rozdílu oproti klasické telefonii. Dále se podíváme na protokoly SIP, H.323 a RTP, které patří mezi základní stavební kameny IP telefonních sítí.

7.1 Úvod

Technologie přenosu hlasu přes Internet, nazývaná též jako IP telefonie, byla vytvořena jako náhrada stávající telefonních přenosů TDM pro síť postavené nad IP. Přenos hlasu lze implementovat nad více technologiemi, např. ATM (VoATM), Frame Relay (VoFR) či přímo nad IP (VoIP). Poslední přenos se v dnešní době nejvíce prosazuje a my se mu budeme věnovat v této kapitole.

Po počátečním nadšení pro IP telefonii dochází dnes spíše k mírnému růstu. V roce 2004 experti z oboru IT očekávali, že IP telefonie bude hrát dominantní roli v oblasti telekomunikací. Gartner řekl: „Otázkou není, jestli bude IP telefonie zavedena, ale kdy bude zavedena.“ Očekávalo se, že v roce 2010 cca 40% organizací spojí datové a hlasové sítě do jediné sítě. V roce 2004 se zvětšil počet organizací používající VoIP z 3% (v roce 2003) na 12%¹. V současné době se ukazuje, že ne všem organizacím se vyplatí nahrazovat stávající telekomunikační rozvody VoIP technologií, zejména kvůli počátečním investicím. Nicméně při budování zcela nových datových a telefonních rozvodů se obě sítě integrují a používá se jedna datová síť běžící nad IP.

7.2 Klasická telefonní síť

Nejprve si řekneme pár základních informací o architektuře a signalizaci klasické telefonní sítě PSTN (Public Switched Telephone Network)². Pochopení těchto dvou prvků nám pomůže

¹InStat, „Business VoIP: An End-Users Perspective“, 2004

²V češtině se užíval termín JTS (jednotný telefonní systém), který je dnes již zastaralý.

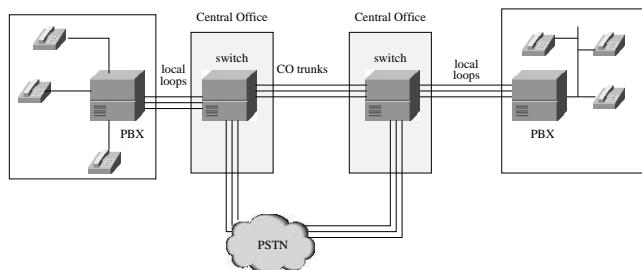
porozumět architektuře a signalizaci telefonních sítí na IP a způsobu propojení těchto dvou technologií.

Klasická telefonní síť vytváří spolehlivé plně duplexní spojení s vysokou kvalitou přenosu hlasu. Hlas se přenáší po kanále s doporučenou přenosovou rychlostí 64 Kb/s. Tato šířka pásma je garantována pro každý telefonní hovor. Telefonní přenos v klasické telefonní síti se přenáší pomocí přepínání okruhů, tj. vytváření přímého elektrického obvodu mezi volajícím a volaným.

Telefonní systém tvoří telefonní přístroj, který konvertuje zvuk na elektrický signál a obráceně. Také slouží k zahájení spojení (vytáčení) a oznamování, že hovor přichází (zvonění). Pro přepojování hovorů slouží ústředny (telephone exchanges), které využívají časový multiplex TDM pro digitalizaci analogového signálu od koncových stanic.

7.2.1 Architektura klasické telefonní sítě

Telefonní síť se skládá z několika základních komponent, viz obr. 7.1. Jsou to jednak koncová



Obrázek 7.1: Základní prvky telefonní sítě

zařízení, kterými účastník komunikuje, a ústředny (přepínače), které slouží k přepínání hovorů. Pro spojení mezi účastnickým zařízením a ústřednou se používá elektrický obvod nazvaný lokální smyčka (local loop). Komunikace mezi ústřednami zase probíhá po páteřních spojích, které se označí v telekomunikacích jako **trunks**.

Koncová zařízení (edge devices) jsou analogové či digitální telefony. Analogové telefony přenášejí zvuk analogovým signálem, digitální telefony obsahují kodek, který převádí analogový signál na digitální. Koncová zařízení mohou být připojena buď do místní (firemní) ústředny (PBX, Private Branch Exchange) nebo přímo do veřejné telefonní sítě PSTN.

Lokální smyčka (local loop) tvoří rozhraní do telefonní sítě. Obvykle jde o jeden pár kabelů (dva či čtyři dráty), který přenáší hovor. Zákazník (např. firma) může mít více lokálních smyček.

Přepínače či ústředny (switches) mohou být *centrální* (CO, Central Office), které ukončují lokální smyčku a zajišťují vytváření a rušení hovorů, poskytují signalizaci a přepínání okruhů. Druhým typem ústředem jsou *privátní* ústředny (PBX, Private Branch

Exchange), které jsou součástí zákaznické sítě. PBX vykonávají podobné funkce jako centrální ústředny.

Základní funkcí ústředny je přepínání hovorů (switching). Pokud jde hovor mimo lokální pobočku, zabezpečují také směrování hovorů. Veřejná telefonní síť PSTN využívá techniku přepínání obvodů (circuit switching), kdy se vytváří na požádání obvod propojující koncová zařízení. Součástí ústředny je napájení telefonních přístrojů, detekce proudu (zvednutí sluchátka), generování vytáčecího tónu, přijímání číslic, generování zvonění a další.

Páteřní spoje (trunks) propojují ústředny. Tyto páteřní linky mohou propojovat privátní ústředny (tie trunk), privátní a centrální ústřednu (CO trunk) nebo propojovat dvě centrální ústředny různých společností (interoffice trunk). Pro páteřní spoje se používají technologie ISDN T1 a E1, které umožňují přenášet souběžně 23, resp. 30 kanálů. Tyto technologie využívají pulzní kódovou modulaci (PCM) a časovým multiplexem TDM. U větších sítí se používají páteřní linky T3 (3x T1), resp. E3, případně linky SONET/SDH.

7.2.2 Signalizace v klasické telefonní síti

Kromě hlasového provozu je nezbytnou součástí telefonní sítě *signalizace*. Podle způsobu použití rozlišujeme tři typy signalizace: kontrolní (supervizory), adresovou (address) a informační (informational).

Pro komunikaci mezi ústřednou a telefonním přístrojem se používá tzv. *kontrolní (supervizory) signalizace*. Tato signalizaci obsahuje tři stavu:

Zavěšeno (On hook) – ve stavu zavěšeno je elektrický obvod s ústřednou přerušen (stav otevřeno). Pouze vyzvánění je aktivní v tomto stavu přístroje.

Zvednuté sluchátko (Off hook) – při zvednutí sluchátka dojde k uzavření elektrického obvodu mezi přístrojem a ústřednou. Procházející proud signalizuje ústředně, že někdo chce uskutečnit hovor. Ústředna v tomto okamžiku pošle vytáčecí tón (dial tone) oznamující připravenost.

Vyzvánění (Ringing) – pokud účastník vytočí volajícího, ústředna pošle na jeho přístroj vyzváněcí signál. Zároveň pošle podobný signál zpět volajícímu.³

Po zvednutí sluchátka může dojít k vytáčení čísla, tzn. adresování. *Adresová signalizace* může používat buď *tónovou volbu* DTMF (Dual Tone Multifrequency), při které každé vytáčené číslo se liší svou frekvencí, nebo *pulzní volbu* (pulse dialing). U pulzní volby přístroj vysílá různý počet elektrických impulzů pro každé vytáčené číslo.

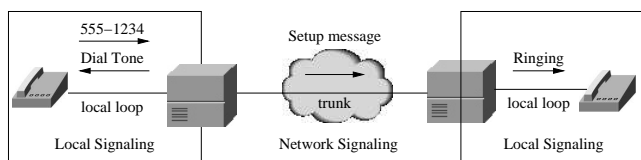
Třetím typem signalizace je *informační signalizace*, která informuje o vytváření hovoru a zjištění stavu volajících. Informační signalizace definuje několik typů tónů – vytáčecí tón (dial tone), obsazovací tón (busy), zpětné vyzvánění (ringback), obsazovací tón (congestion), neznámé číslo (no such number), atd.

Všechny tyto typy signalizace musí být implementovány pro analogový i digitální telefonní přenos. Digitální ústředny využívají přenosové linky T1 a E1 se signalizací, kdy každý kanál přenáší signalizaci speciálními bity na daném hlasovém kanálu (CAS, Channel Associated

³Vyzváněcí tón v USA je dvě sekundy vyzvánění a čtyři sekundy ticha. V Evropě se používá dvojitý vyzvánění následované dvěma sekundami ticha.

Signaling, in-band signaling). Příkladem mohou být signalizace D4 či ESF u T1. Druhý způsob je využití speciálního kanálu vyhrazeného pro signalizaci, kde nedochází ke sdílení kanálu pro hlas a signalizaci. Tento způsob se nazývá CCS (Common Channel Signaling). Příkladem může být kanál D u ISDN (Q.931), signalizace SS7 a další. Signalizační protokol SS7 (Signaling System #7) se používá jako interní protokol pro signalizaci mezi ústřednami.

Na následujícím obrázku 7.2 si ukážeme jednotlivé fáze vytváření telefonního hovoru. Při vytváření se použijí všechny tři typy signalizací, o kterých jsme se už zmínili. Podle místa použití (lokální smyčka, mezi ústřednami) můžeme signalizaci na místní (lokální) a síťovou (network).



Obrázek 7.2: Vytvoření telefonního spojení

Prostřednictvím lokální signalizace uživatel oznamuje ústředně, že chce telefonovat. Využívá všechny základní typy signalizací. Když zvedne sluchátko, uzavře se obvod a oznámí ústředně, že chce volat (kontrolní signalizace). Ústředna odpoví vytáčením tónem (informační signalizace), že je připravena přijmout číslo. Poté uživatel začne vytáčet číslo (adresová signalizace) a pošle ho ústředně.

Ústředna vyhledá cestu k volanému účastníkovi a předá požadavek na další ústřednu přes páteřní spojení. Koncová ústředna pošle signál vyzvánění do telefonního přístroje volaného účastníka.

7.2.3 Výhody klasické telefonie

Mezi hlavní výhody klasické telefonie patří garantovaná šířka pásma a spolehlivý přenos, dobrá kvalita přenosu, napájení telefonních přístrojů z ústředny, spolehlivost a bezchybnost i standardizace. V digitální telefonie nabízí i pokročilé služby, například hlasové schránky, konferenční hovory, pozastavení a přeměrování hovorů, případně pokročilé služby, například lokalizaci účastníka při volání nouzového telefonního čísla.

7.3 Digitální přenos hlasu

Převod analogového signálu na digitální (tzv. digitalizace analogového signálu) se využívá při přenosu telefonního hovoru přes digitální linky a ústředny. Používá se také při převodu hlasového analogového signálu volajícího na digitální signál u IP telefonie. Pro digitalizaci se používají speciální procesory DSP (Digital Signal Processor), které převádí analogový signál na digitální a zpět. Kromě toho provádějí procesory DSP hlasovou kompresi, překódování mezi různými kodeky či umožňují vytvářet konference mezi více účastníky. Vlastní převod signálu se provádí podle vybraného algoritmu pro kódování/dekódování, kterému se zkráceně říká kodek. Základní typy kodeků si ukážeme v dalším textu.

DPS procesory jsou součástí telefonních přístrojů připojených na digitální telefonní linku. Jsou také základní součástí IP telefonů. Činnost DSP lze provádět i softwarově, například u softwarových IP telefonů. V telefonních ústřednách se převážně používají hardwarové DSP převodníky. U IP telefonie se DSP procesory vyžadují zejména na zařízeních, které propojují klasickou telefonní síť s IP telefoníí, případně v místech, kde dochází k překódování (tzn. převod kódování hlasu z jednoho kodeku na jiný). U telefonní sítě, která je postavena pouze na IP telefonii, se nemusí provádět DSP převody (vyjma převodu hlasu na elektrický signál a zpět v telefonu).

Digitalizace hlasového signálu se skládá z několika kroků: vzorkování analogového signálu, kvantifikace vzorků, kódování vzorků na binární čísla a komprese.

Vzorkování analogového signálu probíhá v periodických intervalech. Rychlost vzorkování musí být dvakrát tak větší než nejvyšší frekvence analogového signálu, aby byla možná rekonstrukce (Nyquistův teorém). Pokud se dodrží podmínka Nyquistova teorému, nasnímané vzorky obsahují dostatek informací pro úspěšnou rekonstrukci snímaného signálu.

Lidské ucho vnímá zvuk v rozmezí cca 20 – 20.000 Hz. Řeč zahrnuje zvuky v rozsahu 200 – 9000 Hz. Tradiční telefonie byla navržena pro přenos signálu ve frekvenčním pásmu 300 – 3400 Hz. Toto pásmo je kompromisem mezi ekonomickými náklady na šířku pásma a rozeznáním hlasu volajícího. Nyquist rozšířil digitalizaci na 4000 Hz pro přenos vyšších frekvencí. Pro digitalizaci hlasu o frekvenci 4000 Hz se používá (podle Nyquistova teorému) vzorkování 8000 vzorků za sekundu, tj. každý vzorek za 125 mikrosekund.

Výstupem vzorkování je signál pulzní amplitudové modulace (PAM, Pulse Amplitude Modulation).

Kvantifikace vzorků přiřazuje nasnímaným vzorkům hodnotu, která odpovídá amplitudě vzorku. Kvantifikace rozděluje vzorky do segmentů (tříd) diskretních hodnot, které jsou nejbližší hodnotě původního analogového signálu. Počet kvantifikovaných hodnot v segmentu nemusí být konstantní. Zejména pro nižší velikosti signálu se používá jemnější stupnice, zatímco pro segmenty s vyšší velikostí amplitudy se používá hrubší dělení, neboť snižuje poměr šumu k signálu (SNR, Signal to Noise Ratio). Obecně platí, že jemnější kvantifikace vzorků vede k menšímu zkreslení signálu. Mezi techniky kvantifikace vzorků patří lineární (uniformní) kvantifikace, logaritmická kvantifikace či metody μ – law (US, Kanada, Japonsko) a α – law (ostatní země).

Kódování hodnot a komprese – Kódování hodnot slouží k převodu kvantifikovaných hodnot (obvykle dekadických) na binární vyjádření. Společně s tím dochází ke kompresi hlasových dat. Kódovací algoritmy (kodeky) lze rozdělit do dvou základních skupiny – na kodeky používající *kódování ve tvaru vlny* (waveform algorithms) a na kodeky využívající *kódování parametrů zdroje řečového signálu* (source algorithm). U kódování ve tvaru vlny používá vzorkování 8000 vzorků za sekundu. Redukce objemu dat se dosahuje prediktivní diferenční kompresí, která má velký dopad na kvalitu hlasu. Mezi algoritmy využívající kódování ve tvaru vlny patří kodeky PCM, DPCM (diferenční PCM) či ADPCM (adaptivní diferenční PCM). Algoritmy využívající kódování parametrů zdroje signálu, někdy se jim říká hlasové kodeky (vocoders), používají specifické kompresní schéma optimalizované pro kódování lidské řeči LPC (Linear Predictive Coding). Mezi kodeky využívající tyto algoritmy patří například APC (Adaptive Predictive

Coding), CELP (Code Excitation Linear Prediction), LD-CELP (Low Delay CELP) či CS-ACELP (Conjugate Structure Algebraic CELP). Speciální typem je kódování MPMLQ (Multipulse Maximum Likelihood Quantization, ITU-T G.723.1), viz 7.1.

ITU standard	Algoritmus	Kódovací rychlost
G.711	PCM	64 kb/s (2 x 4 kHz) x 8 bitů/vzorek
G.726r32	ADPCM	32 kb/s (2 x 4 kHz) x 4 bitů/vzorek
G.726r24	ADPCM	24 kb/s (2 x 4 kHz) x 3 bitů/vzorek
G.726r16	ADPCM	16 kb/s (2 x 4 kHz) x 2 bitů/vzorek
G.728	LD-CELP	16 kb/s (2 x 4 kHz) x 2 bitů/vzorek
G.729	CS-ACELP	8 kb/s (2 x 4 kHz) x 2 bitů/vzorek
G.729A	CS-ACELP	8 kb/s (2 x 4 kHz) x 2 bitů/vzorek
G.723r63	MPMLQ	6.3 kb/s
G.723r53	MPMLQ	5.3 kb/s

Tabulka 7.1: Typy kodeků

Velikost hlasového vzorku (sample) ovlivňuje potřebné přenosové pásmo. Hlasový vzorek je definován jako digitální výstup z kodeku (DSP), který se vkládá do protokolové jednotky PDU. Zařízení od firmy Cisco používají kodeky, které snímají hlas po deseti milisekundách a zapouzdříjí 20 ms hlasu do každé jednotky PDU bez ohledu na kodek. Pokud zapouzdříme více vzorků do PDU, redukuje se šířku potřebného přenosového pásma. Nicméně příliš velké PDU mohou způsobit zpoždění a problémy při ztrátě. Například pro kódování G.711 dostáváme $0.020 \times 64.000/8 = 160$ bytů na vzorek. Pokud uvažujeme přenos 160-bytového vzorku po Ethernetu s režii hlavičky Ethernetu (18 B), IP (40 B) a UDP (20 B), tj. celkem 78 B, pak dostaneme potřebné přenosové pásmo $(78 + 160)/160 \times 64.000 = 95200$ b/s, což je cca 93 kb/s.

Proces převodu digitálního signálu na analogový zahrnuje dekódování, tj. převod kódovaného vyjádření vzorků na velikost amplitudy, a filtrování, které slouží k rekonstrukci signálu na původní analogový signál. Při převodu hlasu na pakety se obvykle používá pulzní kódová modulace (PCM) popsaná standardem G.711. Analogový signál je vzorkován 8000 krát za sekundu, přičemž každý vzorek může nabývat jedné z 256 hodnot, tj. 8 bitů 8000 krát za sekundu, což je 64 000 bitů za sekundu. Proto je rychlost datového kanálu B v ISDN 64 kbps. Tato hodnota je velmi kvalitní, pro lidské ucho stačí vzorkovat hlas i v nižší kvalitě. Dostatečná kvalita dosahuje přenosové kapacity 6 kbps.

7.3.1 Faktory ovlivňující kvalitu hlasových přenosů

Narozdíl od klasických datových přenosů je kvalita přenosu u IP telefonie kritická. Pokud IP telefonie nebude pracovat bezchybně a spolehlivě jako klasická telefonní síť, uživatelé ji odmítnou. Například pokud uživatel zvedne sluchátko a neuslyší vyzváněcí tón, nemůže vytočit číslo či nerozumí volanému kvůli nízké kvalitě hovoru, taková činnost je pro komerční organizace nepřijatelná. Proto při přechodu na IP telefonii potřeba brát v úvahu kvalitu a nároky systému VoIP.

U hlasových služeb není zásadní přenosová kapacita – nároky jsou poměrně nízké a dají se snížit volbou kodeku. Ani spolehlivost doručení není kritická, neboť existují opravné mecha-

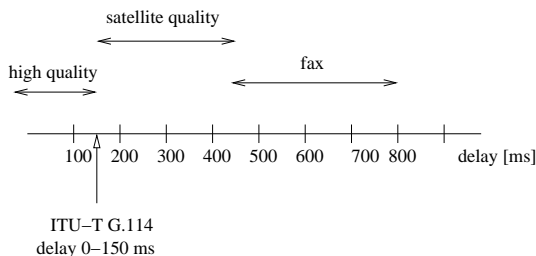
nismy při rekonstrukci signálu, které občasný výpadek paketu nahradí. Zásadní je včasnost doručení (zpoždění) a rozptýl mezi dobou doručení jednotlivých paketů (jitter). Díky tomu není vhodný protokol TCP, ale používá se UDP s nízkou režii při přenosu. Mezi vlivy, které ovlivňují kvalitu hlasového přenosu přes IP, patří následující faktory:

ozvěna (echo) – echo vzniká nežádoucím odrazem hlasového signálu zpět k volajícímu, kde působí rušivě. Akustické echo vzniká částečným přenosem hlasu ze sluchátka do mikrofonu, elektrické echo může vzniknout v místě přechodu čtyřdrátového vedení na dvoudrátové, např. v místech připojení analogového telefonu. Potlačení echa obvykle umožňují mikrofony s potlačovači echa v moderních telefonních přístrojích, případně dochází k eliminaci echa v DSP.

ztrátovost (packet loss) – ke ztrátě paketů může dojít, pokud je síť nestabilní, dochází k zahlcení nebo k různým zpožděním na síti. Velmi důležitá je i volba kodeku, neboť kodeky mají různou odolnost proti ztrátám. Tato odolnost se vyjadřuje faktorem Bpl (Packet-loss Robustness Factor) dle ITU-T G.113. Malé výpadky paketů mohou být rekonstruovány kodeky, které používají algoritmy PLC (Packet Loss Concealment) pro potlačení ztrát. Některé tyto algoritmy jsou součástí kodeků, například kodeky G.723.1, G.728 či G.729 používají lineární predikci CELP (Code Excited Linear Prediction). Ztráty okolo 3% u G.729 pocítí uživatel snížením hodnoty MOS o 0.5. Pokud dojde ke ztrátě souvislé řady hlasových paketů, rekonstrukce většinou není možná a dochází k výpadkům hlasového signálu.

zpoždění (delay) – zpožděním se podle standardu ITU-T G.114 rozumí čas, který uplyne mezi okamžikem promluvy volajícího a okamžikem vyslechnutí volaného. Zpoždění může být pevné či proměnlivé. Pro jeho eliminaci se používá prioritizace hlasových paketů, například pomocí nastavení hodnoty QoS v IP datagramu.

Pevné zpoždění je tvořeno dobou šíření signálu, zpožděním při kódování hlasu a čas strávený tvorbou paketu různými kodeky. Pevné zpoždění by mělo být podle standardu IT G.114 menší než do 150 ms. Při větším zpoždění se výrazně zhoršuje kvalita hlasového přenosu. Pokud je zpoždění větší než 400 ms, nelze hovor dále provozovat. Pro jiné telekomunikační služby (satelit, fax), nemusí být větší zpoždění tak kritické viz obrázek 7.3.



Obrázek 7.3: Zpoždění

Proměnlivé zpoždění je způsobeno zdržením ve výstupních frontách či serializací, která závisí na velikosti paketu a rychlosti linky. Může se pak stát, že krátké hlasové pakety musejí čekat, než se zpracuje jeden dlouhý paket na lince dané rychlosti.

kolísání zpoždění (jitter) se obvykle definuje jako doba mezi očekávaným a skutečným příchodem paketu. Toto zpoždění vzniká během přenosu. Obecně se každý IP datagram posílá nezávisle na ostatních, tzn. po sobě jdoucí datagramy mohou jít jinou cestou. Pokud toto nastane, vzniká jitter. Jitter lze eliminovat použitím vyrovnávacích bufferů na straně příjemce, který pomocí adaptivního algoritmu přizpůsobuje svou velikost aktuální situaci v síti.

kodek – volba kodeku hraje výraznou roli v kvalitě přenášených telefoních hovorů. Algoritmus kódování a dekódování analogového signálu na digitální určuje, kolik hlasových informací bude vloženo do digitálního signálu. S tím souvisí i kvalita hovoru (tab. 7.3), o které si více povíme v další kapitole. Volba kodeku také přímo souvisí s požadavky na šířku přenosového pásma, viz tabulka 7.1.

Obecně lze kvalitu hlasových přenosů nad IP řešit pomocí nastavení kvality služeb QoS v IP datagramech a prioritizací těchto datagramů. Lze použít různé techniky pro obsluhu front a předcházení zhlacení, je možné fragmentovat velké pakety či komprimovat hlasové pakety. Pro zajištění potřebného přenosového pásma se používají Integrované služby (IntServ) či Differenciované služby (DiffServ), které garantují nezbytnou propustnost hlasových přenosů.

7.4 Hodnocení kvality přenosu

V předchozí části jsme se bavili o faktorech, které ovlivňují kvalitu přenosu hlasového signálu. Zmínili jsme echo, zpoždění, ztráty paketů, jitter a další. Uživatelé IP telefonie však tyto charakteristiky obvykle nezajímají, pro ně je důležité, zda se dá telefonování přes IP reálně použít a jaká jsou možností v místě jeho připojení. Z tohoto důvodu existuje několik metod, které umožňuje vyhodnotit kvalitu přenosu hlasového signálu a kvantifikovat ji na předem dané stupnici. V praxi se používají dvě základní metody – metoda absolutního hodnocení ACR (Absolute Category Rating), která je postavena na hodnocení kvality poslechem, a dále metoda využívající E-model, která vypočítává se zadaných parametrů hodnotu tzv. R-faktoru.

7.4.1 Metoda absolutního hodnocení ACR, parametr MOS

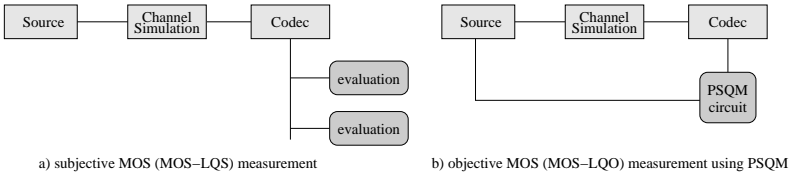
Metoda absolutního hodnocení ACR (Absolute Category Rating) je založena na statistickém měření kvality hlasového přenosu, která pro vlastní hodnocení využívá subjektivního ohodnocení posluchači. Tato metoda je nezávislá na jednotlivých parametrech zkreslení, ať už jsou to různé typy kodeků, ztrátovost paketů, šum či chybovost přenosu.

Metoda je založena na hodnocení skupiny na sobě nezávislých vět při přenosu telekomunikačním zařízením posluchači. Testované vzorky se hodnotí z různých hledisek. Jedním z nich je poslechová kvalita, která se ohodnocuje veličinou MOS (Mean Opinion Score). MOS je hodnota z pětibodové stupnice dané ITU-T P.800, viz tab. 7.2, pomocí které testovací subjekty vyjadřují své hodnocení. MOS může být získán subjektivně, kdy je hodnocení ponecháno na dojmu člověků (MOS-LQS, MOS Listening Quality Subjective), nebo pomocí tzv. intrusivních metod, které porovnávají původní a přenesený vzorek pomocí vhodného algoritmu.

MOS	Kvalita řeči	Stupeň zkreslení
5	vynikající	nepostřehnutelné
4	dobrá	postřehnutelné, málo znepokojující
3	průměrná	postřehnutelné, více znepokojující
2	slabá	znepokojující, ještě přijatelné
1	neuspokojivá	velmi znepokojující, nepřijatelné

Tabulka 7.2: Hodnoty MOS (Mean Opinion Score) podle ITU-T P.800

Tyto metody jsou objektivní a jejím výsledkem je hodnota MOS-LQO (MOS Listening Quality Objective). Mezi používané intrusivní techniky patří PSQM (Perceptual Speech Quality Measurement, ITU-T P.861), která testovaný vstup srovnává s referečním vstupem měřeném na kognitivním modelu, případně metoda PESQ (Perceptual Evaluation of Speech Quality, ITU-T P.862), která bere v úvahu i zpoždění a ztráty. Schéma použití metody absolutního hodnocení ACR pro (a) subjektivní či (b) objektivní výpočet MOS je znázorněn na obrázku 7.4.



Obrázek 7.4: Metoda absolutního hodnocení poslechu hodnotou MOS

Hodnoty MOS pro základní typy kodeků a rychlost vzorkování podle [1] jsou vedeny v tabulce 7.3. Z uvedené tabulky vyplývá, že pokud máme dostatečné přenosové pásmo, tak

Standard	Algoritmus	MIPS	rychlost [kbps]	MOS
G.711	PCM	0	64	4.1
G.726	ADPCM	1	32	3.85
G.728	LD-CELP	30	16	3.61
GSM 06.10	RPE-LTP	10	13	3.5
G.729	CS-ACELP	20	8	3.92
G.723.1	MP-MLQ	16	6.3	3.9
G.723.1	ACELP	20	5.3	3.64

Tabulka 7.3: Algoritmy pro kódování a ohodnocení kvalitou MOS

použití kodeku G.711 (PCM) je nejlepší volbou z pohledu kvality přenosu.

7.4.2 E-model a R-faktor

Měření kvality hovorů pomocí E-modelu patří mezi objektivní metody, která zohledňuje při výpočtu vliv šumu, hlasitosti, kvantizačního zkreslení, způsobu kódování, ozvěny, zpoždění apod. Autorem E-model (Ear) byl v roce 1998 švédský expert Nils Olaf Johannesson, jehož myšlenka se pak stala součástí doporučení ITU-T G.107 vydaného roku 2000 s několika pozdějšími aktualizacemi. Pomocí E-modelu se vypočítá hodnota R (R-faktor), která označuje kvalitu v hovoru v rozsahu 0 až 100, viz tabulka 7.4. Vlastní výpočet hodnoty R-faktoru

R-faktor	MOS	GoB [%]	Spokojenost uživatele
100-90	4.5 - 4.34	100 -97	velmi spokojený uživatel
90-80	4.34 - 4.03	97-89	spokojený
80-70	4.03 - 3.60	89-73	někteří uživatelé nespokojeni
70-60	3.60 - 3.10	73-50	mnoho uživatelů nespokojeno
60-50	3.10 - 2.58	50 -27	téměř všichni uživatelé nespokojeni

Tabulka 7.4: E-model podle ITU G.107 [2]

se odvíjí od základní hodnoty R_0 reprezentující odstup signálu od šumu (SNR, Signal-Noise Ration) původního signálu. Od této hodnoty se odčítají rušivé vlivy ovlivňující kvalitu přenosu a připočítává se faktor zvýhodnění A :

$$R = R_0 - I_s - I_d - I_e + A$$

R_0 – základní hodnota odvozená z vysílaného poměru signál/šum (SNR), která zahrnuje zdroje rušení jako např. např. šum na obvodu, šum v místnosti.

Simultaneous Impairment I_s – parametr lineárního zkreslení (pokles úrovně signálu a šum). Obsahuje kombinace různých zkreslení, které se objeví s hlasových signálem.

Delay Impairment I_d – vyjadřuje zkreslení způsobené zpožděním a echo.

Equipment Impairment I_e – vyjadřuje vliv použitého zařízení na kvalitu hlasového signálu, zejména zkreslení způsobené při kódování signálu (např. malá vzorkovací frekvence kodeku). Tyto hodnoty jsou dané doporučení G.113 [3], výběr je v tabulce 7.5.

Advantage Factor A – zohledňuje výhody a může nabývat hodnot 0 až 20. Standard G.107 navrhuje prozatímní hodnoty faktoru A , které závisí na typu přenosového média: konvenční sítě ($A=0$), mobilní sítě v rámci budovy ($A=5$), mobilní sítě v geografické oblasti, případně v pohybujícím se vozidle ($A=10$), nedostupné lokality, případně satelitní spojení ($A=20$) [2].

Hodnoty R-faktoru a jejich vztah k hodnotě MOS je popsán v tabulce 7.4. Pokud je hodnota faktoru R menší než 50, nedoporučuje se provozovat telefonní spojení.

Výhodou E-modelu je, že výstupní hodnota R-faktoru se dá získávat monitorování jednotlivých parametrů na straně příjemce a narozdíl od intrusivních metod při výpočtu parametru MOS není potřeba žádné další speciální vybavení.

Obecně řečeno, pokud stavíme síť s podporou IP telefonie, je potřeba brát v úvahu faktory, které ovlivňují kvalitu hlasového hovoru. Některé vlivy se dají potlačit vhodnou volbou

Kodek	Rychlost kódování (kb/s)	Hodnota I_e
PCM	64	0
ADPCM	40	2
ADPCM	32	7
ADPCM	24	25
ADPCM	16	50
LD-CELP	16	7
CS-ACELP	8	10
VSELP	8	20
ACELP	7.4	10
MP-MLQ	6.3	15

Tabulka 7.5: Hodnoty I_e pro výpočet R-faktoru podle ITU-T G.113 [3]

kodeku, což je záležitost aplikační vrstvy modelu OSI. Pro zajištění dostatečného přenosového pásma na úrovni vrstvy L2 se používá technologie VLAN (Virtual LAN) pro odlišení hlasového přenosu od ostatních dat a pro prioritizaci paketů. Ethernet narozdíl od standardu IP nerozlišuje prioritu přenášených rámců. Z tohoto důvodu standard 802.1p přináší rozšíření Ethernetového rámce o tři prioritní bity, které umožňují vložit do rámce doporučenou třídu kvality (QoS) z IP hlavičky. Pro hlasový přenos doporučuje firma Cisco použít třídu 5 pro přenos hlasu (hodnota DSCP 46) a pro signalizaci třídu 3 (DSCP 26).

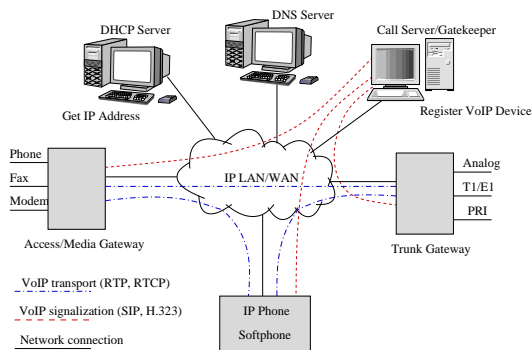
7.5 IP telefonie

Jak už jsme si řekli, systém IP telefonie se zaměřuje na integraci (konvergenci) datových a hlasových služeb na platformu IP přenosů. IP telefonie tedy přenáší hlas prostřednictvím datových sítí. Kombinací datové a telefonní sítě snižuje IP telefonie náklady na vytvoření, údržbu a rozšiřování sítí. Umožňuje také centrální správu systému, podporuje mobilitu účastníků a další pokročilé služby. Hlavní předností IP telefonie je ovšem snížení nákladů na telefonování, zejména při komunikaci mezi vzdálenými pobočkami.

V předchozích kapitolách jsme si ukázali architekturu klasické telefonní sítě, seznámili se s oblastí převodu hlasového signálu na digitální signál a prošli základní požadavky hlasového přenosu na kvalitu přenosových cest nad IP. Nyní se podíváme na základní architekturu IP telefonie, popis základních signalizačních protokolů SIP a H.323, a také na spolupráci IP telefonie s dalšími síťovými službami.

7.5.1 Architektura IP telefonie

Základní komponenty IP telefonie jsou podobné jako u klasické telefonie: tvoří ji telefony, ústředny (gatekeeper, SIP server) a brány do analogové sítě (gateways). Tyto základní prvky jsou propojeny klasickou síťovou infrastrukturou nad IP. Pro uskutečnění hovoru je potřeba pomocí signalizačního protokolu (např. SIP či H.323) navázat spojení s ústřednou a spojit se s volanou stanicí. Následně se vytvoří spojení mezi komunikujícími účastníky a probíhá přenos hlasu transportním protokolem RTP. Základní schéma architektury je na obrázku 7.5. Mezi hlavní stavební komponenty IP telefonie patří:



Obrázek 7.5: Architektura IP telefonie

IP telefon je koncové zařízení, které umožňuje uživateli připojit se do IP telefonní sítě a volat. IP telefon může být buď hardwarový nebo softwarový. Podle typu signalizačního protokolu může IP telefon podporovat signalizaci SIP, H.323, SCCP apod. Narozdíl od klasických analogových telefonů nabízí IP telefon rozšířené služby, například možnost dynamického přidělování profilů podle jména přihlášeného uživatele, interaktivní telefonní seznam (např. s využitím služby LDAP), možnosti komunikace s Internetem přes WWW, vyhledávání informací či prezenční služby (Jabber), apod.

Ústředna (gatekeeper) je síťové zařízení nebo aplikace, která zajišťuje řízení přístup do IP telefonní sítě včetně registrace uživatele a nastavení jeho konfigurace. Dále zajišťuje vyhledávání volaných účastníků, směrování hovorů, přidělování přenosového pásma, překlad adres apod. Příkladem ústředny může být například Asterisk (SIP server), Cisco Call Manager, apod.

Brána (gateway) je zařízení nebo aplikace, která slouží k propojení IP telefonní sítě s veřejnou telefonní sítí PSTN, místní pobočkovou ústřednou PBX či rozhraním do mobilní sítě typu GSM. Umožňuje také propojovat různé IP telefonní sítě, které používají jinou signalizaci (brána SIP/H.323). Bránu je možné implementovat jako součást ústředny (gatekeeper), například ve formě přídatné hardwarové karty a odpovídajícího softwarového rozšíření. Například softwarová ústředna Asterisk umožňuje předávat hovory mezi sítěmi H.323 a SIP, případně připojení k síti PSTN.

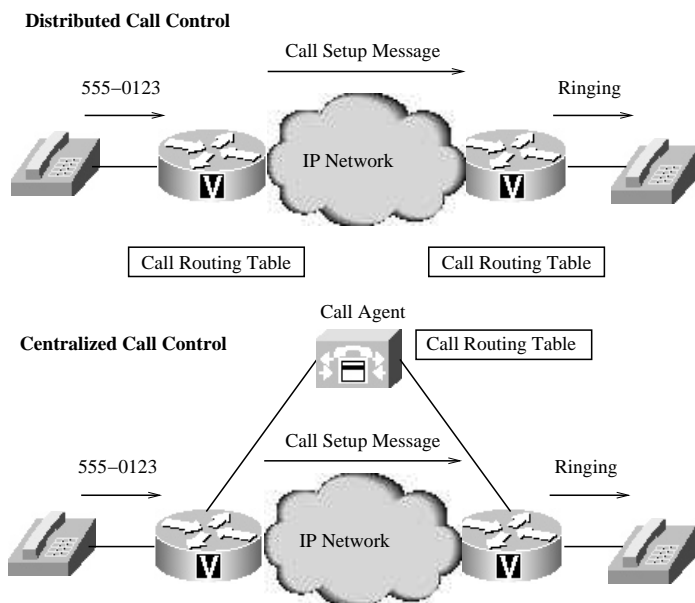
MCU (Multipoint Control Unit) je řídicí jednotka pro komunikaci více koncových zařízení a bran. MCU poskytuje prostředky pro vytváření vícebodové konference. Skládá se z vícebodového řadiče MC (Multipoint Controller) pro řízení koncových zařízení a vícebodového procesoru (Multipoint Processor), který provádí mixování, přepínání a další zpracování datových toků v MCU.

Aplikační servery nabízejí další služby pro uživatele IP telefonie. Patří tam například správa identit, hlasová pošta, přidělování IP adres (DHCP server), překlad ENUM (DNS server), server pro IM (Jabber), XML/WWW služby a další.

7.5.1.1 Řízení hovorů

Základní činností IP telefonní sítě je řízení hovorů prostřednictvím signalizačních protokolů. Řízení se se skládá z (1) vytvoření spojení, (2) údržbu telefonního kanálu a (3) ukončení hovoru napříč IP telefonní sítí. Vytvoření hovoru zahrnuje vyhledání volaného účastníka, což může být lokální uživatel na stejné ústředně nebo uživatel připojen k jiné ústředně. Dále je potřeba zajistit přenosovou kapacitu pro daný hovor. Pokud je pásmo k dispozici, začne se vytvářet spojení. V opačném případě vrátí ústředna obsazovací signál. Po úspěšném navázání telefonního spojení přes IP je potřeba sledovat kvalitu přenosu a vhodně zareagovat, pokud by se výrazně snížila. Pro zjištění kvality se sleduje počet přenesených paketů, ztrátovost či zpoždění. Pokud je hovor ukončen, zajistí signalizace oznámení o ukončení hovoru a uvolnění všech alokovaných výpočetních zdrojů.

Řízení hovorů může být distribuované nebo centrální (obr. 7.6). U distribuovaného řízení se na řízení podílí více zařízení (ústředně). Ústředna nabízí pro své uživatele základní vyvolávací tón, zpracovává požadavek od telefonu (vytáčená čísla), směruje hovor, dohlíží nad ním a řídí ukončení hovoru. Každá ústředna musí obsahovat svou směrovací tabulku pro směrování hovorů do dalších sítí (nejedná se však o směrovací IP tabulku, kterou využívají směrovací protokoly typu RIP, OSPF, apod.). Protože se tato tabulka aktualizuje manuálně, jsou sítě založené na distribuovaném řízení hovorů méně rozšiřitelné. Příkladem může být směrovač s podporou VoIP (např. Cisco ISR s IOS Voice). U centrálního řízení existuje jedno

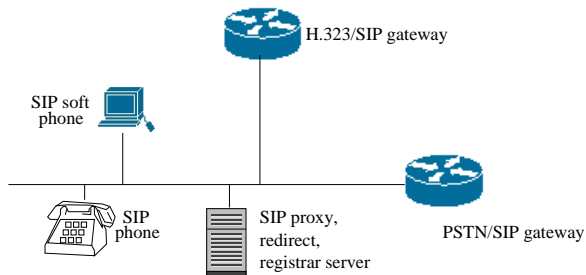


Obrázek 7.6: Distribuované a centrální řízení VoIP hovorů

zařízení (aplikace), které řídí signalizaci a zpracovává žádosti o spojení v celé síti. Ostatní síťová zařízení přeměňují žádosti o spojení na tento centrální prvek například pomocí signalizačního protokolu MGCP. Výhodou centrálního řízení hovorů je soustředění směrovací tabulky a informací o uživateli na jeden síťový prvek. Nevýhodou může být přetížení tohoto prvku a ztráta telefonního spojení v celé síti v případě výpadku centrálního prvku. Příkladem může být Cisco Call Manager.

7.5.2 Signalizace SIP

Protokol SIP (Session Initiation Protocol) (IETF standard RFC 3261 [4]) je signalizační protokol pro IP telefonii. Jedná se o textový protokol s hlavičkami podobnými protokolu HTTP. Komunikace SIP probíhá nad transportním protokolem UDP. Jeho hlavním úkolem je autentizace a autorizace uživatele, sestavení spojení, řízení hovoru a ukončení spojení. Pro adresování používá SIP speciální identifikátor URI ve tvaru `sip:user@domain`. SIP neprovádí rezervaci zdrojů pro přenos ani nastavení parametrů QoS. Architektura SIP je vyvíjívala několik komponent (obrázek 7.7):



Obrázek 7.7: Architektura systému SIP

SIP Server (UAS, User Agent Server) — server, který obsluhuje požadavky uživatele na spojení. Server je obvykle tvořen několika částmi s následujícími funkcemi:

Registrační server (Registrar) provádí registraci uživatele (autentizaci a autorizaci). Informace o registraci se používá pro směrování hovorů určených pro daného uživatele. Provádí aktualizaci dat na lokalizačním serveru.

Server pro směrování hovorů (Redirect Server) informuje klienta o dalším skoku (hop) při připojování na alternativní SIP server.

Lokalizační server (Location Server) slouží k vyhledání volaného klienta. Spolupracuje s registračním serverem, se kterým si vyměňuje informace o aktuálně připojených uživateli.

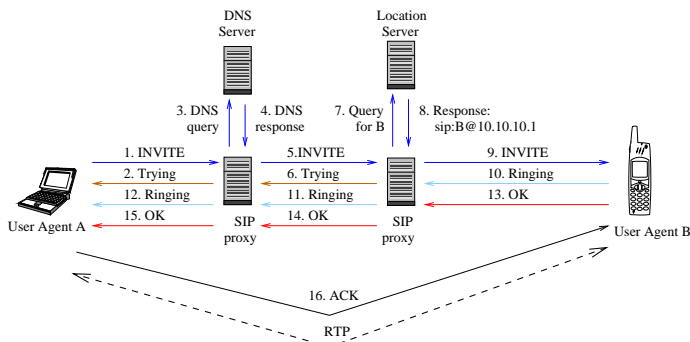
Proxy server analyzuje správy SIP, přepisuje hlavičky a přeposílá zprávy na další SIP server.

SIP telefon (UAC, User Agent Client) je softwarová aplikace či hardwarové zařízení, která iniciuje SIP požadavek. Po vytvoření spojení posílá hlasová data transportním protokolem RTP volanému účastníkovi. Jedná se o koncové komunikační zařízení.

Při vzniku spojení dojednávají protokolu SIP mezi volajícím a volaným důležité parametry spojení, například kódování. Slouží také k určení IP adresy volaného (dotazem na lokalizační server). Během hovoru řídí přenos hovoru, zapojení více účastníků, případně změnu kódování.

Vlastní komunikace se skládá ze dvou základních akcí – (1) registrace klienta a (2) ustavení spojení. Při registraci se vytvoří vazba mezi konkrétní polohou klienta (což je přidělená adresa) a jeho identifikátor v doméně SIP (SIP adresa klienta, ve tvaru URI). Při registraci posílá SIP klient (UAC) příkazem **REGISTER** žádost na registrační server. V příkazu uvádí svou IP adresu a port, na kterém bude komunikovat. Registrační server zkontroluje uživatelský účet a uloží si identifikaci do databáze přihlášených uživatelů. Následující ukázka je popisuje příklad registrace uživatele `sip:matouspcesnet.cz` z IP adresy 147.229.14.140. Z výpisu je také vidět, že uživatel používá softwarového klienta SJphone a pro autorizaci používá zabezpečenou komunikaci pomocí MD5 hashe.

```
REGISTER sip:cesnet.cz SIP/2.0
Via: SIP/2.0/UDP 147.229.14.140;rport;branch=z9hG4bK93e50
Content-Length: 0
Contact: <sip:matousp@147.229.14.140:5060>
Call-ID: F7A05113-86CF-4DC3-BCA1-7C93577C85B8@147.229.14.140
CSeq: 2 REGISTER
From: <sip:matousp@cesnet.cz>;tag=3378934327100
Max-Forwards: 70
To: <sip:matousp@cesnet.cz>
User-Agent: SJphone/1.60.289a (SJ Labs)
Authorization: Digest username="matousp",realm="cesnet.cz",nonce="45537121970525"
```



Obrázek 7.8: Vytvoření spojení pomocí SIP

Po přihlášení následuje ustavení spojení pomocí příkazu **INVITE**. V příkaze můžeme vidět adresu volaného ve tvaru `sip:541141108cesnet.cz`. Pokud je uživatel přihlášený, přepośle SIP server žádost na telefon volanému. Volajícímu vrací server potvrzení, viz obrázek 7.8.

```

INVITE sip:541141108@cesnet.cz SIP/2.0
Via: SIP/2.0/UDP 147.229.14.140;rport;branch=z9hG4bK93e50
Content-Length: 341
Contact: <sip:matousp@147.229.14.140:5060>
Call-ID: 3F40CC00-8686-4CA8-BCC7-5EA1FD4278FF@147.229.14.140
Content-Type: application/sdp
CSeq: 1 INVITE
From: "Petr Matousek"<sip:matousp@cesnet.cz>;tag=2557596820844
Max-Forwards: 70
To: <sip:541141108@cesnet.cz>
User-Agent: SJphone/1.60.289a (SJLabs)

```

```

SIP/2.0 200 OK
Via: SIP/2.0/UDP 147.229.14.140;rport=5060;branch=z9hG4bK93e50
Call-ID: DDFC972C-CD2E-43C3-8AC6-54B0A684AEDA@127.0.0.1
CSeq: 40 OPTIONS
From: <sip:matousp@cesnet.cz>;tag=2557631219784
To: <sip:cesnet.cz>;tag=c10ed4fff3e6fb17efd0bfbdcce87ce2.884e
Accept-Language: en
Server: Sip EXpress router (0.9.5-pre1 (i386/linux))
Content-Length: 0

```

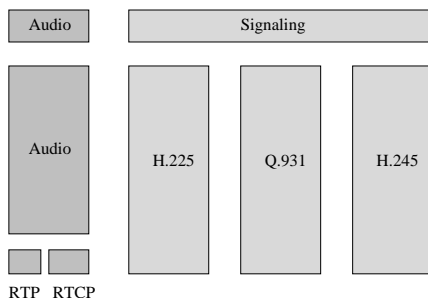
Příklad ustavení spojení pomocí protokolu SIP je na obrázku 7.8. V obrázku můžeme vidět i spolupráci s dalšími servery, například s DNS serverem při překladu vytáčeného telefonního čísla na SIP adresu (tzv. ENUM, viz dále) či lokalizaci uživatele pomocí lokalizačního SIP serveru. Pokud se podaří navázat spojení mezi komunikujícími stranami, vytvoření se posléze datový (transportní) kanál RTP mezi oběma komunikujícími účastníky. Tento kanál je navázán přímo, tzn. bez účasti SIP serverů. Další příkazy signalizačního protokolu SIP jsou uvedeny v tabulce 7.6.

Příkaz	Popis
REGISTER	UAC zasílá SIP serveru oznámení o své pozici
INVITE	UAC posílá žádost o zahájení relace
ACK	potvrzení úspěšného ustavení relace
BYE	zrušení relace, která je v provozu
CANCEL	zrušení relace, která ještě nebyla ustavena
OPTIONS	zjištění možností nastavení UAC
MESSAGE	posílání textových zpráv Instant Messaging (IM) (RFC 3428)
SUBSCRIBE, NOTIFY	přihlášení k odběru informací (prezence) (RFC 3265)
REFER	požadavek jiného UAC na předání relace (RFC 3515)

Tabulka 7.6: Příkazy signalizačního protokolu SIP

7.5.3 Signalizace H.323

Standard pro přenos hlasu nad IP H.323 je původně odvozen od standardu pro vytváření videokonferencí H.320 (přenos nad ISDN). Standard popisuje způsob přenosu audia a video data nad různými komunikačními technologiemi. Standard byl přijat v roce 1996 organizací ITU (International Telecommunication Union). Skládá se z více standardů, z nichž pro IP te-



Obrázek 7.9: Signalizace H.323 u IP telefonii

lefonii jsou nejdůležitější standardy H.245 pro výměnu parametrů přenosového kanálu, H.225 pro vytvoření hovoru, řízení registrace, přístupu a stavu hovoru (RAS, Registration, Admission, Status) a dále standard Q.931, který zajišťuje interoperabilitu s analogovými službami a veřejnou telefonní sítí PSTN, viz obrázek 7.9. Protokoly H.323 jsou binární a pracují nad TCP, narozdíl od SIPu, který je textový a pracuje nad UDP.

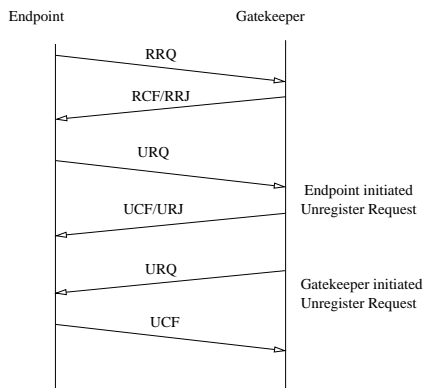
Jak jsme si ukázali, standard H.323 v sobě obsahuje více standardů a protokolů, které plní různé funkce. Mezi základní protokoly patří:

H.225 – Signalizace volání H.225 pracující nad TCP slouží pro navázání spojení mezi dvěma koncovými body na základě výměny zpráv po signalizačním kanálu, a to buď přímo mezi koncovými zařízeními (bez ústředny), nebo mezi koncovými zařízeními a ústřednou (gatekeeper). Součástí definice H.225.0 je signalizace RAS pro registraci a Q.931 pro správu hovorů v sítích PSTN.

RAS (Registration, Admission, Status) je protokol, který zabezpečuje komunikaci mezi koncovými zařízeními (terminál, brána) a ústřednou (gatekeeper) pro zajištění správy, registrace, administrace a stavu. Příklad registrace je na obrázku 7.10. Některé příkazy protokolu RAS jsou uvedeny v tabulce 7.7. Příkazy posílá buď ústředna (gatekeeper) nebo koncová stanice (terminal). Týkají se registrace, odregistrace, přístupu (admission), získání informací, získání lokace, žádost o přidělení přenosového pásma a žádost o uvolnění pásma.

Q.931 – Signalizace Q.931 (standard ISDN) vytváří signalizační kanál (call-signaling channel) s funkcemi pro inicializaci spojení mezi koncovými body (Setup), sestavení tohoto spojení (Call Proceeding), přeposlání vyzvánění (Alerting), vytvoření spojení (Connect), ukončení spojení (Release) a další funkce. Signalizační kanál Q.931 může být vytvořen přímo, pak se jedná o tzv. přímou signalizaci DRC (Direct Routed Call Signaling) nebo prostřednictvím ústředny, tzv. signalizaci GRC (Gatekeeper Routed Call Signaling), viz obr. 7.11. V obou případech však musí dojít k registraci přes RAS na ústředně.

H.245 je řídicí protokol pracuje, který pracuje nad TCP. Tento protokol slouží k vyjednávání nastavení a parametrů mezi koncovými body, případně koncovým bodem a ústřednou.



Obrázek 7.10: Registrace H.323 pomocí RAS

Příkaz		
Gatekeeper Request (GRQ)	Confirm (GCF)	Reject (GRJ)
Registration Request (RRQ)	Confirmation (RCF)	Reject (RRJ)
Unregister Request (URQ)	Confirmation (UCF)	Reject (URJ)
Admission Request (ARQ)	Confirmation (ACF)	Reject (ARJ)
Information Request (IRQ)	Response (IRR)	Status (IRS)
Location Request (LRQ)	Confirmation (LCF)	Reject (LRJ)
Bandwidth Request (BRQ)	Confirmation (BCF)	Reject (BRJ)
Disengage Request (DRQ)	Confirmation (DCF)	Reject (DRJ)

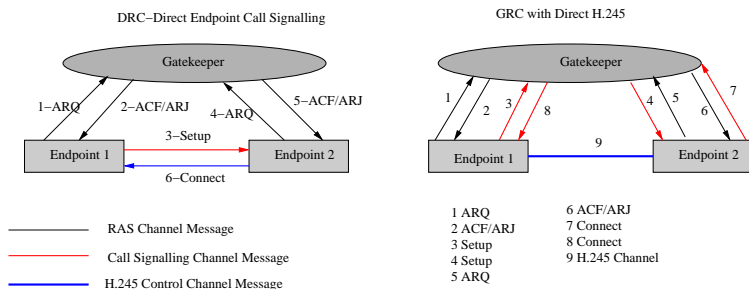
Tabulka 7.7: Registrační protokolu RAS

Mezi vyjednávané parametry patří například kodeky pro audio a video, nastavení časovačů a počítadel, režim přenosu a další. Mezi příkazy H.245 patří Request, Reponse, Command, Indication.

Výše uvedené protokoly slouží k vytváření spojení mezi koncovými stanicemi. Narozdíl od protokolu SIP, který pro všechny funkce používá jeden protokol SIP, standard H.323 rozlišuje protokol pro registraci (RAS), dohadování parametrů (H.245) a vytváření vlastního spojení (Q.931). H.323 dovoluje vytvořit přenosový hlasový kanál jednak mezi koncovými stanicemi (bez použití ústředny), mezi koncovou stanicí a ústřednou a mezi ústřednami. Pokud dochází ke komunikaci bez ústředny, není možné provádět registraci RAS a vysílající stanice musí mít v konfiguraci manuálně zadanou IP adresu cílového stanice.

Vytvoření spojení H.323 bez ústředny

1. Vysílající stanice (brána či terminál) vytvoří spojení H.225.0 s cílovou stanicí na portu TCP/1720.



Obrázek 7.11: Signalizace H.323

2. Spustí se procedura pro vytvoření signalizačního kanálu podle Q.931. Vytvoří se signalizační kanál.
3. Koncová bod vytvoří další kanál pro řízení H.245. Pomocí tohoto kanálu se dohodnou parametry spojení a vymění popis logického kanálu.
4. Podle popisu logického kanálu se vytvoří relace RTP mezi koncovými body.
5. Koncové body si začnou vyměňovat multimediální data pomocí RTP.

Druhá verze doporučení H.323 umožňuje zkrátit tuto proceduru tak, že místo bodu 2 a 3 se vytvoří pouze jeden kombinovaný kanál pro Q.931 a H.245. Tímto kanálem se dohodnou parametry spojení a vymění se popis logického kanálu. Toto řešení významně urychluje ustavení spojení mezi koncovým zařízeními.

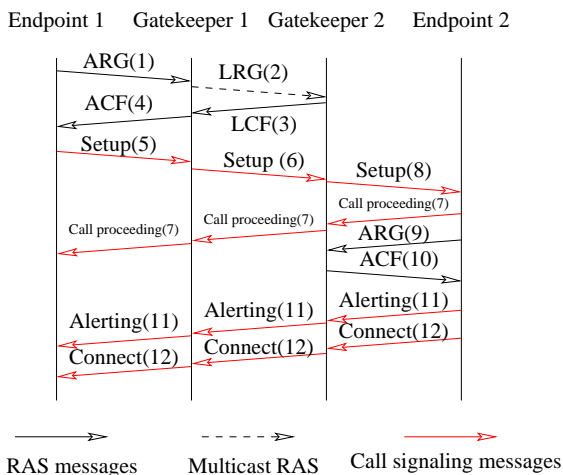
Vytvoření spojení prostřednictvím ústředny (gatekeeper)

Před vytváření spojení mezi koncovými uzly prostřednictvím ústředny, dochází předtím k registraci koncových bodů na ústředně. Proto, aby se mohl koncový bod registrovat, musí znát IP adresu ústředny. Ta může být buď manuálně nastavená v konfiguraci koncové stanice, nebo může koncový uzel vyslat multicastový příkaz GRQ (Gatekeeper Request) na adresu 224.0.1.41. Pokud koncová stanice obdrží odpověď GCF (Gatekeeper Confirmation), použije zaslanou IP adresu. Vlastní registrace na ústředně probíhá pomocí komunikace RAS (obr. 7.10). Pro identifikaci může stanice použít své ID a IP adresu. Poté následuje vytvoření spojení:

1. Vysílající stanice pošle žádost ARQ (Admission Request) na ústřednu.
2. Ústředna žádost potvrdí (ARC, Admission Confirmation) a pošle IP adresu volané stanice.
3. Vysílající stanice začne vytvářet spojení s volanou stanicí (příkaz Setup protokolu Q.931).
4. Volaná stanice se zaregistruje na ústředně (ARQ přes RAS) a zkontroluje svá práva.

5. Pokud proběhne úspěšně registrace volané stanice na ústředně (ACF), volaná stanice odpoví na žádost o vytvoření spojení přes Q.931.
6. Pomocí kanálu H.245 se vymění logické parametry spojení a otevře se transportní kanál RTP mezi koncovými stanicemi.
7. Přes RTP dochází k výměně multimediálních dat.

Příklad vytvoření spojení mezi koncovými zařízeními H.323 prostřednictvím ústředny je zakreslen na obrázku 7.12.



Obrázek 7.12: Vytvoření spojení pomocí signálního protokolu H.323

Základní komponenty systému H.323

Mezi základní prvky systému H.323 patří terminál, brána (gateway), ústředna (gatekeeper) a jednotka MCU.

Terminál je koncové zařízení na LAN, které vytváří dvoucestnou komunikaci v reálném čase s dalším terminálem, bránou či jednotkou MCU. Terminál musí podporovat H.245 pro vytvoření přenosového kanálu pro zvuk a video, H.225 pro signalizaci, RAS pro případnou komunikaci s ústřednou, RTP/RTCP pro zajištění správného pořadí audio/video paketů a kódování G.711 (PCM, 64 kbps).

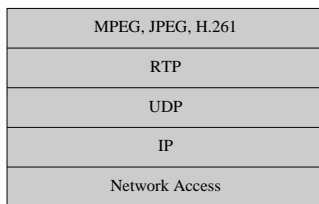
Ústředna (Gatekeeper) provádí služby řízení hovorů. Jedná se o nepovinnou část systému H.323. Poskytuje řízení přístupu, adresaci, účtování, přidělování pásma či řízení volání. Její implementace bývá součástí terminálu, jednotky MCU či brány H.323.

Brána (Gateway) je koncový bod v LAN, který propojuje terminály H.323 na LAN s dalšími branami H.323 či terminály ITU na WAN. Brána zajišťuje komunikaci s terminály v jiných sítích, například ISDN (H.320), ATM (H.321), GSTN (H.324), mobilní (H.323M) či SS7. Brána H.323 překládá komunikaci různými protokoly do sítě H.323. Pro řízení brány se používá protokol MEGACO/H.238 a H.248.1. Speciálním typem brány je *Proxy*, která navzájem propojuje různé relace H.323.

Jednotka MCU (Multipoint Control Unit) je řídicí jednotka pro komunikaci více terminálů a bran. MCU poskytuje prostředky pro vytváření vícebodové konference. Skládá se z vícebodového řadiče MC (Multipoint Controller) pro řízení terminálů a vícebodového procesoru (Multipoint Processor), který provádí mixování, přepínání a další zpracování datových toků v MCU.

7.5.4 Přenos hlasových dat pomocí RTP a RTCP

Protokol RTP (Real-Time Transport Protocol) [5] je transportní protokol pro přenos multimediálních dat (audio, video) zpracovávaných v reálném čase. RTP pracuje nad UDP, viz obr. 7.13. RTP obsahuje identifikaci typu obsahu, sekvenční čísla paketů, časové značky, podporuje



RTP Protocol Architecture

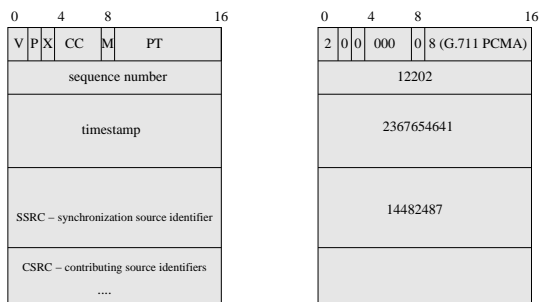
Obrázek 7.13: Architektura protokolu RTP

také monitorování přenosu. RTP neredukuje celkové zpoždění ani nezaručuje kvalitu služby QoS.

Paket RTP se skládá z hlavičky o velikosti 20B, viz obrázek 7.14 a z digitalizovaného hlasu (velikost 20 až 160 bytů, viz další kapitola). Celková velikost datového paketu na úrovni přenosové linky je větší, neboť se ještě připojuje hlavička UDP (8 bytů) a hlavička IP (20 bytů). U dvoubodových spojů se občas používá komprese záhlaví RTP (na velikost 2-4 bytů), což může ušetřit přenosové pásmo.

Standard RTP definuje také kontrolní protokol RTCP (RTP Control Protocol), který slouží k monitorování kvality distribuce dat a přenosu řídicích informací. RTCP sleduje počet přenesených paketů, počet ztracených paketů, zpoždění, kolísání rychlosti a další. RTCP přenáší pakety nejméně každých pět sekund a provádí zpětnou vazbu aktuálního stavu sítě. RTCP tedy slouží pro předávání řídicích informací pro tok RTP.

RTP je kritickou aplikací systému VoIP, protože umožňuje cílovému zařízení přeskldat a upravit hlasové pakety předtím, než jsou přehrány uživateli. Hlavička RTP obsahuje časové razítko a sekvenční číslo. Tyto informace mohou ve vyrovnávací paměť příjemce odstranit ko-



Obrázek 7.14: Hlavička protokolu RTP

lísání jitter a zpoždění tak, aby přehrávání záznamu bylo souvislé. Sekvenční čísla v hlavičce se používají pouze pro uspořádání. RTP znovu nepřenáší ztracené pakety.

7.6 ENUM – mapování telefonních čísel na URI

ENUM (tElephone NUmbers Mapping) je způsob mapování identifikátorů telefonních sítí ITU-T E.164 na jmenné identifikátory URI uložené v DNS. Tento způsob je definovaný ve standardu RFC 3761 [6]. Pro překlad se využívá systém DNS a speciální záznamy typu NAPTR. V kořenovém stromu doménových jmen existuje speciální doména **e164.arpa**, která obsahuje podstrom telefonních čísel, kde každé číslo tvoří jednu úroveň stromu. Správa jednotlivých podstromů je delegovaná na státy s příslušným mezinárodním volacím kódem. Například doménu 0.2.4.e164.arpa patříčí České republice spravuje registrátor NIC.CZ.

Při převodu telefonních čísel na subdoménu ve stromu e164.arpa v DNS se jednotlivé číslice telefonního čísla zapisují v opačném pořadí a oddělejí tečkou. Například telefonní číslo "+420 541 141 108" se namapuje na doménovou adresu 8.0.1.1.4.1.1.4.5.0.2.4.e164.arpa.

Pro tuto doménovou adresu pak existuje v DNS speciální záznam záznam NAPTR. Tento záznam obsahuje regulární výraz, která vrátí DNS server po dotazu na NAPTR záznam pro danou doménovou adresu. Aplikací regulárního výrazu na doménovou adresu získám adresu ve tvaru URI.

Následující příklad uvádí záznam pro překlad telefonních čísel ve formátu doménového jména podstromu e164.arpa na adresu SIP, případně H.323:

```
$ORIGIN 4.1.1.4.5.0.2.4.e164.arpa.
IN NAPTR 10 50 "u" "E2U+sip" "!~\\+(.*)$!sip:\\1@vutbr.cz!".
$ORIGIN 4.1.1.4.5.0.2.4.e164.arpa.
IN NAPTR 10 50 "u" "E2U+h323" "!~\\+(.*)$!h323:\\1@h323.vutbr.cz!".
```

Výsledkem překladu aplikace regulárního výrazu z DNS na telefonní číslo +420 541 141 108 bude SIP adresa sip:420541141108@vutbr.cz.

Překlad ENUM za pomoci dynamický záznamů s regulárními výrazy v DNS má velký význam pro propojování veřejné telefonní sítě a IP telefonie, zejména překladu telefonních čísel na SIP adresy a jejich následné směrování. Velkou výhodou těchto dynamických záznamů

je to, že není nutné mít v DNS samostatný záznam pro každé telefonní číslo, ale lze například sdružit všechny telefony se stejným předčíslem pod jeden záznam. Výše uvedený záznam umožňuje překlad jakéhokoli telefonního čísla s prefixem 420 541 14x xxx, čímž uspoříme cca $10^4 - 1$ položek v DNS.

7.7 Bezpečnost VoIP

Další důležitou otázkou související s architekturou sítí je bezpečnost. Bezpečností zde nechápeme nejen zajištění ochrany hlasových přenosů proti odposlechu či podvržení spojení, ale patří tam i dostupnost, která může být narušena určitým type útoku, například DoS (Denial of Service). Pokud má být IP telefonie rovnocenným partnerem klasické telefonie, musí nabízet i vysokou úroveň dostupnosti, což je u PSTN typicky 99,999% (znamená celkové výpadky maximálně do pěti minut za rok). Klasická telefonie PSTN používá vyhrazené přepojované okruhy (circuit-switched technology), což jednak garantuje šířku pásma a současně zajišťuje vysokou bezpečnost hovorů. Protože IP telefonie využívá pro přenos běžné datové sítě nad IP, musí zajistit spolehlivý přenos a bezpečnost jinými mechanismy. Zároveň je potřeba brát v úvahu velkou citlivost hlasových přenosů na zpoždění, ztrátovost či jitter, což klade další zvýšené nároky na implementaci zabezpečení. Mezi základní bezpečnostní hrozby IP telefonie patří [7]:

- Odposlech
V případě odposlechu je potřeba si uvědomit, že u IP telefonie se používají dva typy spojení (signalizační a transportní), kde jedno přenáší informace týkající se zajištění spojení a druhé vlastní hlasová data. Protože se pohybuje nad IP, může být každý datagram těchto toků směrován jiným způsobem. Z tohoto důvodu není jednoduché zachytit VoIP provoz kdekoli v síti, ale je potřeba mít přístup k síťovému prvku na cestě spojení, nejlépe na straně volajícího či na straně volaného. Proto k odposlechům může dojít spíše z vnitřní sítě než z vnějšího prostředí.
- Viry
- SPIT (SPam over Internet Telephony)
- Phishing – vishing (voice phishing), PHIT (Phishing over IP Telephony)
- Přerušení služby útokem typu DoS (Denial of Service) nebo DDoS (Distributed DoS)
- Neautorizované použití služby

Bezpečností VoIP se zabývá aliance VOIPSA (Voice over IP Security Alliance, www.voipsa.org), která detekuje různé typy útoků na VoIP a navrhuje způsoby zabezpečení. Rozdělení hrozeb a útoků na VoIP je možné najít například v [8]. Pro zabezpečení a zajištění spolehlivé VoIP komunikace lze použít následující technologie [9]:

Řízení přístupu k síťovému médium prostřednictvím 802.1x – Technologie 802.1x definuje rámec pro autentizaci klientských zařízení v lokálních sítích. Základem architektury je autentizační prvek (tzv. authenticator), což může být prepínač nebo přístupový bod WiFi, ke kterému se musí každý uživatel (tzv. supplicant) připojit a ověřit svou

identitu. Vlastní ověření může být oproti lokální databázi uživatelů na straně autentizačního prvku, většinou se však používá speciální autentizační server typu Radius či TACACS+. Autentizace a autorizace prostřednictvím 802.1x umožňuje sledovat všechna síťová zařízení na úrovni portů a řídit jejich přístup do různých částí sítě, například pomocí přidělení VLAN skupiny.

Oddělení hlasového a datové provozu pomocí VLAN je základním bezpečnostním postupem doporučeným pro provoz VoIP. Přestože hlasové přenosy chodí fyzicky po stejné kabeláži jako ostatní datový přenos, VLAN technologie umožňuje logicky oddělit jednotlivé rámce na úrovni Ethernetu a portů přepínače. Použití VLAN pro hlas má výhodu nejen pro zabezpečení, ale také pro zjednodušení návrhu sítě, správu konfigurace IP telefonie a zajištění dostatečného přenosového pásma pomocí QoS aplikovaných na vyhrazenou VLAN síť pro hlasové přenosy.

IPSec (IP Security) zajišťuje zabezpečení na úrovni síťové vrstvy pomocí protokolů AH (Authentication Header) a ESP (Encapsulating Security Payload). Pomocí IPSec lze vytvářet virtuální privátní síť (VPN), které umožňují bezpečný datový přenos přes nezabezpečené (např. veřejné) síť. Hlavní nevýhodou zabezpečení pomocí IPSec je velká režie, neboť ke každému paketu RTP se přidávají zabezpečovací informace.

Secure RTP (SRTP) [10] je rozšíření protokolu RTP pro zajištění důvěryhodnosti, autentizaci a ochranu proti podvržení přenosů RTP a RTCP. Při návrhu byl také kladen důraz na menší výpočetní náročnost zabezpečení, úsporné přenášení zabezpečovacích informací a nezávislost na transportní, síťové a fyzické vrstvě využívané protokolem RTP.

7.8 Zajištění přenosového pásma pro hlasové služby

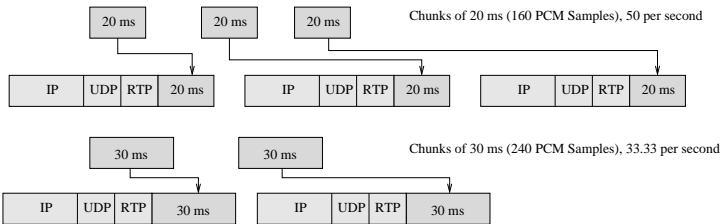
Jedním z důležitých úkolů pro provozování IP telefonních služeb, je zajištění dostatečné kapacity přenosového pásma pro telefonní hovory nad IP. Jak už jsme naznačili v předchozím textu, lze využít oddělení provozu VoIP od ostatního datového provozu pomocí speciální VLAN sítě (obvykle se nazývá Voice VLAN, VVLAN) a prioritizací paketů s hlasovými daty. Tato prioritizace se týká jak signálních protokolů (SIP, H.323), tak i transportního přenosu pomocí RTP. Pro zajištění kvality přenosu QoS (Quality of Service) je možné použít dva základní přístupy – (1) rezervaci výpočetních zdrojů a přenosového pásma pomocí rezervčního protokolu RSVP a integrovaných služeb IntServ nebo (2) vyčlenění hlasových paketů do speciální třídy přenosu (CoS, Class of Service) a nastavení přenosových politik na aktivních prvcích v síti tak, aby preferovaly přenos prioritních tříd. Tento druhý přístup se také nazývá diferenciované služby (DiffServ).

7.8.1 Výpočet přenosového pásma

Pro výpočet přenosového pásma, které je potřebné pro přenos jednoho hovoru, je potřeba brát v úvahu více parametrů, které dohromady tvoří reálné přenosové pásmo:

Rychlost generování paketu (Packet rate) určuje počet paketů, které jsou vyslány za daný časový interval. Obvykle se udává v paketech za sekundu (pps). Je to převrácená hodnota velikosti vzorku (packetization size).

Velikost hlasového vzorku (chunk) (Packetization size) udává počet bytů, které jsou potřeba pro uložení hlasové informace po digitalizaci hlasu do jednoho RTP paketu. Tato reže se přidává k potřebnému přenosovému pásmu daného kodekem a závisí na rychlosti posílání paketů. Pokud budeme posílat vytvářet menší pakety a posílat je častěji, narůstá nám reže. Vztah mezi délkou vzorku (packetization period) a velikostí vzorku (packetization size) je na obrázku ?? . Na prvním obrázku se ukládají do paketu



Obrázek 7.15: Vztah mezi délkou hlasového vzorku a velikostí VoIP paketu

kódovaný hlasový tok o délce 20 ms. Paket se generuje každých 50 sekund (1/20). Pro hlasový záznam 60 ms se pošlou tři pakety. Každý paket obsahuje 20 ms hlasového záznamu a režii (3x hlavičky IP, UDP a RTP). Ve druhém příkladě vidíme záznam hlasu každých 30 ms. Na 60 ms celkového záznamu jsou potřeba dva pakety. Jde tedy o redukci IP režie o jednu třetinu.

Reže IP definuje počet bytů, které se k hlasovému paketu přidávají během vytvoření IP datagramu. Reže obvykle zahrnuje velikosti IP hlavičky, UDP hlavičky a RTP hlavičky.

Reže linkové vrstvy udává počet bytů přidávaných na linkové vrstvě. Reže závisí na typu linkového protokolu, viz tabulka 7.8.

Reže tunelování se použije v případě přenosu zabezpečeným kanálem. Jedná se o režii 802.1Q (trunks), tunelování pomocí IPSec, GRE (Generic Route Encapsulation) či MPLS (Multiprotocol Label Switching). Toto je velice důležité mít na paměti, pokud se rozhodneme tunelovat provoz VoIP. Reže tunelování pro různé linkové technologie je v tabulce 7.8. Pokud je například přenášen hlasový paket IPSec tunel v tunelovém

Technologie	Ethernet	Frame Relay	Trunk 802.1q	MPLS
Reže	18	6	22	4
Technologie	IPSec transport	IPSec tunel	L2TP/GRE	PPPoE
Reže	30-53	50-73	24	8

Tabulka 7.8: Reže L2 a L3 technologií

režimu s hlavičkou ESP, je celková délka režie 94 bytů, tj. IP (20) + ESP (30+4) + IP (20) + UDP (8) + RTP (12).

Firma Cisco na svých zařízeních vytváří hlasové pakety obsahující vzorky o délce 20 ms. Při kódování G.711 (PCM) o rychlosti hlasového toku 64 kb/s je velikost hlasových dat 1280 bitů (20 ms x 64 kb/s), tj. 160 bytů. Tato data tvoří užitečný obsah RTP paketu. Režie přenosu, která zahrnuje hlavičky RTP (12 bytů), UDP (8 bytů), IP (20 bytů) a Ethernetu (18 bytů), činí dohromady 58 bytů, tj. 464 bitů. Při rychlosti generování paketů 50 pps (64 000 b/s / 1280 b) je potřebné přenosové pásmo cca 85 kb/s, tj. $(1280 + 464) \text{ bitů} \times 50 \text{ pps} = 87,200 \text{ b/s}$. Pro jiné kodeky je rychlost uvedena v tabulce 7.9. Pro obecný výpočet celkové šířky pásma

Kodek	G.711	G.711	G.729	G.729
Délka hlasu v paketu	20 ms	30 ms	20 ms	40 ms
Šířka pásma kodeku	64 kb/s	64 kb/s	8 kb/s	8 kb/s
Velikost obsahu paketu	160 B	240 B	20	40
Režie IP	40	40	40	40
Velikost VoIP paketu	200	280	60	80
Rychlost generování paketů	60 pps	33.333 pps	50 pps	25 pps

Tabulka 7.9: Příklad VoIP zapozdření pro různé kodeky a rychlosti

potřebujeme provést následující kroky:

1. Znát délku hlasového vzorku v paketu (packetization period) a výstupní šířku pásma kodeku.
2. Určit režii linkové vrstvy a režii případného tunelování (b).
3. Vypočítat velikost hlasového vzorku (b).
4. Připočíst k velikosti hlasového vzorku režii dalších vrstev (b).
5. Spočítat rychlost generování paketů (pps).
6. Určit celkové potřebné přenosové pásmo (b/s).

Obecný vzorec pro výpočet je naznačen následujícím vztahem:

```
total_bandwidth = total_packet_size * packet_rate
total_packet_size = packetization_size + IP_overhead + Link_overhead + Tunnel_overhead
packetization_size = packetization_period * codec_bandwidth
packet_rate = 1 / packetization_period
```

Pro demonstraci lze použít už výše uvedený příklad. Pokud hlasové vzorky jsou dlouhé 20 ms (packetization-period), potom rychlost generování paketů je 50. Délka paketu je dána délkou hlasového vzorku (20 ms) a šířkou pásma kodeku (64 kb/s), což je 1280 bitů. Spolu s režii činí celková délka paketu 1744 bitů (1280 + 464). Potřebné přenosové pásmo je tedy $1744 * 50 = 87\,200 \text{ b/s} = \text{cca } 85 \text{ kb/s}$. V tabulce 7.10 je ukázka požadavků přenosového pásma pro různé kodeky.

7.9 Přenos hlasu nad IP v sítích LAN a WAN

IP telefonie je technologie, která může pracovat nad sítěmi LAN i WAN, neboť signalizační i datové protokoly pracují nezávisle na přenosové technologii. Nicméně u sítí LAN a WAN se výrazně se liší kvalita přenosu.

Kodek	G. 711	G.729	G.723.1	G.726	G.726	G.726
Šířka pásma kodeku	64 kb/s	8 kb/s	6.3 kb/s	32 kb/s	24 kb/s	16 kb/s
Velikost vzorku	160 B	20 B	24 B	80 B	80 B	60 B
Délka hlasového vzorku	20 ms	20 ms	30 ms	20 ms	20 ms	30 ms
Rychlost generování	50 pps	50 pps	34 pps	50 pps	50 pps	34 pps
Šířka pásma Frame Relay	82.8 kb/s	26.8 kb/s	18.9 kb/s	50.8 kb/s	42.8 kb/s	28.5 kb/s
Šířka pásma Ethernet	87.2 kb/s	31.2 kb/s	21.9 kb/s	55.2 kb/s	47.2 kb/s	31.5 kb/s

Tabulka 7.10: Celková šířka pásma pro různé kodeky

7.9.1 VoIP v sítích LAN

LAN používá Ethernet pro transport, standard 802.3. Operace nad LAN na rychlostech 10/100/1000 Mbps mají velmi krátké odezvy, nedochází tam ke kolísání zpoždění (jitter), nejsou tam velké ztráty paketů a chyby.

Pro lepší přenos se doporučuje pomocí VLAN oddělit zařízení pro přenos dat a hlasových služeb. Je to také z důvodu bezpečnosti a lepšího výkonu. Kvalita hlasu a rychlost signalizace je stejně dobrá jako u TDM PBX. Další možností je prioritizace rámců pomocí rozšíření 802.1p.

7.9.2 VoIP v sítích WAN

Přenos VoIP po sítích WAN přináší několik omezení. Jedním limitem je omezená šířka pásma, která je mnohem menší než u lokálních sítí. Také doba přenosu mezi koncovými zařízeními je delší. Výrazný podíl na nižší kvalitě zvuku má kolísání zpoždění mezi pakety (jitter) a také ztráty paketů.

Cílem IP telefonie je zajistit zpoždění hlasových paketů menší než 150 ms. Přijímající IP telefon musí kompenzovat kolísání zpoždění mezi jednotlivými datovými pakety předtím, že jsou data konvertována na analogový zvuk. Rekonstrukce hlasu je také potřeba pro řešení ztracených paketů.

Technickým řešením těchto otázek může být použití speciálního přenosového pásma nebo použití kvality služeb QoS. Hlas vyžaduje při přenosu pásmo 80 Kbps, pokud není použita komprese (G.711) a asi 25 Kbps (G.729) při použití komprese. Aktuální šířka pásma se mění podle typu komprese a velikosti paketů. U LAN sítí zabírá hlasová přenos maximálně 10% až 20% přenosového pásma. U WAN je tento poměr mnohem vyšší a může vést k degradaci volání.

Kvalita služby QoS je podporována standardem IEEE 802.1p. Směrovače implementují QoS pomocí diferenciování služeb (DiffServ), které musí také podporovat IP telefony a brány. Kvalitu multimediálních přenosů popisuje doporučení G.1010 (End-User Multimedia QoS Requirements), viz tabulka 7.11.

Shrnutí

Přenos hlasových služeb nad IP, IP telefonie, je dnes běžnou součástí služeb poskytovaných počítačovými sítěmi. Oproti klasické telefonní síti přináší úsporu prostředků na vybudování sítě, neboť využívá stávající strukturu datových sítí. Jde o jednoduše spravovatelnou strukturu s dynamickým připojováním uživatelů a snadnou rozšiřitelností. Mezi základní stavební

odezva	interaktivní (zpoždění << 1s)	reagující (cca 2s)	včasný (cca 10s)	nekritické (>> 10s)
Tolerování chyb	Konverzace (hlas, obraz)	messages (hlas, video)	streaming (hlas, video)	fax
Netolerování chyb	příkazy/řízení	transakce (WWW, email)	messages, soubory (ftp)	aplikace na pozadí

Tabulka 7.11: Požadavky uživatelů na multimediální přenos podle G.1010

prvky IP telefonie patří koncové stanice (IP telefony), ústředny (gatekeeper) pro registraci účastníků, přepojování a směrování hovorů, brány (gateway) pro přístup do veřejné telefonní sítě a další prvky. Na komunikaci se podílejí jednak signalizační protokoly (např. SIP, H.323., SCCP), které zajišťují registraci a lokalizaci uživatelů, předávání signalizačních zpráv, vytváření, udržování a ukončování spojení. Vlastní hlasová data se přenášejí po transportním protokolu RTP.

Pro zajištění dostatečné kvality hlasového přenosu nad IP (VoIP) je nutné prioritizovat hlasové pakety a snížit nepříznivé faktory, které se podílejí na nižší kvalitě. Jsou to například zpoždění, ztrátovost paketů, echo, jitter či kvalita kodeku. Kodek je algoritmus, který popisuje převod analogového hlasového signálu na digitální signál, kompresi dat a uložení do paketů pro přenos po síti. Podle typu kodeku můžeme dosáhnout větší hlasové kvality, často však za cenu zvýšení přenosového pásma potřebného pro hlasový hovor.

Sledováním kvality hlasového přenosu se zabývá několik standardů ITU-T. Jedním z metod pro hodnocení přenosu je metoda absolutního hodnocení poslechu, která počítá hodnotu MOS. Tato hodnota udává na pětibodové stupnici kvalitu hovoru. Pro výpočet se používá buď statistické ohodnocení na základě subjektivního poslechu, případně objektivní měření porovnávající čistý signál ze zdroje s přijatým signálem. Druhou běžnou metodou pro výpočet kvality hlasového přenosu je E-model, který od vlatního signálu analyticky odpočítává různá rušení. Výsledkem je hodnota tzv. R-faktoru, který na stupnici 0–100 udává kvalitu hovoru.

Důležitým úkolem současné IP telefonie je zabezpečení hovorů pro zneužití, odposlechu, či útoku prostřednictvím virů, spamu či DoS. Pro zajištění bezpečnosti je dobré oddělit hlasových přenos od ostatního datového přenosu pomocí VLAN sítí, dále je možné použít šifrovaný přenos přes IPSec, TLS či bezpečný transportní kanál Secure RTP. Vytvoření VLAN sítí umožňuje i označení hlasových paketů a umístění do prioritní třídy zpracování při průchodu sítí. Tím zajišťujeme nejen bezpečnost, ale i kvalitu služby a dostupnost.

Otázky

1. Jaké jsou požadavky na kvalitu hlasového přenosu oproti datovému?
2. Vysvětlíte převod hlasu na datový signál.
3. Popište architekturu veřejné telefonní sítě.
4. Popište architekturu IP telefonní sítě.
5. Porovnejte signalizační protokoly veřejné telefonie a IP telefonie.

6. Jaké znáte faktory ovlivňující přenos hlasových dat? Jak lze eliminovat účinek těchto faktorů?
7. Popište metodu absolutního hodnocení kvality přenosu ACR. Jaké dva typy měření můžeme použít.
8. Vysvětlete E-model a jeho použití. V čem jsou jeho výhody a nevýhody oproti metodě ACR?
9. Vyjmenujte a stručně popište základní komponenty IP telefonie.
10. Čím se liší centrální a distribuované řízení hovorů u IP telefonie? Porovnejte výhody a nevýhody.
11. Popište architekturu IP telefonie SIP.
12. Popište architekturu IP telefonie H.323.
13. Popište ustavení hovoru pomocí SIP, resp. H.323.
14. Jaké standardy a protokoly používá H.323? Stručně vysvětlete jejich použití.
15. Co je to ENUM a k čemu slouží?
16. Jaké jsou bezpečnostní problémy VoIP a jak je lze eliminovat?
17. Pro zadaný kodek a režii přenosu vypočítejte potřebné přenosové pásmo.
18. Porovnejte použití VoIP v sítích LAN a WAN.

Doporučená literatura a standardy

- [1] M. Vozňák and D. Zukal. *Kvalita hovoru v prostředí VoIP*. Technical Report 11/2005, Cesnet, 2005.
- [2] ITU-T. *The E-model, a computational model for use in transmission planning*. G.107 (2000), May 2000.
- [3] ITU-T. *Transmission impairments due to speech processing, Appendix I*. G.113 (2002), May 2002.
- [4] J.Rosenberg, H.Schulzrinne, G.Camarillo, A.Johnston, J.Peterson, R.Sparks, M.Handley, and E.Schooler. *SIP: Session Initiation Protocol*. RFC 3261, June 2002.
- [5] H.Schulzrinne, S.Casner, R.Frederick, and V.Jacobson. *RTP: A Transport Protocol for Real-Time Applications*. RFC 3550, July 2003.
- [6] P.Faltstrom and M.Mealling. *The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)*. RFC 3761, April 2004.
- [7] Miroslav Vozňák and Rita Pužmanová. *Kvalita VoIP souvisí se zabezpečením. Professional Computing*, (4):31–33, 2009.

- [8] Jonathan Zar. *VoIP Security and Privacy Thread Taxonomy*. Technical report, VOIPSA, www.voipsa.org, 2005.
- [9] Tomáš Vaněk. *I hlas si žádá bezpečí. Zabezpečení VoIP komunikace*. *Connect!*, 14(5):10–13, 2009.
- [10] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman. *The Secure Real-time Transport Protocol (SRTP)*. RFC 3711, March 2004.
- [11] G. Audin. *VoIP Basics for IT Technicians*. White paper, Fluke Networks, 2005.
- [12] články o IP telefonii. *Connect*, (5), 2005.
- [13] M. Vozňák and D. Zukal. *Vyhodnocení hovoru pomocí R-faktoru v sítích VoIP*. Technical report, CESNET, 2004.
- [14] F.Andreasen and B.Foster. *Media Gateway Control Protocol (MGCP) Version 1.0*. RFC 3436, January 2003.
- [15] M.Handley, V.Jacobson, and C.Perkins. *SDP: Session Description Protocol*. RFC 4566, July 2006.
- [16] M.Handley, C.Perkins, and E.Whelan. *Session Announcement Protocol*. RFC 2974, October 2000.
- [17] H.Schulzrinne, A.Rao, and R.Lanphier. *Real Time Streaming Protocol (RTSP)*. RFC 2326, April 1998.

Kapitola 8

Zajištění kvality služeb

Tato kapitola se zabývá způsoby, jak zajistit požadovanou kvalitu služeb u přenosů v počítačových sítích. Na začátku se seznámíme s obecným problémem přizpůsobení toku dat a s pojmy rozložení provozu v čase (traffic shaping) či omezení provozu (traffic policing). Dále se budeme zabývat mechanismy řízení průchodu paketů síťovými prvky (nejčastěji směrovači) pomocí různých algoritmů. Tyto algoritmy využívají dva základní přístupy k zajištění kvality služeb – model integrovaných služeb nebo model diferenciovaných služeb. Ukážeme si princip obou přístupů, jejich výhody i omezení. V závěru kapitoly si popíšeme techniku pro předcházení zahlcení ve frontách aktivních prvků pomocí technologií RED a WRED.

8.1 Úvod

Původní návrh provozu Internetu založený na protokolu IP počítal s jednoduchým modelem provozu služeb, který se nazývá doručení s největší úsilí (best-effort delivery). Požadavky na zajištění kvality služeb z pohledu zejména multimediálních aplikací (přenos obrazu, hlasu) si vyžádal během devadesátých let 20. století vytvoření dalších modelů, které by lépe odlišovali různé třídy provozu a jejich požadavky na přenosové pásmo. Vznikl jednak model integrovaných služeb založený na rezervaci přenosové pásmo a výpočetních zdrojů na síťových prvcích pro jednotlivé toky. Po něm následoval dnes velmi populární model diferenciovaných služeb, který využívá značení paketů pomocí polí DSCP (původně ToS) v hlavičce IP datagramu. Pomocí kódu DSCP definuje model diferenciovaných služeb několik tříd provozu podle důležitosti. Pro tyto třídy se pak definují politiky pro zpracování dat na síťových zařízeních.

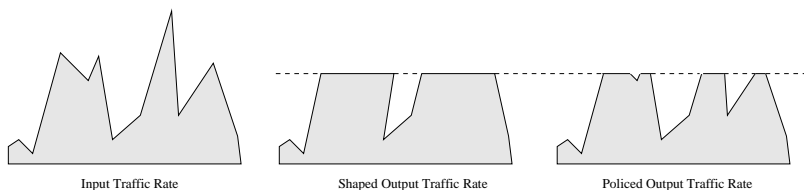
Zajištění kvality služeb vychází z požadavků uživatele sítě. Tyto požadavky jsou často definovány ve smlouvě nazvané *SLA (Service Level Agreement)*. Smlouva SLA specifikuje nároky na přenos zákaznických dat. Obsahuje jednak požadavky na fyzické připojení (typ síťové infrastruktury), které zahrnují například dostupnost sítě (availability) či přenosovou kapacitu (propustnost). Příkladem může být dostupnost 99,9% s propustností 1 Gb/s.

SLA obsahuje také požadavky na síťový přenos, který obsahuje například požadavky na ztrátovost (loss), celkové čas odezvy (round-trip delay), rozptyl zpoždění (jitter), což je zejména pro interaktivní přenosy typu IP telefonie či videokonference kritické. Podle typu provozu (data, hlas, video) se jednotlivé požadavky liší, například zpoždění pro hlasové pakety musí být mnohem menší než pro signalizaci u IP telefonie.

Příklad SLA může být smlouva mezi zákaznickou firmou a poskytovatelem síťového připojení ISP, která obsahuje následující body:

- **Dostupnost sítě.** Síť bude dostupná zákazníkovi 99,95% času za standardních podmínek. V případě neplánovaného výpadku sítě bude mít zákazník nárok na deseti procentní zvýšení přenosového pásma na měsíc. ISP může dočasně přerušit službu z důvodů plánovaných aktualizací a oprav.
- **Síťový přenos.** Zákazník bude mít k dispozici přenosové pásmo 10 Mb/s pro příjem dat (download) a 5 Mb/s pro odesílání dat (upload). Průměrné přenosové pásmo nebude menší než tyto specifikované hodnoty s odchylkou 0,1% za měsíc.
- **Zpoždění a rozptyl.** Průměrná ztrátovost paketů na páteřní síti nepřesáhne 0,2%. Průměrné měsíční zpoždění páteřní sítě bude menší nebo rovno 50 ms. Průměrný rozptyl (jitter) nejvýše 250 μ s. Maximální rozptyl na páteřní síti nepřesáhne 10 ms více než 0,2% za měsíc.

Pro zajištění požadavků SLA je nastavit parametry přenosu. K tomu se používá klasifikace provozu například pomocí *značení paketů* (*packet marking*), *rozložení provozu* (*traffic shaping*) či *ořezání provozu* (*traffic policing*). Rozdíl mezi rozložením a ořezáním provozu je znázorněn na obrázku 8.1.



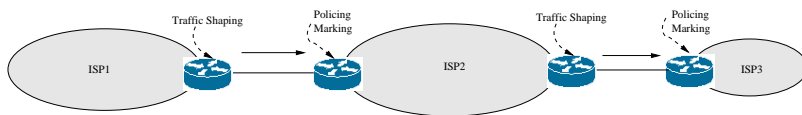
Obrázek 8.1: Charakteristika provozu po rozložení (shaping) a ořezání (policing)

Rozložení provozu (traffic shaping) slouží k regulaci rychlosti a objemu provozu jednotlivých toků či agregovaných toků. Primárním úkolem rozložení provozu je přizpůsobit přenos paketů dané rychlosti specifikované pomocí SLA. Rozložení provozu se používá k zmírnění zahlcení tím, že rozprostře v čase shluky paketů. Rozložení provozu způsobí vyhlazení výstupní rychlosti provozu a tedy i lepší využití přenosového pásma. Rozložení provozu lze v síťových zařízeních implementovat pomocí modelu tekoucího vědra (Leaky Bucket, viz 8.2.5) či model Token Bucket (viz 8.2.6).

Ořezání provozu (traffic policing) slouží k omezení maximální rychlosti individuálního toku či agregovaných toků. Obvykle se používá v hraničním bodě sítě pro ořezání vstupního toku. Každému přicházejícímu toku je přiřazen odpovídající profil přenosu dat. Profil definuje rychlostní a přenosové parametry toku. Pokud daný tok překročí tyto parametry, jsou pakety porušující SLA zahozeny nebo označeny nižší třídou přenosu. Podobně jako rozložení provozu (traffic shaping) lze i ořezání provozu implementovat pomocí modelu tekoucího vědra.

Oba přístupy slouží k řízení rychlosti přenosu paketů po síti. Hlavní rozdíl je v tom, že rozložení provozu (shaping) nejen omezí rychlost toku, ale ukládá provoz přesahující povolený

limit do paměti (buffers). Oproti tomu ořezání provozu (policing) neprovádí ukládání přesahujícího provozu, ale pouze ořezává špičky. Výsledná výstupní rychlost obsahuje výběžky a útlumy rychlosti a využití přenosového pásma není tak efektivní.

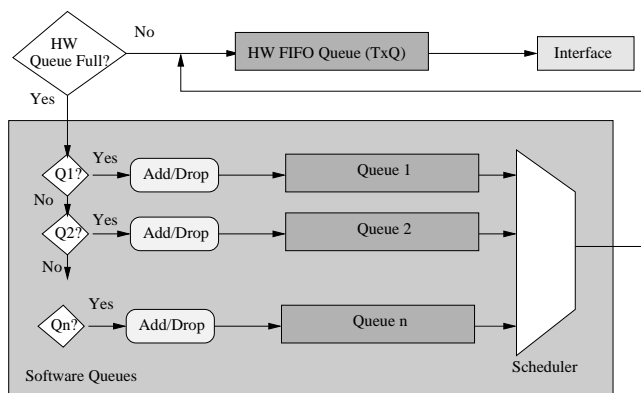


Obrázek 8.2: Rozložení a ořezání provozu v sítích

Rozložení provozu vyžaduje oproti ořezání dostatečnou paměť pro ukládání dat do front. Rozložení se obvykle aplikuje na výstupní provoz ze sítě. Ořezání (policing) se používá jak pro vstupní, tak i pro výstupní provoz. Protože nedochází k ukládání do front, nevzniká další zpoždění provozu. Příklad nasazení metod rozložení a ořezání provozu v praxi je na obrázku 8.2.

8.2 Mechanismy plánování

Při síťové komunikaci dochází k prokládání různých datových toků mezi sebou a vytváření front na výstupech síťových prvků. Pokud má síťový prvek více portů, vytváří se zvlášť výstupní fronta ke každému portu. Způsob, jak řadit a vybírat pakety z front, se nazývá plánování (scheduling) a provádí ho speciální proces na směrovači zvaný plánovač (scheduler). Způsob plánování průchodu paketů zařízením a obsluha výstupních front hraje důležitou roli při zajišťování kvality přenosu QoS. V této kapitole si uvedeme některé základní mechanismy obsluhy front. Obecný princip používání front je na obrázku 8.3. Vidíme zde schéma směrovače

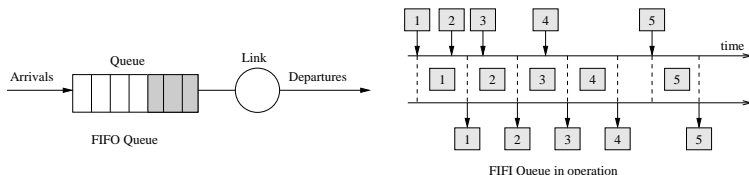


Obrázek 8.3: Použití front na modelu směrovače

při práci z pakety. Paket přijde do směrovače, kde je nejprve umístěn do fronty a pak poslán na výstupní rozhraní. Každému rozhraní náleží jedna hardwarová fronta, do které se uloží paket (či pakety), které jsou určeny pro poslání na výstup. Pokud je tato fronta plná, což může být v důsledku nižší kapacity výstupní linky, délky paketu či vytíženosti procesoru při zpracování dat, je paket poslán do softwarové fronty (či front), kde se uloží. Pokud je fronta plná, dojde k zahození paketu. V dalším textu si ukážeme jednotlivé typy front. Pro hardwarovou frontu se používá obsluha FIFO, ostatní typy se implementují pouze v softwaru.

8.2.1 Fronty typu FIFO (First-In-First-Out)

Fronta typu FIFO je nejběžnější způsob obsluhování požadavků. Používá tzv. mlynářský algoritmus (Kdo dřív přijde, ten dřív mele), tzn. pakety přicházející na výstup linky jsou v případě obsazení linky řazeny do fronty v pořadí, jak přicházejí. Pokud je výstupní fronta zaplněna, mechanismus pro zahazování paketů rozhodne, zda bude nově příchozí paket zahazen nebo zda budou jiné pakety odstraněny z fronty. Ve frontě typu FIFO pakety odcházejí ve stejném pořadí, v jakém přišly, viz obrázek 8.4. Výhodou tohoto postupu je předvídatelné



Obrázek 8.4: Fronta typu FIFO a její použití

zpoždění paketu. Pokud rychlost linky je R (v bitech za sekundu) a B je maximální velikost fronty (v bitech), pak zpoždění ve frontě D je ohraničeno vztahem

$$D \leq \frac{B}{R}.$$

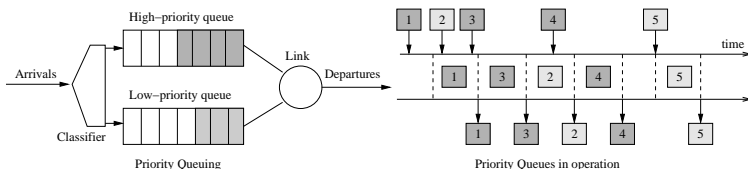
Fronta typu FIFO nemá žádný mechanismus pro zacházení s pakety citlivými na zpoždění či pakety s vyšší prioritou. Pokud je ve frontě příliš velký paket, způsobí zpoždění všech menších paketů, které jsou za ním. Jestliže nějaký agresivní tok obsahuje příliš mnoho paketů ve shlucích (bursts), může fronta FIFO způsobit odepření služby (podobné útoku DoS) pro ostatní toky, dokud není tento shluk obslužen. Fronta FIFO se používá jako implicitní metoda správy fronty výstupní linky, pokud není implementována žádná jiná správa front.

8.2.2 Prioritní fronty (Priority Queues)

U prioritních front dochází ke klasifikaci paketů do jedné nebo více prioritních tříd ve výstupní frontě. Klasifikace závisí na zvolené politice. Může se použít značek paketů (marking), například bity v poli Type of Service (ToS), nověji pole DSCP v hlavičce IP datagramu [1]. Je také možné vytvářet prioritní fronty na základě IP adres (zdrojová či cílová adresa), čísel portů či zvolit jiné kritérium. Každá taková třída má svou výstupní frontu. Když dochází k výběru paketu na výstup, plánovač vybírá nejprve pakety ze tříd s vyšší prioritou. Teprve

když je fronta této třídy prázdná, začne obsluhovat ostatní fronty. Zpracování paketů v rámci jedné prioritní fronty je obvykle typu FIFO.

Na obrázku 8.5 je ukázka prioritizace paketů. Pakety 1,3,4 mají vyšší prioritu než pakety 2 a 5. Pokud se používá nepreemptivní plánování, nedochází k přerušení odesílání paketu, i když obsluhovaný paket má nižší prioritu než právě došlý. Výhodou prioritních front je rozlišení



Obrázek 8.5: Prioritní fronty a jejich použití

toků do různých tříd a nastavení priorit při obsluhování. Například datový provoz VoIP a signalizace IP telefonie má přednost před jiným provozem, např. přenosem ftp. Prioritní fronta negarantuje žádné konkrétní parametry přenosu ani maximální hranici využití přenosového pásma, pouze upřednostňuje pakety určité třídy.

Při použití prioritních front může dojít k tzv. *vyhladovění* (starvation), kdy pakety s vyšší prioritou neustále předbíhají pakety s nižší prioritou. Pokud je fronta s vyšší prioritou stále neprázdná, nikdy nedojde k obsluze front s nižší prioritou. Tomu se říká striktní prioritizace (strict priority queueing). Striktní prioritizaci je vhodné použít například pro přenos VoIP, kdy potřebujeme zaručit, že žádné pakety VoIP nebudou zahozeny. Podobně je vhodné využít striktní prioritní fronty pro aktualizace směrovacích informací, například v případě zahlcení je toto žádoucí. Pokud je prioritní fronta prázdná, zpracovávají se pakety z dalších front a mohou přitom využívat veškerou dostupnou kapacitu linky. Další variantou prioritních front jsou fronty řízení rychlostí (rate-controlled priority queueing), které upřednostňují pakety z prioritních front do té doby, dokud nepřesáhne prioritní fronta daného rychlostního limitu.

8.2.3 Cyklické fronty (Round Robin Queues)

Řešením problému vyhladovění je například použití front s cyklickou obsluhou (round robin), kde je stejný počet paketů cyklicky odebírán z každé fronty. Pokud jsou pakety různě velké, jsou upřednostňovány toky s většími pakety. Proto se často používají tzv. spravedlivé fronty (fair queues), které jsou zvláštním typem cyklickým front. Podobně jako cyklické fronty odebírají z front stejný počet paketů, nicméně tento počet je normalizován na délku paketů.

U cyklických front (round robin) jsou pakety řazeny do prioritních front. Každá fronta odpovídá jednomu toku. Obsluha front nezávisí striktně na prioritě, jako je tomu u prioritních front, ale všechny fronty jsou cyklicky obsluhovány. V jednom cyklu se odebere z každé fronty stejný počet paketů či bitů. Pokud je rychlost linky R , tak při počtu n front je minimální rychlost zpracování toku R/n bitů za sekundu. Jestliže nejsou všechny fronty plně obsazené, výsledná rychlost bude ještě větší. Při plné vytížení všech front rychlost obsluhy neklesne pod tuto hodnotu.

Mechanismus cyklické fronty zaručuje, že nemůže dojít k situaci, kdy by byla nějaká fronta opakovaně předbíhána. Pokud je fronta prázdná, pokračuje se obsluhou nejbližší neprázdné

fronty.

Implementace cyklických front využívá výpočtu virtuálního času ukončení přenosu E_i pro daný paket i . Necht A_i je doba příchodu paketu i do fronty, S_i doba začátku přenosu a E_i doba ukončení přenosu. Pak čas přenosu paketu P_i bude $P_i = E_i - S_i$. Tato hodnota souvisí s výstupní rychlostí fronty. Je důležité připomenout, že pro každý tok dat (soubor paketů od stejného odesílatele stejnému adresátovi) se vytváří samostatná fronta. Hodnota začátku zpracování S_i bude stejně jako A_i , pokud není při příchodu obsluhován jiný paket. V opačném případě musíme vyčkat obslužení předchozího paketu $i - 1$, což znamená, že vlastní obsluha paketu i začne buď v čase jeho příchodu A_i nebo v čase ukončení obsluhy předchozího paketu E_{i-1} . Doba ukončení zpracování paketu i (někdy se jí říká virtuální doba ukončení přenosu [2]) bude

$$E_i = \max\{A_i, E_{i-1}\} + P_i.$$

Implementace obsluhy cyklických front vypočte pro každý paket, který přijde na vstup, hodnotu E_i . Paket je následně zařazen do fronty paketů seřazené podle hodnoty E_i . Při obrovském počtu toků (front) může být doba vložení do výstupní fronty velice dlouhá, neboť složitost vkládání $\mathcal{O}(\log N)$ roste logaritmicky s počtem toků [3]. Jakmile se dokončí přenos zpracovávaného paketu, odešle se další paket s nejmenší hodnotou E_i .

8.2.4 Váhové fronty WFQ (Weighted Fair Queues)

Váhové fronty tvoří další s mechanismů pro plánování průchodu dat zařízením. Přicházející pakety jsou řazeny klasifikátorem do toků a každému toku je přiřazena jedna fronta. Tok se identifikuje na základě informací z hlaviček IP a TCP/UDP, například zdrojové a cílové IP adresy, čísla protokolu (Protocol number), typu služby (ToS), čísla zdrojové a cílového portu. Na základě těchto informací se vypočítá hašovací funkce ukazatel do fronty. Pokud fronta pro daný tok existuje (tj. nejedná se o první paket toku), umístí se paket do příslušné fronty. Pokud se jedná o první paket toku, vytvoří se nová fronta a do ní se vloží paket. Fronty obsluhuje plánovač pro řízení váhových front WFQ. Váhové fronty WFQ jsou fronty, které rozšiřují vlastnosti cyklických front. Počet odebraných paketů u WFQ závisí na váze, která se přiřadí jednotlivým frontám.

U jednoduchých cyklických front RR (Round robin) se odebírá pouze jeden paket z každé fronty. U váhových front WFQ (Weighted Fair Queues) se pro každou prioritu nastaví váha w_i a při každém cyklu se z příslušné fronty odebere takový počet paketů, který odpovídá poměru

$$\frac{w_i}{\sum_{j=0}^n w_j}, 0 \leq i \leq n,$$

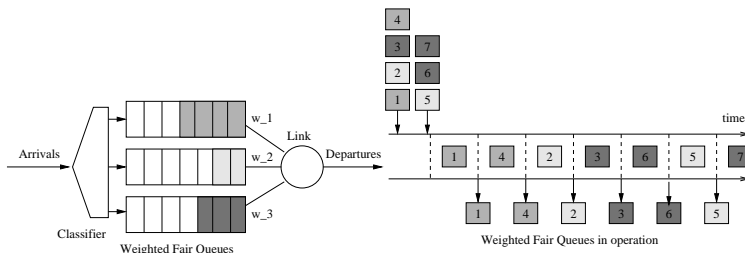
kde n je počet toků. U linky s rychlostí R bude rychlost obsluhy paketů r_i toku i dána vztahem

$$r_i = R \cdot \frac{w_i}{\sum_j w_j}.$$

Pokud zdroj požaduje pro daný tok (službu) zajištění rychlosti r_i , může uzel přidělit požadované zdroje (pokud jsou dostupné) a nastavit příslušnou váhu fronty pro zajištění kvality služby. Zároveň je možno garantovat limit pro maximální zpoždění. Toto je zajištěno mechanismem Leaky Bucket (viz 8.2.5), kde b_i je velikost vědra a r_i je rychlost generování žetonů. Maximální doba zpoždění D_i toku i je dána vztahem

$$D_i \leq \frac{b_i}{r_i}.$$

Na obrázku 8.6 je ukázka fronty WFQ se třemi prioritními frontami. Pakety 1,4 patří do první fronty, 3, 6 a 7 do druhé fronty a 2 a 5 do třetí fronty. V případě, že nastavíme váhy $w_1 = 3, w_2 = 1, w_3 = 2$, budou se pakety odebírat v tomto pořadí: tři pakety z první fronty,



Obrázek 8.6: Váhové fronty WFQ a jejich použití

jeden paket z druhé fronty a dva pakety ze třetí fronty v jednom cyklu. Pokud je některá fronta prázdná, tak se přejde na další frontu. Proto se na výstupu objeví nejprve pakety 1 a 4 z první fronty, paket 2 z druhé fronty a pakety 3 a 6 ze třetí fronty. V následujícím cyklu to bude paket 5 z druhé fronty a paket 7 ze třetí fronty.

Počet front v systému WFQ závisí na počtu aktivních toků. Mechanismus WFQ dynamicky vytváří a ruší fronty. Z tohoto důvodu WFQ neumožňuje přesné zajištění přenosového pásma. Pokud je systém front zaplněn, další přicházející pakety jsou zahazovány.

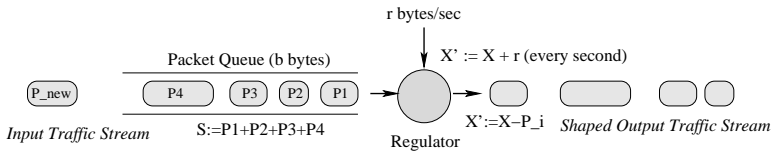
8.2.5 Mechanismus Leaky Bucket (tekoucí vědro)

Mechanismus tekoucího vědra slouží k řízení toku dat, zejména pro rozložení provozu (shaping). Poskytuje mechanismus pro vyhlazení vstupního provozu na tok, který má stálý výstup. Tento mechanismus vnutí vstupnímu provozu, který je obvykle tvořen nepravidelně přicházejícími shluky dat, konstantní přenosovou rychlost.

Hlavní myšlenka algoritmu je jednoduchá. Představme si vědro s malou dírou na dně. Bez ohledu na to, jakou rychlostí přitéká do vědra voda, rychlost odtékající vody je konstantní a závisí na velikosti díry. Pokud je vědro prázdné, bude výstupní rychlost nulová rychlost. Jestliže je vědro plné, tak další přitékající voda se přelije přes okraje a zmizí, tzn. neobjeví se ve výstupním toku u díry na dně vědra.

Myšlenka tekoucího vědra se používá pro rozložení datového provozu. Do vědra (fronty) přichází neregulovaný datový tok. Pakety odcházejí dírou ve dně vědra a jsou přenášeny do sítě konstantní rychlostí r bytů za sekundu. Velikost vědra (hloubka) je omezena na b bytů, viz obrázek 8.7. Pokud je vstupní rychlost paketů do vědra větší než rychlost na výstupu r , vědro se časem zaplní. Po zaplnění se nově příchozí pakety do vědra (fronty) nevejdou, a tak se zahazují.

Příklad 8.1. Mějme například tekoucí vědro o rychlosti $r = 1.2 \text{ Mb/s}$, což znamená, že jeden byte bude přeposlán za cca $6.7 \mu\text{s}$. Pokud přijde paket o délce 1500 bytů a vědro bude prázdné, bude paket přeposlán za 10 ms .



Obrázek 8.7: Model tekoucího vědra (leaky bucket)

Jak vidíme z obrázku 8.7, můžeme algoritmus tekoucího vědra jednoduše implementovat jako frontu FIFO s časovačem t a čítačem X . Časovač t vyprší každou sekundu a zvýší čítač X o hodnotu r bytů. Velikost prvního paketu P_1 ve frontě se porovná s aktuální hodnotou X . Pokud $X > P_1$, pak se hodnota čítače X se sníží o P_1 a paket se pře pošle. Přeposílání paketů pokračuje tak dlouho, dokud platí podmínka $X > P_i$. Pokud je čítač $X < P_i$, přenos se pozastaví o sekundu a čeká se na vypršení časovače a inkrementaci čítače X . Pak se znovu porovná hodnota čítače s velikostí aktuálního paketu ve frontě a podle výsledku se paket buď pře pošle nebo se opět čeká na vypršení časovače a zvýšení čítače.

Jestliže ve frontě čekají další pakety a přijde nový paket P_{new} , porovná se velikost tohoto paketu a velikost čekajících paketů S s velikostí vědra b . Jestliže $P_{new} + S > b$, došlo by k přeplnění fronty (buffer overflow) a paket P_{new} se zahodí. V opačném případě se paket P_{new} přidá do fronty.

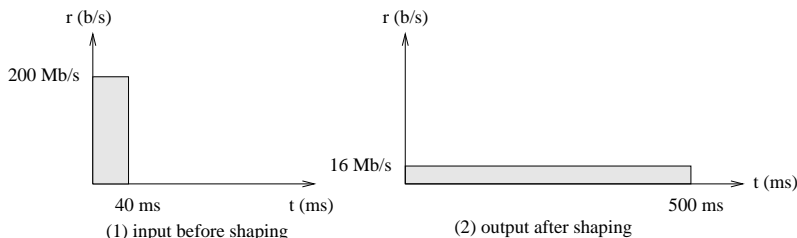
Algoritmus tekoucího vědra řídí tok dat v síti tak, že pakety nejsou přeposílány rychlostí větší než je povolená. Velikost vědra b ohraničuje maximální zpoždění, které paket při rozložení provozu může získat. Parametry algoritmu jsou přenosová rychlost r a velikost fronty b .

Příklad 8.2. Uvažujme počítač, který posílá do sítě data rychlostí 200 Mb/s. Směrovače na síti jsou schopny zpracovávat data o této rychlosti pouze po velmi krátkou dobu. Dlouhodobě mohou přijímat data o rychlosti 16 Mb/s. Předpokládejme, že na vstupu je shluk paketů o velikosti 1 MB. Abychom omezili rychlost na požadovaných 16 Mb/s, použijeme tekoucí vědro s $r=16$ Mb/s a kapacitou fronty $b = 1$ MB. Tato kapacita stačí, aby mohl být jeden shluk o velikosti 1 MB přijat bez zahození. Protože je rychlost redukována na 16 Mb/s, dojde ke zpoždění ve frontě. Vysílání tohoto shluku o velikosti 1 MB potrvá celkem 500 ms, což je i maximální zpoždění, které tekoucí vědro generuje viz obrázek 8.8.

Jak vidíme, algoritmus tekoucího vědra omezuje výstupní rychlost na danou hodnotu r . Pokud obsahuje frontu, tj. $b > 0$, jedná se o rozložení provozu (shaping), při kterém může docházet ke zpoždění paketů přicházejících větší rychlostí než je r . Jestliže je velikost fronty nulová, jedná se o řezání provozu (policing).

8.2.6 Mechanismus Token Bucket (zásobník žetonů)

Mechanismus tekoucího vědra slouží k omezení rychlosti na konstantní rychlost bez ohledu na to, jaké shluky dat se mohou v síti objevit. Tekoucí vědro vytváří pevnou charakteristiku rozložení výstupního provozu. Pro mnoho aplikací je vhodnější povolit krátkodobě překročení výstupní rychlosti tak, aby vstupní shluk dat mohl projít bez zahození. K tomu slouží mechanismus zásobníku žetonů (token bucket), který v zásobníku uchovává žetony (tokens).



Obrázek 8.8: Rozložení provozu pomocí algoritmu tekoucího vědra

Žeton odpovídá povolení přeposlat jeden byte. Žetony se generují určitou rychlostí a ovlivňují průchod vstupního toku.

Model tekoucího vědra neumožňuje uchovávat si povolení a posílat větší shluky dat později. Naopak zásobník žetonů umožňuje posílat shluky dat až do velikosti vědra b a to naráz a bez čekání. Krátkodobě může rychlost výstupu přesáhnout požadovanou průměrnou (střední) rychlost, velikost povoleného shluku dat je však omezena velikostí vědra. Zásobník žetonů tedy vytváří jiný typ rozložení provozu, než tekoucí vědro. Na výstupu se totiž mohou objevit omezené shluky dat, což tekoucí vědro neumožňuje.

Algoritmus zásobníku žetonů dovoluje propouštět určité množství shluků při dané průměrné rychlosti. Zároveň definuje, jaká je maximální povolená velikost shluku. K popisu přenosu využívá tyto veličiny: průměrnou rychlost CIR (Committed Information Rate), maximální velikost shluku CBS (Committed Burst Size) a maximální dobu trvání shluku T (Time Interval).

Průměrná rychlost CIR (Committed Information Rate) určuje velikost dat, které se pošlou za daný časový interval. CIR reprezentuje dlouhodobou průměrnou rychlost, kterou pakety toku vstupují do sítě. Někdy se mluví o střední rychlosti (mean rate). Rozhodujícím kritériem je interval, ve kterém se CIR zjišťuje. Tok s rychlostí 100 paketů za sekundu je více omezen než tok 6 000 paketů za minutu, i když oba toky mají v delším časovém intervalu stejnou rychlost. Je to tím, že druhý limit umožní přenést tisíc paketů během minutového intervalu (což může být i tisíc paketů za sekundu), zatímco v případě prvního limitu dojde po dosažení limitu 100 paketů za sekundu k zahazování.

Maximální velikost shluků CBS (Committed Burst Size) definuje počet bytů, které mohou být krátkodobě přeposlány po síti.

Časový interval T definuje dobu trvání shluku.

Rychlost ve špičce (PIR, Peak Information Rate). Zatímco průměrná rychlost CIR omezuje velikost provozu poslaného po síti za relativně dlouhou dobu, rychlost ve špičce PIR definuje maximální počet bytů, které lze přenést během kratší časové periody. Mějme například tok, který je omezen na rychlost 12 000 b/s. Tento tok může přenést 3000 bitů za 100 ms. V tomto intervalu dosáhneme přenosové rychlosti $PIR = 30\,000\text{ b/s}$. Pokud tok nepřeneseme ve zbytku intervalu více jak 9 000 bitů, průměrná rychlost CIR zůstává menší než 12 000 b/s.

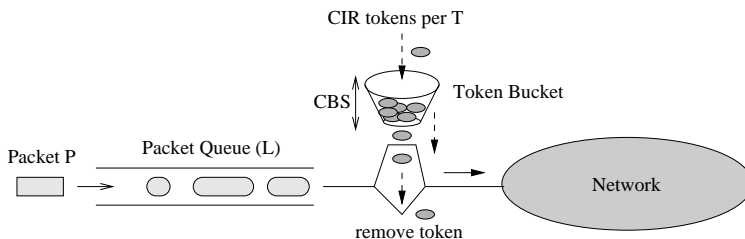
Průměrnou rychlost CIR můžeme vyjádřit jako poměr CBS a T, tj.

$$CIR = \frac{CBS}{T}$$

Tento vztah nám umožňuje vypočítat velikost zásobníku pro uložení shluků dat o délce trvání T . Zároveň platí, že rychlost provozu na síti během celočíselného násobku časového intervalu T nepřesáhne průměrnou rychlost CIR. Ukážeme si to na následujícím příkladu [4, kapitola 5.4.2]:

Příklad 8.3. Předpokládejme, že výstupní provoz do sítě je omezen průměrnou rychlostí 2,4 Mb/s a shluky dat mohou trvat 10 ms. Velikost shluku pro danou rychlost a interval je $CBS = CIR \times T = 2,4 \cdot 10^6 \times 0,01 = 24\,000$ b = 3000 B. Abychom nepřekročili CIR, rychlost generování žetonů povolujících provoz musí být 300 000 B/s, tj. každých 10 ms se vygeneruje 3000 žetonů. Výstupní provoz může v nejhorším případě tvořit 100 shluků o maximální velikosti 3000 bajtů. Rychlost CIR při tom zůstane 2,4 Mb/s.

Výše uvedený způsob řízení datového toku s krátkodobými výkyvy lze implementovat pomocí algoritmu Token Bucket. Algoritmus používá pro každý datový tok zásobník žetonů o velikosti CBS. Žeton lze považovat za povolení přenést ze vstupu na výstup určitý počet bytů, viz obrázek 8.9.



Obrázek 8.9: Model zásobníku žetonů (token bucket)

Velikost zásobníku CBS odpovídá maximálnímu shluku, který je možné přenést do sítě. Nové žetony se generují rychlostí CIR žetonů za časový interval. Pokud zásobník obsahuje méně žetonů než je hodnota CBS, přidají se nově vygenerované žetony do zásobníku. Pokud je zásobník plný nebo se zaplní, žetony nad kapacitu CBS se zahazují.

Když přijde paket o velikosti P na vstup a zásobník žetonů obsahuje X žetonů, kde $X \geq P$, odebere se P žetonů ze zásobníku a paket se přepoše na výstup. Žeton tedy odpovídá povolení přenést jeden byte. Pokud je zásobník prázdný nebo počet žetonů $X < P$, paket čeká ve frontě, dokud se nevygeneruje potřebný počet žetonů. Pak se paket přepoše.

Jestliže přijde na vstup shluk paketů a v zásobníku je určitý počet žetonů, pak se přepoše na výstup tolik paketů, jejichž velikost odpovídá aktuálnímu stavu žetonů v zásobníku, maximálně do velikosti CBS. Protože se do zásobníku přidávají žetony rychlostí CIR během intervalu T , maximální počet bytů, které je možné poslat do sítě za čas t , je $CBS + t \times CIR = CBS + t \times \frac{CBS}{T}$. Maximální rychlost přenosu shluku paketů ve špičce o délce trvání t je dána vztahem

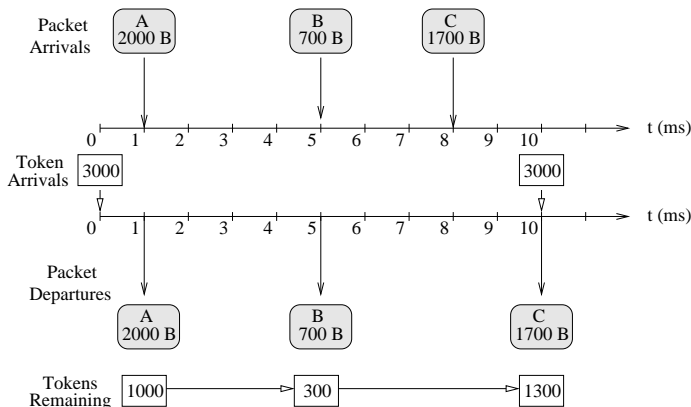
$$PIR = \frac{CBS}{t} + CIR$$

Na následujících příkladech si ukážeme, jak je možné využít zásobník žetonů pro řízení rychlosti výstupního toku dat.

Příklad 8.4. Uvažujme zásobník z příkladu 8.3, kde CBS je 3000 bytů a CIR 2,4 Mb/s. Maximální délka shluky T je 10 ms, tj. každých 10 ms se vygeneruje do zásobníku 3000 žetonů tak, abychom nepřesáhli rychlost CIR.

Předpokládejme, že v čase $t = 0$ je zásobník žetonů plný. Na vstup systému přichází následující pakety: v čase $t = 1$ ms přijde paket A o velikosti 2000 bytů. Protože je zásobník plný, může paket A okamžitě projít. Počet žetonů v zásobníku se sníží o 2000, v zásobníku tedy zůstává 1000 žetonů. V čase $t = 5$ ms přijde paket B o velikosti 700 bytů. I tento paket může okamžitě projít. Aktuální stav zásobníku bude 300 žetonů.

V čase $t = 8$ ms přichází paket C o velikost 1700 bytů. Protože nestačí počet žetonů, uloží se paket C do fronty, kde čeká. V čase $t = 10$ ms dojde k vygenerování nové sady žetonů. Protože v zásobníku zůstávalo 300 žetonů, přidá se do zásobníku pouze 2700 žetonů a zbytek se zahodí. Nyní už může paket C, projít. Obsah zásobníku se sníží o 1700 žetonů. Paket C prošel se zpožděním 2 ms, v zásobníku nyní zůstává 1 300 žetonů, viz obrázek 8.10.



Obrázek 8.10: Příklad rozložení provozu

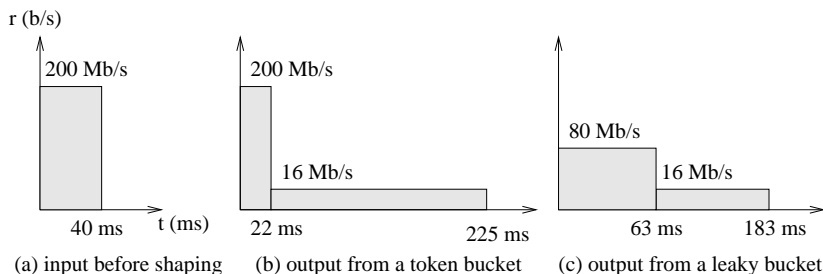
Můžeme vidět, že celkem bylo za 10 ms přeneseno 4400 bytů. Nedošlo však k překročení průměrné rychlosti CIR, neboť v intervalu $\langle 0; 10 \rangle$ bylo přeneseno 2700 bytů a v intervalu $\langle 10; 20 \rangle$ dalších 1700 bytů (maximálně to mohlo být 3000 bytů). Rychlost ve špičce v intervalu $\langle 0; 10 \rangle$ dosáhla hodnoty 3.52 Mb/s, nicméně v delším intervalu, například $\langle 0; 20 \rangle$ to bylo pouze 1,2 Mb/s. Vidíme, že asymptoticky je přenos omezen střední rychlostí CIR = 2,4 Mb/s. Povolené shluky jsou regulovány hodnotami CBS a T.

Jak ukazuje výše uvedený příklad, může krátkodobě dojít k překročení průměrné rychlosti CIR. Ve výše uvedeném příkladu může za čas 10 ms dosáhnout PIR rychlosti 4,8 Mb/s. Toto dvojnásobné zvýšení nám nemusí vždy vyhovovat. Pokud chceme tyto špičky nějakým způsobem upravit, je to možné vhodnou volbou hodnot CIR a CBS. Ne ve všech případech

nám to však vyhovuje. Proto se používá ještě jeden způsob – kombinace zásobníku žetonů a tekoucího vědra tak, že umístíme za zásobník žetonů tekoucí vědro o rychlosti r , kde r je větší než CIR, ale menší než PIR. Takto omezíme výstupní rychlosti na námi definovanou hodnotu. Použití se ukážeme v následujícím příkladu.

Příklad 8.5. Mějme datový tok o rychlosti 200 Mb/s, který má být omezen na 16 Mb/s pomocí zásobníku žetonů o velikosti 500 kB. Ukážeme si, jak bude vypadat výstup, pokud na vstup přijde shluk dat o velikosti 1 MB.

Pokud předpokládáme, že zásobník je před příchodem dat plný, pak mohou data o rychlosti 200 Mb/s procházet po dobu maximálně $(500 \times 10^3 \times 8) / (200 \times 10^6 - 16 \times 10^6) \doteq 22$ ms. Tedy za dobu 22 ms se přenesou shluk o velikosti 550 kB vstupní rychlostí 200 Mb/s. Poté se zásobník žetonů vyprázdní, přenesou se zbývajících 450 kB dat rychlostí CIR=16 Mb/s během $(450 \times 10^3 \times 8) / (16 \times 10^6) = 225$ ms, viz obrázek 8.11 (b).



Obrázek 8.11: Příklad rozložení provozu

Protože krátkodobá špička o rychlosti 200 Mb/s je příliš velká, chtěli bychom ji omezit. Proto přidáme za výstup zásobníku žetonů tekoucí vědro s rychlostí maximálně $r = 80$ Mb/s. Na vstup přijdou během prvních 22 ms data o velikosti 550 kB. Pokud předpokládáme, že velikost vědra je dostatečná, data se uloží do vědra a postupně odchází maximální možnou rychlostí 80 Mb/s, viz obrázek 8.11 (c).

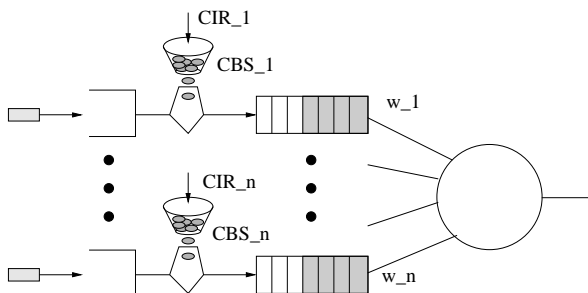
Odesílání dat trvá 55 ms. Během odesílání začnou přicházet další data rychlostí 16 Mb/s. Dokud je zásobník neprázdný, data se odesílají maximální možnou rychlostí, tj. 80 Mb/s. Za 33 ms, po kterých se odesílá první shluk dat, přišla rychlostí 16 Mb/s data o velikosti 66 kB dat. Ta se odešlou rychlostí 80 Mb/s, což bude trvat 6,6 ms. Během této doby opět přibudou do vědra nějaká data, konkrétně 13,2 kB, která se také odešlou nejvyšší možnou rychlostí. Velikost dat ve vědru se bude postupně snižovat, neboť rychlost vstupu 16 Mb/s je nyní pětikrát pomalejší než rychlost 80 Mb/s na výstupu.

Za jak dlouho se vyprázdní vědro? Vidíme, že přírůstek dat do zásobníku se snižuje rychlostí geometrické řady s kvocientem 0,2. Jedná se tedy o konvergující geometrickou řadu. Limitní součet této řady je $S = 33 / (1 - 0,2) = 41,25$ ms. Celkově tedy data odchází rychlostí 80 Mb/s po dobu 63,25 ms. Za tuto dobu se přenesou 632,5 kB dat. Zbytek dat se přenesou vstupní rychlostí 16 Mb/s, což potrvá cca 183 ms.

Techniky tekoucího vědra či zásobníku žetonů se používají k rozložení rychlosti toku či k ořezání špiček. Ve výše uvedených příkladech šlo o rozložení toku (shaping) s řízením délky

shluků. Ořezání (policing) by probíhalo podobně s tím rozdílem, že zásobník žetonů neobsahuje žádnou frontu pro ukládání paketů, které přijdou na vstup. Jestliže je v zásobníku dostatek žetonů, paket se přepoše. Pokud není vygenerován dostatečný počet žetonů, paket se zahodí.

Obě techniky řízení rychlosti toku dat můžeme implementovat například ve frontách aktivních síťových prvků. Mějme například směrovač, na jehož výstupu je každý z n toků umístěn do samostatné fronty. Vstup do fronty můžeme omezit například algoritmem zásobníku tokenů. Výstupní fronty jsou obsluhovány cyklickým výběrem s přidělenými váhami, například frontami typu WFQ, viz obrázek 8.12. Maximální zpoždění paketu d_{max} ve frontě i lze vy-



Obrázek 8.12: Rozložení toku dat s frontami WFQ

počítat pomocí vzorce

$$d_{max} = \frac{CBS_i}{R \cdot \frac{w_i}{\sum_{j=0}^n w_j}},$$

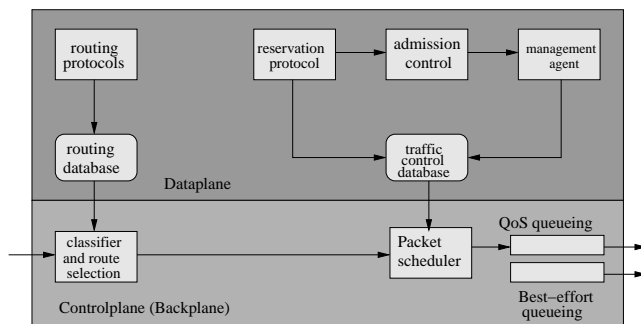
kde CBS_i je maximální velikost shluku paketů, R je maximální rychlost linky a w_i je váha fronty i . Pokud přijde shluk paketů do fronty i , budou obsluhovány rychlostí

$$R \cdot \frac{w_i}{\sum_{j=0}^n w_j}$$

či kratší. Poslední paket shluku bude mít nejdelší zpoždění, neboť před ním musí být obsluženo $(b - 1)$ paketů.

8.3 Integrované služby

V této části si představíme jeden ze způsobů, jak zajistit požadovanou kvalitu služeb pro konkrétní aplikace na Internetu, tzv. integrované služby (Integrated Services, IntServ). Integrované služby poskytují společnou (integrovanou) službu množině provozních požadavků v určité doméně, tj. části Internetu, kde je implementována podpora IntServ. Úkolem integrovaných služeb je zajistit podporu QoS v sítích nad IP, zejména jak sdílet dostupnou kapacitu síťových zdrojů v době zahlcení. Integrované služby tvoří několik součástí – rezervace zdrojů, řízení přístupu, způsob řízení front (queueing) a způsob zahazování paketů (discard policy). Příklad implementace integrovaných služeb ve směrovači je na obrázku 8.13. Hardwarová



Obrázek 8.13: Implementace integrovaných služeb ve směrovači

část směrovače (backplane), kterou prochází všechny pakety, je optimalizovaná na rychlost a probíhá v ní klasifikace paketů a výběr cesty ke směrování. Klasifikace určí, do jaké třídy provozu daný paket náleží. Třída může odpovídat jednomu či více tokům. Klasifikace probíhá na základě dat z hlavičky IP datagramu. Plánovač paketů (Packet Scheduler) obsluhuje jednu či více front pro každý výstupní port. Určuje pořadí, v jakém budou pakety vybírány z front, případně zda se zahodí. Rezervace výpočetních zdrojů, správa zdrojů a přidělování probíhá v softwarové části směrovače (dataplane).

Integrované služby zajišťují dvě základní funkce:

Rezervaci zdrojů (Reserved resources) – směrovač musí vědět, kolik zdrojů (velikosti paměti, přenosového pásma) je rezervováno pro probíhající komunikaci.

Vytvoření spojení (Call setup) – Relace vyžadující určitou kvalitu služeb musí být nejdříve schopná rezervovat si dostatečné množství zdrojů na každém síťovém směrovači od zdroje až k cíli. Tomuto se říká vytvoření spojení (call setup, call admission). Na vytvoření spojení musí spolupracovat každý směrovač po cestě. Musí určit lokální zdroje, které jsou požadovány pro dané spojení a zvážit množství již alokovaných zdrojů.

Vytváření spojení se skládá z následujících kroků:

1. *Specifikace provozu a požadavků na kvalitu služeb QoS*

Nejprve musí relace oznámit své požadavky na QoS (RSpec, Reservation Specification) a definovat provoz, který bude přenášet, tzv. TSpec (Traffic Specification). Mezi TSpec patří parametry dostupné přenosové pásmo, minimální doba latence, velikost MTU na cestě. Dále jsou to parametry pro algoritmus Token Bucket: průměrná rychlost generování žetonu r , velikost vědra b , rychlost ve špičce p , minimální jednotka přenosu m a maximální velikost paketu M , viz [5, 6].

2. *Signalizace pro vytvoření spojení*

Pro přenos parametrů relace TSpec a RSpec slouží protokol RSVP (Resource Reservation Protocol) [7]. Protokol je popsán v části 8.3.1.

3. Předběžné přijetí spojení

Jakmile směrovač obdrží specifikace RSpec a TSpec na zajištění kvality služeb, určí, zda spojení přijme. Pokud nemá dostatek volných zdrojů, tak spojení odmítne.

Architektura integrovaných služeb definuje tři základní třídy služeb: garantovanou službu (guaranteed service), službu řízené zátěže (controlled-load service) a službu doručení s nejlepším úsilím (best-effort service).

Garantovaná služba (guaranteed service) [8] definuje striktní zpoždění ve frontách během zpracování ve směrovači. Přenášený provoz je popsán mechanismem Token Bucket, tj. parametry (r, b) a požadovaná kvalita služby je charakterizována přenosovou rychlostí R . Jak jsme si už řekli u popisu algoritmu v části 8.2.6, velikost provozu (v bitech) generovaná za čas t je dána vztahem $r \cdot t + b$. Pokud řízení fronty zaručuje, že fronta bude obsluhována rychlostí nejméně R , pak maximální zpoždění ve frontě bude ohraničeno vztahem b/R , pokud R je větší než r .

Garantovaná služba zajišťuje tedy požadovanou datovou rychlost. Zároveň nedochází k žádným ztrátám z důvodu přeplnění front.

Síťová služba kontrolované zátěže (controlled-load service) [9] poskytuje datovému toku kvalitu služby, která se s velkou pravděpodobností blíží kvalitě nezatíženého síťového prvku. Pokud je prvek přetížen, využívá prvek řízení přístupu (admission control) tak, aby služba byla zajištěna.

Jinak řečeno velké procento paketů daného toku projde úspěšně síťovým prvkem, aniž by bylo zahozeno, a zároveň zpoždění ve frontách se bude blížit nule (nezatížený prvek). Tato definice neříká nic o tom, jak velké je to procento. Narozdíl od garantované služby tato služba totiž nazajišťuje kvantitativní parametry přenosu dat.

Služba s největším úsilím (best-effort service) – do této třídy spadají všechny pakety, které nepatří do žádného toku s rezervovanou službou. Většina toků patřící do této kategorie využívá mechanismus TCP pro řízení rychlost v případě zahlcení.

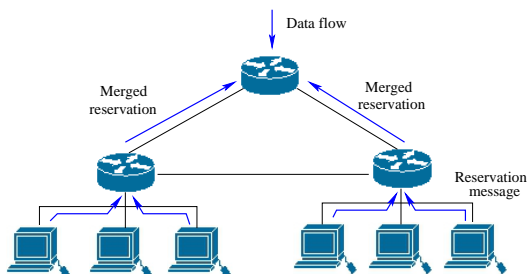
8.3.1 Rezervační protokol RSVP

Rezervační protokol RSVP (Resource Reservation Protocol)[7] je signalizační protokol, který umožňuje skupině vysílajících stanic přenést vysílané toky v požadované kvalitě k přijímajícím stanicím tak, aby přenosové pásmo bylo optimálně využito a předešlo se zahlcení přenosových zdrojů. Požadavek na rezervaci posílá koncová aplikace prostřednictvím protokolu RSVP nejblížejšímu směrovači, který ji přeposílá dále po cestě ke zdroji vysílání. Aby rezervace správně fungovala, musí všechny směrovače na cestě podporovat rezervaci pomocí RSVP.

Protokol RSVP poskytuje rezervaci přenosového pásma v multicastových stromech (multicast trees). Toto vychází z předpokladu, že většina aplikací, které budou žádat o rezervaci pásma, komunikují přes multicast – například sledování televizního či rozhlasového kanálu přes Internet apod. Unicastové vysílání lze v tomto případě považovat za speciální typ multicastu.

Ve své nejjednodušší podobě využívá protokol RSVP multicastové směrování pomocí hledání koster grafu (spanning tree). Multicastové směrovací algoritmy zajistí pokrytí všech uzlů, které chtějí komunikovat se zdrojem. Pro zajištění kvality přenosu může každý z příjemců poslat rezervační paket po multicastovém stromu. Protože rezervaci zdrojů pro požadovaný tok

iniciuje přijímací stanice, patří protokol RSVP mezi tzv. protokol orientovaný na příjemce (receiver-oriented protocol).



Obrázek 8.14: Multicastový přenos a orientace na příjemce u RSVP

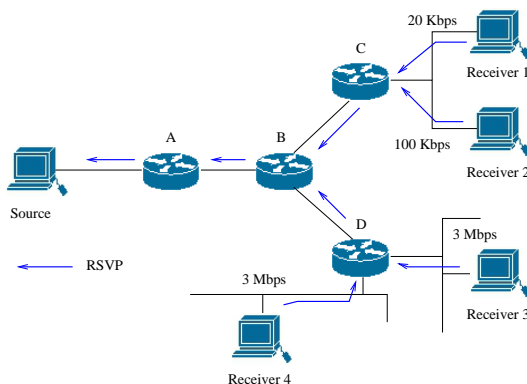
Na obrázku 8.14 vidíme multicastový strom s daty, které jdou od vrcholku stromu (vysílající uzel) ke koncovým stanicím. Ačkoliv data jsou generována odesílatelem, rezervační zprávy posílá přijímací stanice. Při přeposílání rezervačních zpráv směrem ke zdroji dat, může směrovač spojit rezervační zprávy s dalšími rezervacemi přicházejícími stejným směrem. Posílání zpráv ke zdroji multicastového vysílání využívá algoritmu Reverse Path Forwarding (RPF) [4, p.369].

Ačkoliv RSVP je rezervační protokol, tak nedefinuje, jak síťové prvky zajistí požadované přenosové pásmo pro datový tok. Síťové prvky to mohou udělat mnoha způsoby, například nastavením prioritních či váhových front. RSVP není směrovací protokol a tedy neurčuje, kudy data půjdou. Tuto funkci zajišťují klasické unicastové a multicastové směrovací protokoly. Jakmile je síť zkonverguje a je nalezena cesta, RSVP začne provádět rezervaci. Pokud dojde ke změně topologie, RSVP přerezervuje potřebné zdroje. Když je rezervace nastavena, musí plánovač v síťových prvcích zajistit požadované přenosové pásmo. RSVP je signalizační protokol, který umožňuje počítačům nastavit a ukončit rezervaci pro datové toky. Rezervované pásmo je potřeba udržovat prostřednictvím stavových zpráv. Jakmile přestanou docházet stavové zprávy RSVP pro danou rezervaci pásma, relace je ukončena a prostředky uvolněny.

Zajímavým způsobem řeší protokol RSVP rezervaci zdrojů v heterogenním prostředí, tj. v situaci, kdy různí klienti jsou připojeni ke zdroji zpráv linkami o rozdílné kapacitě, viz obr. 8.15. V tomto příkladě jsou připojeny ke stejnému zdroji dat čtyři koncové stanice Receiver 1 až Receiver 4. Předpokládáme, že zdrojová stanice Source chce vysílat multicastové video data na multicastovou adresu. Všechny koncové stanice jsou tedy připojeny do stejné multicastové skupiny. Otázkou je, jak proběhne rezervace přenosového pásma a jakou rychlostí se budou data vysílat.

Než ukážeme řešení, je potřeba připomenout, že rezervace probíhá směrem od koncových stanic. V praxi tedy každý koncový uzel pošle směrem k vrcholu multicastového stromu žádost o rezervaci. Tato žádost bude obsahovat specifikaci přenosového pásma, pomocí kterého chce příjemce načítat data. Jakmile rezervační zpráva dojde na nejbližší směrovač, směrovač si nastaví svůj plánovač paketů tak, aby uspokojil požadovanou rychlost (například nastavením prioritních front). Pak přepošle danou žádost směrem k vrcholu stromu.

Velikost přenosového pásma požadovaného směrovačem směrem ke zdroji dat souvisí s



Obrázek 8.15: Činnost RSVP v heterogenním prostředí

požadavky na pásmo, které směrovač obdržel od připojených uzlů. Například směrovač D dostal od obou koncových stanic požadavek na pásmo o velikosti 3 Mb/s. Protože jde o multicastový přenos typu z jednoho zdroje více příjemců (one-to-many), směrovač D pošle zprávu na B, ve které požádá o rezervaci pásma 3 Mb/s. Podobně i směrovač C pošle na B žádost o pásmo 100 kb/s na kapacitu linky mezi B a C. Aby bylo možno uspokojit oba požadavky, použije zdroj vysílání kódování videa ve dvou vrstvách: v nižší kvalitě pro linku 20 kb/s a ve vyšší kvalitě 100 kb/s. Zdroj tedy pošle video zakódované v obou požadovaných vrstvách maximální rychlosti do multicastového stromu. Příjemce si vybere vrstvu, která odpovídá jeho rychlosti přijímání.

Jakmile směrovač B dostane odpověď od směrovačů C a D, předá rezervační požadavek plánovačům a pošle novou rezervační zprávu směrovači A. Tato zpráva rezervuje pásmo 3 Mb/s mezi A a B, což je opět maximum z obou požadavků od zařízení směrem od zdroje vysílání.

Obecně řečeno, síťový prvek nejprve obdrží požadavek na rezervaci od svých sousedů směrem k příjemci, vyzkouší, zda může provést rezervaci (tzv. test přijetí, admission test), provede rezervaci a pošle nový požadavek, který je maximem všech přijatých požadavků, směrem ke zdroji. Pokud by síťový prvek nemohl provést rezervaci z důvodu nedostatku zdrojů, odmítne ji a odešle zpět chybovou zprávu.

Důležité je si uvědomit, že rezervace probíhá pro jednotlivé toky. Pokud například směrovač D vytvoří rezervaci pro pásmo pro příjemce 4 a přijde mu následně požadavek na rezervaci pro stejný tok od příjemce 3, nemusí směrovač D provádět další rezervaci, ale využije už stávající rezervaci a data od vysílající stanice přeposílá stanicím 3 a 4. Pokud by však stanice 3 požadovala rezervaci pro jiný tok (i kdyby pocházel ze stejné vysílající stanice) není možné požadavek agregovat, ale je potřeba rezervovat nové prostředky pro nový tok. Zde je také vidět slabé místo integrovaných služeb.

8.3.2 Zhodnocení integrovaných služeb

Integrované služby provádí alokaci prostředků pro jednotlivé datové toky v síti (per flow reservation), což umožňuje na jedné straně garantovat kvalitu služeb pro konkrétní toky, na druhou stranu to však přináší určité problémy.

Prvním z nich je rozšiřitelnost. Protože se jedná o rezervaci *per flow*, každý tok procházející směrovačem vyžaduje zpracování rezervačního požadavku a alokaci zdrojů. Zejména pro páteřní směrovače může být alokace zdrojů a výpočetní náročnost rezervace neúnosná.

Druhým problémem integrovaných služeb je, že architektura IntServ byla navržena pro malý počet předem specifikovaných tříd služeb. Množina tříd neumožňuje popsat kvalitativní či vztahové požadavky, například "Služba třídy A bude mít přednost před službou třídy B". Na tyto požadavky reaguje druhý model, diferenciované služby.

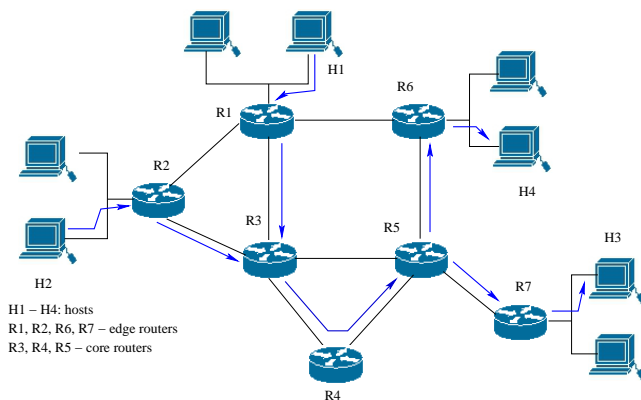
V současné době se řízení kvality přenosu pomocí integrovaných služeb spíše nepoužívá. Nicméně model rezervace přenosového pásma prostřednictvím protokolu RSVP je dneska běžný u přepínání sítí MPLS (Multiprotocol Label Switching) [10], kde se používá modifikovaná verze RSVP-TE (RSVP with Traffic Engineering extension) [11].

8.4 Diferenciované služby

Architektura diferenciovaných služeb (Differentiated Services, DiffServ)[12] se zaměřuje na rozšiřitelnost modelu a flexibilitu, konkrétně schopnost pracovat s různými třídami provozu (podpora diferenciací provozu). Rozšiřitelnost počítá s použitím DiffServ i na páteřních sítích, kde přes směrovače prochází sta tisíce toků. Hraniční směrovače provádějí výpočetně náročnější operace DiffServ, zatímco páteřní směrovače provádějí spíše jednodušší funkce. Flexibilní návrh používá nejen předdefinované třídy, ale umožňuje i vlastní rozdělení provozu a definici tříd. Architekturu diferenciovaných služeb tvoří dva typy prvků:

Hraniční prvky. Hraničními prvky jsou buď koncová zařízení nebo nejbližší směrovače, které podporují diferenciované služby. Tyto hraniční prvky značují (marking) procházející pakety, konkrétně nastavují hodnotu DS (Differentiated Services) v hlavičce IP datagramu. Například na obrázku 8.16 jsou pakety poslané z uzlů H1 na H3 označeny na směrovači R1. Pakety poslané z H2 a H4 jsou zase označeny na směrovači R2. Značení, které daný paket dostane, identifikuje třídu provozu, do které patří. Různé třídy mají různé značení. Standard RFC 2475 [12] používá termín agregace chování (behavior aggregate) místo termínu třída provozu (class of traffic). Po té, co je paket označen, může být (i) poslán do do sítě, (ii) vložen do fronty nebo (iii) zahozen.

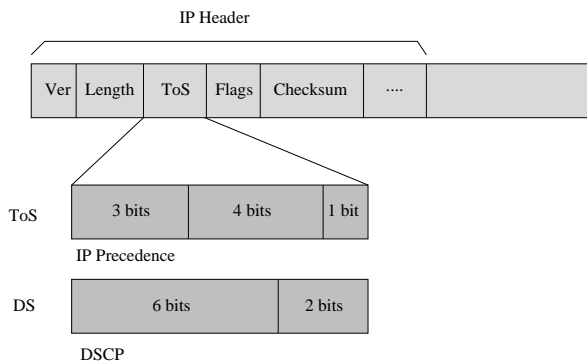
Páteřní prvky. Když přijde označený paket na směrovač s podporou DiffServ, je přeposlán na další uzel podle definovaného chování příslušící jeho třídě (tzv. PHB, Per-hop Behavior). Nastavené chování PHB ovlivňuje sdílení paměti a přenosové pásma mezi různými třídami provozu. Podstatné na architektuře DiffServ je, že na páteřním směrovači se zpracování paketu řídí pouze značením v hlavičce IP. Pokud například pakety jdoucí z počítače H2 na H4 a pakety toku z H1 na H3 obdrží stejné značení, pak je směrovač R3 nebude rozlišovat. Z jeho pohledu půjde o jeden agregovaný tok popsany stejnou třídou provozu.



Obrázek 8.16: Příklad diferenciovanych služeb DiffServ

8.4.1 Klasifikace provozu pomocí DSCP

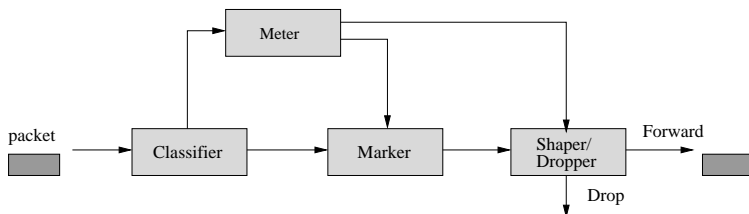
U diferenciovanych služeb je značení paketu uloženo v poli DS (Differentiated Services) v hlavičce protokolu IP. Toto pole není v původní definici protokolu IP [13]. Původní definice obsahuje osmitbitové pole **Type of Service**. Pozdější standardy [1, 14] zavedli jinou interpretaci tohoto pole a změnili jeho název na DS. Pole DS se skládá z šestibitového kódu DSCP (Differentiated Service Code Point), který definuje typ chování PHB (Per-host Behavior), a dvoubitové hodnoty CU (Currently Unused), která se momentálně nepoužívá, viz obr. 8.17.



Obrázek 8.17: IP precedence a DSCP v hlavičce IP datagramu

Nastavení hodnot DSCP provádí hraniční prvek sítě na základě klasifikace provozu, viz

obr. 8.18. Klasifikace může být provedena podle různých kritérií. Nejčastěji to je podle zdrojové či cílové adresy, čísla portu, typu protokolu. Například poskytovatel internetového připojení (ISP) může podle smlouvy SLA označit data pocházející z dané zdrojové adresy daného zákazníka určitým kódem DSCP a přeposílat je s vyšší prioritou po své síti. Může však dojít ke klasifikaci na základě obsahu paketů, což je výhodné například pro provoz tunelovaný přes port 80.



Obrázek 8.18: Klasifikace, označení a práce s pakety u DiffServ

Klasifikátor (obrázek 8.18) tedy rozděluje přicházející pakety do tříd podle hodnot z hlavičky IP či obsahu paketů. Komponenta *Meter* kontroluje procházející provoz a přiřazuje ho do předdefinované třídy provozu (profilu provozu). *Meter* také sleduje, zda daný provoz nepřekročil stupeň garantované služby. Značkovací komponenta *Marker* označuje (či přeznačuje) pakety novým kódem DSCP. Pokud tok překročil úroveň dohodnuté služby, je označen kódem nižší třídy (například BE, best-effort). Pokud se uzel provádějící klasifikaci nachází na hranici dvou ISP (mezi dvěma doménami DS), může provádět přeznačení paketů z hodnot definovaných jedním ISP na hodnoty dohodnuté s druhým ISP. Komponenta *Shaper* provádí rozložení provozu (shaping) v čase tak, aby provoz nepřekročil dohodnutou úroveň služby dle SLA. Pokud tok překročí stanovené hodnoty přenosu, budou pakety zahozeny komponentou *Dropper*. Příklad mapování IP precedence a hodnot DSCP podle [15] je v tabulce 8.1.

Třída provozu	IP Prec	DSCP	Kód třídy	Příklad
Interactive Voice	5	46	EF	RTP/RTCP
Interactive Video	4	34, 36, 38	AF41, AF42, AF 43	RTP
Streaming Media	4	32	CS4	SRTP
Local Mission Critical	3	28, 30	AF32, A33	Oracle, SAP
Telephony Signaling	3	26	AF31	SIP, H.323
Network Management	2	16	CS2	
Bulk Data	1	10, 12, 14	AF11, AF12, AF13	FTP, TFTP
Scavenger	1	8	CS1	Napster, P2P
Best Effort	0	0	BE	WWW, Email

Tabulka 8.1: Mapování hodnot IP precedence a DSCP na standardní třídy provozu

Výše uvedené mapování je jedno z doporučení, které používá firma Cisco. Toto mapování je provedeno na základě hodnot z IP hlavičky. Označení priority lze vkládat i do rámců protokolů druhé vrstvy, například lze využít tříbitové pole PRI u Ethernetu 802.1p, jednobitovou hodnotu DE v hlavičce Frame Relay či tříbitovou hodnotu Exp v hlavičce protokolu MPLS.

8.4.2 Definice chování pomocí PHB

PHB (Per-hop Behavior) popisuje způsob zpracování toků označených pomocí DSCP. Pro různé třídy provozu je definováno různé zpracování. PHB popisuje pouze definici chování a nezabývá se vlastní implementací, která ji zajistí, tzn. neřeší například typy použitých front apod. Příkladem PHB může být garance přenosového pásma o velikost M procent kapacity výstupní linky pro třídu provozu A . Standardy IETF RFC 3246 [16] a RFC 2597 [17] definují dva speciální typy PHB:

Přednostní přeposílání EF (Expedited Forwarding) [16] definuje, že výstupní rychlost odeslání paketů dané třídy ze směrovače musí být stejná nebo vyšší než nakonfigurovaná rychlost. Toto je garantováno bez ohledu na intenzitu ostatního provozu. Pokud provoz ostatních tříd povede k zahlcení směrovače, musí mít směrovač stále dostatečnou rezervu zdrojů, aby zajistil minimální požadovanou rychlost odpovídající požadovanému chování. EF tvoří třídu provozu pro abstraktní linku s minimální garantovanou přenosovou rychlostí. Pro přednostní přeposílání se používá jedna třída EF s doporučenou hodnotou DSCP 46 (101110), viz tab. 8.2.

Garantované přeposílání AF (Assured Forwarding) [17] rozděljuje provoz do čtyř tříd AF1 až AF4, z nichž každé třídě je zaručena určitá přenosová rychlost. Zároveň každá třída definuje tři úrovně pravděpodobnosti zahazení paketu v případě zahlcení, viz tabulka 8.2. Pokud dojde k zahlcení linky, směrovač začne zahazovat nejprve pakety vyšší pravděpodobností zahazení (High Drop Precedence).

Třída	Kód DSCP	Priorita zahazování
BE	000000 (0)	Default (Best Effort)
AF11	001010 (10)	Low Drop Prec
AF12	001100 (12)	Medium Drop Prec
AF13	001110 (14)	High Drop Prec
AF21	010010 (18)	Low Drop Prec
AF22	010100 (20)	Medium Drop Prec
AF23	010110 (22)	High Drop Prec
AF31	011010 (26)	Low Drop Prec
AF32	011100 (28)	Medium Drop Prec
AF33	011110 (30)	High Drop Prec
AF41	100010 (34)	Low Drop Prec
AF42	100100 (36)	Medium Drop Prec
AF43	100110 (38)	High Drop Prec
EF	101110 (46)	

Tabulka 8.2: Tabulka tříd DSCP

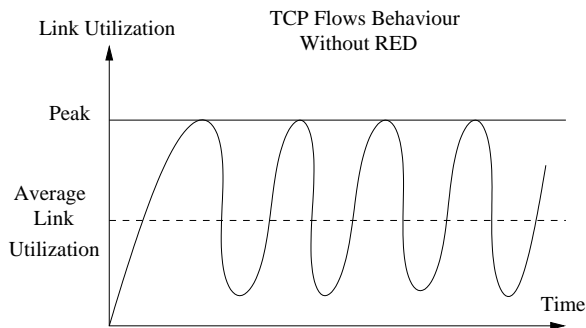
Implicitní třída BE (Best-effort) má první tři bity kódu nastavené na nuly. Jedná se o implicitní třídu, která se použije, pokud není hodnota DSCP v paketu namapovaná na nějakou politiku PHB. Jde o doručení s největším úsilím (best effort).

V současné době se používají diferenciované služby více než integrované služby. Rozšíření diferenciovaných služeb je obvykle omezeno na jednu administrativní doménu (DS doménu),

což většinou zahrnuje síť jednoho poskytovatele internetového spojení. Zaručení kvality služeb toku mezi dvěma libovolnými koncovými stanice je však závislé na všech poskytovatelích, přes něž se tok přenáší. V dnešní době však zpoždění datového toku ovlivňuje spíše kapacita přístupové linky a počet skoků na cestě než zpoždění ve frontách na směrovačích.

8.5 Předcházení zahlcení pomocí RED a WRED

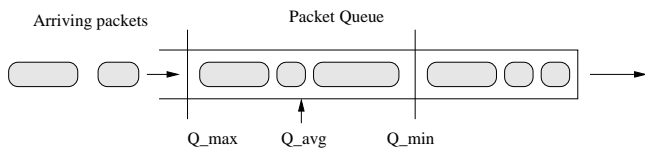
V této části si popíšeme způsob, jak předcházet zahlcení ve frontách na síťových zařízeních. Obecně platí, že každá fronta je konečná a je možné ji někdy zahltit. Pokud dojde k zaplnění fronty, začnou se zahazovat všechny nově příchozí pakety. Jestliže tyto pakety náleží k toku TCP, mechanismus řízení toku TCP zmenší přenosové okno a sníží rychlost toku. To způsobí výrazný pokles nároků na přenosové pásmo a následně i uvolnění front. Jakmile se zlepší průchodnost linky, TCP začne zvyšovat rychlost, až dojde opět k zahlcení. Pak opět následuje snížení rychlosti a celý cyklus se opakuje. Jedné se o tzv. *problém globální synchronizace TCP*, viz obrázek 8.19.



Obrázek 8.19: Problém globální synchronizace toků TCP při zahlcení linky

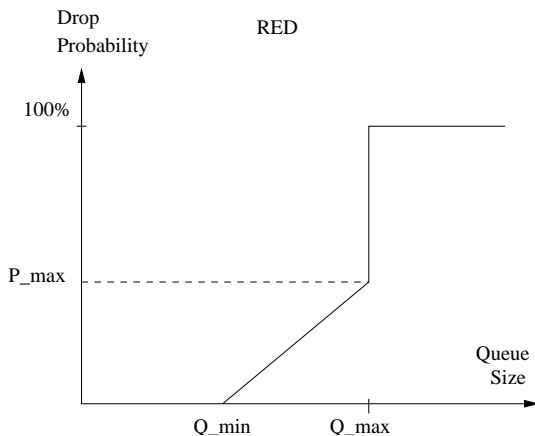
Při zahlcení může dojít ještě k jinému problému. Pokud o přenosové pásmo soutěží toky TCP a UDP, pak v případě ztrát dojde ke snížení rychlosti pouze u přenosů TCP. UDP nemá žádný mechanismus řízení rychlosti. Jestliže prochází zařízením agresivní tok UDP, může tento tok využít snížení rychlosti TCP k dalšímu obsazení přenosové kapacity linky. Tím donutí ostatní toky TCP k ještě většímu snížení rychlosti, až je úplně vytěsní. Toky UDP začnou předbíhat a vytlačí komunikaci TCP. Tento problém se nazývá *vyhladovění TCP (TCP starvation)*.

Řešením výše uvedených problémů je implementace mechanismu pro prevenci zahlcení, který sleduje nárůst paketů ve všech frontách v systému. Pokud velikost fronty daného toku přesáhne minimální práh Q_{min} , začnou se náhodně zahazovat přicházející pakety daného toku. Pravděpodobnost zahození se zvyšuje v čase s délkou fronty. Když přestane růst, pravděpodobnost zahazování se nemění. Pokud zaplnění fronty poroste, dojde k častějšímu zahazování. Pokud délka fronty dosáhne kritické hodnoty maximálního prahu Q_{max} (což může být stejné jako fyzická velikost fronty), dojde ke stoprocentnímu zahazování paketů, viz obr. 8.20.



Obrázek 8.20: Mezní hodnoty RED ve frontě

Tento algoritmus se nazývá RED (Random Early Detection, včasná náhodná detekce). Na grafu 8.21 můžeme vidět vztah pravděpodobnosti zahazování paketů a délky fronty. Algoritmus se může nacházet ve třech stavech: stav nezahazování ($0, Q_{min}$), stav náhodného



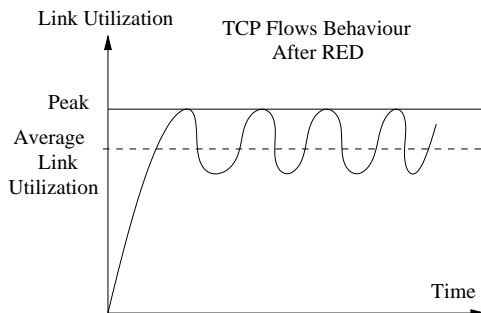
Obrázek 8.21: Pravděpodobnost zahazování mechanismem RED

zahazování ($< Q_{min}, Q_{max}$) a stav úplného zahazování nově příchozích paketů (tzv. tail drop), ($> Q_{max}, \infty$). Algoritmus RED vybírá pro zahození pakety bez ohledu na jejich vlastnosti (prioritu apod.). Pravděpodobnost zahození P_a závisí pouze na aktuální délce fronty a je popsána lineární funkcí

$$P_a = P_{max} \frac{Q_{avg} - Q_{min}}{Q_{max} - Q_{min}},$$

kde Q_{avg} je průměrná délka fronty. Pokud se zvýší průměrná délka fronty, zvýší se i hodnota P_a . Když dosáhne délka fronty hodnoty Q_{max} , bude pravděpodobnost zahození P_{max} . Hodnota se může nastavit podle potřeby v intervalu $0 \leq P_{max} \leq 1$.

Detekce zahltí pomocí algoritmu RED má vliv i na plynulejší využití přenosového pásma. Tento přístup nemůže eliminovat výkyvy, které vzniknout chováním TCP při ztrátách paketů (tj. snížení rychlosti), ale určitým způsobem přizpůsobí výkyvy v rychlostech kapacitě



Obrázek 8.22: Využití linky při použití RED

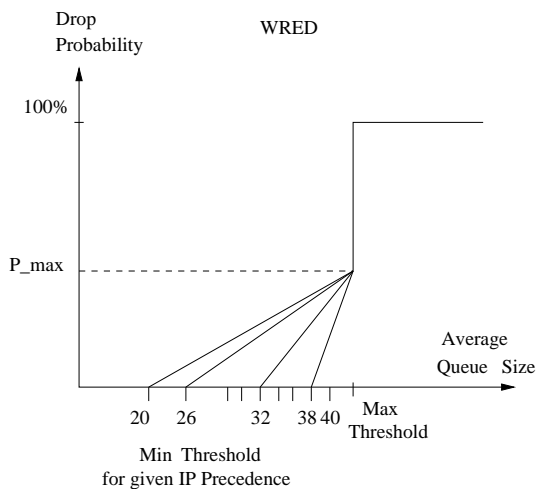
přenosové linky, viz obrázek 8.22. Na tomto obrázku vidíme, že oproti případu na obrázku 8.19 se nám výrazně posunula hodnota průměrného využití linky. Což znamená, že se lépe využívá celková kapacita linky, oproti případu bez použití detekce zahlcení a náhodného zahazování.

Nastavení hodnot Q_{min} a Q_{max} určuje, jak efektivně budeme linku využívat. Pokud hodnota Q_{min} je příliš nízká, povede výskyt shluků paketů v toku dat k zahazování a k nevyužití kapacity linky. Hodnota Q_{min} by měla být dost velká na to, aby směrovač byl schopen přijmout dostatečně velký shluk paketů bez potřeby náhodného zahození. Hodnota Q_{max} určuje zpoždění paketu během přenosu ve směrovači. Velká hodnota Q_{max} znamená, že více paketů bude čekat ve frontě, což zvýší zpoždění pro každý nově příchozí paket. Pro provoz na Internetu je doporučeno nastavit Q_{max} alespoň dvakrát tak velké jako je Q_{min} . Během velkého zatížení se bude průměrná délka fronty pohybovat mezi těmito dvěma hodnotami a zbyde dostatečný prostor pro přijetí shluku paketů bez toho, aniž by se došlo k zahazování. Použití algoritmu RED si ukážeme na následujícím příkladu.

Příklad 8.6. Mějme vstupní frontu s maximální kapacitou 25 paketů. Fronta začíná zahazovat pakety, jakmile obsahuje alespoň pět paketů, tzn. $Q_{min} = 5$, $Q_{max} = 25$. Pokud přijde do fronty šestý paket, pravděpodobnost zahození bude $P_6 = \frac{6-5}{25-5} = \frac{1}{20} = 5\%$. Pokud se fronta zaplní na 11 paketů a přijde další, pak pravděpodobnost jeho zahození bude $P_{12} = \frac{12-5}{25-5} = \frac{7}{20} = 35\%$. Pokud dále poroste fronta a bude obsahovat například 21 paketů, pak další, který přijde se zahodí s pravděpodobností $P_{22} = \frac{22-5}{25-5} = \frac{18}{20} = 90\%$.

Jedním z rozšíření algoritmu RED je přidání váh pro jednotlivé třídy paketů. Modifikace se nazývá *váhový RED* – *WRED* (Weighted RED). WRED nezahazuje všechny pakety se stejnou pravděpodobností, ale rozlišuje jejich důležitost pomocí hodnot IP Precedence nebo DSCP.

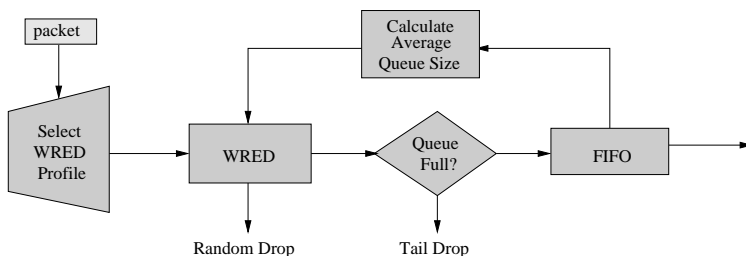
Příklad na obrázku 8.23 ukazuje využití značení klasifikovaných dat a tříd paketů při zahazování WRED. Pro různé třídy paketů definujeme různé fronty s odlišnou politikou zahazování, která se definuje pomocí hodnot Q_{min} a Q_{max} . Například pakety s prioritou IP precedence 0,1 a 2 používají frontu s $Q_{min} = 20$, pakety s prioritou 3,4,5 frontu s $Q_{min} = 26$, pakety s prioritou 6 a 7 využívají délku fronty s $Q_{min} = 32$ a pakety patřící například toku RSVP se ukládají do fronty o délce minimálně 38 [15]. Pakety u toků s nižší prioritou budou



Obrázek 8.23: Algoritmus WRED

zahazovány dříve, čímž umožní průchod paketům s vyšší prioritou.

WRED pro každý paket ve frontě vypočte aktuální délku fronty. Pokud přijde paket, dojde ke klasifikaci podle hodnoty IP **precedence** nebo hodnoty DSCP. Podle typu třídy (profilu) a aktuální délka fronty se určí pravděpodobnost zahazení. Jestliže paket není zahazen na základě průměrné délky fronty, může být umístěn do výstupní fronty FIFO. Jestliže je výstupní fronta plná, je zahazen. Pokud není plná, je přidán do fronty a přepočítá se aktuální velikost fronty, viz obrázek 8.24.



Obrázek 8.24: Schéma chování algoritmu WRED

Použití algoritmu RED a WRED řeší problém globální synchronizace TCP i vyhladovění TCP, neboť pravděpodobnost zahazování je přímo úměrná dravosti toku. Agresivní toky

budou zahazovány s větší pravděpodobností. V konečném důsledku přináší RED a WRED efektivnější využití přenosového pásma, viz graf 8.22.

8.6 Implementace kvality služeb

V předchozích částech jsme si představili různé způsoby pro zajištění kvality služeb při přenosu po IP sítích, zejména mechanismy pro zpracování ve frontách a předcházení zahlcení. Dále jsme si ukázali dvě základní architektury pro implementaci kvality služeb – modely integrovaných a diferenciovanych služeb. Nyní se podíváme na příklad konkrétní implementace diferenciovanych služeb tak, jak je používají zařízení od firmy Cisco [15]. Implementace QoS zahrnuje tři hlavní části:

1. Identifikace typu provozu a požadavky na daný typ provoz

Identifikace typu provozu zahrnuje několik kroků. První z nich je tzv. síťový audit (network audit), který identifikuje provoz na síti. Audit se doporučuje provádět během pracovní doby nebo v čase zahlcení počítačové sítě. Nejjednodušší způsob je sledovat výkonnost procesoru a vytíženost linek. Další částí je stanovení tříd provozu a přiřazení identifikovaných aplikací do těchto tříd provozu. Důležitým kritériem při posuzování je, zda zpoždění či zahození paketů je pro danou aplikaci přijatelné. Určujeme tedy relativní důležitost aplikací. Pro každou třídu je potřeba stanovit požadavky na zdroje, zejména garanci minimálního a maximálního přenosového pásma, maximální zpoždění, maximální jitter a míru zahození paketů. Pro identifikaci typu provozu lze použít speciální aplikace typu Cisco NBAR (Network Based Application Recognition).

2. Klasifikace provozu podle požadavků do tříd (traffic classes) Ve druhé fázi implementace QoS se zaměřujeme na vytvoření několika tříd, do kterých se rozdělí provoz identifikovaný v předchozí fázi. Při identifikaci objevíme desítky či stovky aplikací, které umístíme do několika málo tříd. Třídy musí odrážet cíle, kterých chceme dosáhnout, například zajištění kvality definovaných služeb. Aplikace v dané třídě by měly mít podobný význam a požadavky na přenos. Je vhodné vytvářet speciální třídu, do které dáváme ostatní provoz, který nechceme v síti podporovat (scavenger class, odpadky). Příklad rozdělení podle [15] je v tabulce 8.3.

Třída	Stupeň priority	Typ fronty	Přenosové pásmo
Voice	5	Prioritní fronty	1 Mb/s
Mission Critical	4	Class-based WFQ	1 Mb/s
Signaling	3	Class-based WFQ	400 kb/s
Transactional	2	Class-based WFQ	1 Mb/s
Best-effort	1	Class-based WFQ	max. 500 Kb/s
Scavenger	0	Class-based WFQ	max. 100 Kb/s

Tabulka 8.3: Třídy provozu pro implementaci QoS

3. Definice politik pro zpracování každé třídy provozu (QoS policies)

Pro jednotlivé třídy provozu nyní určíme kvalitu služeb požadovanou podle SLA. Politika pro nastavení kvality služeb (QoS policy) pro jednotlivé třídy zahrnuje stanovení

maximální a minimální šířky přenosového pásma, přiřazení stupně priority a techniky, jak to zajistit, např. pomocí front. V příkladu 8.3 je použita například prioritní fronta s přenosovým pásmem o kapacitě 1 Mb/s pro hlasové přenosy a váhové fronty pro další data. Toto přidělení prostředků znamená, že například hlasová data až do velikosti 1 Mb/s budou využívat své přenosové pásmo a prioritně předbíhat ostatní data. Pokud by hlasových dat bylo více, mohou použít nevyužitou kapacitu ostatních tříd. Pokud by však ostatní třídy plně využili své přidělené pásmo, pak se budou hlasová data nad 1 Mb/s zahazovat.

Shrnutí

Zajištění kvality služeb u datových přenosů nad IP je jedním z hlavních úkolů poskytovatelů internetového připojení. Je to zejména otázka nabídky služeb zákazníkovi a garance kvality těchto služeb. Mezi požadavky na kvalitu služeb patří velikost přenosového pásma, spolehlivost služby, maximální zpoždění dat na síti, přednostní doručení hlasových a videokonferenčních dat apod. Tyto požadavky jsou často specifikovány v dokumentu nazvaném SLA (Service Level Agreement), který zavazuje poskytovatele zaručit dohodnuté služby ve specifikované kvalitě.

Protože přenos po internetu je založen na paketových sítích a sdílení přenosových zdrojů, musí poskytovatel alokovat pro každého zákazníka dostatek zdrojů k pokrytí jeho požadavků. Pro zajištění těchto požadavků se používá označování paketů (marking), rozložení provozu (traffic shaping) a ořezání provozu (traffic policing). Tyto techniky slouží k řízení rychlosti provozu, zajištění maximální dohodnuté rychlosti, prioritizaci paketů s vyššími nároky na doručení apod. Implementují se například pomocí různých typů front na aktivních síťových prvcích, zejména směrovačích.

Směrovače mohou obsahovat různé typy fronty pro vybranou třídu provozu – například fronty typu FIFO, prioritní fronty, cyklické či váhové fronty. Obecný mechanismus pro řízení toku dat (rozložení či ořezání) ve frontě můžeme popsat algoritmem tekoucího vědra (Leaky Bucket) nebo zásobníku žetonů (Token Bucket), které umožňují řídit základní parametry síťového provozu: průměrnou rychlost, rychlost ve špičce a velikost shluků paketů, apod.

Pro zajištění přenosů na Internetu s požadovanou kvalitou služeb se používají dva základní modely: integrované a diferenciované služby. Integrované služby pracují s jednotlivými toky. Rezervují pro ně prostřednictvím signalizačního protokolu RSVP dostatečné síťové zdroje. Integrované služby definují tři základní třídy služeb: (i) garantovanou službu, která definuje striktní zpoždění ve frontách a garantuje požadovanou službu, (ii) službu kontrolované zátěže, kde je toku poskytována taková služba jako u nezátíženého prvku, a (iii) službu s největším úsilím, což je běžný způsob zajištění přenosu pomocí protokolu IP.

Diferenciované služby využijí metodu klasifikace a značení paketů přicházejících do sítě na hraničním směrovači. Pakety dané třídy přeposílají ve vnitřní síti provozovatele podle definovaného chování PHB (Per Hop Behavior). Značené pakety využívají pole DSCP v hlavičce IP datagramu, které definuje čtyři třídy s garantovaným přeposíláním (AF) a jednu třídu se přednostním zasláním (EF). Podle typu třídy a chování provozu může směrovač daný paket přeposlat, přeznačit na třídu nižší kvality nebo zahodit.

Pro předcházení zahlcení na směrovačích používají aktivní prvky na síti mechanismus preventivního zahazování paketů ve frontách s cílem snížit nárůst front a zpomalit agresivní toky na síti. Jedná se o náhodný výběr a zahazování paketů podle pravděpodobnostních mo-

delů RED a WRED. Tyto techniky sledují nárůst paketů ve všech frontách v systému. Pokud velikost fronty přesáhne definovaný práh, začnou se pakety daného toku zahazovat. Pravděpodobnost zahazování roste s délkou fronty. Po dosažení limitu fronty se budou zahazovat všechny příchozí pakety do doby, než se fronta uvolní.

V této kapitole jsme si představili některé možnosti a techniky sloužící k nastavení a zajištění kvality služeb na síťových zařízeních. Tyto techniky se v praxi často kombinují. Pro úspěšné nasazení v provozu existují definované doporučené návrhy, například návrhové vzory (design guides) či doporučené postupy (best-practices), které stanoví, pro jaký typ provozu a pro jakou službu je vhodné použití daný typ front, techniku zahazování apod.

Otázky

1. Co je to SLA (Service Level Agreement), co obsahuje a jak se používá?
2. Jaký je rozdíl mezi ořezáním provozu (traffic policing) a rozložením provozu (traffic shaping). Uveďte výhody a nevýhody obou přístupů.
3. Mějme dva toky s limitem průměrné rychlosti 100 paketů za sekundu u prvního a 6,000 paketů za minutu u druhého. Který tok je více omezen?
4. Popište algoritmus tekoucího vědro (Leaky Bucket) a uveďte příklad jeho použití.
5. Popište algoritmus zásobníku žetonů (Token Bucket) a uveďte příklad jeho použití.
6. Jaké fronty se používají na směrovači? Popište princip jejich použití.
7. Jak velké shluky paketů dovede fronta WFQ bez zahození přijmout? Na čem to závisí?
8. Co jsou to integrované služby, k čemu slouží? Popište, jak se implementují.
9. Jaké základní třídy definují integrované služby? Vysvětlete je.
10. Co jsou to diferenciované služby? Popište jejich základní komponenty a činnost.
11. Na základě jakých informací provádí DiffServ klasifikaci provozu?
12. Jaké znáte základní třídy diferenciovaných služeb? Popište je.
13. Porovnejte použití integrovaných a diferenciovaných služeb.
14. Popište význam a použití protokolu RSVP.
15. Jak lze předcházet zahlcení? Uveďte způsob prevence.
16. Jak se zachová přenos TCP v případě zahlcení linky?
17. Vysvětlete stavy obsluhy fronty při použití algoritmu RED.
18. Čím se liší RED a WRED?
19. Jaké bude chování sítě, když zmenšíme hodnotu Q_{min} ?
20. Jaké bude chování sítě, když zvětšíme hodnotu Q_{max} ?
21. Uveďte základní části implementace kvality služeb v síti.

Použitá literatura a standardy

- [1] K. Nichols, S. Blake, F. Baker, and D. Black. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers*. RFC 2474, December 1998.
- [2] William Stallings. *High-Speed Networks and Internets. Performance and Quality of Service*. Prentice Hall, 2nd edition, 2002.
- [3] D. Medhi and K. Ramasamy. *Network Routing. Algorithms, Protocols, and Architectures*. Elsevier, Inc., 2007.
- [4] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, 4th edition, 2003.
- [5] J. Wroclawski. *The Use of RSVP with IETF Integrated Services*. RFC 2210, September 1997.
- [6] S. Shenker and J. Wroclawski. *General Characterization Parameters for Integrated Service Networks Elements*. RFC 2215, September 1997.
- [7] R. Braden, L. Zhang, S. Berson and S. Herzog, and S. Jamin. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205, September 1997.
- [8] S. Shenker, C. Partridge, and R. Guerin. *Specification of Guaranteed Quality of Service*. RFC 2212, September 1997.
- [9] J. Wroclawski. *Specification of the Controlled-Load Network Element Service*. RFC 2211, September 1997.
- [10] E. Rosen, A. Wiswanathan, and R. Callon. *Multiprotocol Label Switching Architecture*. RFC 3031, January 2001.
- [11] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, and G. Swallow. *RSVP-TE: Extensions to RSVP for LSP Tunnels*. RFC 3209, December 2001.
- [12] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Service*. RFC 2475, December 1998.
- [13] J. Postel. *Internet Protocol*. RFC 791, September 1981.
- [14] D. Grossman. *New Terminology and Clarification for DiffServ*. RFC 3260, April 2002.
- [15] Amir S. Ranjbar. *CCNP ONT Official Exam Certification Guide*. Cisco Press, 2007.
- [16] B. Davie, A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. *An Expedited Forwarding PHB (Per-Hop Behavior)*. RFC 3246, March 2002.
- [17] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. *Assured Forwarding PHB Group*. RFC 2597, June 1999.

Rejstřík

diferenciované služby, 298

DSCP, 299–301, 304

fronty, 283

 cyklické fronty, 285

 FIFO, 284

 prioritní fronty, 284

 váhové fronty, 286, 293

Integrované služby, 293

klasifikace paketů

 použití, 282, 294, 300, 306

Leaky Bucket, *viz* tekoucí vědro

marking, 282

NBAR, 306

ořezání provozu, 282

PHB, definice chování, 301

RED, prevence zahlcení, 302

rozložení provozu, 282

RSVP, protokol, 295

SLA, Service Level Agreement, 281

TCP

 globální synchronizace, 302

 vyhlavování, 302

tekoucí vědro, 287

Token Bucket, *viz* zásobník žetonů

traffic policing, *viz* ořezání provozu

traffic shaping, *viz* rozložení provozu

WRED, váhový RED, 304

zásobník žetonů, 288, 295

Kapitola 9

Klasifikace paketů a filtrování dat

Jedním z hlavních úkolů síťových prvků je rozdělení datového toku do tříd podle různých kritérií. To se nazývá klasifikace (třídění). Klasifikace paketů slouží nejen pro filtrování provozu ve firewallech, ale používá se také pro výběr výstupního rozhraní při směrování podle IP adres.

Filtrování provozu rozděluje pakety vstupního toku na pakety, které vyhovují zadaným podmínkám a pro něž je povolen průchod síťovým rozhraním, a na pakety, které nevyhovují podmínkám a které se zahodí. Pro popis filtrovací funkce se používají filtrovací pravidla, kterým se také říká ACL (Access Control Lists). Vyhledání IP adres ve směrovačích (IP address lookup) můžeme považovat za speciální případ klasifikace podle jednoho kritéria. Výsledkem úspěšného vyhledání adresy je název výstupního rozhraní, kam se zpracovávaný datagram přepošle.

V této kapitole si popíšeme základní algoritmy pro klasifikaci dat a ukážeme si, jak se využívají pro filtrování paketů na směrovačích a na firewallech. Zaměříme se na filtrování dat pomocí informací z hlaviček paketů. Existují také postupy pro filtrování paketů podle obsahu paketů (DPI, Deep Packet Inspection), tomu se však v této kapitole věnovat nebudeme.

9.1 Úvod

Klasifikace paketů je jednou ze základních činností síťových zařízení jako jsou například směrovače či firewally. Pro účely směrování se převážně používá jednoduchá klasifikace podle cílové IP adresy. Směrovač přijme paket, podívá se na cílovou IP adresu a ve směrovací tabulce vyhledá nejvhodnější záznam s adresou sítě, na kterou by se měla adresa směrovat. Směrovač pak přepošle paket na příslušné výstupní síťové rozhraní.

Firewally používají pro klasifikaci a filtrování provozu nejen cílovou IP adresu, ale i další hodnoty z hlaviček IP, TCP a UDP. Patří tam kromě cílové IP adresy také zdrojová IP adresa, typ protolu (např. ICMP, IP, TCP, UDP), třída kvality služeb (ToS/DS), zdrojové a cílové porty TCP či UDP, příznaky u TCP (SYN, FIN, ACK) a další. Klasifikace paketů se používá k filtrování paketů, rezervaci zdrojů, zajištění QoS, či směrování unicastu i multicastu.

Hlavním problémem, který přináší klasifikace, je požadavek na výkonnost procesu klasifikace. Na jedné straně chceme, aby klasifikace probíhala podle co největší množiny kritérií. Na druhé straně požadujeme co nejrychlejší průchod paketu zařízením (propustnost). Při klasifikaci jsme omezení velikostí paměti zařízení, do níž se ukládají pakety během zpracování, omezenou pamětí pro uložení klasifikačních pravidel, rychlostí procesoru a kapacitou

vstupních a výstupních linek.

Hraniční směrovače sítí LAN zpravidla obsahují desítky filtrovacích pravidel. U páteřní směrovačů to mohou být až sta tisíce směrovacích pravidel. Klasifikace proto využívá co nejefektivnější datové struktury pro uložení pravidel, rychlé vyhledávání a porovnávání.

První způsob uložení pravidel, který nás asi napadne, je použití neseřazeného lineárního seznamu, který obsahuje jednotlivá porovnávací pravidla. Když přijde paket, procházíme postupně seznam a porovnáváme hodnoty v hlavičce paketu s pravidly v seznamu tak dlouho, dokud nedojde ke shodě všech parametrů pravidla. Pokud chci najít pouze první vhodné pravidlo, ukončím prohledávání. U některých typů vyhledávání (např. vyhledávání podle nejdelšího prefixu u směrovačů) hledáme také pravidlo, které je nejlepší (nejdelší) shodou porovnávaných bitů. Nestačí nám například, že adresa 147.229.8.12 najde shodu u pravidla 147.229.0.0/16, neboť lepší shoda (delší shodný prefix) je s pravidlem 147.229.8.0/24. Obě pravidla však znamenají shodu prefixu s IP adresou v hlavičce datagramu.

Paměťová složitost lineárního seznamu pravidel závisí pouze na počtu pravidel. Má tedy lineární složitost $\mathcal{O}(N)$. Časová složitost roste nejen s počtem pravidel, ale i s počtem polí v pravidlech, které používáme k porovnání: $\mathcal{O}(N * K)$, kde N je počet pravidel a K počet porovnávaných položek v pravidle (tzv. dimenze). Pokud se počet pravidel pohybuje v řádu stovek, narůstají významně nejen paměťové nároky na uložení pravidel, ale také doba zpracování. Proto se pro ukládání filtrovacích pravidel a vyhledávání používají efektivnější datové struktury než jsou lineární seznamy.

Jestliže máme dostatek paměti, jedním z nejrychlejších způsobů vyhledávání je metoda rekursivní klasifikace toků (RFC, Recursive Flow Classification). Pro vyhledávání do pěti tisíc pravidel je vhodné použít metodu bitového vektoru (Lucent Bit Vector). Pro více pravidel se používají rozhodovací stromy (Decision Tree) či hierarchické rozhodovací stromy HiCuts či HyperCuts, které využívají pro zrychlení heuristické přístupy. Záleží také na tom, zda pro vyhledávání a klasifikaci používáme pouze jedno políčko z hlavičky paketu nebo více. Podle toho rozlišujeme metody na jednodimenzionální pro vyhledávání pouze jednoho parametru (například stromy trie) nebo vícedimenzionální metody pro vyhledávání IP adres, čísel portů apod. Nejběžnější klasifikační metody si představíme v následujících podkapitolách.

9.2 Klasifikace paketů

Klasifikace paketů je algoritmičtý problém, kdy se snažíme roztrždit (klasifikovat) zprávy (pakety) podle zadaných klasifikačních pravidel (rules)¹. Cílem klasifikace je co nejrychleji vyhledat ke každému příchozímu paketu pravidlo, jehož položky se shodují s hodnotami v hlavičce paketu. Například při použití pravidla `drop dstIP=147.229.10.12 dstPort=22` vyhledáváme v příchozích paketech hodnoty cílové IP adresy a cílového portu. Pokud se hodnoty v paketu shodují se všemi položkami klasifikačního pravidla, provede se definovaná akce `drop`, tj. paket se zahodí.

Klasifikaci paketů můžeme rozdělit do dvou fází – předzpracování (pre-processing) a vlastní klasifikaci:

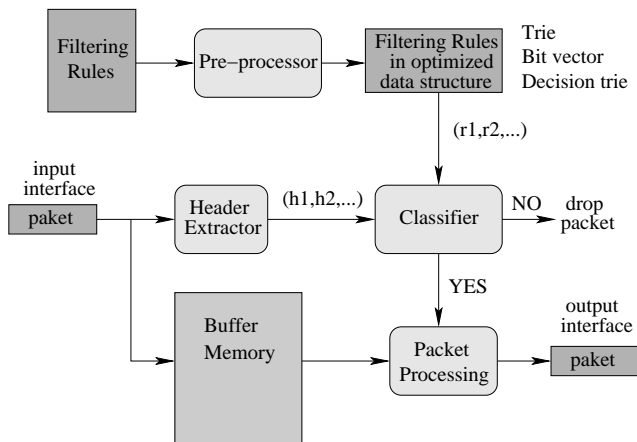
- Cílem *předzpracování* je příprava a uspořádání databáze filtrovacích do datových struktur optimalizovaných pro vyhledávání. Tato fáze se používá v okamžiku přidávání, mo-

¹V odborné literatuře se používají také starší termíny filter (filtrovací pravidlo) místo klasifikačního pravidla a databáze filtrů místo seznamu klasifikačních pravidel.

difikace či rušení pravidel. Protože tyto operace se provádějí zřídka, nejsou tak kritické. Fáze předzpracování probíhá na centrálním procesoru zařízení.

- V *klasifikační fázi* dochází ke zpracování příchozího provozu a vyhledání dat z hlavičky paketu potřebných ke klasifikaci. Poté procházíme datovou strukturou pravidel, která byla vytvořena ve fázi předzpracování. Hodnoty z hlavičky paketu se porovnávají s hodnotami pravidel a vyhledává se nejlepšího klasifikační pravidlo. Klasifikační fáze většinou probíhá v hardwaru síťového zařízení či na specializovaných akcelerovaných kartách.

Příklad klasifikátoru je na obrázku 9.1. Vidíme, že paket je po vstupu do zařízení uložen v paměti (buffer memory). Zároveň jsou z hlavičky paketu získány informace pro klasifikaci (header extractor). Klasifikátor pak vyhledá v databázi filtrovacích pravidel pravidlo, které se aplikuje na paket. Následuje akce spojená s pravidlem: zahození (odfiltrování) paketu nebo předání k dalšímu zpracování (přepínání na výstupní rozhraní, překlad NAT, doručení do vyšší vrstvy apod.).



Obrázek 9.1: Průběh klasifikace paketů

Jednotlivá pole hlavičky paketu, který klasifikujeme, můžeme pokládat za n -tici vybraných polí hlavičky IP paketu H (header): H_1, H_2, \dots, H_K , kde H_i obsahuje sekvenci bitů daného pole. Obvykle se prohledává hlavička datagramu IP na 3. vrstvě či hlavička paketu TCP/UDP na 4. vrstvě modelu TCP/IP. Lze použít i položky z jiných vrstev, například MAC adresu z linkové vrstvy či URL z aplikační vrstvy.

Většinou se pro klasifikaci používá cílová a zdrojová IP adresa (32 bitů), typ protokolu (8 bitů), cílový a zdrojový port (16 bitů) či příznak paketu TCP (8 bitů). Ze zkoumaných dat paketu vytvoříme datovou strukturu H . Příkladem je n -tice $H_i = (77.75.76.3, 147.229.12.8, TCP, 11250, 80, ACK)$, která popisuje paket poslaný z počítače s IP adresou 77.75.76.3 a portu 11 250 na IP adresu 147.229.12.8 a port 80. Jedná se o potvrzení ACK protokolu TCP.

Klasifikátor využívá ke klasifikaci konečnou množinu pravidel (rules) $\mathcal{R} = \{R_1, R_2, \dots, R_N\}$. Pravidla mohou být například u firewallů uspořádána do posloupnosti, neboť firewall vyhodnocuje pravidla podle pořadí. Tím se liší například od směrovačů, kde pořadí pravidel není důležité a vyhledává se nejdelší shoda prefixu IP adresy.

Každé pravidlo R_i je tvořeno K -ticí hodnot, které odpovídají polím z hlavičky paketu. Například pravidlo $R_1 = (147.229.*.*.*., 25, *, \text{permit})$, kde 147.229.*.* je cílová IP adresa a 25 je cílový port, povoluje přístup do sítě 147.229.0.0/16 na port 25 (elektronická pošta).

Pro klasifikaci se obvykle nepoužijí všechna možná pole z hlavičky H paketu. Vybírají se pouze ta pole k , která jsou obsažena alespoň v nějakém pravidle R_i . Pravidlo R_i může použít ke klasifikaci jinou množinu polí než pravidlo R_j . Například pravidlo R_1 bude provádět klasifikaci podle cílové IP adresy a cílového portu, zatímco pravidlo R_2 se omezí na adresu odesílatele.

Každé pravidlo R_i specifikuje *tok dat (flow)*, do kterého paket patří, například přenos HTTP z adresy A na adresu serveru S. Nad tímto tokem dat se provádí *akce*, která je nedílnou součástí pravidla. Akcí nemusí být jenom zahození (deny) či povolení (permit) paketů, ale může dojít například k přepsání třídy provozu v poli ToS (typ služby), překladu IP adresy v NATu, zapouzdření do jiného protokolu při tunelování provozu a další.

Každé pravidlo R_i obsahuje typ porovnání, které se nad daným pravidlem provádí. Základní typy porovnání používané v klasifikátorech jsou:

Přesné porovnání (exact match). U přesného porovnání se musí shodovat hodnota v poli paketu H_k s příslušnou hodnotou v porovnávacím pravidle $R_i(k)$. Přesné porovnání se používá u typu protokolu (TCP, UDP, ICMP) či u příznaků TCP (hodnoty ACK, FIN či SYN).

Porovnání na shodu prefixu (prefix match). Při porovnání na shodu prefixu musí být hodnota pravidla $R_i(k)$ prefixem hodnoty paketu H_k . Prefixové porovnání se používá zejména u směrování, kdy směrovací tabulka obsahuje v položce $R_i(k)$ adresu sítě (např. 147.229.0.0/16). Klasifikovaný paket obsahuje vždy adresu konkrétního počítače (např. 147.229.176.18). V tomto příkladu je adresa sítě $R_i(k) = 147.229.0.0/16$ bitovým prefixem hodnoty $H_k = 147.229.176.18$ z hlavičky paketu.

U směrování může dojít k prefixové shodě ve více pravidlech. Pokud se shoduje více pravidel, vybere se to pravidlo, kde došlo k tzv. *nejdelší prefixové shodě (longest prefix match)*. Například u paketu s cílovou IP adresou 147.229.176.18 dojde k prefixové shodě s pravidly, která obsahují cílové adresy sítí 147.229.0.0/16, 147.229.176.0/24 a 147.229.176.16/28. Nejdelší prefixová shoda, v tomto případě 28 bitů, je pouze u třetí sítě, tedy třetí pravidlo se použije bez ohledu na pořadí v seznamu pravidel.

Intervalové porovnání (range match). U intervalového porovnání musí být hodnota H_k v intervalu, který je zadán v porovnávacím pravidle $R_i(k)$. Tento typ porovnání je vhodný například při omezení povolených portů. Například paket s cílovým portem $H_k = 80$ vyhovuje porovnávacímu pravidlu, které má definuje interval $R_i(k) = \langle 0; 1023 \rangle$. Intervalové porovnávání se dá převést na prefixové porovnávání, neboť libovolný interval lze rozdělit na sjednocení prefixů, viz [8]. Například interval $\langle 0; 1023 \rangle$ lze při šestnácti bitové reprezentaci vyjádřit jako prefix 000000*.

Porovnání regulárního výrazu (regular expression match). Porovnávání pomocí regulárních výrazů se používá při analýze obsahu paketů, například u vyhledávání řetězců

v aplikačních protokolech. Analýze obsahu paketů se říká DPI (Deep Packet Inspection). DPI je u vysokorychlostních sítí velice náročná na zpracování. Proto se implementuje spíše ve specializovaných zařízeních, které využívají hardwarovou akceleraci implementovanou pomocí technologií ASIC či FPGA. U běžných aktivních prvků se vyhledávání regulárních výrazů v paketech implementuje v softwarové části a může způsobit výrazné zpomalení činnosti zařízení.

Množina porovnávacích pravidel \mathcal{R} je umístěna v databázi pravidel daného směrovače či firewallu. Každé pravidlo má své identifikační číslo, které obvykle označuje i pořadí pravidla v seznamu. Pravidla jsou zpravidla vyhodnocována s prioritou odpovídající jejich pořadí². Nejprve se provede porovnání podle prvního pravidla. Pokud nedojde ke shodě se všemi položkami $R_1(k)$, pokračuje se vyhodnocováním podle druhého pravidla R_2 až posledního pravidla R_n . Poslední pravidlo bývá obvykle implicitní pravidlo, které zahazuje všechny pakety, u kterých nedošlo ke shodě v předchozích pravidlech. Jedná se o takzvaný *inkluzivní firewall*, který zahodí vše, co není explicitně povoleno.³). Příklad klasifikačních pravidel pro příchozí komunikaci je v tabulce 9.1.

	Dst IP	Src IP	DstP	SrcP	Proto	Action	Remark
1	147.229.0.0/16	*	25	*	*	permit	povol příchozí emaily
2	147.229.0.0/16	*	53	*	UDP	permit	povol dotazy DNS
3	147.229.0.0/16	*	22	*	TCP	permit	povol přístup přes SSH
4	147.229.5.1	153.13.2.5	123	123	UDP	permit	povol NTP info
5	*	117.16.0.0/16	*	*	IP	decrypt	VPN ze sítě 117.16.0.0
6	*	*	*	*	*	deny	blokuje vše ostatní

Tabulka 9.1: Příklad vstupních filtrovacích pravidel na hraničním uzlu

Tato tabulka obsahuje šest pravidel $R_1 - R_6$, které provádí porovnání až pěti polí z hlavičky paketu. Počet polí, která se používají při porovnání, určují tzv. *dimenzi*. Metody, které umožňují porovnání pouze jedné hodnoty (například zdrojové IP adresy), se nazývají jednodimenzionální. Metody, které používají klasifikaci na základě dvou a více hodnoty, se nazývají dvou či vícedimenzionální.

Hodnota * (*wildcard*) znamená, že dané pole se nepoužije pro porovnání, tj. paket může v daném poli obsahovat libovolnou hodnotu. Výše uvedený příklad ukazuje inkluzivní firewall, který přijímá pouze to, co je explicitně povoleno. Klasifikátor dle 9.1 přijímá emaily (R_1), dotazy na DNS (R_2) či spojení ssh (R_3), které směřují do lokální sítě 147.229.0.0/16. Dále je povolena komunikace ze serveru NTP server 153.13.2.5 na konkrétní stroj v lokální síti (R_4). Veškerá komunikace IP, která přichází ze sítě 117.16.0.0/16, je považována za zapouzdřený provoz VPN a provádí se nad ní vypouzdření (R_5). Všechny ostatní provoz přicházející na vstupní zařízení směrovače je zahozen (R_6).

9.3 Způsoby klasifikace paketů

Jak jsme si již řekli, klasifikace paketů tvoří základ všech firewallů, směrovačů a dalších síťových zařízení. Protože je nutné provádět klasifikaci co nejrychleji, hledáme optimální al-

²Neplatí to například u směrovacích tabulek, kde dochází k vyhledávání nejdelsí prefixové shody.

³Opakem je exkluzivní firewall, který implicitně povolí všechny provoz, a zahodí pouze ty pakety, na které se aplikuje některé pravidlo.

goritmy, které na jedné straně budou podporovat prohledávání ve více dimenzích, na druhé straně budou rychlé, efektivní a jejich časová a paměťová složitost umožní implementaci v prostředí dostupného hardwaru.

V následující kapitole se budeme zabývat klasifikací paketů v jedné dimenzi, což se používá zejména u směrovačů. Pak popíšeme metody klasifikací ve dvou a více dimenzích, které se používají převážně ve firewallech. Ukážeme si různé typy algoritmů a podíváme na jejich složitost z pohledu času a prostoru.

Pro hodnocení klasifikačních algoritmů použijeme následující metriky:

Rychlost. Statistiky [11] ukazují, že 50% příchozích paketů tvoří potvrzení TCP ACK o velikosti 40 bytů. V důsledku toho musí být směrovač schopen zpracovávat neustálý proud krátkých paketů. Důležité je, aby doba procesu klasifikace odpovídala rychlosti příchodu paketů na lince (tzv. wire-speed). To je kritické zejména pro vysokorychlostní sítě.

Například u sítě SONET OC-768 s rychlostí 40 Gb/s může přijít za jednu sekundu až 125 milionů paketů o velikosti 40 B ($40 \cdot 10^9 / (40 \cdot 8)$). Abychom mohli zpracovat každý z těchto paketů, musí být rychlost zpracování do osmi nanosekund. Pro linky 10 Gb/s je to 32 nanosekund a pro 1 Gb/s linky 320 nanosekund. Přitom přístupové doby paměti DRAM se pohybují okolo 40 ns, u SRAM kolem 1 ns (údaje z roku 2012). Proto je rychlost zpracování přímo závislá na typu použité paměti a počtu přístupů do paměti potřebných pro zpracování daného paketu.

Paměťová náročnost. Velikost paměti, kterou potřebujeme pro uložení klasifikačních pravidel, ovlivňuje obvykle cenu zařízení. Paměťová náročnost má také vliv na typ paměti, kterou je potřebujeme. Pokud klasifikátor nevyžaduje velkou paměť, je používají se rychlé statické paměti SRAM s přístupovou dobou kolem jedné nanosekundy. Větší paměti DRAM mají přístupové doby kolem 40–60 nanosekund.

Velikost paměti můžeme také ovlivnit použitím rychlé vyrovnávací paměti *cache*. Oproti klasické paměti jsou přístupové doby těchto pamětí rychlejší, jejich kapacita je však omezená. Efektivita použití paměti *cache* závisí zejména na *časovém rozmístění paketů* toku v provozu. Jestliže je časový rozptýl paketů v toku malý, pak je velká pravděpodobnost, že následující paket využije výsledek klasifikace předchozího paketu, který je uložený v paměti *cache*.

Rychlost aktualizace. Algoritmy a datové struktury pro zpracování prefixů musí počítat s tím, že se například směrovací tabulky mění. Během sekundy se může změnit jeden nebo také stovky prefixů. V porovnání s přístupy do paměti se změny směrovacích tabulek pohybují v milisekundách. Tato čísla jsou řádově jiná, než jsou doby přístupů do paměti, které se pohybují v nanosekundách. Nicméně je užitečné, pokud algoritmus podporuje částečné aktualizace klasifikační tabulky.

Způsob aktualizace klasifikačních pravidel se liší podle typu datových struktur, do kterých se pravidla ukládají. U některých datových typů se pouze přidávají nová pravidla a rozšíří se stávající architektura. Jiné datové struktury vyžadují znovuvytvoření celé datové struktury, což je samozřejmě časově i prostorově náročné. Aktualizace filtrovacích pravidel se zřídka vyskytuje u páteřních směrovačů, které jsou spíše stabilní. Naopak hraniční směrovače často podporují dynamické stavové filtrování či dynamickou reakci na útok, což vyžaduje rychlou aktualizaci databáze pravidel.

Dimenze. Podle počtu hlaviček, které metoda umí zpracovat, rozdělujeme metody na jednodimenzionální, dvoudimenzionální a vícedimenzionální. Ideální algoritmus není omezen počtem dimenzí a je schopen zpracovat libovolný počet klasifikačních polí.

Rozšiřitelnost. Algoritmus pro vyhledávání by měl být rozšiřitelný, tzn. použitelný u směrovačů s tisíci i sta tisíci záznamy. Je jasné, že algoritmus pro uložení klasifikačních pravidel v domácí směrovači se bude lišit od směrovače pro firemní síť. Páteří směrovače se stovkami tisíc záznamů používají jiné algoritmy než menší směrovače. Důležité je, aby přidáním záznamů do směrovací tabulky nedošlo vlivem algoritmu k výraznému omezení výkonnosti zařízení. Proto také specifikace síťových prvků obsahují informaci o maximálním počtu klasifikačních pravidel, které lze zadat pro efektivní zpracování.

Složitost implementace. Složitost implementace algoritmu má vliv na to, zda bude možné algoritmus implementovat přímo na síťové kartě v hardwaru, nebo se zpracovává na procesoru síťového zařízení. Pokud chceme dosáhnout velkou propustnost, volíme metody, kterou jsou implementovatelné na dostupném hardware (například platformy ASIC či FPGA).

Algoritmy pro klasifikaci paketů můžeme rozdělit podle různých kritérií. Deep Medhi a K. Ramasamy [7] rozdělují algoritmy na ty, které používají jedno, dvě či vícedimenzionální vyhledávání. Uvádí příklady struktur *trie*, bitových vektorů, rozhodovacích stromů a také hardwarové řešení pomocí asociativní paměti TCAM. George Varghese používá ve své knize [13] podobné dělení. D. E. Taylor ve studii o technikách klasifikace paketů [10] rozděljuje metody na čtyři skupiny: úplné prohledávání, dekompozice, rozhodovací stromy a vyhledávání pomocí *n*-tic. V této práci se budeme držet dělení algoritmů podle [7] s rozšířením o další zajímavé algoritmy.

Při vyhledávání v jedné dimenzi při rychlostech do 10 Gb/s se používá v hardwaru i softwaru algoritmus vícebitových struktur zvaných *trie* (kapitola 9.4.3). U vyšších rychlostí (do 40 Gb/s) jsou efektivnější algoritmy používající stromové bitové mapy (kapitola ??). Zajímavé je i použití binárního vyhledávání, které popíšeme v kapitole ??.

9.4 Klasifikace v jedné dimenzi

Klasifikace paketů v jedné dimenzi se používá zejména pro směrování podle cílové IP adresy. V literatuře se tato problematika nazývá vyhledávání IP adres (*IP address lookup*) nebo také prefixové vyhledávání (*prefix lookups*).

9.4.1 Prefixové vyhledávání IP adres

Síťové prefixy (IP adresy sítě) jsou tvořeny bity o délce maximálně 32 bitů (u IPv4). I když zapisujeme IP adresu jako čtyři dekadická čísla oddělená tečkou, v počítačích a síťových zařízeních je reprezentována v binární podobě. Např. síťový prefix VUT 147.229.*.* se binárně zapisuje jako 1001001111100101*. S nástupem technik VLSM a CIDR se začaly používat prefixy proměnlivé délky, kde délku prefixu je určena síťovou maskou. Masku můžeme zapsat jako speciální 32-bitovou adresu, kde jedničky označují síťovou část IP adresy a nuly adresu konkrétního počítače v dané podsíti, např. 255.255.0.0. Masku se také zapisuje jako počet bitů síťové části IP adresy uvedený za lomítkem za IP adresou, např. 147.229.0.0/16.

Vyhledávání podle prefixu se provádí především na směrovačích. Poté, co IP datagram přijde na vstupní rozhraní směrovače, procesor zkontroluje cílovou IP adresu a rozhodne, kam bude datagram přeposlán. Při rozhodování se směrovač dívá do tabulky FIB (Forward Information Base, též Forwarding Table)⁴. Tabulka FIB obsahuje množinu prefixů a odpovídajících výstupních rozhraní (next-hop). Záznamy se do tabulky vkládají buď manuálně (statické záznamy) nebo dynamicky pomocí směrovacích protokolů.

Při porovnávání se vyhledá záznam FIB, který obsahuje nejdelší shodný prefix IP adresy zpracovávaného datagramu a označení výstupní linky, na kterou se datagram přepoše. Směrovače tedy provádí dvě základní operace: (i) vyhledání adresy dalšího uzlu a (ii) přeposlání (forwarding) na příslušné výstupní rozhraní. Přestože zpracování paketů a přeposlání na výstupní rozhraní vyžaduje určitou dobu, kritickou operací směrovače je vyhledání správného záznamu. Proto se v této části budeme zabývat algoritmy pro efektivní vyhledávání IP adres ve směrovacích tabulkách.

Příklad tabulky FIB je na obr. 9.2. Každý záznam tabulky obsahuje prefix sítě (ve speciální případě i konkrétní IP adresu zařízení) a jméno výstupního rozhraní (next-hop). Například

Číslo záznamu	Prefix	Next-Hop
1	98.1.1.0/24	eth3
2	171.1.0.0/16	so6
3	171.1.1.0/24	fe4

Tabulka 9.2: Příklad tabulky FIB

první záznam vyhledává pakety, kde hodnota 98.1.1.0 je prefixem cílové adresy v hlavičce zpracovávaného IP datagramu. Pokud je shoda nalezena, datagram se přepoše na výstupní rozhraní **eth3**. Pokud přijde do směrovače IP datagram s cílovou adresou 171.1.1.2, dojde k prefixové shodě ve druhém a třetím záznamu. Protože prefix 171.1.1.0/24 je delší, vybere se záznam číslo 3 a datagram se přepoše na rozhraní **fe4**.

Vyhledání nejdelšího prefixu přináší dvě úskalí. Jednak cílová IP adresa datagramu neobsahuje informace o masce, takže nevíme, kolik bitů IP adresy tvoří adresa sítě (network ID) a kolik adresa počítače (host ID). Do směrovacích tabulek se vkládají převážně adresy sítě. Druhým problémem je, že prefixy sítí ve směrovací tabulce mohou mít různou délku podle stupně agregace. Agregace znamená, že místo několika záznamů se stejným prefixem a stejným výstupním rozhraním, se umístí do tabulky pouze jeden záznam obsahující společný prefix. Například místo čtyř záznamů 10.10.0.0/24, 10.10.1.0/24, 10.10.2.0/24 a 10.10.3.0/24 odkazující se na výstupní rozhraní **em0**, vložíme do směrovací tabulky pouze jeden agregovaný záznam 10.10.0.0/22. Vyhledávání nejdelšího prefixu tedy znamená (1) určení délky největšího prefixu a (2) nalezení tohoto prefixu. Příkladem může být tabulka 9.3.

V této tabulce vidíme, že některé síťové prefixy jsou prefixem jiného záznamu. Například prefix P_4 je prefixem adres P_5 až P_9 . Také je vidět, že rozsahy prefixů nejsou vždy spojitě. Dekadický zápis v tabulce není přesný, protože obsahuje 32-bitovou adresu. Přesný zápis by byl například 0/1 místo 0.0.0.0/1 u pravidla P_1 .

Při vyhledávání v tabulce prefixů FIB hraje roli nejen počet záznamů v tabulce N (tj. počet klasifikačních pravidel), ale také maximální délka prefixu W . Tato délka určuje například výšku stromu při reprezentaci pravidel ve stromové struktuře *trie* (čti jako tree).

⁴FIB je obecný název databáze obsahující směrovací informace o sousedních sítích. Pro vlastní směrování se používá část informací FIB, které je uložena ve směrovací tabulce (routing table).

Číslo pravidla	Prefix	Dekadický zápis
P_1	0*	0.0.0.0/1
P_2	00001*	8.0.0.0/5
P_3	001*	32.0.0.0/3
P_4	1*	128.0.0.0/1
P_5	1000*	128.0.0.0/4
P_6	1001*	144.0.0.0/4
P_7	1010*	160.0.0.0/4
P_8	1011*	176.0.0.0/4
P_9	111*	224.0.0.0/3

Tabulka 9.3: Příklad tabulky síťových prefixů.

Při návrhu algoritmů lze využít následující heuristiky, které vycházejí ze zkušeností při používání směrovačů v praxi:

- V páteřních směrovačích většina prefixů je délky 24 či menší (platí pro síť IPv4).
- Ve směrovacích tabulkách není mnoho záznamů, které by byly prefixem jiných záznamů. Podle zkušeností je maximálně sedm prefixů, které jsou součástí jiného prefixu.

9.4.2 Binární stromy *trie*

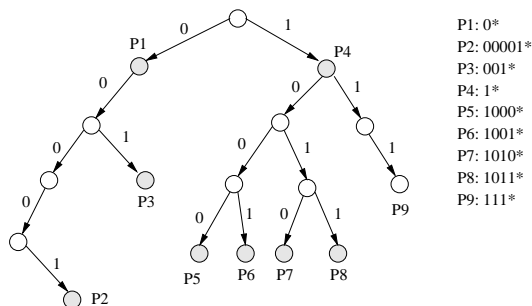
Jedna z nejvíce používaných datových struktur pro uložení prefixů je binární stromová struktura *trie* (výslovnost jako *tree*). Termín *trie* pochází ze slova *retrieval* (vyhledání). *Trie* je binární stromová struktura, do které ukládáme prefixy IP adres ze směrovací tabulky. Uzly tvoří části IP adres, ohodnocené hrany odpovídají jednotlivým bitům adresy. Výška stromu odpovídá nejdelšímu prefixu. Ve stromu lze uložit i celou 32-bitovou IP adresu, kterou můžeme pokládat za speciální prefix o délce 32 bitů.

Prefix připojený k uzlu u je konkatencí všech bitových prefixů na cestě od kořene k uzlu u . Jedná se tedy o cestu od uzlu u ke kořeni. Při čtení postupujeme směrem od nejdůležitějšího bitu IP adresy (tzv. MSB, the most significant bit) k méně významným bitům. Ne každý uzel obsahuje prefix ze směrovací tabulky. Například uzel s hodnotou 00 není prefixem ze směrovací tabulce. Jedná se o pomocný uzel struktury *trie*.

Uzly, které odpovídají prefixům ze směrovací tabulky, jsou v obrázku 9.2 vybarveny šedě a obsahují číslo pravidla, ve kterém je kterýs obsažen daný prefix. Například uzel P2 obsahuje prefix 00001. Hrany spojující uzly jsou ohodnoceny bity 0 nebo 1. Každý vnitřní uzel stromu má jeden nebo dva potomky, listové uzly neobsahují žádné potomky. Listy odpovídají nejdelším prefixům ve směrovací tabulce.

Struktura *trie* umožňuje jednoduše zapsat tabulku a prefixů a efektivně v ní vyhledávat. Detekci společných prefixů nemusíme při vyhledávání řešit, protože se vybere vždy ten nejdelší, který odpovídá klasifikovanému paketu. Jak vidíme na příkladu, struktura *trie* nevytváří úplný binární strom. Obsahuje pouze větve, které odpovídají pravidlům ve směrovací tabulce. Neobsahuje například uzel 01.

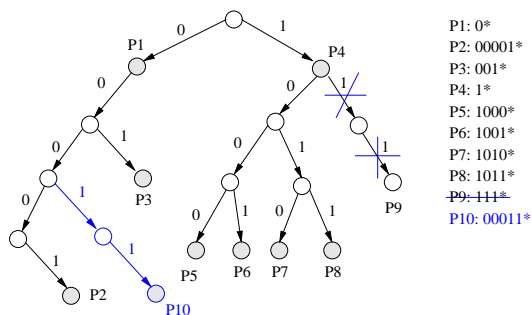
Struktura *trie* se hojně využívá pro jednodimenzionální vyhledávání ve směrovacích tabulkách. Při vyhledávání procházíme strom od kořene po ohodnocených hranách tak dlouho, dokud existuje větev s ohodnocením odpovídající IP adrese porovnávaného paketu. Při průchodu si pamatujeme poslední (nejdelší) prefix P (šedý uzel), kterým jsme prošli. Pokud



Obrázek 9.2: Binární struktura *trie* pro vyhledávání prefixů

dojdeme do uzlu, v němž není další větev odpovídající porovnávanému bitu, vrátíme se k poslednímu uloženému prefixu. Tento prefix je nejdelším shodný prefixem pro danou IP adresu v paketu. Např. u paketu s IP adresou 0001 se jako nejdelší prefix vybere P1.

Operace vkládání a rušení prefixu nad stromem *trie* jsou jednoduché. Při vkládání nejprve hledáme nejdelší shodný prefix. Jakmile dojdeme do uzlu, kde už nejsou další větve pro porovnávání, vytvoříme nový uzel a připojíme ho ke stromu *trie*. Například při vkládání prefixu 00011 vytvoříme dva nové uzly propojené hranou 1 a připojíme je k uzlu 000, viz obrázek 9.3.



Obrázek 9.3: Vkládání a rušení ve stromu *trie*

Při rušení nejprve vyhledáme prefix, který chceme rušit. Pokud jde o vnitřní uzel, který je součástí pravidla ze směrovací tabulky (tj. je označen šedě), zrušíme jeho označení (šedou barvu). To znamená, že daný uzel nebude tvořit platný prefix, ale stane se pomocným uzlem na cestě k jinému prefixu. Pokud rušíme koncový uzel (list stromu), zrušíme všechny uzly s jedním potomkem, které vedou k tomuto uzlu. Například při rušení prefixu P_9 odstraníme pravou větev v bodě 1.

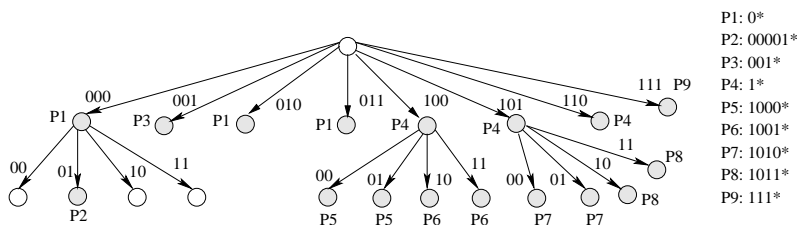
Binární strukturu *trie* můžeme jednoduše implementovat jako záznam, který obsahuje dvě položky: jednu pro bit 0 a jednu pro bit 1. Položka obsahuje buď ukazatel na další uzel nebo hodnotu *null*, pokud nemá uzel potomka pro daný bit.

Časová složitost vyhledávání je $\mathcal{O}(W)$, kde W je maximální délka prefixu v tabulce. Stejnou složitost má i vkládání a rušení uzlů. Paměťová složitost je $\mathcal{O}(N.W)$, kde N je počet prefixů zadaných v pravidlech pro vyhledávání.

9.4.3 Vícebitové stromy *trie*

Nevýhodou binárních stromů *trie* je, že porovnání každého bitu adresy či prefixu je velmi pomalé a někdy i zbytečné. Při porovnání vždy dochází k přístupu do paměti. V nejhrošším případě vyžaduje struktura *tree* pro adresy IPv4 celkem 32 přístupů do paměti. Pokud uvažujeme přístupovou dobu při čtení paměti 10 ns, pak takové porovnání celé IP adresy trvá 320 ns. Při této rychlosti jsme schopni zpracovat 3.125 milionů paketů za sekundu (tj. $\frac{1}{320 \cdot 10^{-9}} = 3,125.000$ paketů za sekundu). Pokud uvažujeme 40 bytové pakety, pak dosáhneme rychlosti maximálně 1 Gb/s (tj., $3,125.000 \times 40 \times 8 = 1.10^9$ b/s). Pro vyšší rychlosti jsou binární (dvoubitové) stromy *trie* nepoužitelné.

Právě proto se začalo uvažovat o stromech *trie*, které umožňují porovnat více bitů v jednom kroku. Pokud jsme schopni například porovnat čtyři bity najednou, zredukuje počet přístupů do paměti z 32 na 8. Toto je hlavní motivace pro konstrukci vícebitových stromů *trie* (multibit trie). Počet bitů, které se kontrolují během jednoho porovnání, se nazývá *krok* (stride). Krok může mít pevnou nebo proměnlivou velikost. Vícebitový strom *trie* obsahuje uzly, které mají 2^k potomků, kde k je krok (tj. počet bitů v jednom porovnání). Pokud všechny uzly jedné úrovně stromu *trie* mají stejnou velikost kroku, jedná se o vícebitový strom *trie* s pevným krokem, v opačném případě jde o *trie* s proměnlivým krokem. Příklad vícebitového stromu *trie* s pevným krokem je na obrázku 9.4.



Obrázek 9.4: Vícebitový strom *trie* s pevnou délkou kroku

Nyní si popíšeme konstrukci takového stromu. Protože vícebitové stromy *trie* používají pro vyhledání a porovnání v jednom kroku vícebitové hodnoty, nemůžeme mít na vstupu prefixy libovolné délky. Vstupní prefixy (prefixy ve směrovací tabulce) se musí transformovat na ekvivalentní prefixy větší délky tak, aby odpovídali délce prefixů vícebitového stromu. Například na obrázku 9.4 je strom, který v prvním kroku používá tři bity pro vyhledání a v dalším kroku dva bity. Proto je potřeba tabulku prefixů rozšířit na prefixy o délce tři nebo pět bitů. Techniku rozšíření si představíme v další části.

Rozšíření prefixů

Rozšíření prefixu (prefix expansion) je jednou z technik, která převádí prefixy určité délky na ekvivalentní prefix větší délky při zachování adresového rozsahu, který prefixy popisují. Říkáme, že prefix je rozšířen, pokud je převeden více delších a konkrétnějších prefixů, které pokrývají stejný adresový rozsah. Například prefix $0*$ lze rozšířit na dvouprvkovou množinu dvoubitových prefixů $00*, 01*$ nebo čtyřprvkovou množinu (2^k) tříbitových prefixů $000*, 001*, 010*, 011*$. Cílem rozšíření prefixů je převést tabulku prefixů, které mají proměnlivou délku, na tabulku prefixů, která obsahuje pouze několik typů prefixů pevné délky. Například tabulka 9.4 obsahuje prefixy o původní délce jedna až pět bitů, které nyní převedeme pouze na dva prefixů o délce tři a pět bitů.

Pravidlo	Původní prefix	Rozšířené prefixy
P_1	$0*$	$000*, 010*, 011*$
P_2	$00001*$	$00001*$
P_3	$001*$	$001*$
P_4	$1*$	$100*, 101*, 110*$
P_5	$1000*$	$10000*, 10001*$
P_6	$1001*$	$10010*, 10011*$
P_7	$1010*$	$10100*, 10101*$
P_8	$1011*$	$10110*, 10111*$
P_9	$111*$	$111*$

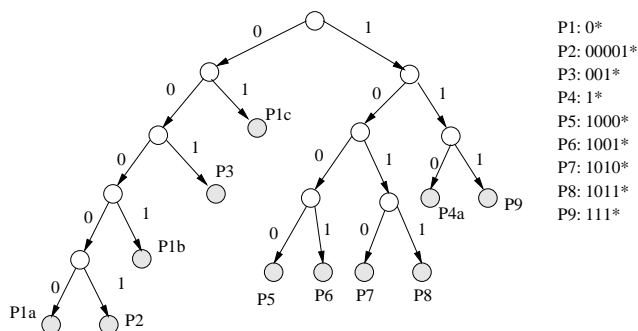
Tabulka 9.4: Tabulka rozšířených prefixů

Při konstrukci ekvivalentních rozšířených prefixů rozšiřujeme nejkratší prefixy na nejbližší možnou délku. Musíme samozřejmě přidat všechny možnosti, které odpovídají danému prefixu. Například pro prefix $0*$, provedeme rozšíření na tři bity na prvky $000*, 001*, 010*, 011*$. Vidíme, že prefix $001*$ už existuje a je použit v pravidle P_3 . Protože vyhledáváme nejdelší shodný prefix, pravidlo P_3 má prioritu před obecnějším pravidlem P_1 . Říkáme, že existující prefix už pokrývá rozšířený prefix. Proto přidáme do množiny rozšířených prefixů pravidla P_1 pouze tři řetězce $000*, 010*$ a $011*$.

Druhým problémem, který musíme řešit, je překrývání prefixů. Pokud se dva prefixy překrývají, znamená to, že jeden interval adres obsahuje jiný interval adres. Při výběru, které pravidlo zvolit, rozhoduje délka prefixu (nejdelší shodný prefix). Pokud strom *trie* obsahuje pouze disjunktní prefixy, jsou prefixy uloženy pouze v listových uzlech, nikoliv ve vnitřních uzlech struktury *trie*.

Abychom vyřešili překrytí prefixů, je potřeba převést množinu prefixů na množinu disjunktních prefixů. Toho docílíme při konstrukci stromu *trie*. Nejprve vytvoříme strom pro zadanou sadu prefixů. Pak strom procházíme a k uzlům, které mají pouze jednoho potomka, přidáme druhý uzel, který bude listem. Tyto nové listy reprezentují nové prefixy, které získají z nejbližšího předchozího prefixu. Jakmile přidáme tyto nové listy, můžeme zrušit označení vnitřních uzlů jako prefixů.

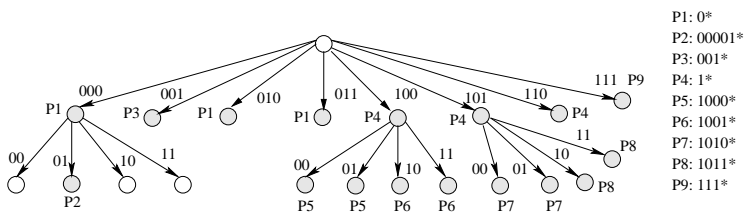
Na obrázku 9.5 můžeme vidět transformovaný strom *trie* bez překrývání prefixů, který vznikl z původní *trie* na obrázku 9.2. Všimněte si nově přidaných uzlů P_{1a} , P_{1b} a P_{1c} , které zdědily informaci z uzlu P_1 . Podobně uzel P_{4a} vznikl z uzlu P_4 . Pokud prohledávání adresy končí prefixem P_1 , pak dojde ke shodě na uzlech P_{1a} , P_{1b} , P_{1c} , které se odkazují ve směrovací tabulce na stejný výstupní rozhraní.

Obrázek 9.5: Binární strom *trie* bez překrývání prefixů

Vícebitový strom *trie* s pevnou délkou kroku

Vícebitový strom *trie* s pevnou délkou kroku obsahuje uzly a hrany, kde ohodnocení hran na dané úrovni (výšce) stromu má daný počet bitů (stejný krok). Abychom toho dosáhli pro směrovací tabulku, která obecně obsahuje prefixy libovolné délky, musíme nejprve rozšířit prefixy na délku odpovídající dané délce kroku, pro kterou je vícebitový strom *trie* vytvářen.

Na obrázku 9.6 je ukázka vícebitového stromu *trie* s pevnou délkou kroku, která je v první úrovni stromu tři bity a v druhé úrovni stromu dva bity. Abychom tohoto dosáhli, musíme rozšířit prefixy na délku tři a pěti bitů, viz tab. 9.4. Takto se nám rozšíří například jednobitový prefix v pravidle P_1 na prefixy 000*, 010*, 011* a 011* o délce tři bity. Protože prefix 001* je už použit v pravidle P_3 , použije se tento existující prefix. Podobně rozšíříme i prefixy P_5, P_6, P_8 a P_8 .

Obrázek 9.6: Vícebitový strom *trie* s pevnou délkou kroku

Vyhledávání v tomto stromu probíhá tak, že rozdělíme hledanou adresu na části odpovídající délce kroku jednotlivých úrovní stromu, v našem příkladu na tři a dva bity. Pak procházíme stromem a pamatujeme si vždy poslední prefix na cestě. Tento prefix označuje nejlepší prefixovou shodu s vyhledávanou adresou.

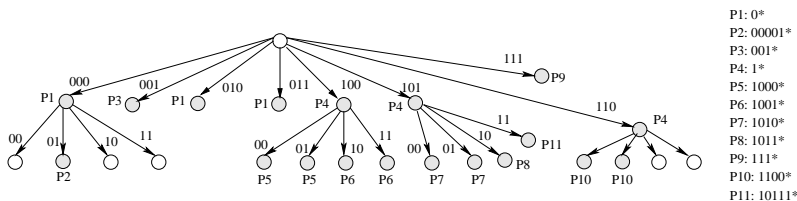
Například při hledání adresy 100111 rozdělíme adresu na tři části 100, 11 a 1. Pomocí prvního řetězce se dostaneme do uzlu P_4 a pak do uzlu P_6 . Protože dále nemůžeme pokračovat,

obsahuje pravidlo P_6 nejlepší prefix pro adresu 100111. Vidíme, že počet porovnání (přístupů do paměti) se zredukoval na dvě, oproti pěti v případě binárního stromu *trie*. Složitost vyhledávání je tedy $\mathcal{O}(W/k)$, kde W je maximální délka prefixu v tabulce a k je počet bitů jednoho kroku.

Aktualizace stromu vyžaduje nalezení nejvhodnějšího uzlu pro rozšíření, tj. nejdelšího prefixu odpovídající vkládanému uzlu. Poté se buď vytvoří nový uzel s vkládaným prefixem, nebo se přepíše rozšířený prefix, neboť nový prefix je delší než původní nerozšířený.

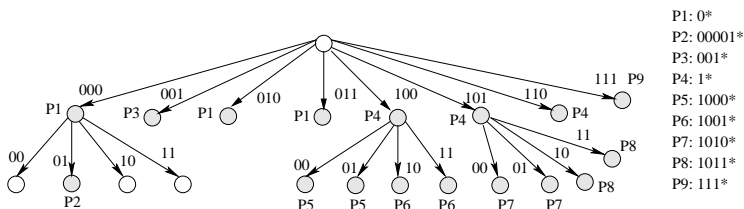
Například při vkládání prefixů $P_{10} = 1100*$ a $P_{11} = 10111*$ do našeho stromu nejprve rozdělíme vkládané prefixy na části odpovídající délce kroků jednotlivých stupňů stromu. Například prefix P_{10} rozdělíme na části 110 a 0. První část vede do uzlu P_4 . Protože zde nejsou žádné další větve, vytvoříme nové větve od velikosti kroku dva. V tomto novém podstromu nám dva uzly s prefixy 00 a 01 pokrývají vkládaný prefix. Zbývající dva prefixy jsou nevyužité.

Při vkládání prefixu P_{11} postupujeme podobně. Po rozdělení na části 101 a 11 postupujeme do uzlu P_4 . Zde vidíme, že už existuje následník s prefixem 11. Protože se jedná o rozšířený uzel P_8 (tuto informaci si musíme v uzlu vždy udržovat), který je původně menší než vkládaný prefix P_{11} , přejmenujeme tento uzel na P_{11} , neboť prefix 10111* již pokrývá prefix 1011*. Abychom toto mohli provést, musí každý uzel obsahovat informaci o původní délce před rozšířením. Příklad je znázorněn na obrázku 9.7.



Obrázek 9.7: Vícebitový strom *trie* po vložení prefixů P_{10} a P_{11}

Rušení prefixů u vícebitových stromů je náročnější, zejména u prefixů, které byly rozšířeny. Musíme totiž nejen zrušit rozšířené prefixy, ale i opravit hodnoty v uzlech na hodnoty, které tam byly nastaveny před rozšířením. Navíc u některých uzlů mohly být původní hodnoty přepsány. Uvažujme například vícebitový strom *trie* na obrázku 9.8. Pokud do tohoto stromu



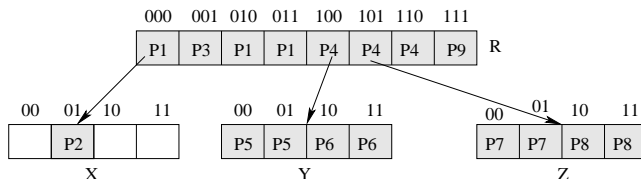
Obrázek 9.8: Vícebitový strom *trie* s pevnou délkou kroku

přidám nově prefixy 100*, 101* a 110*, tak přepíši rozšířený prefix $P4$, který již ve stromě nebude existovat. Pokud pak budu chtít později zrušit takto přidáný prefix 110*, musím se obnovit hodnota v uzlu na rozšířený prefix $P4$. Znamená to, že si musím uchovávat informaci o původních prefixech.

Co je zajímavé na aktualizacích vícebitového stromu *trie* je, že vkládání či rušení se týkají pouze určitého podstromu, tzn. provádějí se lokálně. Krok k podstromu určuje, kolik uzlů bude ovlivněno modifikací, maximálně to bude $2^k - 1$ uzlů. Časová složitost aktualizace zahrnuje jednak dobu vyhledání podstromu $\mathcal{O}(\frac{W}{k})$ a čas modifikace $2^k - 1$ záznamů. Celkově je časová složitost aktualizace vícebitového stromu $\mathcal{O}(\frac{W}{k} + 2^k)$.

Implementace vícebitových stromů *trie*

Vícebitové stromy *trie* s pevným krokem lze implementovat pomocí polí propojených ukazateli. Délka pole bude odpovídat délce kroku na dané úrovni stromu. Velikost pole (tj. počet uzlů na dané úrovni kroku) bude 2^k , kde k je krok dané úrovně stromu. Uložení vícebitového stromu *trie* z příkladu na obrázku 9.8 je ukázáno na obrázku 9.9. Protože krok na první úrovni



Obrázek 9.9: Implementace vícebitového stromu *trie*

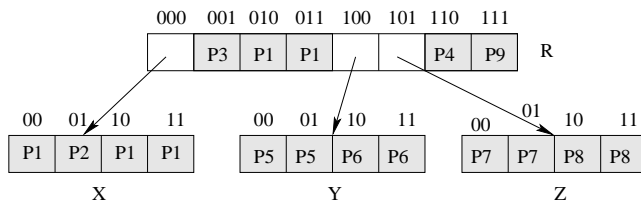
je 3, potřebuje pro uložení první úrovně stromu $2^3 = 8$ prvků pole. Na druhé úrovni stromu jsou to čtyři prvky pole. Prefix se používá jako index do pole, tzn. není uložen v poli. Například prefix 110 slouží jako index na sedmý prvek pole. Tento prvek obsahuje dvě položky. První je *nhop*, který odkazuje na výstupní rozhraní, tzn. našel jsem nejdelší shodný prefix a odpovídající výstup. Pokud obsahuje nenulovou hodnotu v druhé poloze *ptr*, pak že daný prvek odkazuje na podstrom. Některá pole mohou být zaplněna jen částečně (viz první pole v druhé úrovni stromu), kde tři políčka jsou prázdná. Pro efektivní implementace se používají různé kompresní algoritmy. V další části si představíme jeden z nich.

Kompresí vícebitového stromu *trie* metodou Lulea

Při vytváření vícebitových stromů *trie* s pevnou délkou kroku je nutné provést rozšíření prefixů, abychom snížili počet prefixů různé délky. Toto rozšíření způsobuje, že některé uzly obsahují duplicitní informace. Abychom ušetřili paměť, lze vícebitové stromy komprimovat. Zde si představíme algoritmus Lulea pro eliminaci redundantních informací ve vícebitovém stromu *trie*[4]. Algoritmus Lulea ukládá stromy *trie* do bitmapové struktury, kdy se odděluje uložení vlastních hodnot pole a výskytu hodnoty v poli. Využívá se při tom dvou polí: **valarr**(X) pro uchování hodnot pole a **bitarr**(X), které v bitové mapě uchovává informace o výskytu těchto hodnot v původním poli. Například pole P_5 , P_5 , P_6 , P_6 se dá reprezentovat jako

pole hodnot $\text{valarr}(X)$ s hodnotami P_5, P_6 a bitové pole popisující výskyt hodnot $\text{bitarr}(x)$ 1, 0, 1, 0, kde bit 1 označuje začátek nového hodnoty a 0 opakování hodnoty.

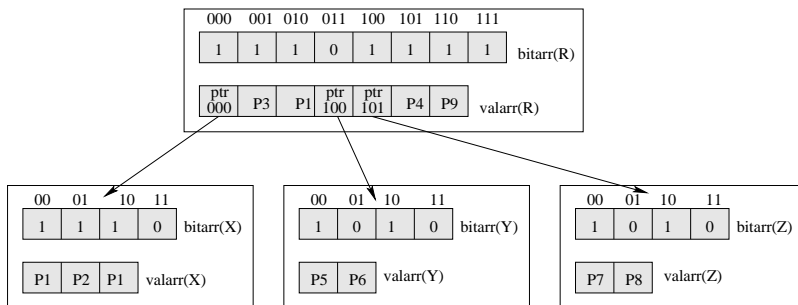
Na obrázku 9.9 vidíme, že některé prvky pole R v poli obsahují jednak hodnotu a jednak ukazatel (například první položka 000). Pro kompresi podle algoritmu Lulea je nutné, aby položka obsahoval buď hodnotu nebo ukazatel, nikoliv však obě informace zároveň. Toho dosáhneme jednoduchou transformací, kdy hodnoty z polí, které se odkazují na další uzly, posuneme o úroveň níže, viz obrázek 9.10. Zde vidíme, že prefix P_1 , který se posunul z prvku



Obrázek 9.10: Komprese vícebitového stromu *trie* – 1. fáze

000 pole R do pole X druhé úrovně na pozice 00, 10 a 11. Prefix P_4 v položkách 100 a 101 první úrovně je nadbytečný a lze ho zcela vypustit. Po této transformaci obsahuje každé políčku buď jenom hodnotu nebo ukazatel.

Dalším krokem je převedení uzlů pole do datové struktury Lulea, která obsahuje pole hodnot $\text{valarr}()$ a pole výskytů hodnot $\text{bitarr}()$, viz obrázek 9.11. Na obrázku můžeme vidět

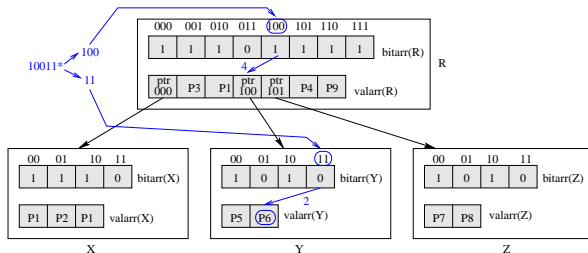


Obrázek 9.11: Komprese pomocí bitové mapy (Lulea) – 2. fáze

například kompresi pole $R = (\text{ptr}(000), P_3, P_1, P_1, \text{ptr}(100), \text{ptr}(101), P_4, P_9)$ z obrázku 9.10 do polí $\text{bitarr}(R)$ a $\text{valarr}(R)$. Předpokládáme, že původní pole R obsahuje položky o délce 32 bitů potřebných pro uložení ukazatele či hodnoty). Původní kořenový uzel R tedy vyžaduje pro uložení hodnot paměť o velikosti $8 \times 32 = 256$ bitů. Každý ze synovských uzlů X, Y a Z vyžaduje paměť 128 bitů, což je dohromady za celou strukturu *trie* 640 bitů. Pokud vezmeme komprimovaný strom dle obrázku 9.11, pak kořenový uzel R vyžaduje $8 + 7 \times 32 = 232$ bitů,

uzel X 100 bitů, uzly Y a Z každý po 68 bitech, což je celkem 486 bitů, tedy úspora 172 bitů oproti nekomprimovanému stromu. Tato úspora paměti je důležitá zejména při použití stromů *trie* s velkou hodnotou kroku.

Vyhledávání je podobné jako u nekomprimovaného vícebitového stromu *trie*. Začínáme v kořenovém uzlu R v bitovém poli $valarr(R)$, kde hledaný prefix o délce kroku k je indexem do bitového pole $bitarr(R)$. Například při hledání prefixu 10011 použijeme řetězec tříbitový řetězec 100 ($k = 3$) jako index bitového pole, viz obrázek 9.12. Nyní musíme určit počet jedničkových bitů od začátku pole na toto místo. Na to lze použít optimalizované algoritmy



Obrázek 9.12: Příklad vyhledání prefixu 10011*

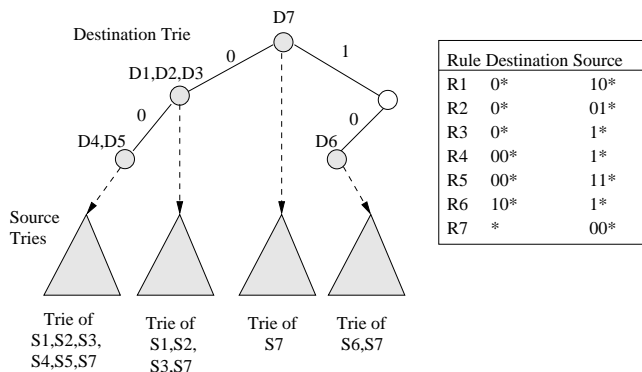
uvedené např. v [7]. Tato hodnota (v našem případě čtyři) se stává indexem do pole hodnot $valarr(R)$, kde nalezneme ukazatel na další uzel Y s adresou $ptr(100)$. Přejdeme tedy do uzlu Y , kde hledáme podobným způsobem, tj. další část hledaného prefixu (sekvence 11) se stane indexem do pole $bitarr(Y)$. Dále zjistíme počet předchozích jedničkových bitů (dva), který se stane indexem do pole $valarr(Y)$. V tomto poli je uložena hodnota pravidla P_6 , tj. identifikátor výstupní rozhraní pro daný prefix. P_6 je tedy nejdelším shodným prefixem pro vstupní hodnotu 10011.

9.5 Klasifikace ve dvou dimenzích

Pravidla používající dvě dimenze pro klasifikaci jsou vhodná pro určité typy aplikace a zařízení. Najdeme je například v páteřních směrovačích, které obsahují velké množství pravidel typy cílová IP adresa–zdrojová IP adresa pro směrování IP adres, VPN sítě či multicastového vysílání. Existuje také způsob jak převést problém klasifikace v n -dimenzích na řešení ve dvou dimenzích. Zde se podíváme, jak je možné rozšířit stromovou strukturu *trie* pro vyhledávání ve dvou dimenzích.

9.5.1 Struktura *trie* ve dvou dimenzích

Klasická stromová struktura *trie* se používá pro vyhledávání v jedné dimenzi, například pro vyhledávání nejdelšího prefixu u IP adres. Pro vyhledávání ve dvou dimenzích lze použít její zobecnění nazývané *síť stromů* *trie* (grid of tries). Tento návrh byl poprvé prezentován v roce 1998 v [?]. Síť stromů *trie* vytváří pro první dimenzi jeden strom *trie*, například strom *trie* cílovou IP adresu. Uzly, které obsahují prefixy P , ukazují na další strom *trie* pro vyhledávání v druhé dimenzi, například podle zdrojové adresy, viz obrázek 9.13.

Obrázek 9.13: Síť stromů *trie* pro dvě dimenze

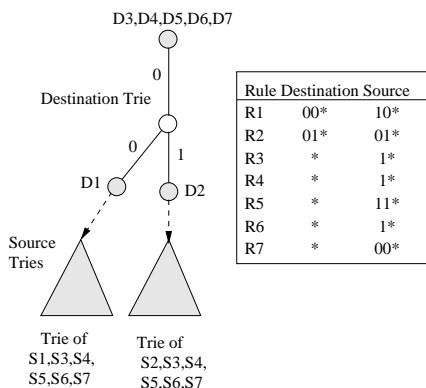
V síti stromů *trie* se nejprve vyhledává podle první dimenze, kdy získáme například nejdelší prefix pro cílovou IP adresu. Pak probíhá vyhledávání ve stromu zdrojové IP adresy. Pokud jsou obě vyhledávání úspěšná, došlo tedy k nalezení záznamu obsahující nejdelší shodný prefix v cílové i zdrojové adrese. V opačném případě nevyhovuje hledaný prefix žádnému pravidlu klasifikátor, paket je pak zpravidla zahozen.

Výše uvedená implementace sítě stromů *trie* pro dvě dimenze není efektivní, zejména kvůli rozšiřitelnosti a prostorové složitosti. Dochází zde k exponenciálnímu nárůstu prostorové složitosti, neboť stromy pro uložení zdrojové IP adresy $S_1 - S_7$ (Source) se kopírují do více koncových uzlů prvního stromu prefixu D_i (Destination). Například k prefixu $D_i = 00*$ jsou vztaženy dvě hodnoty druhé dimenze S_4, S_5 . Zároveň jsou k němu přidány hodnoty S_1, S_2, S_3 , neboť prefix $0*$ je zároveň prefixem pro D_i .

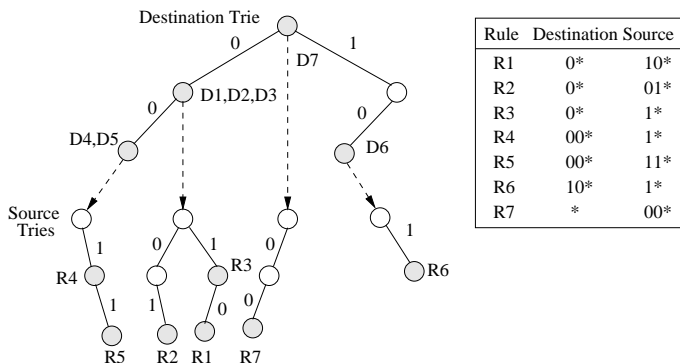
Obecně platí, že pokud je pravidlo první dimenze D_i prefixem jiného pravidla D_j , pak musíme strom *trie* druhé dimenze S_i přidat (duplikovat) ke stromům druhé dimenze pravidla R_j . Například adresa $0*$ je prefixem adresy $00*$, proto i stromy druhé dimenze S_1, S_2, S_3 se musí použít pro vyhledání paketu s cílovou adresou začínající 0011101 . Vyhledávání v tomto stromě je rychlé, nicméně pro praktické využití je extrémně náročný na paměť.

Nejhorší případ nastane, když každé pravidlo v první dimenzi obsahuje hodnotu $*$. Tato hodnota se shoduje s každou IP adresou klasifikovaného paketu. Potom koncový uzel první dimenze obsahuje N hodnot druhé dimenze, kde N je počet pravidel s hodnotou $*$ v první dimenzi. Paměťová složitost tohoto přístupu roste exponenciálně s počtem dimenzí. Pro dvě dimenze je to N^2 prvků, pro K -dimenzi je paměťová složitost $\mathcal{O}(N^K)$. Ukázka takového stromu je na obrázku 9.14.

Jedním ze způsobů, jak exponenciální paměťové nároky eliminovat, je použít odkazů na opakující se struktury ve druhé dimenzi [3, 6], viz část 9.5.1. Původní stromová struktura se změní v orientovaný acyklický graf DAG (Directed Acyclic Graph). Tuto upravenou reprezentaci pravidel lze použít pro softwarovou implementaci klasifikátorů s rozsahem do jednoho sta pravidel.

Obrázek 9.14: Exponenciální nárůst pravidel u sítí stromů *trie***Strom *trie* se zpětným vyhledáváním**

Dalším způsobem, jak se vyhnout exponenciálnímu nárůstu paměťové složitosti, je vytvořit strom *trie* pro druhou dimenzi tak, že nebude kopírovat pravidla druhé dimenze z předchozích prefixů, ale každé pravidlo bude uloženo pouze jednou. Například pro prefix 00* z obrázku 9.15 budeme mít ve stromu *trie* druhé dimenze pouze pravidla S_4, S_5 .

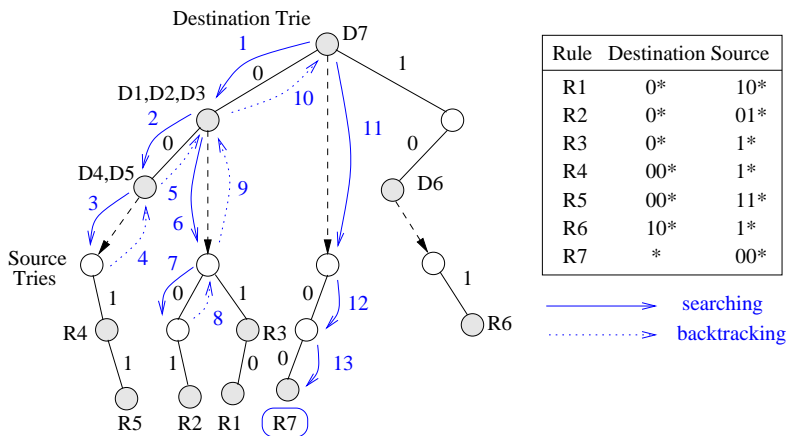
Obrázek 9.15: Strom *trie* se zpětným vyhledáváním (backtracking)

Vlastní algoritmus vyhledávání bude časově náročnější, neboť se prohledává nejen *trie* druhé dimenze pro prefix první dimenze D , ale i další *trie* druhé dimenze, na něž se odkazují předchůdci D . Například pro prefix cílové adresy 00* budeme prohledávat pravidla R_1, R_2, R_3

prefixu 0*, neboť tento prefix je předchůdcem (prefixem) prefixu 00*. Snížíme tedy paměťovou složitost vyhledávání (nedojde k duplikování pravidel) za cenu zvýšení časové složitosti. Vyhledávací algoritmus nejprve projde strom první dimenze s nejdelším prefixem a pokud nenajde shodu, začne se vracet a procházet stromy druhé dimenze u kratších prefixů. Paměťová složitost nyní bude $\mathcal{O}(N.W)$. Časová složitost v nejhorším případě stoupne na $\mathcal{O}(W^2)$, kde W je maximální počet bitů hledané dimenze. Přestože tento způsob implementace je v extrémních případech pomalý, v praxi se ukazuje, že může pracovat dobře.

Strom *trie* s ukazateli

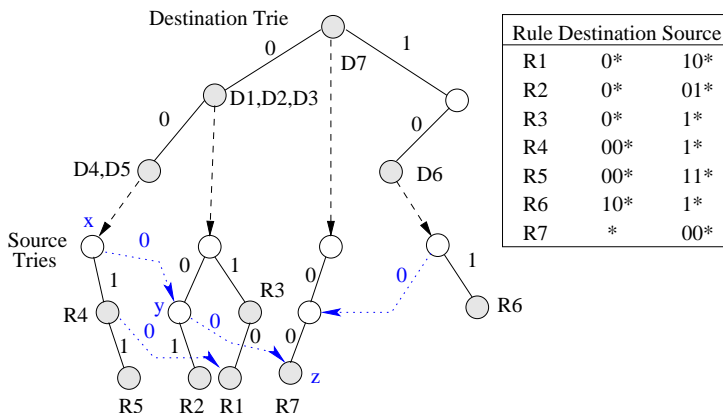
Struktura *trie* s ukazateli (switch pointers) zlepšuje algoritmus prohledání dvoudimenzionální struktury *trie*. Například při vyhledávání prefixu cílové IP adresy 000 a prefixu zdrojové IP adresy 001 ve stromě podle obr. 9.15, začne nejprve v uzlu 00*, kde však nedojde ke shodě na druhé podmínce. Vracíme se tedy po hranách od uzlu 00 směrem ke kořeni, viz obrázek 9.16. Nejprve se posuneme do uzlu 0*, což je předchůdce 00*. Toto vyhledávání je zbytečné, neboť při porovnání bitů prefixu zdrojové adresy 001 nedojde ke shodě.



Obrázek 9.16: Příklad hledání prefixu (000, 001) ve stromě se zpětným vyhledáváním

Toto prohledávání je možné zkrátit tím, že místo zpětného prohledávání uzlů první dimenze začneme u stromů předchůdců prohledávat stromy druhé dimenze s hodnotami, pro které se nepovedlo najít větve v předchozím vyhledávání. To znamená, že pro každý uzel v druhé dimenzi, kde porovnávání neuspělo, předpočítám přímý ukazatel na další možný strom ve druhé dimenzi, kde bude porovnávání úspěšné. Rozšíření pomocí ukazatelů pro náš příklad je zakresleno na obrázku 9.17.

Na tomto stromě *trie* si ukážeme vyhledání adresy (000, 001). Nejprve provedeme porovnání adresy první dimenze, tj. adresy 000. Přitom dojdeme až do uzlu D_4, D_5 stromu *trie* první dimenze. Odtud přejdeme přímo do stromu S_4 druhé dimenze, do bodu označeného písmenem x . Zde zahájíme porovnání zdrojové adresy 001. První bit této adresy nás vede po

Obrázek 9.17: Vyhledávání ve stromu *trie* s ukazateli

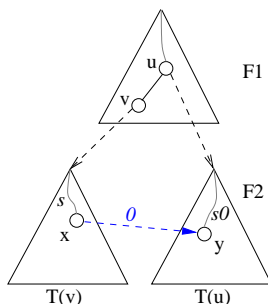
hraně s označením 0 z bodu x do bodu y stromu S_2 . Místo zpětného vyhledávání ve stromu první dimenze, jak jsme to dělali v předchozím příkladu, jsme se rovnou posunuli na zdrojový strom S_2 . Nyní pokračujeme ve vyhledávání podle druhého bitu adresy, kterým je opět hodnota 0. Pomocí dalšího ukazatele s označením 0 se přesuneme z bodu y do bodu z . Protože tento uzel je listový a obsahuje prefix R_7 , skončili jsme úspěšně vyhledávání v pěti krocích (oproti třinácti v předchozím příkladu).

Použitím odkazů lze zlepšit složitost vyhledání z $\mathcal{O}(W^2)$ na $\mathcal{O}(2.W)$ pro dvě dimenze. Toto vylepšení se nazývá zpětné vyhledávání s ukazateli (backtracking search with pointers). Zároveň zůstává zachována lineární paměťová složitost, tzn. stromy *trie* s další dimenze se neduplikují. Tento přístup vyžaduje při vytváření stromů *trie* fázi, kdy se předpočítávají ukazatele. Protože se to děje před vlastní klasifikací, tak není pro nás tato fáze časově kritická. Při vlastním vyhledávání můžeme úspěšně využít strukturu stromů *trie* s předpočítanými ukazateli.

Formálně můžeme definovat ukazatele následujícím způsobem.

Teorém 1. Nechť v je uzel ve stromu F_1 a reprezentuje prefix $P(v)$. Nechť u označuje uzel odpovídající nejbližšímu předchůdci $P(v)$ ve stromu F_1 . Nechť $T(v)$ a $T(u)$ jsou stromy dimenze F_2 napojené na uzly u a v v první dimenze. Předpokládejme, že x je uzel stromu $T(v)$, který reprezentuje prefix s a že z x neexistuje hrana 0 do dalšího uzlu reprezentující prefix $s0$, viz obrázek 9.18. Jestliže strom $T(u)$ v dimenzi F_2 obsahuje takový uzel u , který reprezentuje prefix $s0$, pak vytvoř ukazatel mezi uzly x a y . Tento přístup se aplikuje na všechny uzly v dimenzi F_2 .

Časová složitost vyhledávání v síti stromů je $\iota(W)$, prostorová složitost $(W.N)$, kde W je počet bitů nejdelšího prefixu v tabulce a N je počet pravidel.

Obrázek 9.18: Definice ukazatelů ve stromě *trie* s ukazateli

9.6 Klasifikace ve více dimenzích

Ke klasifikaci paketů ve více dimenzích se používají různé heuristické algoritmy, které byly vytvořeny na základě analýzy činnosti firewallů a směrovačů. Výsledkem analýzy jsou následující pozorování, která se používají při návrhu klasifikačních metod pro více dimenzí [7, str. 551–552], [13, str. 270–301]:

- P1: Počet polí.** Přestože většina implementací používá pravidla s osmi poli (dimenzemi), více jak 50% pravidel obsahuje pouze čtyři pole (dimenze).
- P2: Malá množina hodnot v poli protocol.** Většina klasifikátorů obsahuje v poli *protocol* hodnoty *TCP*, *UDP* či ***. Zřídka se používají typy *ICMP*, *GRE*, *IGMP*, *(E)IGRP* a *IP-in-IP*.
- P3: Porty používají převážně intervalové porovnávání.** Hodně pravidel obsahuje interval u zdrojových či cílových portů. Hodnoty jsou zadány buď pomocí znaku *** nebo intervalů ≤ 1024 či ≥ 1024 . Interval ≥ 1024 lze zapsat pomocí šesti prefixových intervalu (1024, 2047), (2048, 4095), (4096, 8191), (8192, 16383), (16384, 32767), (32768, 65535).
- P4: Počet disjunktních klasifikačních regionů je malý.** Toho se využívá zejména u geometrického porovnávání.
- P5: Porovnání zdrojová adresa – cílová adresa.** Při zkoumání činnosti páteřních směrovačů se zjistilo, že většina paketů se úspěšně porovná na pěti kombinacích dvojice zdrojová IP adresa–cílová IP adresa [9]. V nejhorším případě došlo k úspěšné shodě paketu na maximálně dvaceti pravidlech s různou kombinací páru zdrojová IP adresa–cílová IP adresa.
- P6: Sdílení stejné hodnoty v poli.** Klasifikační pravidla běžně obsahují v různých pravidlech stejné hodnoty pro klasifikaci. Jedná se zejména o obklopní provozu mezi dvojicemi konkrétních počítačů. Toto opakování vede ke sdílení hodnot zdrojové a cílové adresy.

P7: Redundantní pravidla. Analýzou se zjistilo, že téměř 15% pravidel je redundantních. O pravidle R řekneme, že je redundantní, pokud existuje pravidlo T , které je umístěno před R v seznamu pravidel, a zároveň R je podmnožinou T . V tomto případě žádný paket nebude úspěšně porovnán v pravidle R .

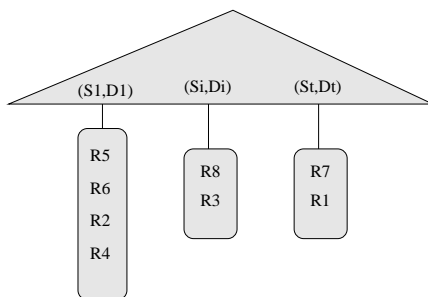
P8: Prefixy IP adres maximálně 24 bitů. Téměř všechny prefixy pro vyhledání IPv4 adres mají maximálně 24 bitů.

P9: Průnik prefixů není obvyklý. Není běžné mít prefixy, které jsou součástí jiných prefixů, například prefixy $00*$ a $0001*$.

Nyní si představíme několik metod pro klasifikaci paketů ve více dimenzích. Krátce si ukážeme, jak lze rozšířit dvoudimenzionální vyhledávání na více dimenzí, poté se podíváme na vyhledávání pomocí bitového vektoru a vektorového součinu.

9.7 Lineární seznam pravidel

Lineární seznam pravidel je jednoduchý způsob pro klasifikaci paketů ve více dimenzích, který vychází z pozorování P5, tj. že většina porovnávacích pravidel používá k vyhledávání převážně dvojici *zdrojová IP adresa-cílová IP adresa*, tj. (S, D) , kde S je zdrojová a D je cílová IP adresa. Pro reprezentaci k -rozměrných pravidel použijeme vhodnou 2D reprezentaci (např. síť stromů *trie*, viz část 9.5.1), na níž navážeme pro různé dvojice (S, D) lineární seznam pravidel, které odpovídají daným hodnotám (S, D) v pravidle (obrázek 9.19).



Obrázek 9.19: 2D porovnávání podle zdrojové a cílové adresy

Při klasifikaci nejprve prohledáváme prostor 2D se zdrojovou a cílovou IP adresou pomocí vhodného algoritmu. Výsledkem prohledání je množina všech dvojice (S_i, D_i) , které se shodují se zdrojovou a cílovou IP adresou klasifikovaného paketu. V dalším kroku pro každou takto nalezenou dvojici projdeme příslušný lineární seznam pravidel a porovnáme hodnoty paketu s položkami ve zbývajících dimenzích.

Výhodou tohoto přístup je, že každé pravidlo je reprezentováno pouze jednou. Případné intervaly hodnot (například u portů) tvoří obvykle vyšší dimenze a jsou tedy uloženy v připojeném seznamu. Algoritmus nejprve projde pro první dvojici (S_1, D_1) seznam pravidel a pokud nedorazí ke shodě, přesune se na další dvojici (S_i, D_i) .

Tento algoritmus předpokládá, že platí pravidlo P5, tj. je možné rychle vyhledávat podle dvojice *zdrojová-cílová adresa*. Čas vyhledávání závisí na čase dvoudimenzionálního vyhledávání a na době průchodu v lineárním seznamu pravidel o maximální délce $c < K$. Podle pravidla P5 zároveň platí, že $c \leq 20$. Příkladem může být použití sítě stromů *trie* a jejího rozšíření o další dimenze do tak zvané *rozšířené sítě stromů trie* (EGT, extended grid of tries), viz [1].

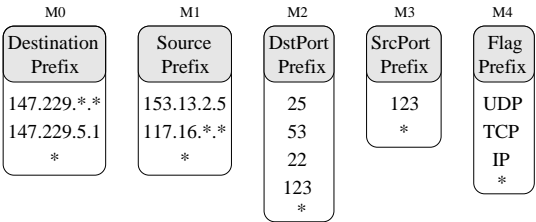
9.8 Metody typu rozděl a panuj

Metoda rozděl a panuj (divide-and-conquer) využívá myšlenku rozdělení problému na jednodušší podčásti, které se zpracují nezávisle a celkový výsledek se složí z dílčích výsledků. Příkladem může být rozdělení do jednotlivých dimenzí, kde každá položka obsahuje disjunktní prefixy dané dimenze.

	Dst IP	Src IP	DstP	SrcP	Proto	Akce	Poznámka
1	147.229.0.0/16	*	25	*	*	permit	povol příchozí emaily
2	147.229.0.0/16	*	53	*	UDP	permit	povol dotazy DNS
3	147.229.0.0/16	*	22	*	TCP	permit	povol přístup přes SSH
4	147.229.5.1	153.13.2.5	123	123	UDP	permit	povol NTP info
5	*	117.16.0.0/16	*	*	IP	decrypt	VPN ze sítě 117.16.0.0
6	*	*	*	*	*	deny	blokuj vše ostatní

Tabulka 9.5: Příklad vstupních filtrovacích pravidel na hraničním uzlu

Například pravidla z tabulky 9.5 lze rozdělit na pět množin $M_0 - M_4$ pro jednotlivé dimenze tak, že například množina M_0 pro dimenzi *cílová IP adresa* obsahuje čtyři různé položky, množina M_1 pro dimenzi *zdrojová IP adresa* tři různé položky a tak dále, viz obrázek 9.20.



Obrázek 9.20: Metoda rozděl a panuj

Metoda Rozděl a panuj prohledává jednotlivé množiny (dimenze) pomocí efektivních jednodimenzionálních metod (např. metod pro hledání nejdelšího shodného prefixu). Poté vezme výsledky dílčích hledání a pomocí vytvoří z nich celkový výsledek. Nyní si ukážeme tři způsoby implementace této metody – vyhledávání pomocí bitového vektoru, použití kartézského součinu a metodu rekurzivní klasifikaci toků.

Protože tato dílčí vyhledávání jsou nezávislá, je možné klasifikátor rozdělit na několik modulů, které paralelně zpracovávají hledání dané dimenze. Pro každou dimenzi lze použít

jinou metodu, která bude efektivnější pro hodnoty dané dimenze, například IP adresy, čísla portů apod. Každá z dílčích metod by měla vrátit nejen nejlepší nalezené pravidlo, ale spíše množinu pravidel tak, abychom při kombinaci s dalšími dílčími výsledky našli takové pravidlo, které není jen nejlepší v jedné dimenzi, ale vyhovuje i pro ostatní dimenze pravidla. Z tohoto důvodu je důležité podívat se také na způsob složení výsledku z dílčích částí.

9.8.1 Lineární prohledávání pomocí bitového vektoru (Lucent Bit Vector)

První metodou je použití bitového vektoru [12], který obsahuje výskyt prefixů v jednotlivých pravidlech, viz obrázek 9.21. Základní myšlenkou algoritmu je nejprve hledání příslušného

Destination Prefixes	Bit vector	Source Prefixes	Bit vector	DstPort Prefixes	Bit vector
147.229.*.*	111011	153.13.2.5	111101	25	100011
147.225.5.1	111111	117.16.*.*	111011	53	010011
*	000011	*	111001	22	001011
				123	000111
				*	000011

SrcPort Prefixes	Bit vector	Flags Prefixes	Bit vector
123	111111	UDP	110111
*	111011	TCP	101011
		IP	100011
		*	100001

Obrázek 9.21: Vytvoření bitového vektoru pro jednotlivé prefixy

pole z hlavičky paketu v seznamu pravidel podle jedné dimenze. Výsledek tohoto dílčího hledání se uloží do bitové mapy. Celkový výsledek je pak průnikem všech dílčích výsledků, tedy průnikem vektorů reprezentující bitové vektory (bitové mapy) jednotlivých dimenzí.

Každému prefixu M v tabulce pravidel je přiřazen vektor výskytu $S(M)$, který obsahuje informaci o výskytu prefixu v jednotlivých pravidlech tabulky 9.5. Pokud je j -tý bit vektoru $S(M)$ nastaven na hodnotu jedna, znamená to, že prefix M se vyskytuje v pravidle R_j (tj. dochází ke shodě na tomto pravidle). Například prefix 147.229.*.* ve sloupci cílová IP adresa se vyskytuje u pravidel R_1, R_2, R_3 . Zároveň dojde ke shodě při hledání, zda je tento prefix součástí prefixu * v pravidlech R_5 a R_6 , proto bitový vektor výskytu bude 111011. Délka vektoru $S(M)$ odpovídá počtu všech porovnávacích pravidel.

Tabulka bitových vektorů pro jednotlivé prefixy všech dimenzí se předpočítá dopředu z vytvořených filtrovacích pravidel. Pokud přijde paket P , který obsahuje hlavičky H_1, \dots, H_K (pro K -dimenzí), provede se nejprve vyhledání nejdelšího shodného prefixu M_i pro každou dimenzi i . Vyhledání provedeme pouze v jedné dimenzi. Výstupem je bitový vektor $S(M_i)$, který označuje množinu vyhovujících pravidel prefixu M v dimenzi i .

Toto jednodimenzionální vyhledávání provedeme pro každou dimenzi. Algoritmus se dá implementovat paralelně, neboť výpočetní operace i datové položky jsou nezávislé. Bitové vektory dílčích hledání v jednotlivých dimenzích poskládáme pomocí operace AND . Výsledný bitový vektor označuje výskyt klasifikovaného paketu v seznamu filtrovacích pravidel.

Uvažujme například klasifikaci paketu jdoucího z IP adresy 147.229.15.1 a portu 1029 na adresu 147.229.5.1 a port 53, tj. $H = (147.229.5.1, 147.228.15.1, 53, 1029, UDP)$. Pro jednotlivé dimenze dostáváme vektory výskytu 111011, 111001, 010011, 111011 a 110111. Například první vektor výskytu 111011 pro cílovou IP adresu odpovídá položce 147.229.*.* první dimenze, kde došlo k nejlepší prefixové shodě na cílovou IP adresu. Bitovým součinem těchto vektorů dostaneme vektor 010001, což znamená, že se uplatní pravidla R_2 a R_6 . Protože menší index (tzn. větší prioritu) má pravidlo R_2 , použije se tedy toto pravidlo.

Z předchozího příkladu není jasné, jak vypadá implementace vyhledávání v jedné dimenzi. Myslím, že nestačí jen říci, že pro adresu 147.229.5.1 vyhledáme příslušný bitový vektor 111011. Přemýšlivého čtenáře jistě napadne, jak k tomu dojde, aby to bylo rychlé a efektivní na uložení a zpracování.

Demonstraci vyhledávání a klasifikace si ukážeme na jiném příkladu, který je snažší na zobrazení. Místo vyhledávání 16 a 32 bitů zvolíme zjednodušená pravidla, která budou vyhledávat řetězce o délce maximálně dva bity. Předpokládejme klasifikátor, který má osm pravidel $R_1 - R_8$ a dvě dimenze F_1 a F_2 , viz obrázek 9.22.

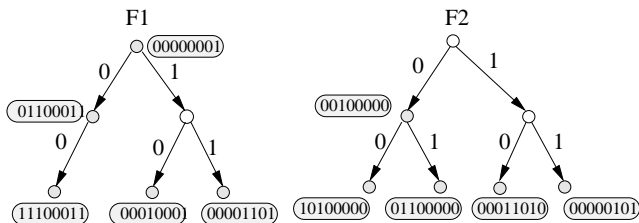
Rule	F1	F2
R1	00*	00*
R2	0*	01*
R3	0*	0*
R4	10*	10*
R5	11*	10*
R6	11*	11*
R7	0*	10*
R8	*	11*

F1	Bit vector
0*	01100011
00*	11100011
10*	00010001
11*	00001101
*	00000001

F2	Bit vector
0*	00100000
00*	10100000
01*	01100000
10*	00011010
11*	00000101

Obrázek 9.22: Klasifikátor pro dvě dimenze pomocí bitového vektoru

Implementace tohoto klasifikátoru pomocí binárních stromů *trie* pro obě dimenze je zakreslena na obrázku 9.23. Z tohoto obrázku je patrné, jak probíhá klasifikace. Například při



Obrázek 9.23: Implementace klasifikátoru pomocí bitového vektoru

hledání paketu s poli 001 a 010, proběhne paralelní vyhledání prvního prefixu ve stromu F_1 , kde výsledek je uzel bitovým vektorem 11100011. Ve druhé dimenzi, tj. ve stromu F_2 , se vyhledá bitový vektor 01100000. Průnikem těchto dvou bitových vektorů dostaneme výsledný

bitový vektor 01100000, který říká, že ke shodě došlo podle pravidla R_2 a R_3 . Protože pravidla jsou uspořádána podle priority, vybere se pravidlo podle nižšího bitu výsledného bitového vektoru, tj. R_2 .

Použití metody bitového vektoru má časovou složitost vyhledání $\mathcal{O}(N)$, kde N velikost bitového vektoru. Ze vztahu též vyplývá, že časová složitost je nezávislá na počtu dimenzí, tj. přidáním nové dimenze se nám nezhorší. Je to způsobeno právě tím, že dochází k dekompozici vyhledávání na jednotlivé dimenze. Vyhledávání v jednotlivých dimenzích je paralelní, tzn. nezhorší časovou složitost. Pro lepší využití paměti se používá modifikace bitových vektorů, která se nazývá *agregované bitové vektory* (ABV, Aggregated Bit Vector) [2].

Použití bitových vektorů se doporučuje pro databáze střední velikosti. Přidávání nových pravidel je pomalé a vyžaduje aktualizaci celé databáze.

9.8.2 Klasifikace pomocí kartézského součinu

Síť stromů *trie* (část 9.5.1) s ukazateli je efektivní pro klasifikaci při použití dvojice *cílová-zdrojová IP adresa*. Uložení do paměti má pouze lineární složitost. Časová složitost vyhledávání je ekvivalentní době vyhledávání nad jednotlivými dimenzemi, což je dostačující pro směrovače.

Chceme-li použít obecnější filtry, které obsahující více než dvě dimenze (např. ve firewallch), potřebujeme jiný přístup. Jedním z možných rozšíření je *kartézský součin* (cross producting)⁵ nad množinami hodnot jednotlivých dimenzí [8]. Motivací pro použití kartézského součinu ke klasifikaci je pozorování, že počet neopakujících se prefixů dané dimenze (například zdrojová IP adresa) je výrazně menší než celkový počet pravidel klasifikátoru, zejména u klasifikátorů s velkým počtem pravidel. Kartézský součin tvoří množina n -tic nad neopakujícími se prefixů každé dimenze.

Příklad na obrázku 9.24 zobrazuje množiny neopakujících se prefixů M_0 – M_5 , které vychází z klasifikátoru popsaného tabulkou 9.5. Každá množina prefixů M_i obsahuje jedinečné prefixy

M0	M1	M2	M3	M4
Destination Prefix	Source Prefix	DstPort Prefix	SrcPort Prefix	Flag Prefix
147.229.*.* 147.229.5.1 *	153.13.2.5 117.16.*.* *	25 53 22 123 *	123 *	UDP TCP IP *

Obrázek 9.24: Rozdělení množiny pravidel z tabulky 9.5 do dimenzí

dané dimenze. Nad těmito k množinami (kde k je počet dimenzí), vytváříme kartézský součin, tj. množinu vektorů o rozměru k (k -tice), kde každá položka vektoru obsahuje jednu hodnotu z dané dimenze.

Nechť M_0, M_1, \dots, M_{k-1} jsou množiny dimenzí. Množina k -tic $C = M_0 \times M_1 \times \dots \times M_{k-1}$ se nazývá *kartézský součin*. Celkem je možné vytvořit $|M_0| \cdot |M_1| \cdot \dots \cdot |M_k| \cdot \dots \cdot |M_{k-1}|$ k -tic, kde

⁵Původní anglický termín *cross product*, se někdy překládá jako vektorový a někdy jako kartézský součin

$|M_i|$ je počet prvků (hodnot) dimenze i . Tyto množiny se použijí pro vyhledání nejdelšího shodného prefixu nad danou dimenzí.

Hlavní myšlenka pro využití kartézského součinu při hledání nejlepšího prefixové shody je následující tvrzení:

Teorem 2. Nechť je dán paket P . Pokud provedeme hledání nejdelšího prefixu nad každým polem $P[i]$ a spojíme výsledek do podoby kartézského součinu C , pak nejlepším klasifikačním pravidlem pro paket P je nejlepší klasifikační pravidlo nalezené pro paket C .

Důkaz. Důkaz tohoto tvrzení se provádí sporem, viz např. [7, str. 560]. Předpokládejme, že dané tvrzení neplatí, tzn. že nejlepší klasifikační pravidlo pro paket P není stejné jako pro paket C . Protože každé pole v C je prefixem odpovídajícího pole v P , pak každé pravidlo, u něhož dojde ke shodě nad paketem C , najde také shodu u paketu P . Pokud dojde u P ke shodě na jiném klasifikačním pravidle, pak to znamená, že existuje jiné pravidlo R , kde dojde ke shodě s paketem P , ale nikoliv s paketem C . Toto je možné pouze v případě, kdy toto pravidlo R obsahuje v i -té dimenzi položku $R[i]$ takovou, že $R[i]$ je prefixem $P[i]$, ale nikoliv $C[i]$, kde $C[i]$ i -tá položka kartézského součinu C . Protože je $C[i]$ prefixem $P[i]$, tak se toto může stát pouze tehdy, když $R[i]$ je delší než $C[i]$ (ale vyhovuje $P[i]$). Toto je ale v rozporu s předpokladem, že $C[i]$ je nejdelší shodný prefix pro položku i . \square

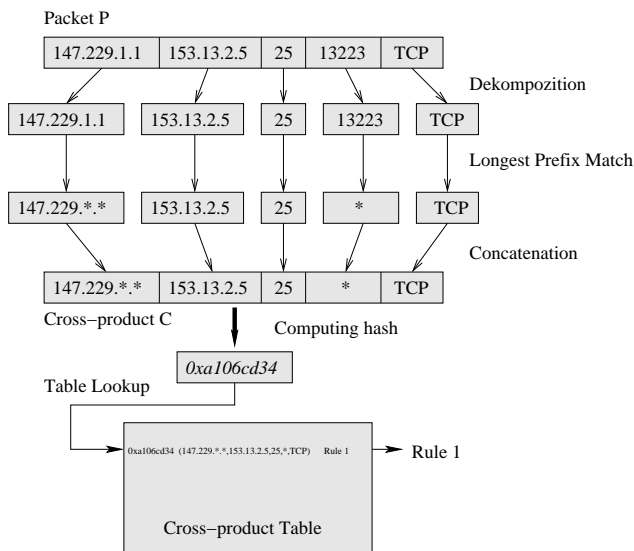
Metoda kartézského součinu začíná tím, že vytvoří vhodné datové struktury pro hledání v jednoduchých množinách prefixů M_i . Tyto struktury (například jednodimenzionální stromy *trie*), provádějí nejdelší prefixovou shodu nad odpovídajícími položkami s hlavičky paketu P . Abychom určili nejlepší prefixové pravidlo, musíme vytvořit tabulku C_T kartézského součinu $M_0 \times M_1 \times \dots \times M_{k-1}$. Tato tabulka obsahuje pro každý vektor kartézského součinu předpocítané pravidlo pro nejlepší shodu, viz obr. 9.25. Pro každý vektor tabulky C_T je

Number	Cross Product	Matching Rule
1	(147.229.*,*,153.13.2.5, 25,123,UDP)	Rule 1
2	(147.229.*,*,153.13.2.5,25,123,TCP)	Rule 1
3	(147.229.*,*,153.13.2.5,25,123,IP)	Rule 1
4	(147.229.*,*,153.13.2.5,25,123,*)	Rule 1
5	(147.229.*,*,153.13.2.5,25,*,UDP)	Rule 1
6	(147.229.*,*,153.13.2.5,25,*,TCP)	Rule 1
○	○	○
○	○	○
○	○	○
359	(*,*,*,*,IP)	Rule 6
360	(*,*,*,*,*)	Rule 6

Obrázek 9.25: Tabulka kartézského součinu

v tabulce odpovídající pravidlo pro nejlepší shodu. Pokud klasifikátor neobsahuje univerzální pravidlo $(*,*,*,*,*,*)$, může se stát, že pro danou kombinaci položek vektoru neexistuje žádné pravidlo. Pokud v příkladu dle tabulky 9.5 nebude pravidlo č. 6, tak pro vektor (147.229.5.1, 10.10.1.1, 25, 123, UDP) není žádné vyhovující klasifikační pravidlo a jedná se o tak zvaný *prázdný kartézský součin*.

Vyhledávání pomocí kartézského součinu probíhá následujícím způsobem. Když přijde paket P , určíme nejprve pro jednotlivé položky $P[i]$ nejlepší prefixovou shodu nad množinou prefixů M_i . Vybrané položky uspořádáme do kartézského součinu C . Vektor C pak použijeme jako *hash* do tabulky C_T , kde vyhledáme nejlepší pravidlo. Například pro paket s hodnotami (147.229.1.1, 153.13.2.5, 25, 13223, TCP) vyhledáme pro jednotlivé dimenze nejlepší prefixovou shodu a vytvoříme kartézský součin $C = (147.229.*.* , 153.13.2.5, 25, *, TCP)$. *Hash* tohoto vektoru je indexem do tabulky C_T , kde výsledkem je pravidlo číslo 1, viz obrázek 9.26.



Obrázek 9.26: Klasifikace paketu P pomocí tabulky kartézského součinu

Pro malé databáze je možné prefixové vyhledávání provádět přímo ve vyrovnávací paměti a pak jednou přistoupit do hlavní paměti, kde je uložena celá předpočítaná tabulka kartézského součinu.

Prefixové vyhledávání v jednotlivých dimenzích lze – podobně jako u bitového vektoru – implementovat paralelně. Jak je však vidět na předchozím příkladu, metoda kartézského součinu má exponenciální nárůst prostorové složitosti tabulky C_T . Její velikost je v nejhorším případě N^k , kde k je počet dimenzí a N počet prvků. I v případě malých seznamů pravidel se může jednat o velký počet položek. V našem případě to bylo $3 * 3 * 5 * 2 * 4 = 360$ položek. Tento počet s každým dalším prvkem v kterékoli dimenzi exponenciálně roste.

Jedním z řešení problému prostorové exploze je budovat tabulku postupně během výpočtu (on-the-fly). Datové struktury pro výpočet nejdelších prefixů pro jednotlivé dimenze se vypočtou a uloží v paměti. Tabulka vektorů kartézského součinu je však na začátku prázdná. Začne se plnit až s příchodem prvního paketu, kdy se hledá nejdelší shoda prefixu v dané

dimenzi. Pokud v tabulce vektorů chybí vektor C , začne se (nejčastěji pomocí lineárního prohledávání) hledat nejlepší pravidlo pro vektor C . Nalezené pravidlo se spolu s vektorem C vloží do tabulky.

Další variantou je metoda rekurzivní klasifikace toků RFC, která pro úsporu prostoru vytváří pro vektory kartézského součinu třídy ekvivalence. O této metodě si povíme v další kapitole.

9.8.3 Rekurzivní klasifikace toků RFC

Metoda rekurzivní klasifikace toků (RFC, Recursive Flow Classification) [5] nebo také metoda kartézského součinu s ekvivalencí (equivalenced cross-producting) [8], je rozšířením metody kartézského součinu v kombinaci s bitovým vektorem.

Metoda rekurzivní klasifikace toků řeší exponenciální nárůst záznamů při použití kartézského součinu prohledávaných dimenzí. Metoda vychází z pozorování, že mnohé kombinace prefixů jednotlivých dimenzí, tj. vektorů, odpovídají stejné množině klasifikačních pravidel. Tyto vektory, které mají stejná pravidla, tvoří třídu ekvivalence. Jinými slovy můžeme říci, že dva prefixy, jsou ve stejné třídě ekvivalence, pokud vyhovují stejným klasifikačním pravidlům.

Princip této metody je založen na výpočtu dílčích kartézských součinů (například pro dvě dimenze) nad třídami ekvivalence, což je efektivnější než výpočet jednoho kartézského součinu nad všemi dimenzemi, jak je to uvedeno například na obrázku 9.26. Princip a použití metody rekurzivní klasifikace toků si ukážeme na příkladu pravidel z tabulky 9.5.

Můžeme například vytvořit jeden kartézský součin pro dvojici *cílová – zdrojová IP adresa*, další součin pro dvojici *cílový a zdrojový port* apod. Pro každý takový součin určíme bitový vektor výskytu v jednotlivých pravidlech, viz část 9.8.1. Pro náš příklad vyjdeme z tabulka bitových vektorů pro jednotlivé dimenze, která je vypočtena na obrázku 9.27. Za pomoci

Destination Prefixes	Bit vector	Source Prefixes	Bit vector	DstPort Prefixes	Bit vector
147.229.*.*	111011	153.13.2.5	111101	25	100011
147.225.5.1	111111	117.16.*.*	111011	53	010011
*	000011	*	111001	22	001011
				123	000111
				*	000011

SrcPort Prefixes	Bit vector	Flags Prefixes	Bit vector
123	111111	UDP	110111
*	111011	TCP	101011
		IP	100011
		*	100001

Obrázek 9.27: Vytvoření bitového vektoru pro jednotlivé prefixy

těchto bitových vektorů pak tvoříme tabulku dílčích kartézských součinů nad dvojicemi dimenzí. Pokud mají dílčí kartézské součiny (dvojice) stejný bitový vektor, tvoří stejnou třídu ekvivalence.

Celkový výsledek netvoříme skládáním dílčích kartézských součinů, ale kombinací vypočtených tříd ekvivalence. Tím významně redukuje paměťové nároky, neboť dílčí kartézské

součiny se stejným bitovým vektorem se mapují do jedné třídy ekvivalence.

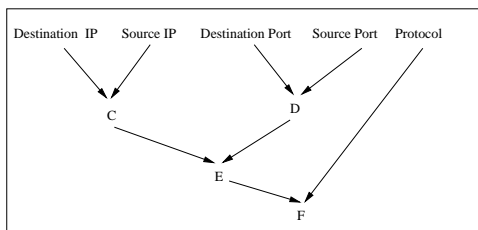
Příklad vytvoření tříd ekvivalence pro kartézský součin *zdrojová a cílová IP adresa* a pro součin *zdrojový a cílový port* je v tabulce 9.6.

Dest. IP – source IP prefix	Rule Bitmap	Class	Dest – Src port	Rule Bitmap	Class
(147.229.*, 153.13.2.5)	111001	C1	(25,123)	100011	D1
(147.229.*, 117.16.*,*)	111011	C2	(53,123)	010011	D2
(147.229.*,*)	111001	C1	(22,123)	001011	D3
(147.229.5.1, 153.13.2.5)	111101	C3	(123,123)	000111	D4
(147.229.5.1, 117.16.*,*)	111011	C2	(*,123)	000011	D5
(147.229.5.1,*)	111001	C1	(25,*)	100011	D1
(*, 153.13.2.5)	000001	C4	(53,*)	010011	D2
(*, 117.16.*,*)	000011	C5	(22,*)	001011	D3
(*,*)	000001	C4	(123,*)	000011	D5
			(*,*)	000011	D5

Tabulka 9.6: Kartézský součin se třídami ekvivalence (metoda RFC)

Pro každou dvojici (cílová IP adresa, zdrojová IP adresa) vybereme pravidla, která vyhovují obou těmto položkám. Například dvojice (147.229.*, 147.229.5.1) vyhovuje pravidlům R_1, R_2, R_3 a R_7 v tabulce 9.5. Bitový vektor výskytu této dvojice hodnot je 1110001. Bitový vektor lze jednoduše vypočítat jako průnik (bitový AND) prefixů zdrojové a cílové IP adresa, jak jsou uvedeny na obrázku 9.21, pro výše uvedený příklad je to průnik vektoru 1110101 pro cílovou IP adresu a vektoru 1111011 pro zdrojovou IP adresu.

Bitové vektory vypočítaných dílčích kartézských součinů tvoří třídy ekvivalence, tj. součiny se stejným bitovým vektorem patří do jedné třídy ekvivalence, například třídy C1. Všechny dílčích kartézských součinů je devět (tři hodnoty první dimenze krát tři hodnoty druhé dimenze), tříd ekvivalence je pouze pět. U dvojic *cílový a zdrojový port* nám deset kartézských součinů vytvoří pět tříd ekvivalence. Oproti původnímu počtu kartézských součinů nad všemi dimenzemi, který byl 360, máme nyní pouze 120 součinů, které jsou uloženy v paměti. To významně zrychluje vyhledávání při klasifikaci paketů. Touto redukcí však nemusíme skončit. Můžeme pokračovat kombinací tříd ekvivalence C a D a vytvořením nových tříd ekvivalence, viz obrázek 9.28.



Obrázek 9.28: Kombinace tříd ekvivalence pro více prefixů

Nad těmito třídami ekvivalence budeme vytvářet další dílčí kartézské součiny. Například kartézský součin pro třídy ekvivalence C a D je ukázán v tabulce 9.7. Pro konstrukci této

Cross Product	Bitmap	Class	Cross Product	Bitmap	Class	Cross Product	Bitmap	Class
(C1,D1)	100001	E1	(C3,D1)	100001	E1	(C5,D1)	000011	E8
(C1,D2)	010001	E2	(C3,D2)	010001	E2	(C5,D2)	000011	E8
(C1,D3)	001001	E3	(C3,D3)	001001	E3	(C5,D3)	000011	E8
(C1,D4)	000001	E4	(C3,D4)	000101	E9	(C5,D4)	000011	E8
(C1,D5)	000001	E4	(C3,D5)	000001	E4	(C5,D5)	000011	E8
(C2,D1)	100011	E5	(C4,D1)	000001	E4			
(C2,D2)	010011	E6	(C4,D2)	000001	E4			
(C2,D3)	001011	E7	(C4,D3)	000001	E4			
(C2,D4)	000011	E8	(C4,D4)	000001	E4			
(C2,D5)	000011	E8	(C4,D5)	000001	E4			

Tabulka 9.7: Tabulka kartézského součinu tříd ekvivalence C a D

tabulky a vytvoření tříd ekvivalence jsme opět použili bitové vektory. V tabulce můžeme opět vidět redukci položek způsobenou použitím ekvivalenčních tříd – v dalším stupni kartézského součinu místo 25 položek použijeme pouze devět tříd ekvivalence $E1$ - $E9$.

V posledním kroku vytváření tabulek dílčích kartézských součinů zkombinujeme třídu ekvivalence E a číslo protokolu. Nad tímto součinem opět spočítáme třídu ekvivalence a získáme devět tříd F - $F9$, místo 36 kombinací, viz tabulka 9.8.

Cross Product	Bitmap	Class	Cross Product	Bitmap	Class	Cross Product	Bitmap	Class
(E1,UDP)	100001	F1	(E4,UDP)	000001	F3	(E7,UDP)	000011	F7
(E1,TCP)	100001	F1	(E4,TCP)	000001	F3	(E7,TCP)	001011	F8
(E1,IP)	100001	F1	(E4,IP)	000001	F3	(E7,IP)	000011	F7
(E1,*)	100001	F1	(E4,*)	000001	F3	(E7,*)	000001	F3
(E2,UDP)	010001	F2	(E5,UDP)	100011	F5	(E8,UDP)	000011	F7
(E2,TCP)	000001	F3	(E5,TCP)	100011	F5	(E8,TCP)	000011	F7
(E2,IP)	000001	F3	(E5,IP)	100011	F5	(E8,IP)	000011	F7
(E2,*)	000001	F3	(E5,*)	100001	F1	(E8,*)	000001	F3
(E3,UDP)	000001	F3	(E6,UDP)	010011	F6	(E9,UDP)	000101	F9
(E3,TCP)	001001	F4	(E6,TCP)	000011	F7	(E9,TCP)	000001	F3
(E3,IP)	000001	F3	(E6,IP)	000011	F7	(E9,IP)	000001	F3
(E3,*)	000001	F3	(E6,*)	000001	F3	(E9,*)	000001	F3

Tabulka 9.8: Tabulka kartézského součinu tříd ekvivalence E a *Protocol*

Posledním krokem je určení klasifikačního pravidla pro třídu F . Zde bude klasifikační pravidlo odvozeno od bitového vektoru příslušné třídy s tím, že nejlepší pravidlo (v případě výše pravidel) odpovídá nejnižšímu bitu bitového vektoru. Příslušná pravidla jsou uložena v tabulce mapování tříd ekvivalence na číslo pravidla, viz tab. 9.9.

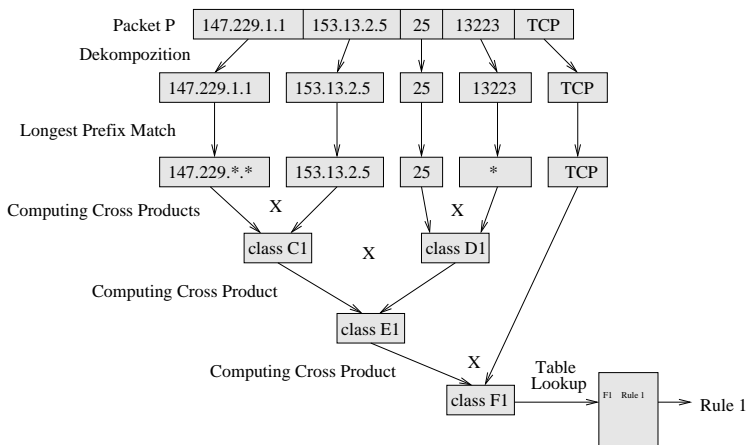
Proces klasifikace je podobný jako u metody kartézského součinu s tím, že se vždy vytvoří částečný vektorový součin dvojice. Příslušná třída ekvivalence se dále použije jako ukazatel

Class	Bitmap	Rule
F1	100001	Rule 1
F2	010001	Rule 2
F3	000001	Rule 6
F4	001001	Rule 3
F5	100011	Rule 1
F6	010011	Rule 2
F7	000011	Rule 5
F8	001011	Rule 3
F9	000101	Rule 4

Tabulka 9.9: Tabulka mapování ekvivalentních tříd na pravidla

do tabulky kombinací tříd. Nakonec získáme pravidlo pro nejlepší shodu.

Nyní si ukážeme klasifikaci paketu $P = (147.229.1.1, 153.13.2.5, 25, 13223, TCP)$ nad takto vytvořenou tabulkou RFC, viz obrázek 9.29. Začneme - podobně jako u metody



Obrázek 9.29: Klasifikace paketu P metodou RFC

kartézského součinu - rozdělením na jednotlivé dimenze (fáze dekompozice) a nalezením nejlepších prefixů. Tyto operace mohou probíhat paralelně. Pro paket P bude výsledkem vektor $(147.229.*.8, 153.13.2.5, 25, *, TCP)$. Nyní začneme vytvářet dílčí kartézské součiny, například pro IP adresy a porty. Dílčí vektory $(147.229.*.8, 153.13.2.5)$ a $(25, *)$ použijeme jako indexy do příslušných tabulek dílčích kartézských součinů. Výsledkem hledání budou třídy ekvivalence $C1$ a $D1$. Další krokem je kartézský součin tříd ekvivalence $C \times D$, viz tab. 9.7, kde je výsledkem třída $E1$.

V posledním kroku použijeme kartézský součin třídy E a dimenze *Protocol*, viz tab. 9.8. Výsledkem našeho hledání bude třída $F1$, která se podle tab. 9.9 mapuje na pravidlo číslo 1.

Jak vidíme z příkladu, bitový vektor se používá pouze pro konstrukci tříd ekvivalence, nikoliv pro vlastní vyhledávání. Použitím tříd ekvivalence významně zredukuje prostor pro uložení pravidel. Oproti 360 položkám v původní tabulce máme nyní v tabulkách kartézských součinů pouze 77 položek.

Paměťová složitost je dána počtem různých hodnot v jednotlivých dimenzích. Pro dvě dimenze je to složitost $\mathcal{O}(|M_i| \cdot |M_j|)$, kde $|M_i|$, resp. $|M_j|$ je počet prvků v dimenzi i , resp. j . V nejhorším případě to může degradovat na složitost $\mathcal{O}(N^K)$ pro K -dimenzi a N hodnot v každé dimenzi. Pro databázi 8000 pravidel autoři uvádějí, že všechny studované databáze měly lineární paměťovou složitost $\mathcal{O}(N)$ [5].

Použitá literatura

- [1] F. Baboescu, S. Singh, and G. Varghese. *Packet classification for core routers: Is there an alternative to CAMs? Proceedings of IEEE INFOCOM*, 2003.
- [2] F. Baboescu and G. Varghese. Scalable packet classification. In *ACM SIGCOMM*, pages 199–210, 2001.
- [3] D. Decasper, Z. Dittia, G. Parulkar, and B. Plattner. *Router plugins: A software architecture for next-generation routers. Proceedings of ACM SIGCOMM*, 1998.
- [4] Mikael Degermark, Andrej Brodnik, Svante Carlsson, and Stephen Pink. Small forwarding tables for fast routing lookups. In *ACM Sigcomm*, pages 3–14, 1997.
- [5] P. Gupta and N. McKeown. *Packet classification on multiple fields. Proceedings of ACM SIGCOMM*, pages 147–160, 1999.
- [6] G. Malan and F. Jahanian. *An extensible probe architecture for network protocol measurement. Proceedings of ACM SIGCOMM*, 1998.
- [7] D. Medhi and K. Ramasamy. *Network Routing. Algorithms, Protocols, and Architectures*. Elsevier, Inc., 2007.
- [8] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and Scalable Layer Four Switching. *SIGCOMM Comput. Commun. Rev.*, 28:191–202, October 1998.
- [9] S. Singh, F. Baboescu, and G. Varghese. *Packet classification using multidimensional cutting. Proceedings of ACM SIGCOMM*, 2004.
- [10] David E. Taylor. *Survey and taxonomy of packet classification techniques. ACM Comput. Surv.*, 37(3):238–275, 2005.
- [11] Kevin Thompson, Gregory J. Miller, and Rick Wilder. Wide-area internet traffic patterns and characteristics. *IEEE Network*, 11:10–23, 1997.
- [12] T.V. Lakshman and D. Stidialis. *High-speed policy-based packet forwarding using efficient multidimensional range matching. Proceedings of ACM SIGCOMM*, 1998.
- [13] George Varghese. *Network Algorithmics*. Elsevier, Inc., 2005.

Kapitola 10

Správa sítě

Nárůst počítačové komunikace a služeb klade důraz na efektivní využití zdrojů. Netýká se to pouze nároků na procesorový výkon serverů a připojených uživatelských stanic, ale také zařízení, která zajišťují konektivitu a spolehlivý přenos dat. Mezi sledovaná zařízení patří aktivní síťové prvky (přepínače, směrovače, firewally), serverové aplikace (server DNS, webový portál, poštovní server, souborový server) a další služby (tiskové server, ústředny VoIP, video servery, apod.).

Síťové zdroje stále využívá rostoucí počet uživatelů. Na počátku devadesátých let 20. století si webové stránky vytvářela pouze hrstka nadšenců přímo v kódu HTML, který psali v textových editorech typu `vi` či `joe`. Stránky si prohlíželi pomocí prehistorických prohlížečů typu NSCA Mosaic, Netscape Navigator a další. Dnes je situace opačná. Většina dnešních studentů (a nejen vysokoškolských) má své vlastní webové stránky, účty na sociálních sítích apod. Používají služby jako Facebook, Twitter, Youtube, Google docs, Alfresco apod. Nemít emailovou adresu či účet na Facebooku je dnes pro spoustu lidí stejně podivné jako nevlastnit mobilní telefon.

Uživatelé těchto služeb požadují rychlé a spolehlivé připojení. Kdo ho má pro ně zajišťovat? Samozřejmě že poskytovatelé internetového připojení ISP a síťoví administrátoři. Problém je, že počet šikovných administrátorů neroste stejně rychle jako počet uživatelů. Co s tím dělat? Napadají mě dvě řešení. Abychom mohli uspokojit stoupající počet požadavků na správu sítě, je potřeba, aby se více studentů zaměřilo na studium počítačových sítí. Což není zrovna evropský trend, ale existují i výjimky, například FIT v Brně :). Druhým řešením je vytvářet inteligentní systémy pro správu a řízení sítí, které mohou mnohé administrativní a monitorovací činnosti vykonávat bez zásahu člověka. Jaké nástroje a systémy pro správu se používají, si ukážeme v této kapitole.

10.1 Úvod do správy sítí

Správa sítě (angl. network management) je pojem, který nezahrnuje pouze instalaci kabelů, připojování počítačů a konfiguraci IP adres. Správa sítě zahrnuje také sledování a monitorování aktivit na síti, detekci chyb, útoků, vyvažování výkonu sítě, plánování a další. Jak přistupovat k tomuto úkolu?

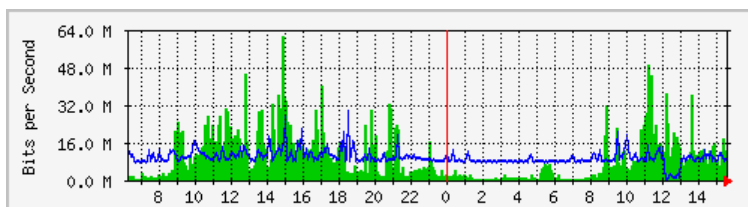
Nejjednodušší scénář může vypadat takto: síťový administrátor sedí ve své kanceláři, čte denní tisk, chatuje s kolegy a popije výbornou brazilskou kávu. Náhle zazvoní telefon a vedoucí projekce se ptá, proč nejde elektronická pošta. Administrátor dočte fejton a začne zjišťovat,

co se děje. Po nějaké době přijde na to, že vypadla linka připojující firemní IMAP server k lokální síti. Zajde do serverovny, kde předtím přičinlivá uklízečka důkladně vytřela podlahu, zapojí vypadlý kabel do přepínače a ejhle, služba se rozběhla. Výborně, úkol splněn.

Přestože by tento přístup leckomu vyhovoval, opravdový síťový administrátor by se měl o chybě dozvědět dříve, než to zjistí uživatel. Ne vždy je to možné. Existují však přístupy a nástroje, které například testují dostupnost zařízení, sledují počty přenesených bytů, apod. Je to především starobylý, leč výkonný a stále nezastupitelný protokol ICMP (Internet Control Message Protocol) [15] pro síť IPv4. Pro komplexní správu síťových zdrojů se dnes hojně používá neméně starobylý systém SNMP (Simple Network Management Protocol) [19] a také systém pro monitorování toků NetFlow [7].

10.1.1 Koncepce a cíle správy sítě

Východním bodem pro účinnou správu sítě je zjištění aktuálního stavu sítě. K tomu potřebujeme informace o topologii sítě (jaká zařízení jsou zapojena, jaké mají síťové adresy), aktuální stav zařízení (přehled běžící rozhraní, procesů a poskytovaných služeb), statistické údaje o přenosu (počet přenesených bytů, paketů, broadcastových paketů). Pro řešení některých problémů na síti je někdy potřeba znát i obsah přenášených dat. Pomocí analyzátorů síťového provozu (softwarových či hardwarových) lze tato data načítat a analyzovat. Zjišťování infor-



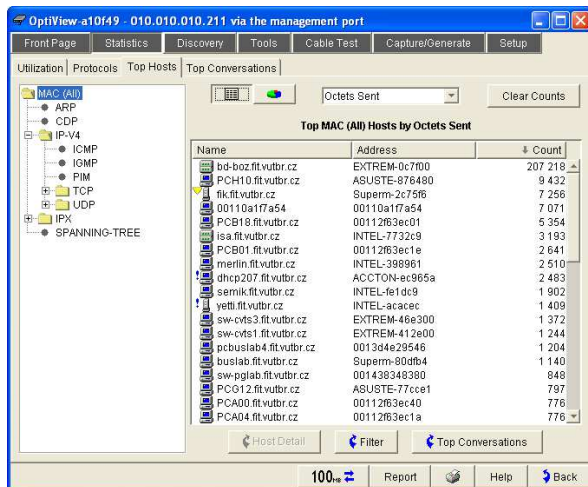
Obrázek 10.1: Příklad zobrazení vytiženosti linky

mací o síti lze rozdělit do dvou základních kategorií:

Monitorování sítě

Monitorováním sítě rozumíme zjišťování stavu síťových zařízení a služeb. Monitorování může být pasivní či aktivní. Při *pasivním monitorování* sledujeme probíhající komunikaci na síti a sbíráme z ní informace o stavu sítě a zařízeních na síti, dále informace o chybách či nedostupnosti zdrojů, případně pokusy o neoprávněný přístup k síťovým službám. Pasivní monitorování využívá především logovací informace aplikací a služeb (webového serveru, poštovního serveru, FTP serveru) sbírané u administrátora např. pomocí protokolu syslog [10]. Dále to mohou být nevyžádané asynchronní zprávy SNMP typu *traps*, o kterých se budeme více bavit v další kapitole, hlášky od aplikací zasílané např. přes email či SMS, data ze sond NetFlow a další.

U *aktivního monitorování* nečeká administrátor pouze na zprávy, které pošlou síťová zařízení a servery samy, ale aktivně se dotazuje na dostupnost linek, aplikací či síťových prvků. Jednoduchý a účinný způsob je například pomocí zpráv ICMP, dále to mohou být data získaná



Obrázek 10.2: Sledování přenosu dat na počítačích v síti

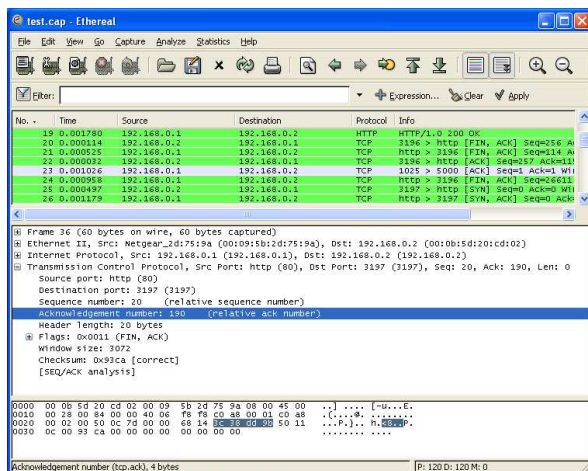
protokolem SNMP, či telnet na TCP port sledovaných služeb. Aplikace na aktivní monitorování většinou pracují v nekonečném cyklu. V případě, že aplikace či zařízení neodpovídá, zkusi dotazující se stanice navázat spojení opakovaně během daného časového intervalu. Pokud se ani při opakovaném dotazování neobnoví komunikace, je zařízení prohlášeno za nedostupné a správce sítě je informován formou nějaké zprávy (např. výpis na administrátorské konzoli, zaslání emailu, zprávy SMS apod.).

Analýza přenášovaných dat

Analýza dat je proces, který se zabývá podrobnějším zkoumáním a vyhodnocováním nasbíraných dat. Může využívat již dříve nasbíraná data nebo přímo sledovat procházející datové toky. Analýza zahrnuje nejen zkoumání hlaviček protokolů na vrstvě L2 a L3, ale dívá se také na obsah paketů až do vrstvy L7. Datový analyzátor obvykle naslouchá na sdíleném médiu (či zkoumá replikovaný provoz z jiného portu) a zobrazuje data z linky, kterou zkoumá.

Zachytávání a zpracování dat v reálném čase je při přenosových rychlostech v řádu stovek Mb/s prakticky nerealizovatelné úkol, s výjimkou specializovaných zařízení. Z tohoto důvodu většina analyzátorů data nejprve data nasnímá (data capturing) a teprve poté nad nimi provádí analýzu, která zahrnuje klasifikaci, dekodování a prezentaci dat. Výsledkem analýzy může být např. detekce a zobrazení obsahu ARP datagramů, které vysílá špatně nakonfigurovaný přepínač, nalezení paketů s privátními adresami na vnějším rozhraní firemní sítě či jiné problémy.

Pro sledování dat v reálném čase slouží nástroje, které zobrazují data, která se objeví na sledovaném síťovém rozhraní. Oblíbené jsou softwarové nástroje Wireshark (typu obrázek 10.3, Microsoft Network Monitor či tcpdump. Tyto nástroje obvykle umožňují i jednodu-



Obrázek 10.3: Analýza obsahu dat v nástroji Wireshark

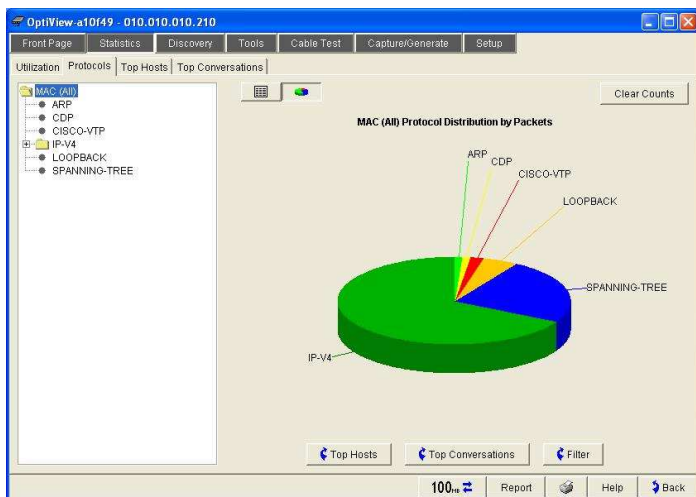
chou analýzu odchycených dat, např. interpretaci položek IP hlavičky, transportního paketu a mnohých aplikačních protokolů. Síťové analyzátoři zasahují výrazným způsobem do koncepte operačního systému (přidělování síťových prostředků) a bezpečnosti, proto je možné je spouštět pouze s právy administrátora systému. K programování aplikací, které komunikují přímo se síťovým rozhraním a čtou všechna příchozí data, lze použít síťové knihovny *libpcap* (pro Unix), viz kapitola 2.5.3, či *WinPcap* pro operační systém Windows.

Sledování dat v reálném čase je náročné zejména na procesorový výkon (analýza paketů, interpretace hlaviček) a v případě, že data ukládáme na disk, tak i na prostor. Je to zřejmé, když uvážíme kolik dat přijde každou sekundu po gigabitové lince. Na plně vytížené gigabitové lince můžeme za 10 minut nasbírat až 76 GB dat.

Obvykle nás nezajímá pouze aktuální obsah dat, ale také *dlouhodobé statistiky* získané analýzou provozu. Ty získáme tak, že z monitorovaných dat si vybíráme pouze hodnoty, které nás dlouhodobě zajímají (například počty přenesených bytů na dané lince), a ty ukládáme. V rámci daného časového okna pak můžeme zjistit například vytíženost aplikačních serverů během posledních 24 hodin, za týden, měsíc a podobně (například pomocí nástroje MRTG, viz obrázek 10.1).

Cíle analýzy a monitorování

Výsledkem analýzy monitorovaných dat jsou například statistické údaje ve formě grafů, které pomáhají detekovat slabá místa v síti, špičky v provozu datových spojů a jejich rozložení, vytíženost služeb apod. Údaje jsou důležité pro plánování dalších investic – např. zvýšení kapacity linky, rozdělení služby na více serverů, přesun rutinních přenosů a činností mimo hlavní provozní hodiny apod.



Obrázek 10.4: Využití přenosového pásma v síti různými protokoly

Pomocí síťových analyzátorů můžeme sledovat nejvíce komunikující stanice (tzv. Top Ten Hosts, viz např. obrázek 10.2) či protokolové statistiky - např. poměr paketů TCP/UDP/ICMP v provozu, poměr unicastových, broadcastových či multicastových rámců atd., viz obrázek 10.4. Tyto statistiky slouží například k detekci chyb na síti - Při enormním nárůstu broadcastových paketů zkoumáme, zda v lokální síti není smyčka, případně nějaká stanice s vadnou síťovou kartou.

Techniky monitorování sítě a analýzy dat pomáhají hledat odpovědi například na následující otázky týkající se správy sítě:

- Jaká je vytíženost jednotlivých linek, např. páteřních spojů, z hlediska časového (doba zátěžových špiček) a aplikačního (které typy požadavků převažují?)
- Jaký je objem přenášených dat, jaký je směr největších toků? Které uzly nejvíce komunikují? Jaké protokoly zabírají největší procento šířky přenosového pásma?
- Je komunikace uživatelů bezpečná? Používají uživatelé šifrovaná spojení?
- Co znamená náhlé zvýšení objemu přijímaných dat, např. přes víkend? Jedná se o cílený útok na síť? Jak můžeme detekovat a izolovat útočníka?
- Kde se nachází síťové zařízení (počítač, přepínač, apod.), které posílá do sítě chybná data a způsobuje výpadky?

10.2 Správa sítě na úrovni IP

Pro správu sítě na úrovni IP vrstvy modelu TCP/IP se používá protokol ICMP (Internet Control Management Protocol). Tento protokol je součástí implementace protokolu IP, tzn. je implementován v každém TCP/IP stacku síťových zařízení. Popis protokolu je v RFC 792 z roku 1981 [15].

Protokol ICMP používají počítače, směrovače či jiné síťové prvky pro předávání síťových informací, zejména chybových hlášek. Důvodem je, že protokol IP nezaručuje spolehlivé doručení. V případě zahození není uživatel informován o tom, že IP datagram nebyl doručen. ICMP plní úlohu jednoduché správy a řízení přenosu nad IP. Přestože jde o jednoduchý protokol obsahující několik málo příkazů, jeho pro IP sítě nezbytný. Protože se jedná o protokol pracující na vrstvě IP, musí ho podporovat všechna L3 zařízení – počítače, směrovače, L3 přepínače a další zařízení.

Většina z nás se s protokolem ICMP setkala. Typickým příkladem je situace, kdy zadáme chybnou adresu do webového prohlížeče. Pokud dotýčný server neexistuje nebo neposkytuje webové služby, ICMP protokol vrátí hlášku, kterou webový prohlížeč interpreтуje jako “nedostupný server” apod. Kromě komunikačních problémů (neznámá adresa, nefunkční služba), informuje ICMP také o ztrátách paketů, např. v důsledku směrování (neexistuje cesta k cíli), či zahlcení front na směrovači a zahození IP datagramu. Nástroje pro sledování dostupnosti síťových zařízení ping či traceroute využívají pro svou činnost protokolu ICMP, jak si ukážeme později.

10.2.1 Funkce a struktura ICMP

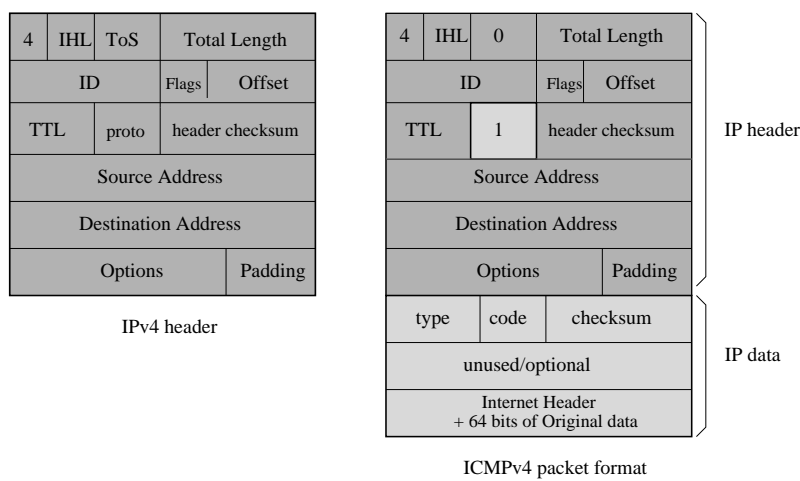
Protokol ICMP slouží k detekci chyb, řeší problémy s fragmentací IP datagramů, detekuje nedostupnost zařízení. Poskytuje také informace o zahlcení, umožňuje přesměrovat data při výpadku implicitní brány sítě či jednoduchým způsobem sleduje stav linky.

Struktura ICMP se příliš neliší od IP datagramu, viz obr. 10.5. Jedná se vlastně o speciální IP datagram, jehož tělo tvoří informace důležité pro ICMP. Typ protokolu je 1. Hlavička protokolu obsahuje typ ICMP zprávy, kód dané chyby, který závisí na typu, a prvních 64 bitů původního IP datagramu, na který ICMP reaguje. Typy zpráv ICMP jsou uvedeny v tabulce 10.1.

Zpráva	Typ	Popis
Destination Unreachable	3	Cílová síť je nedostupná
Time Exceeded Message	11	TTL dosáhlo hodnoty 0
Parameter Problem Message	12	chyba při zpracování IP hlavičky
Source Quench Message	4	přeplnění bufferu brány, kontrola zahlcení
Redirect Message	5	informace o přesměrování datagramu (brána)
Echo, Echo Reply	8,0	data posílaná pomocí Echo se vrátí
Timestamp, Timestamp Reply	13,14	posílání časového razítka
Info Request, Info Reply	15,16	zjišťování adresy počítače

Tabulka 10.1: Typy zpráv protokolu ICMP

Mezi nejvíce používané zprávy patří zpráva typu 3 **Destination Unreachable**, dále zprávy 11, která se generuje poté, co hodnota TTL IP datagramu klesne na 0 a datagram je zahozen,



Obrázek 10.5: Struktura ICMP

a zprávy 8 Echo. Zpráva 5 slouží ke změně implicitní brány poté, např. když se původní brána stane nedostupnou.

Nejvíce používané zprávy typu 3 **Destination Unreachable** určuje blíže kód zprávy ICMP. Seznam kódů je uveden v tabulce 10.2.

Typ	Kód	Popis	Možná příčina
3	1	destination network unreachable	chyby na lince k cíli
3	2	destination host unreachable	chyba komunikace počítače
3	3	destination protocol unreachable	nedostupný protokol
3	4	destination port unreachable	nedostupná služba
3	6	destination network unknown	chyba ve směrování
3	7	destination host unknown	chyba v cílové adrese

Tabulka 10.2: Seznam kódů ICMP typu 3

Podle typu kódu jsme schopni přesněji detekovat, o jakou chybu se jedná a určit, kde chybu hledat – zda se jedná o nedostupnost počítače na fyzické či linkové vrstvě vrstvě (ICMP s kódem 1, 2), zda není implementován požadovaný komunikační protokol (kód 3) či hledaná služba neběží (chyba 4). Chyby s kódem 6 a 7 se týkají směrování – směrovače na cestě nejsou schopny určit cestu, kam přeposlat došlý paket. Buď je chybná cílová adresa nebo nemá směrovač potřebné směrovací informace.

10.2.2 Aplikace využívající ICMP

Nejběžnějšími nástroji pro kontrolu síťové spojení založenými na ICMP jsou programy **ping**, **trace route** či **tracert** (ve Windows). Tyto programy zasílají zprávy ICMP (např. ping) nebo využívají zprávy ICMP (tracert).

Ping posílá zprávy typu 8 kódu 0 (Echo) na vzdálený počítač. Ten mu odpovídá pomocí typu 0 kódu 0 (Echo Reply).

Tracerouter posílá sekvenci stejných IP datagramů na cílový počítač. První datagram má nastavené TTL na hodnotu 1, druhý na hodnotu 2 atd. Pro každý odeslaný paket zapne traceroute časovač a čeká, na zprávu ICMP. Když n-tý datagram přijde na n-tý směrovač, TTL se sníží na 0. Směrovač zahodí datagram a pošle odeslateli zprávu ICMP, typ 11, kód 0 (Time Exceeded Message). Po přijetí této zprávy zjistí vysílající stanice čas přenosu (RTT, round-trip time) a IP adresu vzdáleného směrovače, který datagram zahodil.

10.3 Model OSI pro správu sítě

Jak jsme již řekli v úvodu této kapitoly, správa sítě je spojena s návrhem a rozšířením Internetu. Standardy pro správu sítě se objevily už v osmdesátých letech 20. století pod názvem OSI SMAE (System Management Application Entity). Hlavní funkcí systému SMAE bylo umožnit správci sítě vzdáleně spravovat síťové objekty – protokoly, přepínače, směrovače apod. – nezávisle na typu a architektuře zařízení či sítě.

Součástí definice systému SMAE byl popis rozhraní služeb pro správu sítě CMISE (Common Management Information Service Element) definovaný standardem OSI X.710 [13] a přenosový protokol CMIP (Common Management Information Protocol) definovaný standardem OSI X.711 [12]. Standardy CMISE a CMIP se staly základem pro standard IETF SNMP (Simple Network Management Protocol) [3], který patří dodnes k nejvíce rozšířeným systémům pro správu sítí.

10.3.1 Model FCAPS

Standard ITU-T M.3400 [14] definuje základní požadavky na správu sítě pomocí tzv. modelu FCAPS (Fault, Configuration, Accounting, Performance, Security). Následující popis vychází z [21].

Správa poruch (Fault Management) zajišťuje detekci, izolaci a opravu poruch. Porucha je obvykle charakterizována jako neobvyklý stav, který vyžaduje zásah správce. Příkladem může být fyzické přerušení linky, porucha síťové karty, atd. Většinou zjistíme nejprve důsledky poruch – pomalý přenos, špatná odezva, apod. Správa poruch zahrnuje logování událostí, detekce poruch a reakce.

Pokud se objeví v síti porucha, je důležité co nejrychleji provést následující akce:

- Určit, kde porucha přímo je.
- Izolovat poruchu v síti tak, aby ostatní části sítě mohli pracovat bez přerušení.
- Rekonfigurovat či upravit síť tak, aby to mělo minimální dopad na ty zařízení, kde k poruše nedošlo.

- Opravit či nahradit porouchané zařízení či komponenty sítě a nastavit počáteční bezchybový stav.

Během řešení poruchového stavu je důležité, aby uživatelé byli informováni o výpadku či omezení služeb. Vhodné je také odhadnout dobu řešení problému a odstranění poruchy.

Systémy pro monitorování poruch by měli také sloužit k izolaci poruch a diagnostice. Příkladem testů, které může systém pro správu poruch obsahovat, jsou například:

- test konektivity,
- test integrity dat,
- test odezvy,
- test funkčnosti, a další.

Účtování (Accounting Management) zajišťuje správu uživatelských účtů a poplatků za služby. Obsahuje specifikaci, logování a řízení přístupu uživatelů a zařízení k síťovým zdrojům. Definuje uživatelské kvóty, alokaci prostředků a další.

Správa účtování se týká například i interního účtování přenosů ve firmě tak, aby se kontrolovala oprávněnost nákladů na komunikační síť. Statistiky přenosů jednotlivých uživatelů lze využít i pro plánování sítě. Příkladem může být definice maximálního přenosu pro zaměstnance na bezdrátové síti. V případě překročení měsíčního limitu, mohou se další data označit nižší prioritou a přesunout do pomalejšího přenosového pásma.

Pro účely účtování je obvykle potřeba sbírat nejrůznější typy dat – jednak identifikační údaje (kdo) a jednak provozní údaje (kolik dat, odkud a kam, apod.). Například účtování zákazníka u poskytovatele internetového připojení bude obsahovat následující údaje:

- identifikaci uživatele (např. uživatelské jméno zadané při autentizaci při přístupu do sítě),
- cíl (identifikace cílového zařízení, kam uživatel přistupuje),
- počet paketů,
- úroveň zabezpečení,
- časové razítko,
- stav spojení (spojení proběhlo v pořádku, výpadky, apod.),
- použité zdroje.

Správa konfigurací (Configuration Management) umožňuje správcům sítě sledovat, které monitorované objekty jsou připojeny k síti a jaká je jejich konfigurace. Správa konfigurací přiděluje jména (identifikace) síťovým zařízením, definuje, jaký software má být na nich nainstalován a standardní konfigurace. Konfigurace objektů se mohou měnit během aktualizací či rozšiřování sítě, či v době uzávěrky a pravidelné údržby sítě.

V databázi konfigurací monitorovaných objektů by měly být základní informace o zařízení (typ, verze, firmware, procesor, paměť, rozšiřující karty), dále seznam nainstalovaného softwaru a verze, seznam běžících služeb, otevřených portů apod. Konfigurace by měla popisovat požadovaný stav chování konkrétního zařízení.

Správa konfigurací zahrnuje následující informace:

Konfigurační data zahrnují názvy zdrojů (koncový systém, směrovač, server) a jejich atributy (jméno, adresa, stav, verze softwaru). Konfigurační data se ukládají do databáze, což může být jednoduše strukturovaný seznam položek s jednou hodnotou (používá SNMP), relační či objektově orientovaná databáze.

Nastavení a modifikace atributů objektů. Řídící stanice by měla měnit nastavení atributů. Některé atribut objektů (například počet portů) jsou určeny pouze pro čtení, jiné (například směrovací tabulka, stav síťového rozhraní) mohou být modifikovány. S tím souvisí i zabezpečení přístupu k objektům.

Inicializace a přerušení síťové operace zahrnuje například nastavení a vzdálené spuštění webového serveru, nastavení filtrování na směrovači, konfigurace statického překladu NAT apod.

Distribuce software. Správa konfigurací by měla zajišťovat také distribuci softwaru (aplikací, operačních systémů či aktualizací) na koncové stanice či aktivní síťové prvky. S tím souvisí i správa verzí, zálohování softwaru i nastavení systémů a aplikací.

Správa výkonnosti (Performance Management) se zaměřuje na dvě oblasti – monitorování a řízení. Monitorování sleduje chování na síti, sbírá informace o stavu monitorovaných objektů, pak je analyzuje a vyhodnocuje. Řídící funkce slouží ke zlepšování výkonnosti sítě na základě zjištěných hodnot.

Na základě zjištěných informací poskytuje správa výkonnosti informace o průměrném zatížení sítě, odezvě sítě atd. Výstupy monitorování opět slouží pro účely údržby, plánování a rozvoje sítě. Lze z nich i zjistit, jaké části sítě tvoří slabé místo ve výkonnosti sítě a jak lze výkon vylepšit.

Mezi základní parametry, které slouží ke sledování výkonnosti sítě, patří:

Dostupnost (Availability) se vyjadřuje v procentech času, kdy je systém dostupný uživateli. Požadavky se liší podle důležitosti a složitosti aplikace, například požadavek na dostupnost webového serveru nebo systému pro rezervaci letenek bude asi jiný.

Dostupnost závisí na spolehlivosti jednotlivých síťových komponent (serveru, linek, aktivních prvků, apod.). Spolehlivost je pravděpodobnost, že daná komponenta bude provádět svou funkci po daný čas a za daných podmínek. Pravděpodobnost poruchy je vyjádřena hodnotou MTBF (Mean time between failures), což je střední dobou mezi dvěma poruchami. Pak můžeme dostupnost A vyjádřit jako poměr MTBF a MTTR (Mean time to repair, střední doba opravy) následujícím vztahem:

$$A = \frac{MTBF}{MTBF + MTTR}.$$

Odezva (Response Time) je doba, za kterou systém odpoví na daný vstup. U interaktivního systému to může být doba mezi úhodem na klávesnici a zobrazením výsledku na vzdálené konzoli. Doba odezvy se může pohybovat v řádě sekund (načítání webové stránky) nebo může být z pohledu člověka okamžitá (zlomky sekund).

Doba odezvy je se skládá z více položek jako jsou například zpoždění při vysílání (serialization delay), doba přenosu (propagation delay), zpoždění ve výstupních frontách (queueing delay) či zpoždění při zpracování na zařízeních (forwarding/processing delay). Některá zpoždění mohou být fixní (například zpoždění při vysílání, při přenosu či překódování), jiná jsou proměnlivá a závisí na stavu sítě (zpoždění ve frontách, zpracování, komprese, apod.). Tato zpoždění potom určují výslednou dobu odezvy.

Propustnost (Throughput) udává rychlost, při které se objevují události. Obecně se propustnost nemusí počítat pouze v bitech za sekundu (bps) či paketech za sekundu (pps), ale může zahrnovat i aplikačně orientovanou metriku, která udává například počet zpracování určitých transakcí za daný čas, počet navázaných spojení pro danou aplikaci za čas či počet navázaných telefonních spojení za jednotku času.

Využití (Utilization) je parametr, která vyjadřuje v procentech poměr využívané kapacity zdroje (například přenosového pásma) vůči teoretické hodnotě. Sledování využití nám pomáhá určit potencionální slabé místo systému, sledovat historii a chování sledovaných objektů a stanovit, zda jejich využití je očekávané a optimální. Výsledky sledování využití objektů slouží k rozložení zdrojů, plánování rozšíření či zálohování.

Výsledkem měření výkonnosti systému mohou být například následující výstupy:

- matice nejvíce komunikujících koncových stanic (source x destination),
- histogram typů paketů (např. TCP, UDP, ICMP, DNS, HTTP, apod.),
- histogram velikostí paketů,
- histogram zpoždění paketů,
- histogram přenesených paketů, apod.

Jak uvidíme v následující kapitole, tyto údaje můžeme získávat například pomocí systému SNMP a jeho rozšíření RMON.

Bezpečnost (Security Management) se zaměřuje na řízení přístupu k síťovým zdrojům podle specifikované bezpečnostní politiky. Zahrnuje distribuci klíčů a certifikátů, používání firewallu a další.

Řízení bezpečnosti zahrnuje jednak zabezpečení koncových zařízení (computer security) a jednak bezpečnost síťových prvků a síťové infrastruktury (network security). Bezpečnost se řídí tzv. bezpečnostní politikou (security policy), což je formální dokument, který definuje pravidla kdo, co a proti čemu má chránit. Doporučení pro bezpečnostní politiku upravuje například RFC 2196 [9] či standard ISO 27002.

Bezpečnostní politika obvykle stanoví i hrozby, proti kterým zajišťuje bezpečnost zdrojů. Hrozby mohou být pasivní (odposlechy, analýza provozu) či aktivní (přerušení služby, útoky DoS či DDoS, průnik do systému, modifikace dat).

Systém pro správu bezpečnosti sbírá provozní informace a sleduje chování na monitorovaných objektech. V případě, že se objeví podezřelá aktivity či události, zareaguje. Reakcí může být hlášení administrátorovi, což provádí systémy IDS (Intrusion Detection Systems), nebo může systém nastavit definovaná protiopatření, například

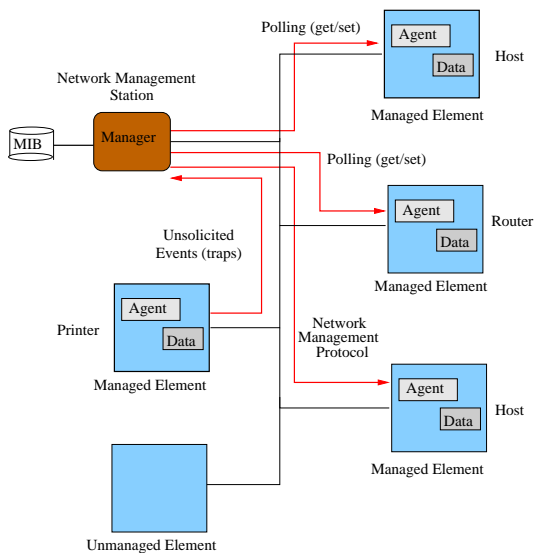
filtrování útočnickova provozu, což provádí zařízení typu IPS (Intrusion Prevention Systems).

Bezpečnostní systém může využívat například následující provozní informace:

- sledování událostí (event logging),
- výstupy bezpečnostního auditu (security audit trails),
- hlášky o porušení bezpečnostních pravidel (security violation reports),
- sledování logovacích informací (maintaining and examining security logs) a další.

10.3.2 Systém pro správu sítě

Systém pro správu sítě (Network Management System) tvoří soubor nástrojů a utilit pro monitorování sítě a její řízení. Protože jsou monitorované objekty sítě obvykle rozmístěny na vzdálených místech, je systém správy distribuovaný, tzn. zahrnuje jednak síťové monitorované zařízení NME (Network Management Entity) a jednak řídicí systém správy nazývaný NMS (Network Management System), viz obrázek 10.6.



Obrázek 10.6: Architektura správy sítě

Řídící jednotku NMS (Managing Entity) tvoří aplikace, která běží na centrálním serveru a která řídí sběr, zpracování, analýzu a prezentaci dat od zařízení z monitorované sítě.

Monitorované zařízení (Managed Device) je síťové zařízení (počítač, směrovač, most, rozbočovač, tiskárna, modem), na němž probíhá sběr provozních dat a statistik. Zařízení se skládá z tzv. monitorovaných objektů (managed objects), což jsou konkrétní komponenty zařízení (například operační systém, síťové karty, výstupní porty apod.) s množinou konfiguračních parametrů.

Monitorované objekty obsahují informace, které jsou ukládány lokálně do databáze MIB (Management Information Base). Data z MIB sbírá pomocí přenosového protokolu (Network Management Protocol) řídicí jednotka NMS.

Každé monitorované zařízení obsahuje agenta (klienta) (Network Management Agent), což je proces, který komunikuje s řídicí jednotkou. Podle příkazů NMS vykonává v monitorovaném zařízení lokální akce (například čtení dat, zápis, reset čítačů, nastavení stavu, apod.).

Protokol pro správu sítě (Network Management Protocol) je protokol pro komunikaci mezi řídicí jednotkou a monitorovanými zařízeními. Umožňuje zjišťovat stav monitorované jednotky a provádět akce prostřednictvím lokálního agenta pro správu sítě.

Komunikace mezi řídicí stanicí a monitorovanými zařízeními může být dvojího typu. Nejčastěji se jedná o tzv. *vyzývání (polling)*, kdy centrální řídicí stanice pošle požadavek na agenta, kde specifikuje, aby provedl danou akci (například poslal stav daného čítače). Agent vyhledá informaci v lokální databázi MIB a odešle ji řídicí stanici. Řídicí systém se může pravidelně dotazovat monitorovaných zařízení, generovat průběžná hlášení o stavu sítě a zobrazovat je pro potřeby správy sítě. Toto je nejčastější způsob komunikace.

Druhou možností je *hlášení událostí (event reporting)*, což je asynchronní komunikace, kterou zahajuje monitorované zařízení zpravidla v okamžiku dosažení nějaké neočekávané události (změna stavu rozhraní, chyba, překročení maximálního počtu přenesených bytů, apod.).

Nasazení obou typů předávání zpráv se liší. U telekomunikačních systémů je běžnější hlášení událostí, naproti tomu správa nad IP využívá spíše vyzývání.

10.4 Architektura SNMP

Správa sítí vytvořených nad protokolovým profilem TCP/IP byly z počátku založena pouze na protokolu ICMP. S rostoucím počtem uzlů v síti však nebylo možné kontrolovat stav zařízení jednoduchými dotazy ICMP a začalo se uvažovat o návrhu systému pro správu sítí.

První začátkem byl protokol SGMP (Simple Gateway Monitoring Protocol) [8] vydaný roku 1987. Z něho se po krátké době (v roce 1988) vyvinul protokol SNMP (Simple Network Management Protocol) [2]. Původně měl být použit jako přechodný mechanismus pro síť TCP/IP, než se přejde na skutečně propracovaný ISO OSI model se správou sítí pomocí ISO CMIS/CMIP [6]. Z toho důvodu Internet Architecture Board (IAB) doporučil navrhnout protokol SNMP jako velmi jednoduchý a snadno implementovatelný. Protože "přechodový stav" trvá až dodnes, není překvapením, že podporu SNMP můžeme najít na téměř všech aktivních prvcích.

Architekturu systému SNMP tvoří čtyři základní prvky, viz obrázek 10.7.

Řídicí stanice NMS (Network Management Station). Jedná se o serverovou aplikaci, kterou tvoří nástroje pro sběr a analýzu monitorovaných dat, ukládání statistik a pre-

- **trap**: agent pomocí příkazu **trap** oznamuje řídící stanici, že nastala nějaká důležitá událost na monitorovaném zařízení

Současná verze pro systému SNMP je popsán standardy RFC 1155 [20], RFC 1213 [18] a RFC 1157 [3], které definují strukturu monitorovaných objektů, databázi standardních objektů a přenosový protokol pro přenos objektů. V následující části si popíšeme tyto standardy.

10.4.1 Popis struktury a identifikace monitorovaných objektů

Standard RFC 1155 [20] popisuje strukturu a identifikaci monitorovaných dat (objektů) vyjádřených pomocí jazyka SMI (Structure Management Information). Jazyk SMI definuje pravidla, jak vytvářet objekty SNMP popsané notací ASN.1 (Abstract Syntax Notation) [1].

Jazyk ASN.1 se používá pro popis abstraktních datových struktur a datových typů, které se vyměňují komunikačními protokoly. Použití notace ASN.1 umožňuje komunikovat se zařízeními od různých výrobců, s různou architekturou a s různými operačními systémy, které mohou přijatá data různě interpretovat. ASN.1 proto definuje abstraktní datové struktury, kterým pak přiřazuje konkrétní interpretaci na daném zařízení. SNMP využívá ASN.1 proto, aby hodnoty, které generuje monitorované zařízení, byly stejně interpretovány řídící stanicí.

Proto se každý objekt SNMP popisuje prostřednictvím ASN.1. Každý objekt popisovaný pomocí ASN.1 má své *jméno*, *syntax* a *kódování*, které se použije při přenosu po síti.

Jméno objektu SNMP je dáno jednoznačným identifikátorem OID.

Syntax objektu definuje abstraktní datový typ spojený s objektem, například typ INTEGER, OCTET STRING (základní datové typy ASN.1). Důležitým datovým typem je také OBJECT IDENTIFIER, který obsahuje identifikátor OID pro daný objekt.

Jazyk SMI definuje pomocí ASN.1 nové datové typy užitečné pro správu sítí, např. NetworkAddress, IpAddress či Counter. SMI využívá i rozšířené datové typy, např. SEQUENCE pro definici seznamu či OBJECT-TYPE pro vlastní definici objektu.

Kódování určuje, jak jsou instance objektu reprezentovány při přenosu po síti. SMI používá pro kódování standard BER (Basic Encoding Rules) [11].

Pomocí jazyka SMI můžeme tedy definovat monitorované objekty. Běžné objekty jsou standardizovány v dokumentech RFC. Můžeme si však vytvářet i popisy vlastních objektů, kterým přiřadíme identifikátor tak, aby byly jednoznačně určené a mohly se obecně používat při správě sítí.

Základní datové typy ASN.1 použité pro tvorbu informací o monitorovaných objektech, jsou uvedeny v tabulce 10.3. Kromě těchto základních datových typů můžeme vytvářet rozšířené datové typy, viz tabulka 10.4.

Na základě těchto datových typů si můžeme vytvořit vlastní objekt s položkami, které o něm chceme sledovat. Příkladem může být čítač doručených IP datagramů:

```
ipInDelivers OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS  read-only
    STATUS      current
    DESCRIPTION "The total number of input datagrams successfully
```

Datový typ	Popis
INTEGER	32-bitové celé číslo definované v ASN.1
OCTET STRING	binární či textový řetězec (až 64kB) definovaný ASN.1
OBJECT IDENTIFIER	přiřazeno ASN.1
Integer32	32-bitové celé číslo
Unsigned32	kladné 32-bitový celé číslo
IPAddress	32-bitová IP adresa, síťový formát
NetworkAddress	pro reprezentaci jiných typů adres
Counter32	32-bitový čítač, po přetečení se nastaví na 0
Counter64	64-bitový čítač
Gauge32	32-bitový čítač, po přetečení uchová max. hodnotu až do resetu
TimeTicks	čas měřený v setinách sekundy od dané události
Opaque	neinterpretovaný řetězec podle ASN.1

Tabulka 10.3: Základní datové typy

Datový typ	Popis
IMPORTS	položky definované v jiném modulu MIB
MODULE-IDENTITY	modul s administrativními informacemi (kontakt, revize)
SEQUENCE	seznam základních i rozšířených datových typů ASN.1
OBJECT-TYPE	datový typ (syntax), přístup k objektu (čtení, zápis, vytvoření), stav (aktuální, platný, zastaralý) a textový popis objekt
MODULE-ENTITY	umožňuje spojit příbuzné objekty do jednoho modulu
NOTIFICATION-TYPE	informace týkající se zpráv SNMPv2-Trap a InformationRequest
MODULE-COMPLIANCE	definuje množiny monitorovaných objektů v modulu, které musí agent implementovat
AGENT-CAPABILITIES	specifikuje podporované možnosti agenta SNMP

Tabulka 10.4: Rozšířené datové typy

```

        delivered to IP user-protocol (includiong ICMP)."
 ::= { ip 9 }

```

Výše uvedený objekt `ipInDelivers` obsahuje 32-bitový čítač, který je určený pouze pro čtení, je platný a obsahuje celkový počet vstupních datagramů (včetně ICMP), které jsou úspěšně doručeny procesu na zpracování IP. Obecná definice objektu je dána následující syntaxí:

```

<name> OBJECT-TYPE
    SYNTAX <datatype>
    ACCESS <read-only|read-write|write-only|not-accessible|accessible-for-notify>
    STATUS <mandatory, optional, obsolete, current, deprecated>
    DESCRIPTION "textový popis objektu"
    ::= {<Unique OID>}

```

Jak vidíme, definice objektu musí obsahovat minimálně čtyři parametry popisující objekt:

- *Syntax* definuje datový typ objektu, viz tabulky 10.3 a 10.4.
- *Max-Access* určuje, jak k proměnné přistupovat – zda jsou její hodnoty pouze pro čtení, či lze do ní zapisovat nebo je nedostupná.

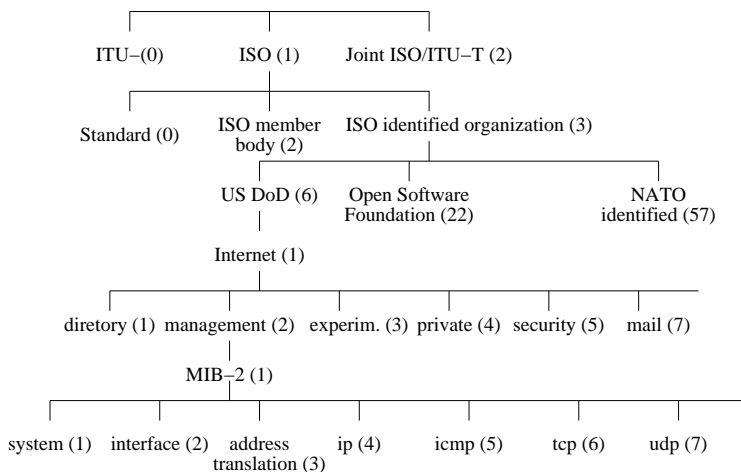
- *Status* určuje stav proměnné, tj. zda je povinná, volitelná, platná, zastaralá, či neplatná.
- *Description* je textový popis v ASCII popisující použití objektu.

Standard RFC 1155 také popisuje způsob identifikace objektů, tzn. jak se na objekty a jejich položky můžeme odkazovat. Je to určitá forma adresování (na aplikační vrstvě). Jméno (adresu) monitorovaného objektu v systému SNMP tvoří identifikátor OID (Object ID), který má číselnou nebo textovou podobu. Příkladem je adresa OID pro objekt "popis systém" (sysDescr):

číselná podoba: 1.3.6.1.2.1.1.1

textová podoba: iso.org.dod.internet.mgmt.mib-2.system.sysDescr

Identifikátory jsou strukturovány hierarchicky (podobně jako například IP adresy), viz obrázek 10.8. Na obrázku vidíme, že objekty na Internetu mají společný prefix 1.3.6.1.

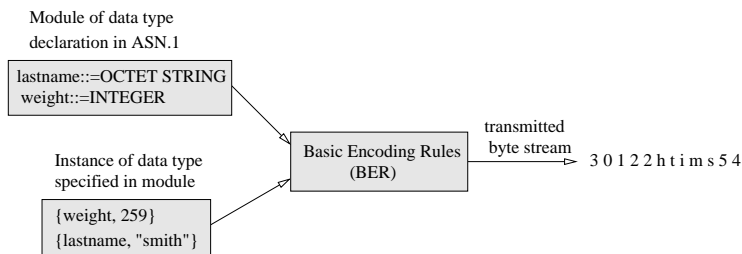


Obrázek 10.8: Identifikace objektů v MIB-2

Hodnota objektu SNMP se pro přenos po síti zapisuje do binárního formátu pomocí kódování BER (Basic Encoding Rules) [11]. BER reprezentuje přenášená data v oktetech. Každá přenášená hodnota se zapisuje pomocí struktury TLV (Type–Length–Value), která slouží nejen k přenosu vlastní hodnoty, ale i k identifikaci obsahu. Přijímající stanice pak přesně ví, jakého typu jsou přijatá data a kolik paměti vyhradit pro jejich uložení. Příklad kódování objektu zapsaného v ASN.1 je na obrázku 10.9.

10.4.2 Databáze monitorovaných objektů

RFC 1213 [18] definuje databázi jednotlivých objektů a skupiny monitorovaných, tzv. MIB-II (Management Information Base). Původní databáze MIB [17] obsahovala osm skupin s celkem 114 monitorovanými objekty. Základní skupiny objektů MIB byly **System**, **Interface**,



Obrázek 10.9: Příklad kódování dat pomocí BER

Address Translation, IP, ICMP, TCP, UDP, EGP. Z názvů skupiny vidíme, které informace jsou pro správu sítě nejdůležitější – jde převážně o data popisující přenos základních protokolů (IP, ICMP, TCP, UDP), dále stav směrovacích protokolů (EGP), směrovací tabulky (včetně tabulek ARP, viz skupina Address Translation) a informace stavu o operačním systému (System) a síťových rozhraní systému (Interface).

Původní databáze MIB byla na základě dalších požadavků upravena a rozšířena. Rozšířená verze se nazývá MIB-II obsahuje více skupin a objektů. Oproti MIB-I přibyla skupina **Transmission**, která zohledňuje různé typy přenosových médií připojených k zařízení, a skupina **SNMP Group**, která obsahuje statistické informace. Deset základních skupin objektů v MIB-II je uvedeno v tabulce 10.5.

Jméno podstromu	OID	Popis
system	1.3.6.1.2.1.1	operační systém: jméno OS, systémový čas, kontakt
interface	1.3.6.1.2.1.2	stav síťového rozhraní
at	1.3.6.1.2.1.3	překlad adres (address translation) - nepoužívá se
ip	1.3.6.1.2.1.4	IP adresa, směrovací informace
icmp	1.3.6.1.2.1.5	sleduje chyby ICMP
tcp	1.3.6.1.2.1.6	stav spojení TCP (closed, listen, synSent)
udp	1.3.6.1.2.1.7	statistiky UDP - příšle/odešlé pakety
egp	1.3.6.1.2.1.8	statistiky EGP
transmission	1.3.6.1.2.1.10	objekty závislé na přenosovém médiu
snmp	1.3.6.1.2.1.11	počet vyslaných/přijatých SNMP paketů

Tabulka 10.5: Skupiny objektů definovaných v MIB-2

Každá skupina obsahuje množinu objektů, které obsahují další proměnné. Například skupina **system** obsahuje objekty uvedené v tabulce 10.6. V tabulce můžeme vidět objekty, které popisují daný systém. Objekty obsahují konkrétní proměnné (například sysDescr, sysUpTime, apod.), případně identifikátor OID. Hodnoty typu sysDescr, sysObjectID, sysUpTime a sysService jsou určeny pouze pro čtení, tzn. správce sítě do nich nemůže prostřednictvím protokolu SNMP zasahovat. Proměnné typu sysContact, sysName či sysLocation jsou určeny pro čtení i pro zápis, tzn. je možné nastavit jejich hodnotu příkazem **set**.

Mezi nejčastěji sledované objekty patří objekty ze skupiny **interface** a **ip**. Skupina **interface** obsahuje informace o fyzických rozhraních zařízení, včetně nastavené konfigurace či statistiky

Object	Syntax	Access	Description
sysDescr	DisplayString	RO	A description of the entity, such as hardware, operating system, etc.
sysObjectID	OBJECT IDENTIFIER	RO	The vendor's authoritative identification of the network management subsystem contained in the entity
sysUpTime	TimeTicks	RO	The time since the network management portion of the system was last reinitialized
sysContact	DisplayString	RW	The identification and contact information of the contact person for the managed node
sysName	DisplayString	RW	An administratively assigned name for this managed node
sysLocation	DisplayString	RW	The physical location of this node
sysService	INTEGER	RO	A value that indicated the set of services this entity primarily offers.

Tabulka 10.6: Příklad skupiny objektů `system`

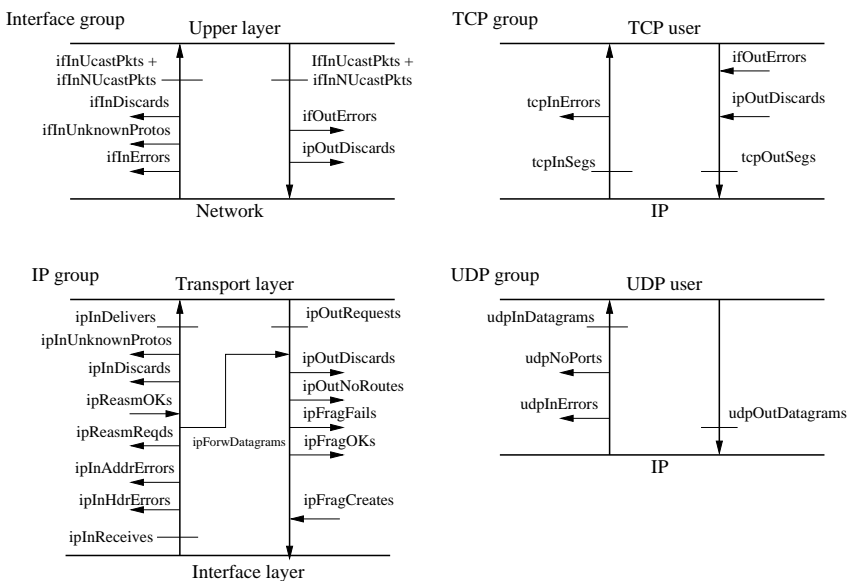
událostí, které se objevily na daném rozhraní, viz další příklad.

```
snmpwalk -v 1 -c public host.fit.vutbr.cz interfaces
IF-MIB::ifNumber.0 = INTEGER: 12
IF-MIB::ifIndex.1 = INTEGER: 1
IF-MIB::ifDescr.1 = STRING: em0
IF-MIB::ifType.1 = INTEGER: ethernetCsmacd(6)
IF-MIB::ifMtu.1 = INTEGER: 1500
IF-MIB::ifSpeed.1 = Gauge32: 1000000000
IF-MIB::ifPhysAddress.1 = STRING: 0:1c:c0:6c:61:60
IF-MIB::ifAdminStatus.1 = INTEGER: up(1)
IF-MIB::ifOperStatus.1 = INTEGER: up(1)
IF-MIB::ifInOctets.1 = Counter32: 2383522562
IF-MIB::ifInUcastPkts.1 = Counter32: 289122241
IF-MIB::ifInNUcastPkts.1 = Counter32: 0
IF-MIB::ifInDiscards.1 = Counter32: 0
...
IF-MIB::ifInErrors.1 = Counter32: 0
IF-MIB::ifInErrors.2 = Counter32: 0
IF-MIB::ifOutOctets.1 = Counter32: 122577245
IF-MIB::ifOutOctets.2 = Counter32: 0
IF-MIB::ifOutUcastPkts.1 = Counter32: 101179266
IF-MIB::ifOutUcastPkts.2 = Counter32: 0
```

Například výše uvedený příklad vypisuje údaje o rozhraních na zařízení `host.fit.vutbr.cz`. Toto zařízení má 12 rozhraní, z nichž první je rozhraní č.1 s názvem `em0`. Vidíme, že jde o ethernetovou síťovou kartu s rychlostí 1 Gb/s, MTU 1500 B a fyzickou adresou `0:1c:c0:6c:61:60`. Rozhraní je ve stavu `up`. Doposud tam bylo přijato 2,383 miliard oktetů (bytů) v celkem 289 mil. unicastových paketů. Dále vidím, že na rozhraní se neobjevily žádné chybné pakety. Můžeme též vidět počet odchozích bytů a paketů.

Z výše uvedeného příkladu už asi tušíme, v čem je systém monitorování objektů SNMP užitečný. Pokud si pravidelně zjišťujeme statistiky na daných rozhraních, může detekovat chyby v komunikaci, nadměrný přenos dat či nefunkční rozhraní.

Protože každý objekt obsahuje spoustu hodnot, které mají na první pohled podobný název, je důležité vědět, jak se které hodnoty generují. Význam a vztahy hodnot objektů v SNMP můžeme popsat speciálním popisem, kterému říkáme případový diagram (Case Diagram). Tyto diagramy slouží k upřesnění významu a jsou také zdrojem pro implementaci objektů MIB. Na obrázku 10.10 vidíme případové diagramy skupin `interface`, `ip`, `tcp` a `udp`. Diagramy znázorňují, jak se počítají data při zpracování na zařízení. Například ve sku-



Obrázek 10.10: Případové diagramy skupin `interface`, `ip`, `tcp` a `udp`

pině IP vidíme, že při vstupu se počítá celkový počet přijatých IP datagramů. Dále jsou tam čítače chyb (`ipInHdrErrors`, `ipInAddrErrors`, `ipInDiscards`, `ipInUnknownProtos`). Tyto hodnoty se odečítají od původního čítače `ipInReceives` (šipka doleva znamená odečítání), naopak se připočítávají pakety, které byly úspěšně defragmentovány (`ipReasmOKs`). Celkově pak přichází do transportní vrstvy počet datagramů popsanych hodnotou `ipInDelivers`.

Databáze MIB je strukturovaná hierarchicky ve formě stromu, kde listy jsou konkrétní monitorované objekty, viz obr. 10.8. Tyto objekty jsou identifikovány pomocí číselným identifikátorem OID (Object Identifier).

Databáze objektů MIB není jen jedna konkrétní databáze. Máme například databázi objektů MIB-I a MIB-II. Tyto dvě databáze obsahují objekty standardizované dokumenty IETF RFC. Některé databáze MIB jsou privátní. Většina výrobců síťového hardware a software si pro své konkrétní potřeby definují vlastní privátní podstromy MIB, kde jsou popsány objekty a atributy potřebné pro správu zařízení těchto zařízení. Většinou jde o rozšíření oproti MIB-I

a MIB-II. Rozšiřující privátní databáze MIB jsou obvykle volně k dispozici tak, aby aplikace pro správu sítí mohly s těmito daty pracovat. Přehled privátních databází MIB lze získat například na webové stránce <http://www.oidview.com/mibs/detail.html>¹.

10.4.3 Protokol SNMP

Standard RFC 1157 [3] popisuje protokol SNMP pro práci s monitorovanými objekty. Jak jsem již uvedl na začátku kapitoly 10.4, protokol se vyvinul z protokolu SGMP (Simple Gateway Management Protocol). Jedná se o aplikační protokol nad TCP/IP přenášený transportním protokolem UDP. Jeho hlavní úlohou je komunikace mezi řídicí stanicí a agentem na monitorovaném zařízení.

Komunikační protokol SNMP je protokol typu dotaz-odpověď (request-response). Protože pracuje nad UDP, jedná se o nestavový protokol, tzn. každá výměna dat typu dotaz-odpověď je nezávislá.

Komunikace probíhá tak, že řídicí jednotka pošle dotaz agentovi SNMP. Ten načte dotaz, vykoná akci a pošle výsledek operace, např. výpis či změnu hodnot objektu MIB v zařízení. Jak jsme si již řekli, agent umí také vysílat nevyžádaných zprávy typu **trap** směrem k řídicí jednotce. Tyto zprávy se používají k oznámení nějaké výjimečné situace při změně hodnot v MIB objektech, např. pád síťového rozhraní, zaplnění front atd.

Jméno zprávy	Hodnota	Popis
coldStart	0	reboot, všechny řídicí proměnné se resetují
warmStart	1	agent se znovu inicializoval, proměnné zůstávají
linkDown	2	síťové rozhraní nedostupné
linkUp	3	síťové rozhraní opět v provozu
authenticationFailure	4	nepovolený přístup
egpNeighborLoss	5	sousední EGP nedostupný
enterpriseSpecific	6	specifikováno výrobcem zařízení

Tabulka 10.7: Asynchronní zprávy typu **trap**

Obvyklá činnost systému SNMP je taková, že řídicí stanice v pravidelných intervalech (např. 5 minut) sbírá provozní informace z monitorovaných zařízení, vyhodnocuje je a ukládá do databáze. Agent se tedy po většinu času chová jako pasivní účastník komunikace, který pouze čeká na přicházející žádosti. Jedná se tedy o komunikaci typu vyzývání (polling). Teprve při výskytu mimořádných událostí posílá agent aktivně zprávy typu **trap**. Příklad události **trap** je v tabulce 10.7.

První verze protokolu SNMP byla definována v roce 1988 [2], později aktualizovaná RFC 1157 [3]. SNMPv1 definuje pět základních příkazů: **Get-Request**, **Get-Next-Request**, **Set-Request**, **Get-Response** a **Set**. Další verze SNMPv2 přidala další dva příkazy: **Get-Bulk-Request** a **Inform-Request**[4], viz tabulka 10.8. Platnou verzi protokolu SNMPv2 popisuje standard RFC 3416 z roku 2002 [19].

Již od počátku měl protokol SNMP velký problém týkající se zabezpečení. Veškeré hodnoty z monitorovaných objektů se přenášely v otevřené textové podobě kódované pro přenos metodu BER (Basic Encoding Rules), viz dále. Kdokoliv mohl data odposlechnout, kdokoliv si je mohl vyžádat. První náznak autentizace přináší SNMPv2c, který definuje heslo (community

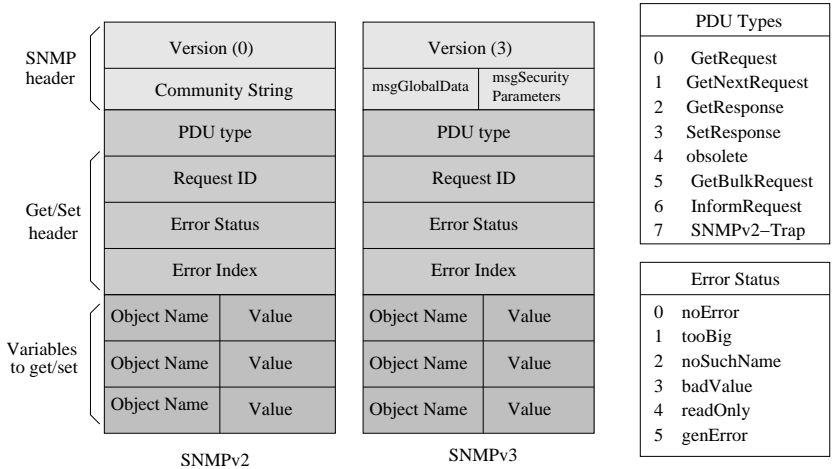
¹Jde o webové stránky nástroje OidView pro získávání a analýzu dat SNMP, březen 2013

Příkaz	Typ	Směr	Popis
get	0	manager-to-agent	získání hodnoty jednoho či více objektů MIB
get-next	1	manager-to-agent	hodnota dalšího objektu v seznamu/tabulce
get-response	2	agent-to-manager, manager-to-manager	vrací odpověď na předchozí dotaz
set	3	manager-to-agent	nastaví hodnotu jednoho či více objektů MIB
get-bulk	5	manager-to-agent	získání bloku dat
inform	6	manager-to-manager	informace o vzdálené řídicí jednotce o přístup- ných hodnotách v MIB
snmpv2-trap	7	agent-to-manager	informace o vyjimečné události

Tabulka 10.8: Příkazy protokolu SNMP verze 2

string) pro přístup k objektům. Nicméně heslo bylo možné odchytit během přenosu, neboť samotný protokol nebyl nijak chráněn a heslo bylo přenášeno v otevřené podobě. Teprve od verze SNMPv3 definované v roce 2002 standardem RFC 3410 [5], které definuje bezpečnostní mechanismus zajišťující autentizaci odesílatele a integritu dat pomocí algoritmu HMAC-MD5 a HMAC-SHA. Šifrování paketů (privátnost) protokol SNMP nezajišťuje.

Formát paketů SNMP je na obrázku 10.11. Na obrázku můžeme vidět strukturu paketů



Obrázek 10.11: Formát paketů SNMPv2 a SNMPv3

SNMPv2 a SNMPv3. Hlavička obsahuje verzi protokolu a autentizační údaje (buď řetězec `community string` nebo autentizační údaje HMAC-MD5/SHA u SNMPv3). Dále popisuje typ PDU (GetRequest, GetNextRequest, atd.) a hodnoty objektů, které načítá či nastavuje.

Pro přenos hodnot se používá zejména vyzývání (polling), tj. posílání požadavků od řídicí stanice k monitorovaným agentům. Důležitou otázkou administrace sítě je, jak zvolit frekvenci

načítání dat. Předpokládejme, že chceme monitorovat v síti N zařízení. Pokud označíme proměnnou Δ průměrný čas na zpracování jednoho dotazu, pak maximální čas dotazování T je dán vztahem

$$N \times \Delta \leq T$$

Tento vztah nám říká, že počet monitorovaných stanic a doba zpracování dotazu od jedné stanice musí být menší než frekvence zasílání dotazů.

Hodnota proměnné Δ je ovlivněna mnoha faktory: časem, po který se generuje dotaz, dobou přenosu na síti, zpracování dotazu v agentovi, generováním odpovědi na straně agenta, přenosem k řídicí stanici a zpracováním odpovědi v řídicí stanici.

Příklad 10.1. *Uvažujme síť LAN, kde chceme sledovat zařízením jednou za 15 minut. Uvažujme čas zpracování na straně agenta i klienta max. 100 ms a zpoždění při jednom přenosu 1 ms. Celkové zpoždění pak bude 202 ms. Počet zařízení, které jsme schopni na dané síti v intervalu 15 minut sledovat, je*

$$N \leq \frac{T}{\Delta} = \frac{15}{0.202} \approx 4,450.$$

Pokud bychom se chtěli dotazovat každých pět minut, počet sledovaných zařízení může být maximálně 1,485.

Z výše uvedeného příkladu můžeme vidět, že efektivní činnosti systému SNMP ovlivňují čtyři kritické parametry: počet monitorovaných agentů, frekvence dotazování, doba zpracování dotazu a odpovědi a celkové zpoždění přenosu na síti. SNMP tedy není příliš vhodné pro sledování rozsáhlé sítě a pro přenos velkých objemů dat. Dalším omezením SNMP je, že je přenášeno přes UDP, tj. jedná se o nespolehlivý přenos. Toto je kritické například při ztrátě zpráva typu **traps**, kdy si agent neověřuje, zda řídicí stanice zprávu přijala. Proto se zprávy **traps** nedají spolehlivě použít pro sledování výpadků. Na tyto omezení SNMP reaguje rozšíření zvané RMON, o kterém si povíme v další kapitole.

10.5 RMON – Remote Monitoring

Jak jsme si ukázali v předchozím příkladu, intenzivní monitorování sítě pomocí SNMP může významným způsobem vyčítat či přetížít procesory aktivních síťových zařízení a také zaplnit přenosové pásmo. Představme si síť ISP tvořenou desítkami a stovkami zařízení, ze kterých každých pět minut vyčítá řídicí stanice NMS provozní informace. Všechny tyto informace se přenášejí do jednoho místa, kde se ukládají, analyzují a vyhodnocují. Tento proces probíhá opakovaně.

Nabízí se otázka – je nutné přenášet všechna data do jednoho místa a tam je vyhodnotit? Není možné získat přenosové údaje a vyhodnotit je lokálně v podsíti a na řídicí stanici přenést až zpracované statistiky?

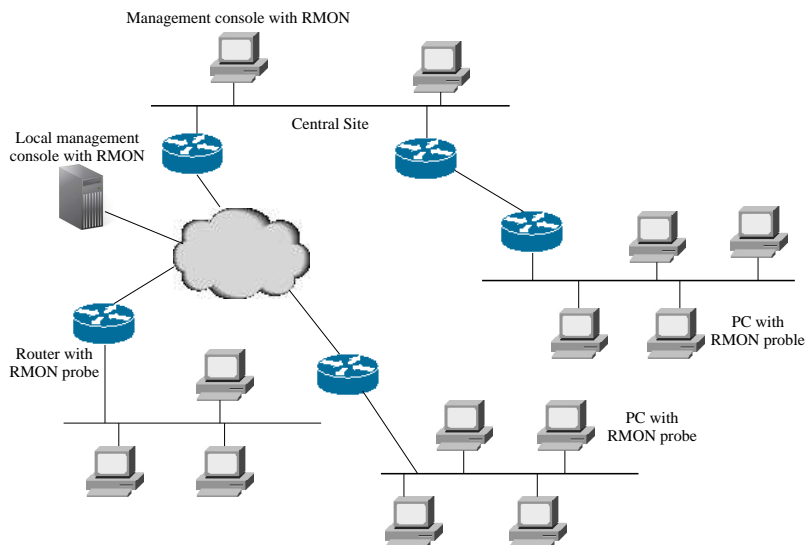
Je tady ještě druhá stránka věci – pomocí databáze MIB-II může řídicí stanice získat pouze informace, které jsou lokální pro monitorované zařízení. Pokud máme síť LAN, ve které se nachází desítky zařízení, může se řídicí stanice NMS dozvědět spoustu informací o přenosu na každém z těchto zařízení, ale je není jednoduché dozvědět se mnoho informací o LAN síti jako celku. Proto, abychom se dozvěděli něco o jednotlivých částech sítě, je nutné, aby v každé podsíti LAN byl umístěno jedno monitorovací zařízení (analýzátor, sonda), které

v promiskuitním režimu sleduje veškerý procházející provoz na daném segmentu sítě a vytváří souhrnné statistiky o provozu, včetně chybových statistik, rozložení provozu podle typu paketů, apod. Monitorovací sondou může být buď specializované zařízení (hw sonda) nebo standardní aktivní prvek, který obsahuje funkcionalitu sondy.

Přesně touto cestou se vydal návrh standardu RMON, který je zaměřený na vzdálené monitorování LAN sítí. RMON není přenosový protokol. Standard RFC 2819 z roku 2000 [22] definuje RMON MIB jako doplněk k MIB-II. Standard RFC 4502 [23] rozšiřuje možnosti monitorování o další skupiny MIB, zaměřené převážně na protokoly a aplikace. Pro přenos zachovává RMON standardní protokol SNMP.

10.5.1 Architektura RMON

Základem jsou monitorovací zařízení pro vzdálenou správu zvaná monitory (monitors) nebo sondy (probes). Zařízení jsou rozmístěny v jednotlivých segmentech LAN sítě, kde monitorují provoz, viz obrázek 10.12. Monitorovací zařízení může být buď specializované sonda nebo



Obrázek 10.12: Příklad správy sítě pomocí RMON

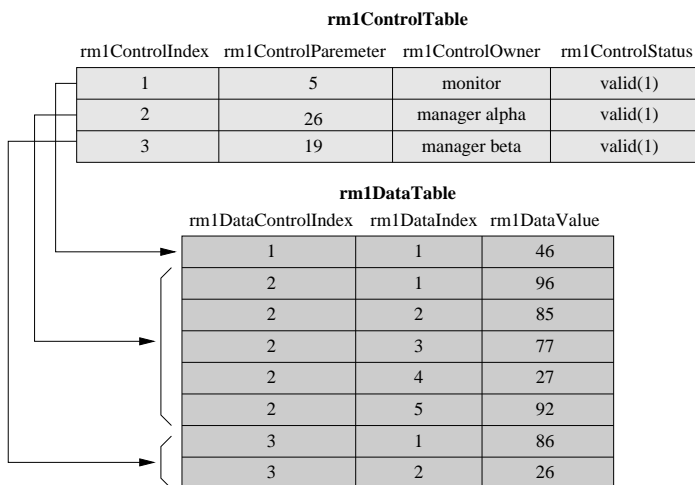
aktivní prvek (směrovač, PC), který kromě vlastní činnosti provádí i monitorování segmentu sítě. Tato monitorovací zařízení komunikují s centrální řídicí stanicí. Zde vidíme posun od klasického modelu SNMP, kde každé monitorované zařízení na síti komunikuje s řídicí stanicí.

Protože sonda funguje jako nezávislé zařízení, které zpracovává data a vyhodnocuje je, není nutné, aby řídicí stanice pravidelně načítala údaje z monitorovaných zařízení. Rozšíření RMON tedy umožňuje nepřímé (offline) zpracování.

Architekturu RMON tedy tvoří:

Sonda (RMON probe, agent) monitoruje tok dat na vzdálené síti, sbírá je a analyzuje monitorovaná data. Konfigurace sondy definuje data, která sonda sbírá. Konfigurace obsahuje řídicí tabulku MIB (control table), která popisuje, jaká data monitorujeme, a dále datové tabulky (data tables) obsahující výsledky monitorování. Řídící stanice RMON zapisuje do řídicí tabulky sondy a definuje, jaká data je třeba monitorovat. Řídící tabulka může obsahovat například objekty, které specifikují zdroj monitorovaných dat, typ dat, frekvenci sběru dat, apod.

Obecný formát řídicích a datových tabulek RMON-1 je na obrázku 10.13. Zde vidíme



Obrázek 10.13: Řídící a datové tabulky RMON-1

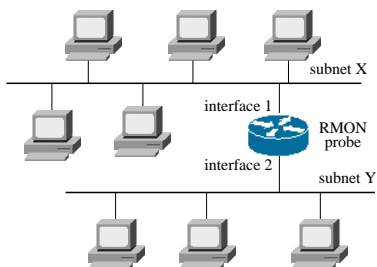
řídicí tabulku (objekt `rm1ControlTable`), což je vlastně seznam jednotlivých řádků tabulky (objekty `rm1ControlEntry`). Každý tento záznam je seznamem sloupců tabulky, v našem případě seznamem objektů `rm1ControlIndex`, `rm1ControlParameter`, `rm1ControlOwner` a `rm1ControlStatus`. Pomocí indexu se identifikují jednotlivé řádky tabulky. Index také odkazuje na řádky do datové tabulky, což jsou datové záznamy, které řídí daný záznam v řídicí tabulce. Datová tabulka obsahuje opět index (objekt `rm1DataIndex`), který ukládá hodnoty pro dané záznamy (objekty `rm1DataEntry`).

Řídící stanice může prostřednictvím protokolu SNMP vytvářet, modifikovat či rušit záznamy tabulek příkazem `SetRequest`. Rušení se provádí tak, že stav záznamu se nastaví na hodnotu `invalid`.

Na obrázku 10.13 můžeme například vidět, že druhý záznam řídicí tabulky je spravován řídicí stanicí **manager alpha**. Tento záznam řídí pět záznamů v datové části s index 1 až 5. Jednotlivé záznamy v datové tabulce obsahují hodnoty 96, 85, 77 a další.

Příkladem může být skupina objektů RMON MIB `host`, která sbírá statistiky o konco-

vých stanicích na dané LAN síti, viz obrázek 10.14. Zde vidíme sondu RMON připojenou



Obrázek 10.14: Zapojení RMON sondy

dvěma rozhraními ke dvou podsítím X a Y. Na podsíti X (rozhraní č.1) je pět monitorovaných koncových uzlů, na podsíti Y (rozhraní č.2) jsou tři monitorované stanice.

Sonda se z procházejících rámců naučí MAC adresy připojených stanic, které si zanesou do seznamu sledovaných koncových stanic. Pro každou sledovanou stanicí si vytváří množinu statistik zahrnující například počty příchozích a odchozích paketů, počty příchozích a odchozích bytů, chybných paketů apod. Sonda si ukládá tato data do RMON MIB databáze, konkrétně do objektů skupiny *host*. Tato skupina obsahuje jednu řídicí tabulku *hostControlTable* a dvě datové tabulky *hostTable* a *hostTimeTable*, viz obrázek 10.15. Na obrázku 10.15 je zobrazena řídicí tabulka *hostControlTable* a jedna datová ta-

hostControlTable						
hostControlIndex	hostControlDataSource	hostControlTableSize	...	hostControlStatus		
1	D1	N1	...	valid(1)		
2	D2	N2	...	valid(1)		
...		

hostDataTable						
hostAddress	hostCreationOrder	hostIndex	hostInPkts	hostOutPkts	...	hostOutErrors
M(1,1)	C(1,1)	1
M(1,2)	C(1,2)	1
M(1,3)	C(1,3)	1
M(1,4)	C(1,4)	1
M(1,5)	C(1,5)	1
M(2,1)	C(2,1)	2
M(2,2)	C(2,2)	2
M(2,3)	C(2,3)	2

Obrázek 10.15: Řídící a datové tabulky skupiny *host*

bulka hostDataTable. Každý řádek řídící tabulky odpovídá jednomu rozhraní, kde sonda naslouchá. Pokud na daném rozhraní detekuje například pět různých koncových stanic, přidá do datové tabulky pět záznamů pro každou stanic, kde index tabulky hostIndex odpovídá indexu prvního rozhraní, což je v našem případě hodnota 1. Každý záznam obsahuje identifikační MAC adresu hostAddress M(1,x) a pořadí přidání záznamu o koncovém uzlu hostCreationOrder C(1,x). Další položky obsahují statistiky vztahující se k danému uzlu, například počet příchozích paketů hostInPkts, počet odchozích paketů hostOutPkts, počet příchozích bytů na danou MAC adresu hostInOctets, počet odchozích bytů hostOutOctets, počet chybných paketů hostOutErrors, počet odchozích broadcastových paketů hostOutBroadcastPkts či počet odchozích multicastových paketů hostOutMulticastPkts.

Další tři řádky datové tabulky se vztahují k rozhraní 2 monitorovací sondy. Zde jsou statistiky k třem koncovým stanicím, které komunikují na daném segmentu LAN sítě připojené k dané sondě.

Řídící stanice (Network Management Station) načítá data z RMON sond pro celkovou analýzu. Protože protokol SNMP nemá žádné příkazy pro řízení monitorovacích stanic, používá se operace **Set** pro vyvolání procesů (action invocation). RMON MIB k tomu využívá speciální objekty, které reprezentují stavy. Pokud řídící změní hodnotu takového stavu, vyvolá to požadovanou akci na sondě.

Základním cílem rozšíření RMON je specifikace rozhraní mezi systémem SNMP a prvky pro vzdálenou správu sítě. RMON redukuje přenosy mezi monitorovanými prvky a řídící stanicí a snižuje nároky na správu sítě jak na straně agenta, tak na straně řídící stanice.

Standard [22] uvádí hlavní výhody, které použití RMON nabízí:

- *Zpracování offline*

Z důvodu snížení nároků na přenosový kanál nebo kvůli chybě na síti může dojít k výpadku spojení mezi řídící stanicí (management station) a RMON sondou (probe). MIB RMON umožňuje sondě provádět diagnostiku a sbírat statistické údaje i v případě přerušení spojení.

- *Proaktivní monitorování*

Při správě zařízení je výhodné sbírat průběžně diagnostické informace a logovat výkon sítě, pokud to procesorový výkon sondy umožňuje. Monitorování je možné i během výpadků. Sonda může oznámit řídící stanicí výpadek na části sítě a uložit historii výpadku. Tyto informace mohou přispět k rychlému zjištění příčiny výpadku.

- *Detekce a reportování*

Monitorovací zařízení umí rozpoznat chybové stavy a události. Aktivně monitoruje stav sítě a pokud dojde k výskytu chybové události, událost se uloží do logovacích souborů. Zároveň se sonda pokusí poslat oznámení řídící stanicí.

- *Přidaná hodnota*

Monitorovací sonda může provádět analýzu nasbíraných dat, čímž ulehčí řídící stanicí. Příkladem může být sledování přenosů a určení nejvíce komunikujících uzlů na daném segmentu sítě.

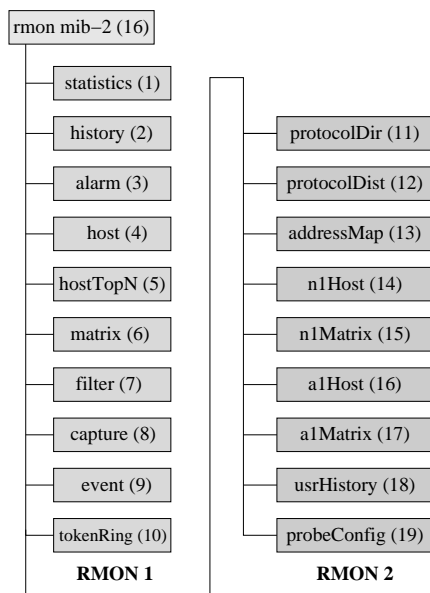
- *Více řídicích stanic*

Systém vzdáleného monitorování sítě umožňuje mít více řídicích stanic, což zvyšuje spolehlivost a distribuuje zátěž na více uzlů. Sonda může být nakonfigurována tak, že paralelně komunikuje s více řídicími stanicemi.

10.5.2 Základní skupiny objektů MIB RMON-1 a RMON-2

Skupina objektů pro vzdálené monitorování RMON MIB je umístěna do podstromu standardní databáze objektů MIB-II. RMON MIB tvoří samostatnou pod-větev s OID 1.3.6.1.2.1.16.

RMON-1 MIB [22] rozděluje monitorované objekty do devíti skupin podporujících technologii Ethernet a jednu skupinu pro podporu sítě Token Ring. Standard RMON-2 [23] přidává dalších devět skupin zaměřených převážně na monitorování vyšších protokolů a aplikací, viz obrázek 10.16.



Obrázek 10.16: Skupiny objektů RMON-1 MIB a RMON-2 MIB

Každá skupina objektů uchovává data a statistiky sebraná RMON sondami. Sonda může obsahovat více jak jedno síťové rozhraní a být připojena k více podsítím. Všechny skupiny RMON MIB jsou volitelné, tzn. nemusí je všechny implementovat RMON sonda. Většina skupin RMON-1 je zaměřena na sběr statistik LAN sítě (skupiny *statistics*, *history*, *host*, *hostTopN*, *matrix* a *tokenRing*). Další skupiny *alarm*, *filter*, *capture* a *event* se

zabývají různými alarmy, podmínka a filtrováním paketů na základě uživatelsky definovaných kritérií.

Nyní si představíme základní skupiny RMON-1 MIB:

1. **Statistics.** Tato skupina monitoruje využití každého segmentu podsítě a sleduje statistiky přenesených rámců, počty broadcastových a multicastových paketů, chyby na Ethernetu (špatný kód CRC, chybná velikost rámce, výskyt kolizí) apod. Zároveň sledují i velikost přenášených rámců – vytváří histogram přenesených rámců o velikosti 64 B, 65 – 127 B, 128 – 255 B, 256 – 511 B, 512 – 1023 B a 1024 – 1518 B.
2. **History.** Skupina *history* uchovává historii monitorování přenosů, která zahrnuje využití zdroje, počet chyb, počet rámců a další statistiky podobné skupině *statistics*. Historii se myslí časový interval, kdy zjišťujeme hodnoty monitorovaných čítačů. Historie časového intervalu se ukládá do jednoho řádku datové tabulky.

Každý řádek datové tabulky tedy obsahuje jeden sebraný vzorek statistik pro dané rozhraní v daném čase (říká se mu historie nebo také vědro *bucket*). Implicitně se vzorky ukládají každých 30 minut.

Vzorek obsahuje hodnoty čítačů monitorovaných událostí za daný čas, například počet přijatých paketů *etherHistoryPkts*. Po ukončení monitorovaného intervalu (vzorku) se vytvoří nový řádek v datové tabulce a uloží se do něj hodnoty monitorovaných čítačů. Pokud počet vytvořených řádků (historií) pro dané rozhraní dosáhne hodnoty *historyControlBucketsGranted*, nastaví se ukazatel na první řádek a nejstarší data se začnou přepisovat. Jedná se tedy o cyklický seznam intervalů monitorovaných dat.

3. **Alarms.** Skupina *alarm* slouží k řízení sběru dat a generování informací na RMON sondě. Alarmy umožňují administrátorovi nastavit vzorkovací interval a práh pro libovolnou položku, kterou agent zaznamenává (absolutní i relativní hodnoty, vzestup či pokles prahových hodnot apod.) Alarm může být například vygenerován, pokud během pěti minutového intervalu se objeví více jak 500 chyb CRC.

Skupina obsahuje pouze jednu tabulku *alarmTable*, kde každý řádek odpovídá vytvořenému alarmu. Alarm je dán sledovanou proměnnou, intervalem vzorkování a parametry prahových hodnot. Alarm může být vygenerován na základě překročení absolutní hodnoty nebo pokud rozdíl současné hodnoty a předchozí je větší než povolený limit.

4. **Host.** Skupina *host* obsahuje statistiky o jednotlivých koncových uzlech na daném segmentu sítě, viz obr. 10.14. Řídící tabulka obsahuje záznamy týkající se jednotlivých rozhraní, datová tabulka k danému rozhraní přiřazuje záznam o monitorované koncové stanici, viz obr. 10.15. Koncová stanice je identifikována pomocí MAC adresy, kterou RMON sonda zjistí v přenášených rámcích.

Každý řádek datové tabulky *hostTable* tedy obsahuje hodnoty čítačů přijatých či odeslaných paketů, počty přijatých a odeslaných bytů, počty chyb, multicastových či broadcastových přenosů. Tento řádek se vždy vztahuje k jednomu počítači.

Pokud je na segmentu příliš mnoho komunikujících stanic, tabulka *hostTable* stále poroste. Aby nedošlo k vyčerpání paměti, začne se tabulka chovat jako cyklický seznam řádků a nejstarší záznamy se budou přepisovat.

5. **HostTopN.** Skupina **hostTopN** poskytuje statistiky o TopN zařízeních na daném segmentu sítě, které v určeném parametru přesahují hodnoty ostatních zařízení. Například může jít o deset nejvíce komunikujících uzlů na LAN síti. Statistiky pro sledování TopN se vybírají z tabulek skupiny **host**. Položka **hostTopNRateBase** definuje sedm proměnných z tabulky **hostTable**, podle nichž se vyhledává TopN: počet příchozích a odchozích paketů, počet přijatých a odeslaných bytů, počet chybných odchozích paketů či počty odchozích broadcastů a multicastů.

Pro sledovanou skupinu je definován interval sledování TopN (**hostTopNDuration**) a také maximální počet reportovaných uzlů. Typ sledovaného parametru a interval je definován v řídicí tabulce **hostTopNControlTable**. Ke každému sledovanému intervalu se vytváří v datové tabulce záznamy o uzlech, které patří do TopN v daném ukazateli.

6. **Matrix.** Skupina **matrix** obsahuje matici provozu mezi jednotlivými dvojicemi komunikujících uzlů na daném segmentu LAN. Hodnoty se ukládají do dvou datových tabulek **matrixSDTable** a **matrixDSTable**. První tabulka je indexována podle zdrojové MAC adresy (source MAC address), druhá tabulka je indexována podle cílové MAC adresy (destination MAC address). Obě tabulky obsahují stejné hodnoty čítačů, tj. počet paketů odeslaných ze zdrojové MAC adresy na cílovou MAC adresu, počet bytů a počet chybných přenosů mezi touto dvojicí adres. Tabulka **matrixDSTable** tato data v opačném pořadí.

Výhodou použití dvou tabulek je například vyhledání informace o tom, jak daná stanice komunikuje s ostatními stanicemi v síti LAN (zde se využije **matrixSDTable**). Pokud nás zajímá, ze kterých stanic přichází data do konkrétní sledované stanice, využijeme data z tabulky **matrixDSTable**.

7. **Filter.** Skupina **filter** umožňuje, aby řídicí stanici přikázala RMON sondě sledovat vybrané pakety na daném rozhraní. Filter může být buď datový, který vyhledává bitové vzorky v paketech, nebo stavový, který umožňuje filtrovat pakety na základě jejich stavu, tj. například platný kód CRC, apod.

Použití logických spojek AND a OR můžeme vytvářet složené filtry obsahující více podmínek. Definovaný filtr je aplikován na procházející provoz a pakety, který jím projdou, tvoří tzv. kanál (channel). Skupina **filter** pak uchovává statistiky provozu týkající se daného kanálu. Pakety definované filtrem je možné zachytávat a ukládat, což dělá skupina RMON **capture**.

Skupina **filter** obsahuje dvě tabulky: tabulku filterů **filterTable** a tabulku kanálů **channelTable**.

8. **Capture.** Skupina **capture** se používá pro zachytávání a ukládání paketů z kanálu definovaném ve skupině **filter**. Obsahuje dvě tabulky: **bufferControlTable**, která specifikuje podrobnosti o ukládání dat, a **captureBufferTable**, která ukládá data do bufferu.

Řídicí tabulka **bufferControlTable** obsahuje ukazatel na kanál, ze kterého se data načítají, definuje velikost bufferu pro ukládání. V případě, že buffer je plný, mohou se přepisovat nejstarší data nebo se buffer uzamkne a další data nepřijímá. Dále definuje max. velikost paketů, které se budou posílat přes SNMP při žádosti o načtení uložených dat.

Datová tabulka `captureBufferTable` obsahuje řádky, kde každý řádek obsahuje zachycený paket. Spolu s ním je uložen identifikátor paketu, délka a čas zachycení.

9. **Event.** Skupina `events` se zaměřuje na logování událostí. Událost může být vybuzena jakoukoliv podmínkou umístěnou v databázi MIB. Událost může také vyvolat určitou akci, například může zapnout či vypnout kanál. Na základě události se mohou začít logovat informace nebo se pošle zpráva typu `trap`.

Řídící tabulka obsahuje definici událostí. Událost obsahuje textový popis, typ události (`none`, `log`, `snmp-trap` či `log-and-trap`), adresáta události a čas poslání.

Pokud se události logují, vytvoří se datová tabulka `logTable`, která obsahuje záznamy odpovídající logovaným událostem.

10. **TokenRing.** Skupina `tokenRing` obsahuje různé statistiky týkající se sítě IEEE 802.5 Token Ring. Tyto statistiky se týkají každé stanice, která je připojena do kruhu Token Ring. RMON sonda může monitorovat více logických kruhů Token Ring a o každém si uchovávat stav a přenosové statistiky.

Databáze RMON-1 MIB umožňuje RMON sondě monitorovat všechny provoz na LAN síti, ke které je sonda připojena. Umí zachytávat veškerý provoz ethernetových rámců a sledovat MAC adresy na síti. Sonda může poskytovat detailní informace o provozu na vrstvě L2. Pokud provoz směřuje mimo LAN, není RMON sonda schopna zjistit nic o cíli, protože nezkoumá IP adresy. Z tohoto důvodu byla přidána nová databáze MIB zvaná RMON-2.

Databáze RMON-2 [23] se zaměřuje na dekodování paketů na vrstvách L3 až L7. Pokud RMON sonda podporu RMON-2, může číst hlavičky vyšších síťových protokolů, zejména IP provozu. Není však omezena na síťovou vrstvu – může také sledovat informace na transportní a aplikační vrstvě modelu TCP/IP.

RMON-2 obsahuje devět nových skupin objektů, viz obrázky 10.16. Některé rozšiřují původní funkcionalitu RMON-1 o podporu informací ze síťové vrstvy (network layer), například `nlHost` či `nlMatrix`, skupiny `alHost` a `alMatrix` obsahují data z aplikační vrstvy (application layer). V této části si představíme čtyři skupiny objektů, které jsou v databázi MIB nové a přinášejí nové možnosti:

1. **ProtocolDir.** Klíčovou skupinou pro monitorování provozu na úrovni L3 až L7 je skupina `protocolDir` (adresář protokolů). Na síti může vyskytovat velké množství různých protokolů, které jsou standardizované či proprietární. Skupina `protocolDir` obsahuje tabulku protokolů `protocolDirTable`, kde každý řádek obsahuje záznam o jednom protokolu.

Záznam obsahuje jedinečný identifikátor pro každý protokol. Identifikátory jsou uspořádány jako strom podle úrovně zanoření. Kořenem je L2 protokol. Každá vrstva je specifikována 32-bitovým kódem zapsaným po oktetech jako `[a.b.c.d]`, například `[0.0.0.1]` pro Ethernet. Další vrstva zanoření přidá další řetězec. Například protokol SNMP zapouzdření v UDP nad IP a Ethernetem, má symbolický identifikátor `ether2.ip.udp.snmp`, kterému odpovídá řetězec ošetřené `[16.0.0.0.1.0.0.8.0.0.0.0.17.0.0.0.161]`. Úvodní hodnota 16 definuje délku v bytech, tj. čtyři protokoly s 32-bitovými identifikátory. Řetězec `[0.0.0.1]` specifikuje Ethernet2, řetězec `[0.0.8.0]` obsahuje hodnotu 16-bitové pole `type` z ethernetového rámce, což pro IP je `0x80`. Další řetězec `[0.0.0.17]` identifikuje protokol na IP, kde hodnota 17 odpovídá UDP. Poslední řetězec `[0.0.0.161]` obsahuje na posledních dvou bytech 16-bitové číslo portu, což je pro SNMP hodnota 161. Vidíme tedy,

že identifikátor protokolu obsahuje identifikaci zapouzdření. V případě výše uvedeného zapouzdření by tabulka `protocolDirTable` obsahovala čtyři různé záznamy:

```
ether2 (4.0.0.1)
ether2.ip (8.0.0.1.0.0.8.0)
ether2.ip.udp (12.0.0.0.1.0.0.8.0.0.17)
ether2.ip.udp.snmp (16.0.0.0.1.0.0.8.0.0.17.0.0.0.161)
```

Pro každý takto identifikovaný protokol obsahuje tabulka `protocolDirTable` jméno protokolu, textový popis, formát adresování a popis pro dekódování protokolu.

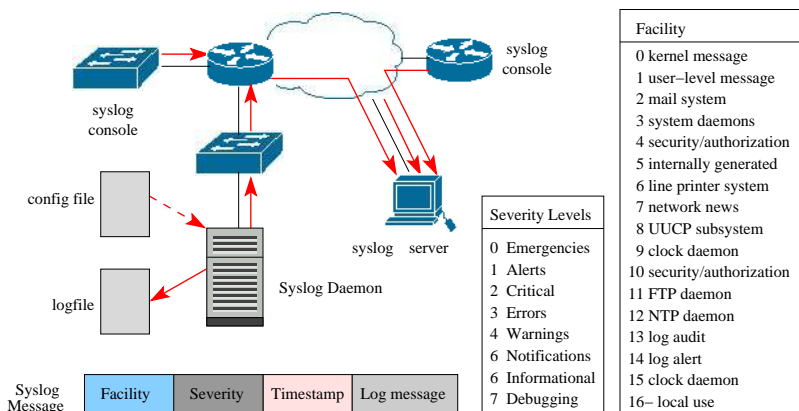
2. **ProtocolDist.** Skupina `protocolDist` (protocol distribution) sumarizuje, kolik bytů a paketů byly poslány každým z podporovaných protokolů. Obsahuje opět dvě tabulky: řídicí tabulku `protocolDistControlTable`, která definuje jednotlivá rozhraní, kde RMON sonda monitoruje provoz. Informace o jednotlivých protokolech na rozhraní jsou ukládána do datové tabulky `protocolDistStatsTable`, kde pro každý protokol je záznam o přenesených paketech a bytech.
3. **AddressMap.** Skupina `addressMap` slouží k mapování síťových L3 adres na konkrétní L2 adresy a na konkrétní porty síťového zařízení. Toto je velmi užitečné pro objevování sousedů a topologie sítě. Datová tabulka `addressMapTable` obsahuje záznamy o mapování IP adresy a MAC adresy. Kromě toho obsahuje i port, na kterém se naposledy daná IP adresa objevila, čas změny a další údaje.
4. **ProbeConfig.** Skupina `probeConfig` slouží k řízení RMON sondy řídicí stanicí NMS. Skupina obsahuje objekty vztahující se k popisu sondy (možnosti sondy, verze softwaru a hardwaru, aktuální datum a čas, jméno image pro čtení z TFTP, apod.). Dále obsahuje konfigurační informace pro sériová rozhraní, konfiguraci na úrovni IP (adresa, maska, implicitní brána), adresu pro posílání zpráv `trap` a další.

Z výše uvedeného výpisu základních skupin MIB RMON-1 a RMON-2 vidíme široké možnosti monitorování provozu na síti, sbírání statistických údajů a chybových hlášek pomocí protokolu SNMP a RMON sond. Implementace podpory RMON-1 a RMON-2 na zařízeních je dost náročná, zpracování rámců a paketů má také velký vliv na výkon zařízení. Proto se v praxi monitorování pomocí RMON-1 a RMON-2 příliš nepoužívá. V poslední době ustoupilo monitorování pomocí statistik Netflow, který umožňuje získat ze sítě informace o provozu podobné statistikám RMON-1 a RMON-2 s daleko jednodušší implementací a menší provozní i přenosovou režii. Popis systému NetFlow je však nad rámec tohoto textu.

10.6 Syslog

Posledním nástrojem pro správu sítí, který si krátce představíme, je syslog. Jedná se o důležitý nástroj pro správu sítí zejména na úrovni aplikační. Syslog je aplikace původně vycházející z Unixu, kde slouží k logování systémových událostí na systémové konzoli. Pracuje tak, že syslog démon (syslogd) sleduje lokálně generované události z aplikací a ukládá je do logovacího souboru lokálně nebo na vzdálený počítač.

Tento systém se postupně rozšířil a používá se pro sledování činností procesů na síťových zařízeních. Jeho důležitou vlastností je, že umožňuje zasílat oznámení o výskytu různých událostí. Syslog se velmi často používá jako doplňující nástroj ke správě sítí, kde syslog démon



Obrázek 10.17: Monitorování sítě pomocí nástroje syslog

sleduje aktivity na spravovaném zařízení a při výskytu kritické události pošle oznámení na řídicí stanici sítě (syslog server).

Pro komunikaci se používá aplikační protokol Syslog, který je definován standardy RFC 3164 a 5424 [16, 10]. Jedná se o textový protokol, který přenáší popis události, která se vyskytla, čas výskytu, úroveň důležitosti zprávy a textový popis. Příklad komunikace pomocí aplikace syslog je na obrázku 10.17.

Jak vidíme na obrázku, syslog definuje osm úrovní důležitosti zprávy (severity), kde 0 (Emergency) označuje nejvíce kritickou událost. Nejméně kritickou událostí jsou debugovací hlášky (úroveň 7). Správce systému se podle nastavení může zvolit, kterou úroveň chce od dané aplikace či zařízení posílat. Obvyklé nastavení je úroveň důležitosti 4, které zahrnuje i zprávy o vyšší úrovni urgency.

Dále se ke zprávě přidává typ odesílatele pomocí pole **Facility**. Kromě definovaných hodnot 0–15 existují i typy 16–23, které jsou určeny pro lokální využití a které si může administrátor sám nadefinovat. Paket také obsahuje část textového popisu chyby, kde lze specifikovat podrobnější popis chyby.

Protože protokol Syslog je podporován většinou operačních systémů i serverových aplikací, používá se k monitorování stavu důležitých serverů. Také spousta výrobců síťových zařízení podporuje tento protokol, takže je možné obdržet notifikaci přes syslog od přepínače, kde došlo k bezpečnostnímu zablokování portu při detekci nepovoleného DHCP serveru apod. Syslog server bývá také součástí NMS systému pro správu sítí.

Shrnutí

V této kapitole jsme si představili přístupy a nástroje pro efektivní správu sítí. Shrnutí jsme důvody a cíle správy sítě i základní postupy, které jsou založeny na sledování statistických dat či analýze obsahu přenášených paketů. Podívali jsem se na standardizovaný model FCAPS,

který se zabývá správou poruch, účtováním, správou konfigurací, sledování výkonnosti a bezpečnosti.

Pak jsme se podívali na různé úrovně monitorování provozu a stavu síťových aplikací a zařízení. Na třetí vrstvě modelu TCP/IP slouží pro správu protokol ICMP a aplikace, které ho využívají.

Komplexnější správu zajišťuje systém SNMP, který vytváří strukturu agentů a řídicích stanic NMS, které pravidelně sledují stav sítě. Jádrem systému SNMP jsou data uložená v databázi MIB. Standardní databáze MIB a MIB-II podporují pouze správu LAN segmentů, kde zjišťují nezávislé statistiky o přenosech a chování monitorovaných zařízení. Další funkčnost přináší rozšíření RMON, které buduje nezávislé měřicí body v různých částech LAN sítě. Tyto takzvané RMON sondy nejen sbírají provozní statistiky, ale také je zpracovávají, dlouhodobě ukládají a analyzují. Rozšíření RMON-II přidává zprávu na úrovni L4 až L7 včetně možnosti filtrování a odchyťování sledovaného provozu.

K monitorování dat na úrovni L7 včetně monitorování činnosti aktivních prvků se také využívá systém Syslog, kde sledovaná zařízení posílají notifikační zprávy na řídicí server.

V dnešní době se používá ke zprávě kombinace systému SNMP, syslog, případně NetFlow. Každý nástroj je určen na sběr a sledování odlišných parametrů, což pomáhá ke zjišťování komplexního pohledu na spravovanou počítačovou síť.

Použitá literatura a standardy

- [1] *Abstract Syntax Notation One (ASN.1): Specification of basic notation*. ITU-T X.680, 2008.
- [2] J. Case, M. Fedor, M. Schoffstall, and C. Davin. *A Simple Network Management Protocol*. RFC 1067, August 1988.
- [3] J. Case, M. Fedor, M. Schoffstall, and J. Davin. *A Simple Network Management Protocol (SNMP)*. RFC 1157, May 1990.
- [4] J. Case, K. McCloghrie, M. Rose, and S. Waldbusser. *Protocol Operations for Version 2 of the Simple Network Management Protocol (SNMPv2)*. RFC 1905, January 1996.
- [5] J. Case, R. Mundy, D. Partain, and B. Stewart. *Introduction and Applicability Statements for Internet Standard Management Framework*. RFC 3410, December 2002.
- [6] Vint Cerf. *IAB Recommendations for the Development of Internet Network Management Standards*. RFC 1052, April 1988.
- [7] B. Claise. *Cisco Systems NetFlow Services Export Version 9*. RFC 3954, October 2004.
- [8] J. Davin, J. Case, M. Fedor, and M. Schoffstall. *A Simple Gateway Monitoring Protocol*. RFC 1028, November 1987.
- [9] B. Fraser. *Site Security Handbook*. RFC 2196, September 1997.
- [10] R. Gerhards. *The Syslog Protocol*. RFC 5424, March 2009.
- [11] ITU-T. *X.209 : Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*. X.209, November 1988.

- [12] ITU-T. *Common Management Information Protocol: Specification*. X.711, October 1997.
- [13] ITU-T. *Common Management Information Service and Protocol*. X.710, October 1997.
- [14] ITU-T. *TMN Management Functions*. M.3400, February 2000.
- [15] J.Postel. *Internet Control Message Protocol*. RFC 792, September 1981.
- [16] C. Longvick. *The BSD syslog Protocol*. RFC 3164, August 2001.
- [17] K. McCloghrie and M. T. Rose. *Management Information Base for network management of TCP/IP-based internets*. RFC 1156, May 1990.
- [18] K. McCloghrie and M. T. Rose. *Management Information Base for Network Management of TCP/IP-based internets:MIB-II*. RFC 1213, March 1991.
- [19] R. Presuhn, J.Case, K.McCloghrie, M.Rose, and S.Waldbusser. *Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)*. Rfc 3416, December 2002.
- [20] M. T. Rose and K. McCloghrie. *Structure and identification of management information for TCP/IP-based internets*. RFC 1155, May 1990.
- [21] William Stallings. *SNMP, SNMPv2, SNMPv3, and RMON 1 a 2*. Addison-Wesley, 3rd edition, 1999.
- [22] S. Waldbusser. *Remote Network Monitoring Management Information Base*. RFC 2819, May 2000.
- [23] S. Waldbusser. *Remote Network Monitoring Management Information Base Version 2*. RFC 4502, May 2006.

