VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

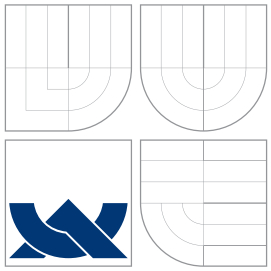
SYSTÉM PRO UŽIVATELEM ŘÍZENÉ QOS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

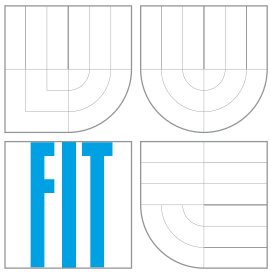
AUTOR PRÁCE
AUTHOR

OLDŘICH PLCHOT

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

SYSTÉM PRO UŽIVATELEM ŘÍZENÉ QOS

USER ORIENTED QOS SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — OLDŘICH PLCHOT
AUTHOR

VEDOUCÍ PRÁCE — Ing. TOMÁŠ KAŠPÁREK
SUPERVISOR

BRNO 2007

Abstrakt

Tento diplomový projekt se zabývá možnostmi operačního systému GNU/Linux v oblasti zajištění kvality poskytovaných síťových služeb. Práce porovnává a hodnotí prostředky k zajištění kvality služeb dostupné v operačním systému GNU/Linux. Cílem práce je diskutovat nedostatky a přednosti těchto prostředků a navrhnout systém, který řeší problematiku zajištění kvality služeb. Navržený systém využívá heuristiky, která umožní uživateli nastavit kvalitu služeb i bez nutnosti studovat specifické vlastnosti komunikačních protokolů na úrovni síťové nebo aplikační vrstvy. Součástí projektu je také teoretický úvod do problematiky kvality služeb a architektury počítačových sítí.

Klíčová slova

Linux, TCP/IP, IPv6, kvalita služeb, QoS, Iptables, router, HTB, jádro, ISO/OSI, paket, hlavička paketu, neuronové sítě, klasifikace, datový tok, pcap

Abstract

This master's thesis deals with the possibilities how to guarantee the quality of service in the area of computer networks using a GNU/Linux operating system. This work compares and evaluates tools which are necessary to guarantee the quality of service. The goal of this work is to discuss the advantages and disadvantages of these tools and to design a system which handles the problem of quality of service. Designed system uses a heuristics, which allows the user to set up the quality of service system without studying specific properties of communication protocols on the network or application layer. This work also includes a theoretical introduction into the quality of service and computer networks.

Keywords

Linux, TCP/IP, IPv6, quality of service, QoS, Iptables, router, HTB, kernel, ISO/OSI, packet, packet header, neural networks, classification, data flow, pcap

Citace

Oldřich Plchot: Systém pro uživatelem řízené QoS, diplomová práce, Brno, FIT VUT v Brně, 2007

Systém pro uživatelem řízené QoS

Prohlášení

Prohlašuji, že jsem tento diplomový projekt vypracoval samostatně pod vedením Ing. Tomáše Kašpárka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Oldřich Plchot
15. května 2007

Poděkování

Rád bych na tomto místě poděkoval především panu Ing. Tomáši Kašpárkovi za odbornou pomoc, poskytnutou při řešení problémů spojených s tímto projektem.

Obsah

Seznam tabulek

Seznam obrázků

Kapitola 1

Úvod

Dnešní svět si nelze představit bez možnosti snadné a rychlé komunikace, internetu a multimédií. Stále více počítačových sítí, a to jak malých, tak rozlehlých, je připojováno k internetu a vzniká potřeba tyto sítě chránit a zajistit jejich uživatelům možnost pohodlně, bezpečně a efektivně pracovat. Většina malých a středních sítí připojovaných do internetu řeší problém úzkého hrdla v případě konektivity do této mezinárodní sítě. V mnoha případech dokáže vhodně nastavený systém pro zajištění kvality služeb na přístupovém bodu do mezinárodní sítě ušetřit čas a peníze připojených uživatelů.

Tato práce se zabývá možnostmi operačního systému GNU/Linux v oblasti sítí a zajištění kvality služeb pro uživatele připojené do internetu. Cílem práce je porovnat možnosti prostředků k tomuto účelu dostupných, diskutovat jejich přednosti a nedostatky a navrhnout systém, který řeší problematiku zajištění kvality služeb. Navržený systém spojuje možnosti poskytované standardními nástroji dostupnými v operačním systému GNU/Linux s novým přistupem ke klasifikaci síťového provozu.

Součástí projektu je teoretický úvod do problematiky kvality služeb a architektury počítačových sítí.

Výsledkem je porovnání možností operačního sysému GNU/Linux sloužit jako platforma pro zajištění kvality síťových služeb a implementace navrženého systému, který v reálném čase reaguje na provoz generovaný jednotlivými uživateli a dokáže se mu přizpůsobovat.

Tento diplomový projekt navazuje na ročníkový projekt Srovnání síťové vrstvy BSD a Linuxu, který tvoří základy kapitoly 3 a 4. Dále potom práce navazuje na semestrální projekt, který je součástí zadání diplomového projektu. V rámci semestrálního projektu byly vypracovány kapitoly 2,3,4 a část kapitoly 5. Během semestrálního projektu byly prozkoumány teoretické materíály a navržena architektura výsledného systému.

Kapitola 2

Kvalita služeb (QoS)

V oblasti dnešních počítačových sítí znamená pojem kvalita služeb schopnost sítě dosahovat požadovaných parametrů pro různé typy síťových aplikací na lince o určité šířce pásma, kterou je síť připojena do další síťě (typicky do internetu). Různá řešení, zabývající se zajištěním kvality služeb pro tyto aplikace, musí počítat nejen s jednoduchým rozdělením šířky pásma, ale také s chybami a problémy,[7] které se vyskytují v nynějších sítích, fungujících na principu přepínání paketů.

ztracené pakety (packet loss) - může se stát, že některé z routerů, vyskytujících se v cestě paketu, jej nedoručí. Tento případ může nastat například při velkém vytížení některé z cest, kterými paket prochází nebo také poruchou některého z routerů. V důsledku této chyby potom vzniká situace, kdy aplikace, která daný paket očekává, požádá o jeho znovu zaslání a způsobí tak zpoždění. Pokud však je služba navržena tak, že není nutné obdržet každý paket, pak pokračuje v činnosti, ale je ovlivněna touto chybou (například krátký výpadek zvuku nebo artefakty v obraze).

zpoždění (delay) - může trvat různou dobu, než paket doputuje do svého cíle díky chybám a nebo proto, že putuje jinými cestami, aby se vyhnul zahlceným trasám. Tato doba je pro některé aplikace kritická a pokud ji nedokážeme udržet pod určitou hranicí, tak dochází ke zhoršení kvality služby, případně k její nefunkčnosti.

zkreslení (jitter) - nastává, když pakety dorazí do cíle s různým zpožděním. Je problémem zejména pro interaktivní přenos audia nebo videa.

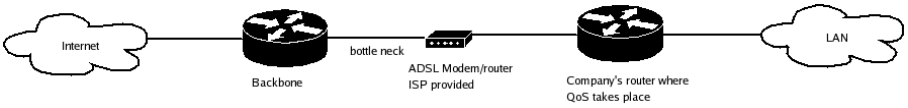
nesprávné pořadí paketů - je běžné, že pakety procházejí velkým množstvím routerů, které je mohou poslat různými cestami, a tím pádem také způsobí různé zpoždění a pakety jsou doručeny v jiném pořadí, než byly vyslány. Pro některé služby to nepředstavuje žádný problém, ale například pro přenos hlasu a videa je správné pořadí obdržených paketů velmi důležité.

chyba v přenosu - pokud během cesty došlo ke změně jednoho nebo více bitů paketu a oprava chyb v transportní vrstvě nedokáže data opravit do správného stavu, tak je s paketem jednáno, jako by byl ztracen a je vyžádáno jeho opětovné zaslání.

Tyto chyby závisí na aktuálním stavu sítě, který nejsme schopni ovlivnit a je nemožné s určitostí předpovědět, kdy nastanou. Nejčastějšími příčinami těchto chyb bývá zahlcení některé z cest, kterými jsou pakety routovány k cíli. Důležitým krokem pro zajištění kvality služeb je eliminování úzkých hrdel sítě. V případě velkých telekomunikačních společností,

poskytujících přístup k páteřním sítím o velmi vysoké propustnosti, musí být síť dimenzována tak, aby nedocházelo k jejímu zahlcování a nebo aby byla předem zajištěna kvalita služby. K tomuto účelu existuje například protokol *RSVP* (Resource Reservation Protocol, RFC 2205), který je navržen za účelem rezervace kapacity pro požadované služby skrze celou síť. Podpora těchto protokolů je potom obsažena v každém routeru páteřní sítě a zajišťuje tak její vysokou úroveň při zajištění služeb. Úzkým hrdlem se tedy v mnoha případech stává bod, ve kterém dochází k napojení do této sítě.

Tímto způsobem jsou většinou připojeni buď menší poskytovatelé připojení k internetu nebo různé firmy, které mají sjednánu linku o garantované kapacitě s operátorem, který má možnosti a prostředky zajistit s velkou spolehlivostí požadovanou šířku pásma. Zajištění kvality služeb pro tyto menší subjekty spočívá tedy především v prevenci zahlcení svého přístupového bodu. K tomu, abychom byli schopni kontrolovat kvalitu služeb na určité lince, musíme mít pod kontrolou obě strany toku dat. Pokud bychom připojili do páteřní sítě přímo nějaké koncové zařízení, pak můžeme účinně kontrolovat pouze tok dat, který na tomto zařízení vzniká a putuje do páteřní sítě. Druhý směr, odkud data přicházejí, má pod kontrolou telekomunikační operátor, který obvykle vyhradí pouze smluvenou kapacitu a další kroky k řízení kvality služeb v tomto bodě zpravidla již neposkytuje. Aby bylo možné kontrolovat oba dva směry toku dat a zajistit nad nimi v rámci možností požadované parametry, tak je třeba, v bodě připojení k páteřní síti, zařadit router, který je schopen definovat kvalitu služeb.



Obrázek 2.1: Typické připojení sítě do internetu

K tomu, aby byl router schopen efektivně kontrolovat tok dat, předcházet zahlcení a kontrolovat tak požadované parametry, je nutné obětovat určitou část ze sjednané kapacity linky ve prospěch řízení kvality služeb. Pokud je linka plně vytížena a fronty na jejím koncovém bodě jsou plné, začne docházet k zahazování paketů, což způsobí zpoždění, a to je pro velkou část aplikací problém. Pokud bychom nesnížili rychlost linky na vstupním routeru, tak by docházelo k tvoření těchto front na výstupní straně poskytovatele a router by tedy nebyl schopen kontrolovat příchozí tok dat. Jakmile je na vstupním routeru nastavena rychlost menší než na vstupní lince, tak se tyto fronty přesunou na výstup do vnitřní sítě, který je pod kontrolou routeru a je možné nad nimi provádět operace, které umožní zajistit parametry požadované jednotlivými službami.

Pokud paket dorazí k routeru a nemůže být okamžitě odeslán, tak je zařazen do fronty. Kdyby nebyla definována žádná pravidla kvality služeb, musel by tento paket čekat, než budou odeslány všechny předchozí pakety. To by však opět způsobilo zpoždění. Vstupní router ale dokáže takový paket zařadit na začátek fronty a umožnit tak například aplikaci pro streamování audia nerušený provoz.

Kapitola 3

Základní principy síťové architektury

3.1 Referenční OSI model

OSI (Open System Interconnection) je ISO standard, který vznikl v roce 1978 a zrevidován byl v roce 1984. Popisuje rámec síťové komunikace pro implementaci protokolů v sedmi vrstvách. OSI model je zkonstruován tak, že řízení je předáváno z jedné vrstvy do druhé, a to takovým způsobem, že každá vrstva může komunikovat pouze se svou sousední vrstvou.

Referenční OSI model má sedm vrstev, pro které platí následující principy:

- vrstva by měla být vytvořena tam, kde je potřeba dosáhnout různého typu abstrakce,
- každá vrstva by měla vykonávat pouze jasně definované funkce,
- funkce každé vrstvy by měla být vybrána s ohledem na definování mezinárodně uznávaných a standardizovaných protokolů,
- rozsah vrstvy by měl být zvolen tak, aby se minimalizoval datový tok mezi zařízeními,
- počet vrstev by měl být natolik velký, aby nemusely být rozdílné funkce slučovány do jedné vrstvy a natolik malý, aby se architektura nestala nepraktickou.

Aplikační vrstva Tato nejvyšší vrstva definuje, jakým způsobem může aplikace běžící na jednom systému komunikovat s aplikací na jiném systému, čímž umožňuje aplikacím přístup ke komunikačním kanálům a následnou spolupráci.

7 Aplikační vrstva – Application Layer	http, ftp, SMTP
6 Prezentační vrstva – Presentation Layer	SSL, TLS
5 Relační vrstva – Session Layer	TCP
4 Transportní vrstva – Transport Layer	TCP, UDP, SCTP
3 Síťová vrstva – Network Layer	IP, ICMP, ARP
2 Vrstva datových spojů – Data Link Layer	Ethernet, ATM, PPP
1 Fyzická vrstva – Physical Layer	DSL, ISDN, 100BASE-T

Tabulka 3.1: ISO/OSI model

Prezentační vrstva Vrstva poskytuje nezávislost na rozdílné reprezentaci dat tak, že převádí data z aplikačního do sítového formátu a naopak. Poskytuje například kompresi dat nebo šifrování a tím zajišťuje, že procházející komunikace je ve správné formě, které rozumí příjemce. Programy v této vrstvě určují tři základní vlastnosti. Datové formáty, například ASCII, nebo binární formát dat, kompatibilitu s operačním systémem hostitelského počítače a zapouzdření dat do zpráv tak, aby mohly být posílány přes počítačovou síť.

Relační vrstva Tato vrstva vytváří, udržuje a ukončuje spojení mezi aplikacemi. V internetových aplikacích je každé sezení přiřazeno k určitému portu, což je číslo přidružené ke konkrétní vyšší vrstvě aplikace. Například http server má obvykle číslo portu 80. Čísla portů přidružených k hlavním internetovým aplikacím jsou přiřazována organizací IANA (Internet Assisgned Numbers Authority) a můžeme je nalézt v souboru /etc/services. Čísla portů menší než 1024, nazývané také privilegované porty, mohou být otevřena pouze superuživatelem. Klienti, kteří se připojují k těmto portům si mohou být jisti, že na nich běží některá ze standardních služeb a ne libovolná služba puštěná uživatelem. Většina portů je nicméně k dispozici k dynamickému přiřazení ostatním aplikacím.

Transportní vrstva Tato vrstva poskytuje transparentní přenos dat mezi jednotlivými koncovými systémy nebo počítači a je zodpovědná za kontrolu chyb a jejich opravu. Kompletně zajišťuje datový přenos.

Síťová vrstva Poskytuje přepínací a směrovací technologie pro přenášení dat, překládá logické adresy a jména na fyzické adresy a určuje optimální cestu paketu ze zdroje do cíle. Vytváří logické cesty, které jsou známy jako virtuální okruhy (virtual circuits). Jejími funkcemi jsou také přesměrovávání, kontrola chyb, kontrola zahlcení a řazení paketů.

Vrstva datových spojů Zajišťuje přenos dat tam i zpět přes fyzické spojení v síti, dělí odchozí data do rámců, kontroluje integritu přijatých dat, spravuje přístup k médiu a použití média a zajišťuje správné řazení paketů. Tyto funkce jsou většinou poskytovány ovladačem síťové karty, obecněji hardwarem zodpovědným za propojení stroje do sítě. Takové zařízení se označuje jako NIC (Network Inteface Card).

Fyzická vrstva Tato vrstva vytváří bitový proud (elektrické impulzy, světelný, nebo radiový signál atd.) přes komunikační médium na elektrické a mechanické úrovni. Zpřístupňuje hardware ve smyslu odesílání a přijímání dat na nosiči.

3.2 Sada protokolů TCP/IP

Základem pro přenos informací vyššími protokoly se stal už roku 1980 protokol IP (Internet Protocol). [1] Je používán na paketově orientovaných sítích a zajišťuje přenos paketu, aniž by garantoval úspěšnost doručení, případně zajišťoval správnost doručených dat. Tyto úkoly přenechává vyšším protokolům, kterým poskytuje abstrakci nad přenosovým médiem a umožňuje tak spojovat sítě různých typů (např. ethernet, ATM, FDDI). Cílem protokolu

[1] Tento protokol byl definován jako standard už v lednu roku 1980 v RFC 760 a do podoby používané dodnes a známé jako IPv4 byl upraven v září 1981 dokumentem RFC 791.

Referenční ISO/OSI model	TCP/IP
Aplikační vrstva	Síťové aplikace
Prezentační vrstva	Rozhraní soketů
Relační Vrstva	
Transportní vrstva	TCP UDP
Síťová vrstva	IP ICMP
Vrstva datových spojů	ARP, RARP, NDIS
Fyzická vrstva	Fyzické rozhraní síťového zařízeni

Tabulka 3.2: Znázornění TCP/IP do ISO/OSI modelu

IP je poskytnout všem zařízením v internetu jedinečnou adresu v jednodném adresním prostoru a umožnit tak jejich komunikaci.

Protokol TCP/IP byl původně vyvinut jako vědecký experiment na univerzitě v Berkeley a postupně se stal klíčovou technologií Internetu. Protokoly TCP/IP poskytují uživatelům základní služby jako například World Wide Web (WWW), E-mail a další. Podobně jako ISO/OSI má vrstvovou architekturu. TCP/IP není v konfliktu s ISO/OSI standardem a naopak, protože oba dva modely byly vyvinuty souběžně, ale existují mezi nimi určité rozdíly. Největší rozdíl mezi OSI architekturou a TCP/IP je ve způsobu řešení problémů nad transportní vrstvou a v síťové vrstvě. OSI má relační a prezentační vrstvu, kdežto TCP/IP kombinuje tyto dvě vrstvy do aplikační vrstvy. Potřeba implementace protokolu, který neudržuje stav spojení také přinutila TCP/IP zkombinovat fyzickou vrstvu a vrstvu datových spojů do jedné vrstvy.

TCP/IP je zkratkou dvou nejpoužívanějších protokolů. TCP (Transmission Control Protocol) je protokol transportní vrstvy, který převádí zprávy do posloupnosti paketů na zdrojovém uzlu a potom je zase sestavuje do původních zpráv na cílovém uzlu sítě. IP (Internet Protocol) je protokol síťové vrstvy, který spravuje adresování tak, aby pakety mohly být směrovány skrze uzly nebo i celé sítě pracující s různými komunikačními protokoly. TCP/IP nám umožňuje propojení počítačů na aplikační úrovni, což nám umožňuje zavádět síťové služby, které jsou identifikovány pomocí portů.

Implementaci síťového stacku v systémech unixového typu si tedy můžeme představit tak, jak je znázorněno v tabulce. Jde rovněž o vícevrstvý model, kde určité části plní jednu nebo více funkcí z modelu ISO/OSI.

V současných systémech nejvíce převládá právě použití sady protokolů TCP/IP díky jeho jednoduché koncepci, a tím pádem také snadné implementaci. Rozšíření tohoto protokolu bylo způsobeno především rozmachem internetu a poptávkou po levných, ale současně rychlých síťových zařízeních.

Kvalitní podpora TCP/IP je absolutní nutností jakéhokoliv dnes používaného operačního systému. GNU Linux i *BSD systémy, kde bylo TCP/IP původně vytvořeno, poskytují v této oblasti služby na nadstandardní úrovni.

3.2.1 Protokol ICMP

Internet Control Message Protocol se stará o distribuci chybových a kontrolních zpráv zasílaných jak routery, tak koncovými zařízeními v síti. Tyto zprávy nejsou většinou generovány uživatelskými aplikacemi, ačkoliv existují aplikace jako například *ping* nebo *traceroute*, pomocí kterých je možno rychle a jednoduše diagnostikovat základní stav sítě.[5]

Datagramy protokolu ICMP jsou přenášeny protokolem IP a poskytují zařízením v síti informace o stavu sítě. Tyto informace jsou využívány především routery, které mohou na jejich základě změnit své chování tak, aby byla zachována jejich funkce na co nejlepší úrovni. Může jít například o výběr nejvhodnější trasy pro směrované pakety, nebo o využití alternativní trasy při výpadku některého sousedního routeru.[2] ICMP byl jako standard definován už v září roku 1981 a je popsán dokumentem RFC 792. Tento protokol je využíván pouze nad komunikací pomocí IPv4. Pro nový protokol IPv6 je definována nová verze ICMP nazývaná také ICMPv6, která v sobě kombinuje funkce několika dalších protokolů používaných spolu s IPv4. ICMPv6 zahrnuje funkce předchozího ICMP, IGMP (Internet Group Protocol) a ARP (Address Resolution Protocol).[6]

3.2.2 Protokol TCP

Tento protokol tvoří základ dnešní síťové komunikace a je používán pro sestavení spojení, které zajišťuje spolehlivý a bezporuchový přenos dat. Navíc je zajištěno, že data budou aplikaci dodána v takovém pořadí, v jakém byla vyslána. Takovýchto spojení mezi jednotlivými koncovými body může být více a každé z nich je jednoznačně určeno čtveřicí zdrojová ip adresa, zdrojový port, cílová ip adresa a cílový port. Protokol TCP pracuje s proudem dat, který generuje příslušná aplikace. Tento proud dat je dále rozdělen na segmenty obvykle o maximální možné velikosti (MTU - Maximum Transmision Unit), které podporuje linková vrstva zařízení připojeného do sítě. Tyto segmenty - *pakety* jsou poté předány do nižší vrstvy, kde jsou zpracovány protokolem IP, který zajistí samotný přenos paketu skrze síť druhé straně. Pak jej IP předá protokolu TCP, který provede kontrolní součet a zjistí, zda při přenosu nedošlo k chybě. Pokud byl paket v pořádku přenesen, pak TCP pošle potvrzení, že byl paket přijat. Pokud dojde k chybě, tak je vyžádáno znovuzaslání paketu. Pokud vysílající strana neobdrží potvrzení v určitém čase (RTT - Round Trip Time), tak nastává *timeout* a odeslání příslušného paketu je opakováno.

Protokol TCP dokáže také určitým způsobem zamezovat zahlcení linky. Díky časovačům a zasílání potvrzení jsou vysílající strany schopny odhadnout podmínky na síti a adekvátně k nim pak přizpůsobit datový tok. Tato kontrola datového toku ale vychází pouze z informací získaných z koncových uzlů spojení. Způsob, jakým tečou data mezi koncovými body, už má na starosti protokol IP, který pracuje na třetí vrstvě podle ISO/OSI. Na čtvrté vrstvě, kde pracuje protokol TCP tedy není možné přesně určit, co způsobuje zahlcení, případně z jakého důvodu. TCP vychází pouze z informací o nutnosti přeposlat určité segmenty dat. Nicméně TCP musí být schopno tuto situaci řešit, aby nedošlo k úplnému zahlcení linky a tím pádem také k nefunkčnosti služeb využívajících TCP.

Každý vysílaný segment dat je umístěn do fronty s časovačem, který určí, za jak dlouho má být paket přeposlán. Pokud by z jakéhokoliv důvodu na síti došlo k velkému zvýšení provozu a k zahlcení a neexistovaly by mechanismy, které se s touto situací vyrovnají, tak by nastalá situace ještě zvýšila současné zahlcení. Došlo by k tomu z toho důvodu, že by některé segmenty nebyly přeneseny nebo byly přeneseny pozdě, což by způsobilo jejich opětovné vyslání a ještě dále zatížilo síť a situace by mohla dojít tak daleko, že by byla síť naprosto nepoužitelná.

Tím, že TCP určí úroveň, při které nejsou vysílané pakety potvrzovány protějškem, získá informaci, která je použita při zamezení zahlcení. Původní standard RFC 793 neobsahoval téměř žádné instrukce, jak se vyhýbat zahlcení linky. Později se ovšem ukázalo, že právě

[2]Routery v páteřních sítích místo protokolu ICMP často využívají vyspělé routovací protokoly jako jsou například OSPF (Open Shortest Path First) nebo BGP (Border Gateway Protocol)

zahlcování linky je velký problém a byly vyvinuty příslušné techniky řešení, které byly nakonec shrnuty v RFC 2001. Jsou to techniky *TCP Slow Start, Congestion Avoidance, Fast Restransmit a Fast Recovery Algorithms.*

V RFC 3168 byl popsán protokol ECN - Explicit Congestion Notification jako doplněk do sady TCP/IP. Je to signalizační protokol, který má předcházet zahlcení.

3.2.3 Protokol UDP

Tento protokol je dalším ze sady protokolů TCP/IP. Je používán pro přenos krátkých zpráv nazývaných *datagramy.* UDP (User Datagram Protocol) ale na rozdíl od TCP nevytváří spojení a neposkytuje bezporuchový přenos dat, přenos dat ve správném pořadí, znovu zasílání ztracených paketů, detekce a zahazování duplicitních paketů a podporu pro zamezování zahlcení linky. Pokud je některá z těchto vlastností aplikací požadována, tak musí být zajištěna ve vyšších vrstvách. Díky absenci režie těchto vlastností je UDP často rychlejší a výkonnější pro aplikace, kde je potřeba přenést rychle mnoho krátkých zpráv k mnoha klientům. Je také používáno k multicastu a broadcastu. Mezi konkrétní aplikace využívající tohoto protokolu patří například DNS systém, aplikace pro streamování multimédií, hlasové služby, počítačové hry atd.

Tím, že chybí jakékoliv techniky pro zamezení zahlcení linky, je nutné implementovat tyto mechanismy do síťových prvků, aby se předešlo případnému kolapsu sítě díky nekontrolovanému provozu, který aplikace využívající UDP generují. V praxi jsou to většinou routery, které díky řazení paketů do front a jejich případnému zahazování dokáží kontrolovat datový tok způsobený protokolem UDP, předcházet tak zahlcení a udržet stanovenou úroveň kvality i pro ostatní aplikace.

V současné době je vyvíjen protokol DCPP - Datagram Congestion Control Protocol (RFC 4340),/indexDCPP (Datagram Congestion Control Protocol) který zajistí kontrolu zahlcení na nižší vrstvě a každá aplikace, která tuto funkčnost bude požadovat, nemusí tuto techniku sama implementovat. DCPP je určeno aplikacím, které potřebují obousměrné unicastové spojení s kontrolou zahlcení, ale nepotřebují spolehlivé doručení datagramů a doručení ve správném pořadí. Jsou to aplikace pro streamování multumédií a hlasové služby.

3.3 IPv6

Tento protokol, původně nazýván IPng (IP next generation), byl navržen jako následník protokolu IPv4 a v roce 1994 byl oficiálně schválen organizací IETF (The Internet Engineering Task Force). Hlavním důvodem jeho definice byla potřeba *zvětšení adresového prostoru*, protože s rozvojem internetu se ukázalo, že adresový prostor, poskytovaný protokolem IPv4, bude zanedlouho vyčerpán. IPv6 adresa má 128 bitů oproti 32 bitům adresy IPv4, což poskytuje obrovské množství jedinečných IP adres a vyřeší se tak na velmi dlouhou dobu problém s nedostatkem adres. Ačkoliv nedostatek IPv4 adres měl značně urychlit nástup IPv6, tak se tomu doposud nestalo díky implementaci technik CIDR[3] a NAT[4], které šetří adresový prostor a oddalují tak nevyhnutelný nástup nového protokolu.

[3]CIDR - Classless Inter-Domain Routing je způsob interpretace IP adres popsaný v RFC 1518 a RFC 1519, který nahradil předchozí syntaxi IP adres založenou na třídách. CIDR přináší rozdělení velkých sítí na podsítě, čímž je umožněno efektivnější využití adresového prostoru a je ulehčeno směrovačům.

[4]NAT - Network Address Translation je technika překladu veřejné IP adresy na neveřejné, která umožňuje přistupovat počítačům v privátní síti do Internetu. Tato technika šetří adresový prostor a také chrání počítače ve vnitřní síti před útokem z internetu. Na druhé straně ale omezuje funkčnost mnoha služeb, zejména těch, které vyžadují umožnění spojení z internetu, nebo služeb využívajících bezstavové protokoly.

Tento nový protokol přinese ovšem mnohem víc, než jen větší adresový prostor. IPv6 poskytne oproti IPv4 model, kdy se každý klient sítě bude moci spojit s jakýmkoliv dalším klientem bez nejrůznějších omezení, která vyvstávají zejména při použití překladu adres. Při návrhu protokolu byl brán velký ohled na možnosti řízení kvality služeb, což je další současná slabina IPv4, který původně s tímto problémem nepočítal. Je zjednodušena hlavička IPv6 paketu a routery tak mají ulehčenou práci se zpracováním nových paketů. Dalšími důležitými vlastnostmi je možnost zabezpečení, autentifikace a mobility. Přesná specifikace protokolu je popsána v dokumentu RFC 2460.

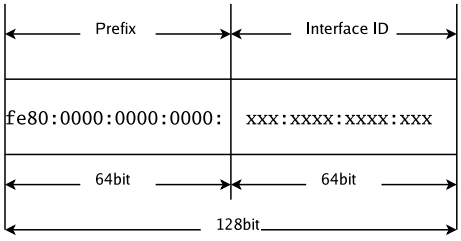
Podpora protokolu IPv6 existuje jak v linuxovém jádře, tak v jádrech systémů *BSD již poměrně dlouhou dobu a tato technologie je již bez problémů použitelná ve všech zmiňovaných systémech. Implementací IPv6 do linuxového jádra se zabývá projekt USAGI (Universal Playground for IPv6) a podpora je začleněna od jádra verze 2.2. *BSD systémy využívají práce projektu KAME a obsahují IPv6 od BSD 4.0. Oba dva tyto vývojové týmy samozřejmě spolupracují a tedy nejsou nijak velké rozdíly v návrhu a konečném řešení podpory pro IPv6 v těchto systémech. Dá se říct, že v této oblasti jsou Linux a *BSD ekvivalentní, a pokud se v některém ze systémů osvědčí nějaké nové postupy a způsoby implementace, tak se většinou importují i do ostatních systémů.

3.3.1 Automatická konfigurace

Další výhodou oproti IPv4 je automatická konfigurace zařízení v síti. Protokol počítá se dvěma způsoby nastavení síťových parametrů. Prvním způsobem je takzvaná *stavová konfigurace*, což vlastně není nic nového, protože se jedná o konfiguraci prostřednictvím DHCPv6[5] serveru nebo PPP, kdy klient vyšle do sítě dotaz a příslušný server mu přidělí všechny potřebné parametry, které potřebuje vědět. Novinkou je *bezstavová konfigurace*[8], která nevyžaduje DHCP server. Je založena na objevování sousedů, kdy klient, který se připojuje do sítě, posílá multicastový požadavek RS (Router Solicitation), na který odpoví router takzvaným RA (Router Advertisement) paketem. Klient nemusí nutně posílat RS paket, ale může si pasivně počkat na RA paket, který směrovač periodicky zasílá. Z odpovědi směrovače se klient dozví prefix identifikující danou síť a sestaví svou IP adresu z tohoto prefixu a identifilátoru rozhraní (většinou 64 bitů), který se jednoznačně vygeneruje z jeho MAC adresy. Pokud je v síti více směrovačů, tak klient používá všechny z nich střídavě a postupně si upravuje svou směrovací tabulku na základě ICMPv6 zpráv generovaných těmito směrovači. Jestliže se klinet připojí do sítě, kde nejsou směrovače, tak je schopen automaticky vygenerovat pouze tzv. link-local adresu, se kterou bude okamžitě schopen komunikovat s ostatními klienty připojenými na stejném segmentu sítě. Tato lokální adresa není směrovatelná do dalších sítí. Jak je vidět na obrázku, tak link-local adresa má daný prefix `fe80`, dalších 48 bitů jsou nuly a část interface ID je vygenerována automaticky podle hardwarové adresy. Na mém počítači je například hardwarová adresa rozhraní eth0 `00:13:D3:9B:E7:17` a z ní vygenerovaná link local adresa `fe80::213:d3ff:fe9b:e717`[6] Proces autokonfigurace se nevztahuje na směrovače, které musí být nastaveny manuálně nebo nějakým jiným způsobem.

[5]DHCP servery jsou aplikace komunikující prostřednictvím protokolu DHCP (Dynamic Host Configuration Protocol) a umožňují zařízením připojených do sítě požádat o přidělení IP adresy a dalších síťových parametrů. DHCP server poté přiděluje zařízením adresy z definovaného rozsahu, který má k dispozici, tak aby nedocházelo ke konfliktům zařízení, kdy má více zařízení stejnou IP adresu.

[6]Pokud se v adrese vyskytne několik po sobě jdoucích nulových skupin, tak je možné je nahradit dvojicí dvojteček. Ta se však v zápisu každé adresy smí objevit jen jednou, aby zápis byl jednoznačný.



Obrázek 3.1: Formát link-local IPv6 adresy

Díky autokonfiguraci je značně ulehčeno vytváření a další propojování sítí. Přináší značné výhody mobilním zařízením, které se musí neustále přihlašovat do jiných sítí a konfigurovat své parametry.

3.3.2 Bezpečnost

Už díky obrovskému rozsahu adres je IPv6 bezpečnější protokol. Výchozí podsíť má 2^{64} adres (přibližně $18 * 10^{18}$), což neumožňuje útočníkům efektivně skenovat celou podsíť. I kdyby byl schopen útočník proskenovat 1 000 000 adres za sekundu, tak by mu trvalo více než 500 000 let, než by nalezl všechny počítače v síti. Útočníci budou muset vyvinout jiné strategie pro získávání IP adres, které ovšem nejspíše nebudou tak přímočaré, jednoduché a rychlé jako současné způsoby.

Díky rozšíření bezstavového protokolu popsaném v RFC 3041 mohou navíc klienti generovat veřejné adresy, které se v čase mění. Změny adresy je dosaženo změnou pole interface identifier, čímž se také samozřejmě změní celá adresa a pro potenciální útočníky je pak mnohem těžší takový stroj identifikovat a napadnout.

IPv6 poskytuje také kryptograficky generované adresy[1]. Kryptograficky generovaná adresa (CGA) je IPv6 adresa získaná prostřednictvím zabezpečeného objevování sousedů (SEND - Secure Neighbour Discovery), pro kterou je vygenerována část *interface identifier* na základě vypočtení hašovací funkce z veřejného klíče a dalších parametrů. Vazba mezi adresou a veřejným klíčem může být ověřena vypočtením hašovací funkce a porovnáním s interface ID. Zprávy zasílané z takovéto adresy mohou být chráněny příslušným soukromým klíčem. Toto zabezpečení ke své funkci nepotřebuje žádnou certifikační autoritu nebo jinou bezpečnostní infrastrukturu.

V IPv6 bylo možné se zbavit protokolu ARP (Address Resolution Protocol), který v IPv4 zajišťuje identifikaci sousedů na linkové vrstvě. Funkce tohoto protokolu je nahrazena několika funkcemi ICMP a dalšími schopnostmi, které jsou dohromady definovány v protokolu objevování sousedů (ND - Neighbour Discovery). Díky absenci ARP bude nemožné provádět útoky typu ARP cache poisioning.[7]

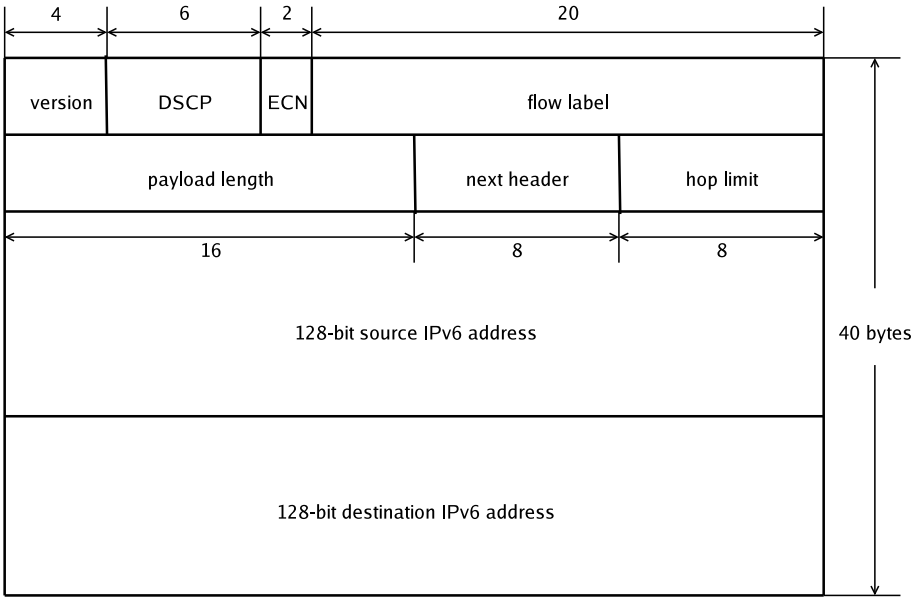
Dalším přínosem pro bezpečnost přenosu dat je sada protokolů IPSec (IP Security) implementovaných přímo v IPv6. IPSec zajišťuje bezpečný přenos paketů na síťové vrstvě a v současné době je používáno především k implementaci virtuálních privátních sítí (VPN). Funkčnost, kterou přináší IPSec je nezbytná pro některé vlastnosti, které IPv6 přímo poskytuje. Jedná se například o podporu mobilních zařízení, kde je vyžadováno autentizované

[7]Útočníkovi se může do sítě podařit vložit nepravou vazbu mezi IP adresou a MAC adresou (ARP paket), čímž docílí toho, že data ze stroje oběti jsou doručena jinam, nebo vůbec nejsou doručena. Situace se stává kritickou, pokud je obětí útoku brána. To může vést k odepření služby pro celou síť (DOS Attack).

spojení mezi zařízením a jeho home-agentem.

3.3.3 Kvalita služeb

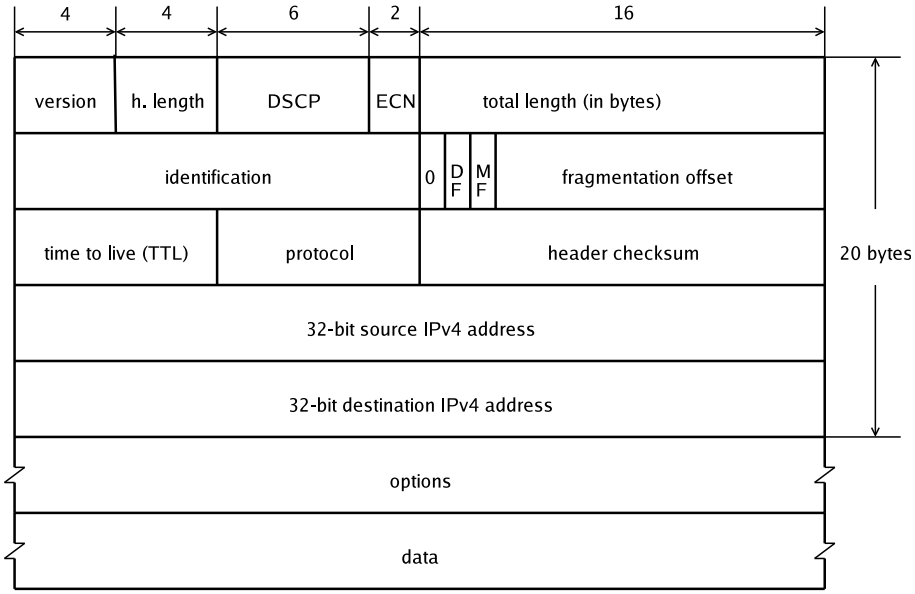
Prvním krokem ke zlepšení kvality služeb, kterou je schopno IPv6 poskytovat, bylo výrazné zjednodušení hlavičky IP paketu oproti IPv4. Tato změna umožní rychlejší zpracování paketů, což bude mít za následek menší nároky kladené na velmi vytížené routery a tím pádem také větší propustnost sítě.



Obrázek 3.2: Formát IPv6 hlavičky

Podle RFC 2474 se v hlavičce paketu nachází pole, které slouží k rozlišení služeb v iternetu a nastavení priority, s jakou bude paket zpracován. Tento model se nazývá *Differentiated Services* a nahrazuje předchozí pole *ToS (Type of Service)* v IPv4 a *traffic class* v IPv6. DS pole definuje *per-hop behavior (PHB)*, což je rozhodnutí o způsobu, jakým bude paket klasifikován a jaká strategie se použije při jeho odeslání. V dokumentu RFC není nařízeno, jakým způsobem mají systémy implementovat takovéto chování. To jak se jednotlivá zařízení postaví k možnosti využít této vlastnosti, závisí tedy pouze na výrobci, případně na administrátorovi systému. To se projevilo jako nevýhoda, protože různé směrovače se mohou k paketům chovat různě a nelze předpovědět chování takového spojení. Z komerčního hlediska je nemožné zajistit různé třídy konektivity na základě tohoto systému, protože to, co jeden poskytovatel může považovat za prioritní provoz, druhý třeba přeřadí do běžného provozu. Tento způsob funguje pouze pokud všichni dodržují stanovená pravidla. Realita je ovšem taková, že některé systémy jednoduše nastaví větší prioritu svému provozu, i když k tomu nemají patřičný důvod.

Do IP protokolů byla také přidána podpora ECN (Explicit Congestion Notification, RFC 3168). Je to způsob signalizace směrovače o zahlcení linky. Díky aktivní správě fronty (např. RED - Random Early Detection) jsou schopny směrovače detekovat zahlcení ještě před tím, než fronta přeteče. A díky ECN už nejsou routery nuceny k zahazování paketů, aby indikovaly zahlcení. Místo zahození paketu mohou nastavit příznak CE (Congestion



Obrázek 3.3: Formát IPv4 hlavičky

Experienced) v IP hlavičce. Díky kombinaci těchto technik jsou omezeny nežádoucí efekty, které vyplývaly z přetečení fronty a výsledkem je několik zlepšení:

- Bude zahazováno menší množství paketů.
- Zvýší se využití linky díky tomu, že bude potřeba méně TCP synchronizace.
- Tím, že se bude udržovat rozumná velikost fronty, se sníží zpoždění a zkreslení v jednotlivých tocích.
- Šířka pásma bude sdílena více rovnocenně mezi jednotlivými toky.

Novinkou oproti IPv4 je 20 bitové pole *flow label* zvolené aplikací nebo jádrem pro daný soket. Flow, neboli tok, je posloupnost paketů vyslaných z určitého zdroje do určitého cíle, pro kterou požaduje zdroj nějaké speciální zacházení příslušnými směrovači. Jakmile je označení toku vybráno, tak se po cestě paketu již nesmí měnit. Pokud je označení rovno nule, což je výchozí hodnota, tak to znamená, že paket nenáleží do žádného toku. Hlavní výhoda označení toku spočívá v tom, že směrovače nemusí zkoumat vnitřek paketů k tomu, aby identifikoval tok, ke kterému paket patří, a to i pokud je použito šifrování. V IPv6 je každý tok jednoznačně identifikován trojicí údajů (cílová adresa, zdrojová adresa a flow label), oproti tomu v IPv4 paketu k takové identifikaci potřebujeme pětici (zdrojová a cílová adresa, protokol, zdrojový a cílový port). Jednoduchá identifikace toků slouží k jejich efektivnímu klasifikování a následnému přiřazení priorit.

V současné době se pracuje na návrhu protokolu[4], který umožní alokaci nezbytných prostředků jednotlivému toku nebo jejich skupině na jejich cestě sítí. Toto signální schéma bude vyžadovat podporu v hardwaru příslušných síťových prvků (nejčastěji směrovačů). Kvalita služby tak bude nastavena v reálném čase bez účasti nějaké externí softwarové signalizační struktury, jakou je například Reservation Protocol (RSVP). Tuto funkčnost bude možné začlenit jak do IPv4, tak IPv6 paketů. U IPv6 bude s výhodou použito pole

next header, které odkáže na další hlavičku, která bude definovat požadavky toku na kvalitu služeb. Výhoda IPv6 spočívá v tom, že dokáže využít tohoto protokolu i při šifrovaných spojeních pomocí IPSec, kdežto s IPv4 to není bez úpravy IPSec možné.

Kapitola 4

Prostředky pro podporu QoS v linuxu

Klíčovými prvky, na kterých je postavena podpora pro řízení kvality služeb v GNU Linuxu jsou tři základní prvky:

- Řadící disciplína - queuing discipline
- Třídy - classes
- Filtry/plánovače/klasifikátory - Filters/Policers/Classifiers

Pakety, přicházející z internetu, spadají přímo do filtru, a odtud jsou přiřazeny příslušné řadící disciplíně, která je následně rozdělí do jednotlivých tříd. Každé síťové zařízení v linuxu vlastní frontu (výchozí frontou je FIFO), která definuje, jak jsou zařazené pakety zpracovávány a ke každé frontě je přiřazena třída. Řadící disciplíny mohou být různých typů. Mezi nejpoužívanější patří:

- Class Based Queuing (CBQ)
- Hiearchical Token Bucket (HTB)
- First In First Out (FIFO)
- Traffic Equalizer (TEQL)
- Stochastic Fair Queuing (SFQ)
- Random Early Detection
- Generalized RED (GRED)
- Priority
- Hierarchical Fair Service Curve (HFSC)

Řazení paketů do front a následná obsluha těchto front s definovanou prioritou nebo propustností je velice důležitým a používaným způsobem zajištění požadované kvality služeb. Toto řazení se děje na úrovni síťové vrstvy podle schématu ISO/OSI a je navrženo tak, že jakmile pakety odesílané z počítače vstupují do síťové vrstvy, tak je řadící mechanismy

přítomné v jádře operačního systému zařadí do front, kde čekají na další zpracování. Modifikací řadících strategií můžeme dosáhnout rovnocenného sdílení šířky pásma mezi různými aplikacemi, uživateli nebo počítači.

Takový způsob ošetření sdílení linky je ale použitelný pouze v *odchozím* směru. Jakmile totiž paket dorazí na síťové rozhraní v příchozím směru, tak je pozdě jej řadit do nějakých front, protože už spotřeboval určitou kapacitu pásma k tomu, že byl přijat vstupním rozhraním. Řešením tohoto problému je nastavení řazení na nadřazeném routeru, a to na vnitřním rozhraní, kde pakety opouštějí router a směřují do vnitřní sítě.

Subsystém, který obstarává řízení toku dat, je postaven na různých modulech, které se zavádějí do jádra operačního systému a uživatelských nástrojů jako jsou ip, iptables a tc. Dohromady jsou tyto programy schopny vytvořit velice komplexní systém pro zajištění kvality různých služeb nabízených na serverech nebo rovnocenné postavení zákazníků ISP.

Klíčovou roli v definování již známých front hraje program *tc*. Pomocí tc se také přiřazují jednotlivé třídy k frontám a můžeme ho též využít k přiřazení paketů do tříd. Příkazem `tc` bez parametrů vyvoláme syntaxi tohoto příkazu:

```
# tc
Usage: tc [ OPTIONS ] OBJECT { COMMAND | help }
where  OBJECT := { qdisc | class | filter | action }
       OPTIONS := { -s[tatistics] | -d[etails] | -r[aw] | -b[atch] file }
```

Každé síťové rozhraní má definován nějaký řadící mechanismus paketů, který určuje, jakým způsobem budou zpracovány. V linuxu se tento mechanismus nazývá qdisc (queueing discipline) a může být zobrazen příkazem `ip link show`. Výchozím algoritmem pro řazení paketů je pfifo_fast, který funguje na principu fronty FIFO. Tuto výchozí hodnotu můžeme změnit právě použitím nástroje `tc`:

```
tc qdisc add dev eth0 root handle 1:0 htb
```

Tímto příkazem jsme předefinovali výchozí řadící algoritmus na HTB[1] na rozhraní eth0. Handle je uživatelem specifikované číslo ve tvaru MAJOR:MINOR. MINOR číslo každého plánovače (qdisc) musí být nula.

Kořenový qdisc můžeme chápat jako hlavní kontejner, do kterého spadá všechen provoz. Nad tímto kořenem můžeme vystavět potřebnou strukturu tříd, do kterých rozdělíme požadované typy provozu tak, aby vyhovovaly daným požadavkům:

```
tc class add dev eth0 parent 1:0 classid 1:1 htb rate 512kbit

tc class add dev eth0 parent 1:1 classid 1:20 htb \
        rate 100kbit ceil 100kbit prio 0
tc class add dev eth0 parent 1:1 classid 1:21 htb \
        rate 300kbit ceil 512kbit prio 1
tc class add dev eth0 parent 1:1 classid 1:22 htb \
        rate 100kbit ceil 512kbit prio 2

tc qdisc add dev eth0 parent 1:20 handle 20:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:21 handle 21:0 sfq perturb 10
tc qdisc add dev eth0 parent 1:22 handle 21:0 sfq perturb 10
```

[1] HTB - Hierarchical Token Bucket. Podrobné informace o tomto plánovači a jeho implementaci lze nalézt v dokumentaci na stránkách projektu http://luxik.cdi.cz/ devik/qos/htb/ .

V tomto příkladě jsme tedy definovali třídu, jejíž propustnost je 512 kilobitů za sekundu a následně ji rozdělili na tři třídy s různou prioritou a kapacitou. Čím menší je parametr `prio`, tím je priorita větší. Parametr `rate` určuje minimální propustnost třídy a parametr `ceil` její maximální propustnost. Navíc byl ke každé koncové třídě přidán SFQ[2] plánovač, který zajistí rovnocenné postavení jednotlivých spojení spadajících do této třídy, obzvláště je-li třída plně vytížená.

Nakonec je třeba zajistit správné rozdělení paketů do jednotlivých tříd. Zde se naskýtá více možností: Jednou z nich je přímé použití iptables a cíle CLASSIFY, další možností je použití cíle MARK v iptables pro označení jednotlivých paketů a následné přiřazení do tříd programem `tc`.

```
iptables -t mangle -A POSTROUTING -o eth0 -p tcp --sport 22 \
        -j MARK --set-mark 20
tc filter add dev eth0 protocol ip parent 1:0 handle 20 fw flowid 1:20
```

Poslední možností je využití u32 selektoru, který analyzuje hlavičku paketu a podle hodnoty určitých bitů nastavuje příslušnou třídu. Použití tohoto nástroje vyžaduje detailní znalost struktury hlavičky paketu. Příklady použití tohoto filtru lze najít na internetu[5].

4.1 Iptables - paketový filter v Linuxu

Linuxová jádra měla paketový filtr již od verze 1.1. Tato první verze byla založena na filtru ipfw z BSD Unixu a byla portována do Linuxu v roce 1997 Alanem Coxem. V Linuxu řady 2.0 byl tento filtr vylepšen a vznikl nástroj *ipwadm*, který nastavoval jádro v oblasti filtrování paketů.

V jádrech řady 2.2 byl uveden nástroj *ipchains*, který přinesl možnost definování vlastních řetězců a jejich struktur ke standardním zabudovaným řetězcům (chains) *input, output a forward*. Každý paket procházející routerem musel projít alespoň třemi základními implicitními řetězci.

Poslední verzí paketového filtru v Linuxu jsou *iptables*. Tento filtr je standardně v jádře od verze 2.4 a v současných jádrech 2.6 je jediným podporovaným filtrem. Velký rozdíl oproti ipchains spočívá v použití řetězců. Pakety, které procházejí routerem (ty které nejsou určeny pro router samotný), projdou pouze přes řetězec FORWARD. Tuto skutečnost je třeba mít na paměti a na takovém routeru, který často bývá i firewallem, je nutno mít dvě sady pravidel, a to jednak pro pakety určené přímo pro router a dále pro pakety, které budou přeposílány do vnitřní sítě.

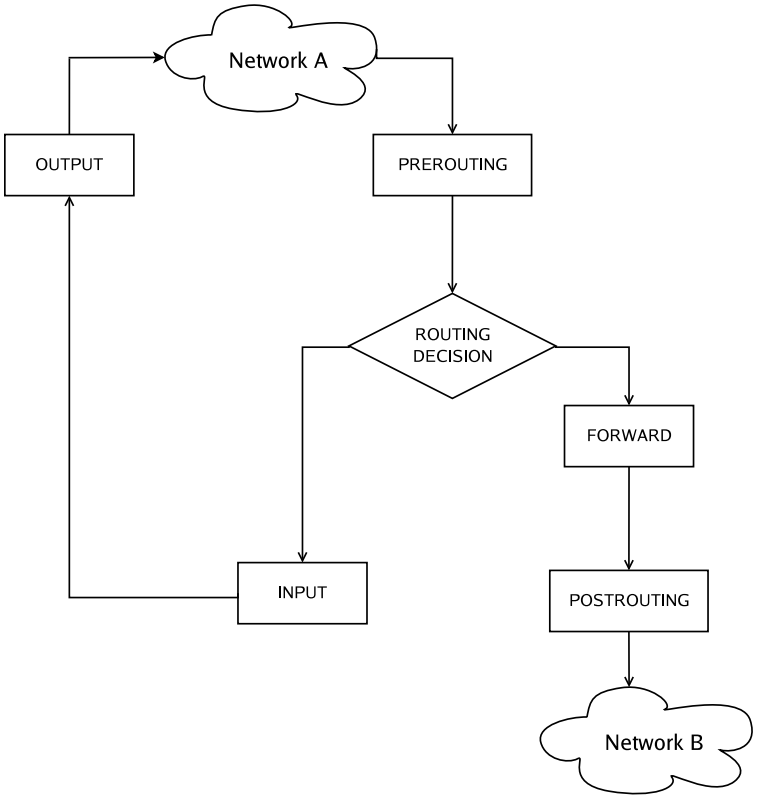
Syntaxe programu Iptables je díky jeho komplexnosti poměrně složitá a pro podrobné informace je vhodné nahlédnout do manuálové stránky programu, která by měla být dostupná na každém počítači s nainstalovaným programem Iptables.

4.1.1 Syntaxe a použití iptables

Pravidla, která určují, jak mají být pakety filtrovány jádrem operačního systému, se definují příkazem `iptables` a následujícími parametry:

- Typ paketu - určuje typ paketu, který bude pravidlo filtrovat (TCP, UDP, ICMP).

[2]SFQ - Stochastic Fair Queueing. V šesté kapitole Traffic Control HOWTO[2] jsou popsány beztřídní (classless) plánovače s dalšími příklady.



Obrázek 4.1: Cesta paketu jádrem linuxu

- Původ nebo cíl paketu
- Cíl - určuje, jaká akce se provede nad paketem, který vyhovuje kritériím definovaných v pravidle.

Silnou stránkou iptables je možnost použití více tabulek k rozhodnutí o osudu příslušného paketu v závislosti na požadované akci, která se má s příslušným paketem provést a jeho typu. Výchozí tabulka se jmenuje *filter* a obsahuje tři zabudované řetězce:

- INPUT pro pakety přicházející na počítač samotný,
- OUTPUT pro lokálně generované pakety opouštějící počítač,
- FORWARD pro pakety routované skrze počítač.

Tato tabulka se použije v případě, že není specifikována explicitně žádná jiná tabulka.

Iptables ale implicitně obsahují další dvě tabulky, které mají svou specifickou úlohu. Tabulka *nat* může být použita k modifikování zdrojové nebo cílové adresy zaznamenané v paketu. Tato tabulka obsahuje tři řetězce:

- PREROUTING pro pozměňování paketů ihned, jakmile vstoupí přes síťové rozhraní,
- OUTPUT pro pozměňování lokálně generovaných paketů před routováním,
- POSTROUTING pro pozměňování paketů, které vystupují přes síťové rozhraní.

Další tabulkou je *mangle*, která se používá pro různé další speciální manipulace s paketem a obsahuje všechny[3] dosud zmíněné řetězce, jejichž použití je na stejných místech cesty paketu jako u předchozích tabulek. Iptables také umožňují vytvořit si další řetězce pro každou z tabulek.
Většina příkazů vložení pravidla má následující strukturu:

```
iptables [-t <table-name>] <command> <chain-name> <parameter 1> \
         <option 1> <parameter n> <option n>
```

Volba `<table-name>` umožňuje vybrat jinou tabulku než implicitní filter. Volba `<command>`[4] je centrum příkazu, kde se specifikuje, jaká akce se provede, například přidání nebo smazání pravidla z příslušného řetězce, který je uveden volbou `<chain-name>`. Za `<chain-name>` jsou dvojice parametrů a voleb, které určují co se stane, jakmile paket vyhoví pravidlu. Pro úplný přehled struktury příkazu `iptables` stačí zadat příkaz `iptables -h`.

[3]Tato tabulka původně obsahovala jen dva řetězce a to PREROUTING a OUTPUT a to až do verze jádra 2.4.17. Od verze 2.4.18 obsahuje také zbývající tři řetězce: INPUT, FORWARD a POSTROUTING.

[4]V manuálové stránce lze nalézt kompletní popis příkazů a parametrů, zde si postupně ukážeme pouze ty, které budou použity v příkladech.

Kapitola 5

Řízení kvality služeb pomocí klasifikace datových toků

Většina současných řešení, které se zabývají zajištěním kvality služeb v síťovém prostředí, jsou založena na principu klasifikace jednotlivých paketů a následným přiřazováním různých priorit. Tato řešení pracují většinou na úrovni síťové nebo aplikační vrstvy.

Pokud je klasifikace paketů prováděna na úrovni síťové vrstvy, tak jedinými informacemi, které slouží pro rozřazení paketů do jednotlivých front s definovanou prioritou, případně i propustností, jsou data uložená v hlavičce paketu. Jsou to zejména čísla portů, případně IP adresy. Při klasifikaci podle čísel portů je možné úspěšně rozlišit pouze omezené množství služeb, které mají své ustálené číslo portu. [1] Některé typy aplikací ale používají náhodná čísla portů a v tomto případě je není možno dále rozlišit, což má většinou za následek to, že skončí ve stejné třídě bez ohledu na to, zda se jedná o aplikaci vyžadující prioritní zpracování (např. VoIP) nebo aplikaci, která je z hlediska zajištění kvality služeb neprioritní (např. P2P). Může být použito i pole Type of Service, ale toto řešení také není spolehlivé, protože velké množství aplikací, případně i některé operační systémy naprosto ignorují správné nastavení tohoto pole v hlavičce paketu. Navíc toto pole může být libovolně změněno každým routerem, kterým paket prochází a nikdy není zaručeno, že hodnota zde nastavená opravdu odpovídá požadovanému způsobu zpracování paketu.

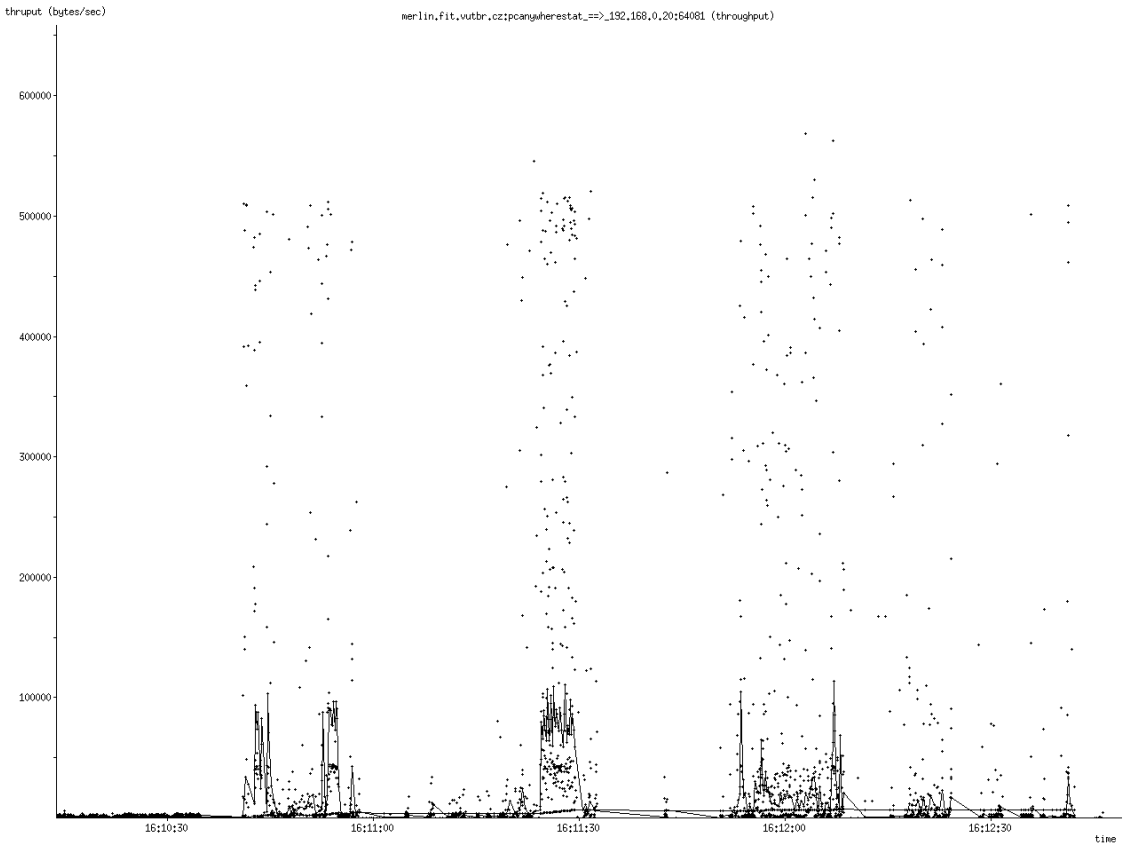
Další možností, jak klasifikovat pakety, je inspekce paketu na úrovni aplikační vrstvy, kdy jsou zkoumána přenášená data, která jsou porovnávána oproti databázi typických řetězců vyskytujících se v datech známých služeb. Toto řešení nejen že je poměrně výpočetně náročné, ale už ze svého principu nedokáže správně klasifikovat jakýkoliv šifrovaný provoz. Problém může nastat taky v případě, že se objeví nějaká nová služba, která není zahrnuta v databází známých služeb a tím pádem mohou být takovéto neznámé pakety špatně klasifikovány.

Ani kombinací obou předešlých metod nelze ve všech případech zajistit správné rozřazení paketů a následné přiřazení priorit. Na klasifikaci síťového provozu lze ale nahlédnout i z jiné strany. Můžeme upustit od snahy přiřadit každému paketu prioritu pouze na základě informací získaných pouze z dat, která jsme schopni vyčíst z jeho hlavičky a těla. Sledujeme-li síťový provoz, který jednotlivé aplikace generují, tak na základě jeho charakteristiky jsme schopni odhadnout, do jaké skupiny daná aplikace patří.

Každá aplikace komunikující přes síť vytvoří spojení, které je jednoznačně identifikovatelné čtveřicí údajů: zdrojová IP adresa, zdrojový port, cílová IP adresa, cílový port.

[1] Většinou se vychází ze souboru /etc/services

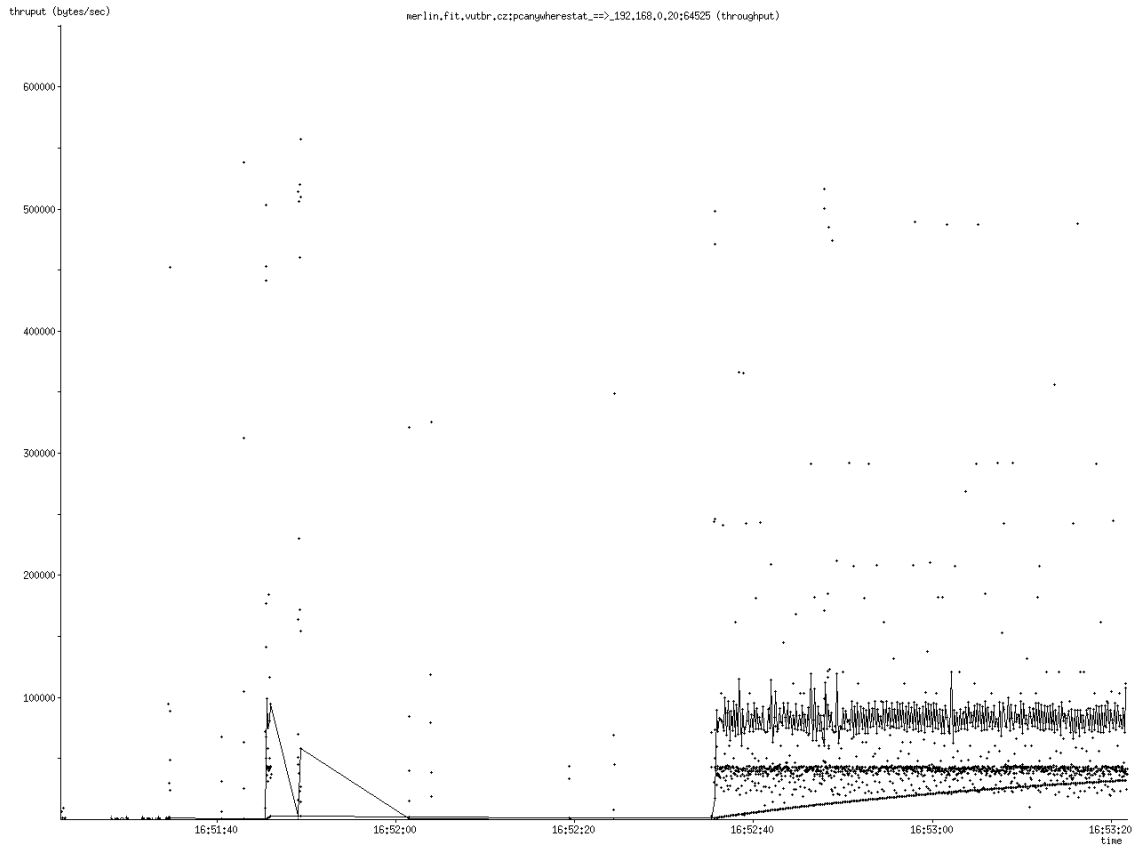
Na základě těchto informací jsme schopni rozřadit všechny pakety do jednotlivých spojení a tím pádem přesně sledovat charakteristiku datového toku, který generují. Stěžejní informací, podle které lze odhadnout o jaký typ aplikace se jedná, je závislost rychlosti přenášených dat na čase. Tyto charakteristiky se u některých aplikací mohou dokonce v čase změnit tak, že interaktivní aplikace přejde do neinteraktivního módu. Pokud tedy jednotlivé datové toky periodicky sledujeme a klasifikujeme, tak i takovou změnu je možno detekovat a náležitě se podle toho zachovat přiřazením jiné priority. Další výhodou tohoto řešení je schopnost správně klasifikovat i šifrovaný provoz.



Obrázek 5.1: Interaktivní provoz přes SSH

Na obrázku 5.1 je znázorněna situace, kdy je přes SSH vzdáleně spuštěna grafická aplikace, se kterou je následně běžně pracováno. Toto je příklad datového toku, který potřebuje větší prioritu, aby bylo uživateli co možná nejlépe umožněno pracovat s aplikací opravdu interaktivně. Na obrázku 5.2 je naopak znázorněno spojení přes SSH, kdy po chvíli běžné interaktivní práce, jakou bylo procházení adresářů, byl spuštěn přenos velkého souboru. Od této chvíle spojení přestává mít interaktivní charakter, protože uživatel čeká na přenesení souboru. Z grafu je jasně patrná změna charakteristiky takového datového toku.

Jak bylo ukázáno na obrázku, tak lze i z charakteristiky přenosu dat v jednom směru odvodit o jaký typ datového toku se jedná. Můžeme ale využít skutečnosti, že pro úspěšnou komunikaci je většinou potřeba přenášet data v obou směrech. Jedná se například o potvrzování přijatých dat při stahování souboru přes protokol http nebo ftp. Jsme tedy schopni sestavit graf, na kterém budou zachyceny charakteristky jak v příchozím, tak v odchozím



Obrázek 5.2: Neinteraktivní provoz přes SSH

směru, čímž dosáhneme zvýšení informační hodnoty takového grafu a zjednodušíme klasifikaci provozu, který daná apliace generuje.

5.1 Návrh systému pro řízení QoS pomocí analýzy datových toků

Tokem budeme rozumět řadu po sobě jdoucích paketů, které se shodují ve čtyřech základních atributech: zdrojová IP adresa, zdrojový port, cílová IP adresa, cílový port. Spojení bude považováno za ukončené, pokud uběhla předem určená doba od přijetí posledního paketu v daném spojení, a to jak v příchozím, tak odchozím směru. Abychom mohli analyzovat jednotlivé datové toky na určitém rozhraní (Ethernet), tak musíme získat všechny pakety přes toto zařízení procházející. Tuto práci obstarává modul aplikace nazvaný *packet sniffer*.

Při analýze problému získávání paketů se jako nejjednoduší, a zároveň z pohledu systému nejbezpečnější řešení nabízí využít knihovnu, která bude kopírovat pakety procházejícími přes síťová rozhraní do uživatelského prostoru samotné aplikace. Knihovna by současně měla být multiplatformní, aby bylo umožněno portovat aplikaci do prostředí jiného operačního systému. Tyto požadavky splňuje knihovna *pcap - Packet Capture library*.[2] Implementace této knihovny je dostupná pro systémy unixového typu (Linux, *BSD systémy, Solaris a

[2]http://www.tcpdump.org/

další) a je známa jako *libpcap*. Existuje také implementace pro Win32 systémy známa jako *WinPcap*.

Získané pakety je nutno v následujícím modulu analyzovat na základě informací obsažených v IP hlavičce a rozřadit do jednotlivých datových toků. Vhodnou datovou strukturou pro ukládání datových toků se jeví kombinace asociativního pole a kruhového bufferu, kdy v kruhovém bufferu budou uloženy zachycené pakety a klíčem do asociativného pole bude struktura jednoznačně identifikující jednotlivé datové toky.

Další modul pracuje nad datovou strukturou obsahující jednotlivé toky a periodicky provádí vyhodnocení rychlosti přenášených dat v čase a vytváří *vektor dat* obsahující informace, na jejichž základě bude provedena klasifikace do jednotlivých tříd reprezentujících typ provozu.

Pro to, abychom mohli určit typ provozu, reprezentovaný vektorem dat zíkaným v předchozím modulu je potřeba se „podívat“ na jeho tvar a porovnat jej s empiricky zjištěnými vektory jednotlivých typů a vybrat třídu provozu, které se nejvíc podobá. K tomuto účelu se dá s výhodou využít možností neuronových sítí a jejich schopnosti naučit se podle trénovací množiny vzorků rozpoznávat vzorky jim podobné.

Samotnému výběru implementace neuronové sítě předcházelo prozkoumání možností a výkonnosti různých knihoven poskytujících rozhraní pro práci s neuronovými sítěmi. Nejdůležitějšími kritérii pro výběr vhodné implementace byly především:

Otevřený zdrojový kód - Svobodná licence knihovny nejen že umožňuje integraci s dalšími open source knihovnami, ale zaručuje také možnost případného rozšiřování jejich funkcí a podrobné studium problematiky neuronových sítí. Dostupnost zdrojového kódu také umožňuje experimentování a další výzkum na akademické úrovni.

Multiplatformnost - Možnosti použití softwaru výrazně rostou, je-li možno jej zkompilovat a provozovat na různých hardwarových platformách a pod různými operačními systémy.

Vazby na různé programovací jazyky - V případě dalšího rozdělení funkcí programu do na sobě nezávislých částí, které mezi sebou komunikují není nutné používat pouze jeden programovací jazyk. Možnost využít knihovny ve spojení s jinými programovacími jazyky může přinést zjednodušení při vývoji dalších rozšíření programu, nebo při jeho integraci do jiného již existujícího systému.

Rychlost a efektivnost - Knihovna by měla poskytovat nejen velikou míru abstrakce pro koncového uživatele, ale měla by současně používat vysoce výkonných algoritmů, čímž bude využitelná pro širokou škálu aplikací a architektur.

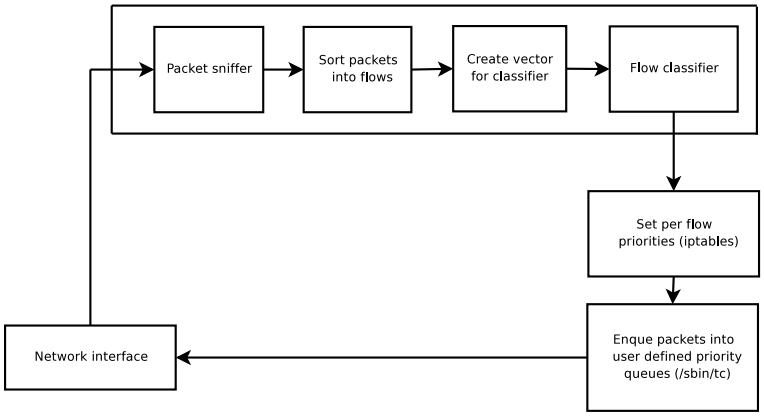
Výše zmíněné požadavky dobře splňuje knihovna Fast Artificial Neural Network Library (FANN).[3] S pomocí této knihovny bude sestavena plně propojená dopředná neuronová síť s jednou skrytou vrstvou. Počet vstupních neuronů se bude rovnat velikosti klasifikovaného vektoru a počet výstupních neuronů bude udávat počet tříd, do kterých bude neuronová síť rozdělovat datové toky.

Rozpoznaný typ a popis datového přenosu jsou vstupem pro další modul, který pomocí paketového filtru *iptables* nastaví příslušnou prioritu tokům dat v příchozím i odchozím směru spojení. Zde již není jiná možnost, než použití systémově závislé implementace, protože v současnosti neexistuje a ani nejspíše nebude existovat jednotné rozhraní nebo

[3] http://leenissen.dk/fann/index.php?p=gsoc.php

nástroj, který umožní nastavení parametrů jádra pro zpracovávání paketů. Tento modul je ovšem navržen tak, že díky použití uživatelského rozhraní iptables není jeho implementace nijak složitá a portace modulu by znamenala jej pozměnit tak, aby využíval jiného programu (například *pf* pro OpenBSD a FreeBSD).

Poslední a také systémově závislou částí systému je skript, který nad síťovým rozhraním definuje pevně dané schéma řadících disciplín. Tento skript nastaví parametry jádra operačního systému, který potom bude řadit pakety do prioritních front na základě informace vložené do paketu pomocí programu iptables. V linuxu k tomuto účelu existuje uživatelský nástroj *tc*. V *BSD systémech se ke stejnému účelu používá například nástroje *altq*. Pokud bychom chtěli portovat systém pod jiný operační systém, tak je nutné přepsat tento skript tak, aby využíval nástroje v tomto systému dostupné.



Obrázek 5.3: Schéma navrženého systému pro řízení QoS

Kapitola 6

Implementace navrženého systému

6.1 Zachytávání paketů

Pro zachytávání paketů procházejících přes definované síťové rozhraní je použita knihovna *pcap*, která poskytuje jednoduché a efektivní rozhraní pro získávání paketů.[1] Tímto způsobem mohou být získány všechny pakety na síti, a to i ty, které jsou určeny pro jiné počítače (procházející routerem).

K získání paketu je použita funkce pcap_open_live():

```
pcap_t *pcap_open_live(char *device, int snaplen, int promisc, int to_ms,
    char *ebuf)
```

Prvním parametrem *device* je řetězec specifikující síťové zařízení, které se má otevřít. Jestliže je tento parametr nastaven na „any“ nebo NULL, tak budeme získávat pakety ze všech dostupných rozhraní. Jméno síťového rozhraní je předáváno programu jako první parametr z příkazové řádky. Druhý parametr *snaplen* určuje kolik bytů z paketu se maximálně získá. Tento parametr je v programu nastaven na pevnou hodnotu. Pokud bychom nechtěli analyzovat i data paketu (payload), tak jej lze nastavit tak, aby byla vždy zachycena alespoň hlavička paketu. Parametr *promisc* může být true nebo false a určuje, zda má být zařízení uvedeno do promiskuitního módu[2]. V programu je tato hodnota nastavena na 1 (true). Další parametr *to_ms* specifikuje timeout v milisekundách. Tato hodnota určuje dobu, po kterou budou čteny pakety ze síťového rozhraní. Čtení tedy neskončí po prvním zachyceném paketu, ale načte se větší množství paketů v jedné operaci. V programu je nastaven timeout na jednu sekundu. Posledním parametrem je pole, do kterého se ukládá chyba nebo varování. Pokud funkce pcap_open_live() skončí neúspěšně, tak vrátí NULL, pokud uspěje, tak vrátí proměnnou *handle*.

Dalším klíčovým bodem je použití funkce pcap_loop():

```
int pcap_loop(pcap_t *p, int cnt, pcap_handler callback, u_char *user)
```

Prvním parametrem je *session handler* získaný funkcí pcap_open_live(), druhým parametrem je celé číslo udávající, kolik paketů se má načíst. Pokud zadáme zápornou hodnotu, tak funkci říkáme, že má číst, dokud nenastane chyba. V programu je nastavena hodnota -1. Třetím parametrem je jméno callback funkce, které je vždy předán získaný paket a ta

[1] Vzorem pro vypracování této části programu byl článek Programming with pcap.[3]

[2] Síťové zařízení pracující v promiskuitním módu zachytává všechny rámce na síti a ne pouze rámce, které jsou mu přímo adresované. Pro uvedení zařízení do tohoto módu jsou nutná superuživatelská práva.

jej dále zpracuje. Tato funkce je vstupním bodem do druhého modulu rozřazování paketů. Poslední parametr může sloužit k předání dalších parametrů callback funkci kromě těch, které již předává funkce pcap_loop(). V našem případě je nastavena na NULL, což znamená, že nepředáváme žádné další parametry.

Prototyp naší callback funkce musí být definován tak, aby funkce pcap_loop() věděla, jak ji použít. Nemůžeme ji tedy definovat libovolně:

```
void get_packet(u_char *args, const struct pcap_pkthdr *header,
    const u_char *packet);
```

První parametr funkce koresponduje s posledním parametrem funkce pcap_loop(), a tedy vše, co je uloženo v posledním parametru pcap_loop(), je předáno jako první parametr callback funkci. Dalším parametrem je pcap hlavička, která obsahuje důležité informace o času získání paketu a jeho délce. Tato struktura je definována v souboru `pcap.h` jako:

```
struct pcap_pkthdr {
        struct timeval ts; /* time stamp */
        bpf_u_int32 caplen; /* length of portion present */
        bpf_u_int32 len; /* length this packet (off wire) */
        };
```

Posledním parametrem jsou data paketu, ze kterých vhodným přetypováním můžeme získávat důležité struktury, jako jsou IP a TCP hlavička, případně UDP hlavička. Z těchto struktur jsou následně čteny informace sloužící k rozřazení do jednotlivých spojení.

6.2 Rozřazení paketů do jednotlivých toků

Jak již bylo řečeno, pro každý zachycený paket se volá funkce, která jej zpracuje a zařadí do datové struktury reprezentující datové toky přes určité rozhraní. Tato funkce se jmenuje get_packet() a je definována v souboru `get_packet.c`. Na začátku funkce jsou získány hlavičky paketu pomocí přetypování části surových dat, předaných jako data paketu, na příslušné datové struktury, které je reprezentují a mají pevný formát. Například pro získání IP hlavičky paketu je potřeba provést následující:

```
/* define/compute ip header offset */
ip = (struct sniff_ip*)(packet + SIZE_ETHERNET);
```

Nejprve je nutno určit, kde přesně se požadovaná data nacházejí. Každý paket na ethernetovém rozhraní začíná ethernet hlavičkou, která má pevně danou velikost 14 bytů, což je konstanta SIZE_ETHERNET. Ihned za touto hlavičkou již následuje IP hlavička, kterou je potřeba získat. Formát IP hlavičky je také pevně dán a odpovídá struktuře `sniff_ip`:

```
/* IP header */
    struct sniff_ip {
        u_char ip_vhl;        /* version << 4 | header length >> 2 */
        u_char ip_tos;        /* type of service */
        u_short ip_len;       /* total length */
        u_short ip_id;        /* identification */
        u_short ip_off;       /* fragment offset field */
    #define IP_RF 0x8000      /* reserved fragment flag */
```

```
    #define IP_DF 0x4000      /* dont fragment flag */
    #define IP_MF 0x2000      /* more fragments flag */
    #define IP_OFFMASK 0x1fff /* mask for fragmenting bits */
        u_char ip_ttl;        /* time to live */
        u_char ip_p;          /* protocol */
        u_short ip_sum;       /* checksum */
        struct in_addr ip_src,ip_dst; /* source and dest address */
};
```

Jelikož ve světě počítačových sítí existuje mnoho různých protokolů a linkových rozhraní, je nutné se ujistit, že pracujeme opravdu s rozhraním *ethernet* a zachytáváme IP pakety. Tyto důležité služby opět zajišťuje knihovna pcap. Ihned po otevření síťového rozhraní otestujeme pomocí funkce *pcap_datalink()*, zda se nacházíme na ethernetovém rozhraní a poté aplikujeme funkcemi *pcap_compile()* a *pcap_setfilter()* filtr,[3]který vybere pouze IP protokol.

Pokud tedy víme, že IP hlavička začíná přesně za čtrnáctým bytem od začátku paketu, tak přetypováním dat od tohoto místa na strukturu IP hlavičky, získáme požadovaná data. Obdobným způsobem jsou získávána všechna ostatní data z paketu.

Z každého paketu budeme získávat data popsaná strukturou `rb_data`:

```
struct rb_data{
    struct  in_addr s_ip;  /* source ip address */
    uint16_t s_port;       /* source port */
    struct  in_addr d_ip;  /* destination ip address */
    uint16_t d_port;       /* destination port */
    struct timeval ts;     /* timestamp from pcap */
    uint32_t len;          /* packet length */
    unsigned char protocol;/* Protocol (TCP or UDP)*/
};
```

Každý datový tok je reprezentován kruhovým bufferem[4] těchto datových struktur. Jednotlivé kruhové buffery jsou dále zařazeny do asociativního pole[5], kde klíčem ke každému toku je čtveřice údajů, které jej jednoznačně identifikují:

```
struct key {
uint32_t one_ip; uint32_t two_ip; uint16_t one_port; uint16_t two_port;
};
```

Toto asociativní pole je globální a přístupné pro ostatní funkce, zejména pak pro funkce modulu, který z jednotlivých datových toků vytvoří charakteristické vektory.

6.3 Vytvoření charakteristického vektoru

Prvním krokem pro vytvoření závislosti rychlosti přenášení dat na čase jednotlivých toků je zpracování dat uložených do asociativního pole toků v předchozím modulu. Tuto a další

[3]Podrobný popis, jak takový filtr sestavit lze nalézt v manuálových stránkách knihovny pcap. Příklady různých filtrů jsou k dispozici například na internetových stránkách projektu Ethereal - http://wiki.ethereal.com/CaptureFilters.

[4]Implementace kruhového bufferu byla převzata z projektu JACK
http://jackit.sourceforge.net/cgi-bin/lxr/http/source/

[5]Implementace asociativního pole byla převzata z projektu Christophera Clarka,
http://www.cl.cam.ac.uk/ cwc22/hashtable/

úlohy spojené se zpracováním toků řeší funkce definované v souboru *compute_flows.c*. Prototypy těchto funkcí, spolu s definicemi konstant a datových struktur jsou uloženy v souboru *compute_flows.h*. Jako první je tedy volána funkce:

```
int compute_flows(struct hashtable *h, struct hashtable *flow_chars);
```

Prvním parametrem funkce je ukazatel na asociativní pole, kde jsou roztřízeny pakety do jednotlivých datových toků. Druhým parametrem je ukazatel na asociativní pole, kam bude funkce ukládat vypočtenou rychlost přenosu dat v jednotlivých intervalech. Funkce *compute_flows()* postupuje tak, že postupně čte všechny toky uložené v tabulce *h* a pro každý paket daného toku vyhodnocuje, zda-li patří do sledovaného intervalu, či nikoliv. Dokud pakety do intervalu náleží, tak se postupně sčítá jejich velikost. Jakmile je zjištěn paket, který patří do následujícího intervalu, tak je posunuta hodnota proměnné *interval_beg*, která byla na počátku čtení nastavena na čas příchodu prvního přečteného paketu daného toku. Pro operace s časovými razítky jsou definovány pomocné funkce:

```
int tv_diff(struct timeval *first, struct timeval second);

void tv_add_ms(struct timeval *tv, int ms);
```

První vrací rozdíl v milisekundách mezi dvěma časovými razítky a druhá přičítá požadovaný počet milisekund k časovému razítku. Pokud náhodou nastane situace, že mezi dvěma po sobě jdoucími pakety je rodíl větší než je délka intervalu, tak funkce doplní do intervalů, kde nebyl přenesen žádný paket, nulové hodnoty. Výsledkem této funkce je tedy opět globální asociativní pole obsahující všechny datové toky, ale tentokrát je datový tok charakterizován kruhovým bufferem struktury:

```
/*structure which contains bandwidth characteristics of a flow*/
struct rb_flow_data {
struct  in_addr s_ip;
uint16_t s_port;
struct  in_addr d_ip;
uint16_t d_port;
double band;
    unsigned char protocol;
};
```

V této struktuře je důležitá především hodnota *band*, která obsahuje počet přenesených bytů za jeden časový interval. Časový interval je definován v souboru `compute_flows.h` jako konstanta INTERVAL a je nastaven na hodnotu 1000ms, tedy jedna sekunda.

Jakmile je vytvořena tabulka obsahující časové charakteristiky rychlosti přenosu dat, tak je nutné sestavit vektor o přesně definované délce, který bude v následujícím modulu klasifikován neuronovou sítí. Velikost tohoto vektoru je definována jako hodnota `NUM_OF_SAMPLES` na hodnotu 30. Tato hodnota musí být vždy sudá, protože výsledný vektor bude sestaven z obou směrů datového spojení. Vektor je sestavován funkcí:

```
classify_flows(struct hashtable *flow_chars);
```

Jejím parametrem je ukazatel na asociativní pole vytvořené funkcí *compute_flows()*. Toto pole je pak procházeno prvek po prvku a jakmile je nalezeno spojení, které pochází

z vnitřní sítě, do které router směruje provoz z internetu nebo jiné vnější sítě, tak jsou podle ip adres a portů sestaveny dva klíče do tabulky *flow_chars* a popis zpracovávaného datového toku, který je uložen do řetězce *flowdesc* ve tvaru:

```
zdrojová_ip_adresa:zdrojový_port-cílová_ip_adresa:cílový_port
```

První klíč je sestaven tak, že jsou hodnoty zrojové a cílové ip adresy spolu s hodnotami zdrojového a cílového portu nakopírovány do struktury klíče. Druhý klíč je sestaven tak, že jsou oproti prvnímu klíči prohozeny zdrojové ip adresy s cílovými ip adresami a zdrojové porty s cílovými. Pomocí těchto klíčů pak vyhledáme v tabulce *flow_chars* příslušné toky a nakopírujeme polovinu vektoru z hodnot prvního toku a polovinu z druhého. Kopíruje se vždy proměnná *band*.

Ještě před samotným klasifikováním vektoru je třeba provést jeho normalizaci:

```
int normalize_vector(double *vec);
```

Parametrem této funkce je ukazatel do vektoru, jehož každý prvek bude normalizován podle funkce:

$$\delta = \frac{d - d^{min}}{d^{max} - d^{min}}$$

Cílem je transformovat data z rozsahu [d^{min},d^{max}] do intervalu [0,1]. Hodnota d označuje puvodní a δ normalizovanou hodnotu. Protože známe hodnoty d^{min} a d^{max}, kdy d^{min} je nulová přenosová rychlost a d^{max} je zadaná maximální přenosová rychlost, tak platí: Když $d = d^{min}$, tak $\delta = 0$, když $d = d^{max}$, tak $\delta = 1$. Můžeme tedy původní funkci zjednodušit na tvar

$$\delta = \frac{d}{d^{max}}.$$

Získaný normalizovaný vektor je spolu s popisem toku předán funkci:

```
int classify_vector(double *vec, char *flow);
```

Tímto krokem je ukončeno vytváření dat potřebných k určení typu provozu. Vstupním bodem do dalšího modulu, jehož úkolem je určení typu provozu, je funkce:

```
int process_flow(int class, char *flowdesc, struct key *k1, struct key *k2);
```

Parametry této funkce jsou: typ provozu zjištěný voláním funkce *classify_vector()*, textový popis datového toku a klíče do asociativních polí identifikující příslušné datové toky v obou směrech.

6.4 Klasifikátor datových toků

Základem pro klasifikaci získaného vektoru je neuronová síť uložená v souboru *train.net*. Její strukturu a parametry lze nejjednodušeji popsat krátkým programem[6], který byl použit k natrénování a uložení sítě:

[6]Tento program je mírnou modifikací vzorového programu dostupného spolu se zdrojovými kódy knihovny.

```
#include <fann.h>
int main()
{
    const unsigned int num_input = 30;
    const unsigned int num_output = 6;
    const unsigned int num_layers = 3;
    const unsigned int num_neurons_hidden = 80;
    const float desired_error = (const float) 0.001;
    const unsigned int max_epochs = 5000000;
    const unsigned int epochs_between_reports = 1000;

    struct fann *ann = fann_create_standard(num_layers, num_input, \
      num_neurons_hidden, num_output);

    fann_set_activation_function_hidden(ann, FANN_SIGMOID);
    fann_set_activation_function_output(ann, FANN_SIGMOID);

    fann_train_on_file(ann, „train.data\, max_epochs, \
      epochs_between_reports, desired_error);

    fann_save(ann, „train.net\);

    fann_destroy(ann);

    return 0;
}
```

Na prvních čtyřech řádcích těla programu je popsána struktura sítě. Jedná se tedy o síť, která má třicet vstupních neuronů, šest výstupních neuronů a 80 neuronů ve skryté vrstvě. Další řádky udávají postupně požadovanou chybu, na kterou se má neuronová síť natrénovat, maximální počet epoch trénování a počet epoch, mezi nimiž budou vypsány zprávy o průběhu trénování.

Funkce *fann_create_standard()* vytvoří standardní plně propojenou dopřednou neuronovou síť a do každé vrstvy kromě výstupní bude přidán *bias* neuron,[7] který bude propojen se všemi neurony další vrstvy a bude vždy emitovat hodnotu 1.

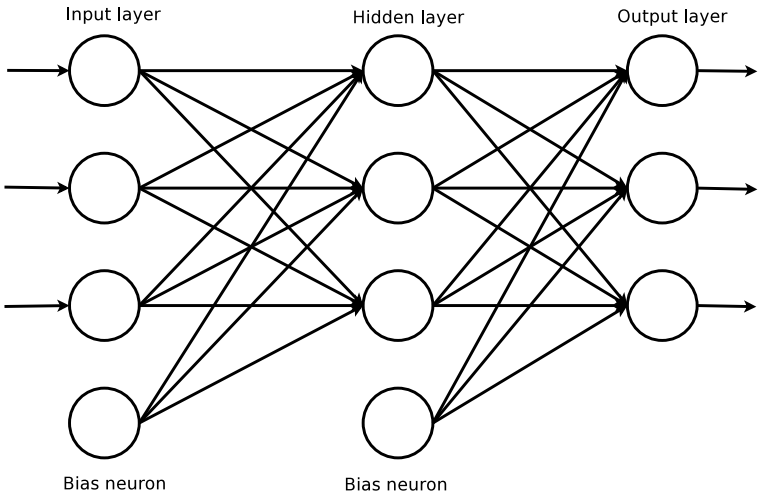
Další dvě funkce nastaví aktivační funkci všech skrytých vrstev a aktivační funkci výstupní vrstvy. Zde byla na základě dokumentace[8] ke knihovně zvolena sigmoidální aktivační funkce, která má rozsah funkčních hodnot v rozmezí [0,1] a je definována jako

$$g(x) = \frac{1}{1 + e^{-2s(x+t)}}.$$

Tento rozsah funkčních hodnot je důležitý, protože do stejného intervalu normalizujeme data vstupního vektoru.

[7] Funkce bias neuronu může být chápána tak, že zajišťuje aktivaci neuronu bez ohledu na jeho vstupy. Pokud by síť neměla bias neuron a všechny vstupy byly nulové, tak by jediným možným výstupem sítě byla nula. S bias neuronem může dojít k aktivaci i kdyby suma příspěvků od všech neuronů byla 0. http://www.codeproject.com/useritems/NeuralNetwork_1.asp

[8] Veškerá dokumentace je v podobě webových stránek je k dispozici na adrese http://leenissen.dk/fann/ . Stejná dokumentace je i součástí archivu se zdrojovými kódy knihovny.



Obrázek 6.1: Plně propojená dopředná síť s jednou skrytou vrstvou a bias neurony

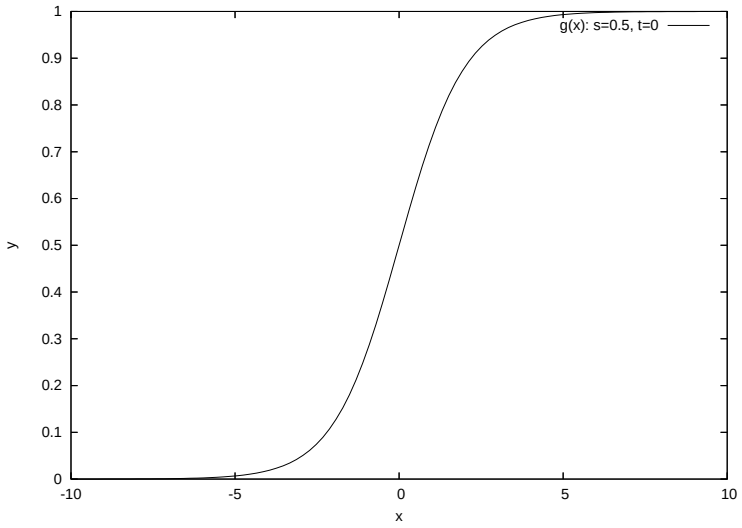
Následuje funkce *fann_train_on_file()*, která natrénuje síť s použitím vzorků uložených v souboru *train.data*. Formát tohoto souboru je definován tak, že na prvním řádku je zapsán počet vzorků v souboru, počet vstupních neuronů a počet výstupních neuronů. Dále následují vždy postupně data každého vzorku následovaná požadovaným výstupním vektorem. Pokud bychom měli tedy čtyři vzorky, čtyři vstupní neurony a dva výstupní vypadal by soubor třeba následovně:

```
4 4 2
1 1 1 1
1 0
1 1 1 0
1 0
0 0 0 0
0 1
0 0 0 1
0 1
```

Jakmile je neuronová síť natrénovaná, tak je pomocí funkce *fann_save()* uložena do souboru *train.net*. Z tohoto souboru je potom neuronová síť načtena při spuštění programu funkcí *fann_create_from_file()*.

Další použití neuronové sítě je velice jednoduché. Pomocí funkce *fann_run()*, jejímiž parametry jsou struktura neuronové sítě a vektor vstupních dat získáme ukazatel na výstupní vektor. Tento vektor následně procházíme funkcí *compute_class()* jejímiž parametry je ukazatel na výstupní vektor a jeho délka. Jakmile nalezneme hodnotu větší, než 0.85, tak získáme požadovanou třídu. Pokud takovou hodnotu nenalezneme a nebo jich nalezneme více, tak to znamená, že neuronová síť vzorek špatně vyhodnotila a funkce vrátí hodnotu výchozí třídy, do které spadá neklasifikovaný a středně prioritní provoz.

Bylo zvoleno šest různých typů provozu, do kterých se program snaží klasifikovat provoz. Každý z těchto typů je charakteristický svým průběhem závislosti rychlosti přenosu dat na čase. Typické znázornění těchto charakteristik je přiloženo v příloze.



Obrázek 6.2: Sigmoidální aktivační funkce

typ provozu	**příchozí směr**	**odchozí směr**
1. Download - FTP, HTTP, sftp	5	3
2. Streamované video, audio	4	3
3. Interaktivní http, interaktivní ssh	2	2
4. Režie spojení	3	3
5. Voice over IP	1	1
6. Upload - FTP, HTTP, sftp	3	5

Tabulka 6.1: Klasifikace do tříd

Po klasifikování datového přenosu je vytvořen záznam do asociativního pole *class_flows*, kam se uloží aktuální čas klasifikace, textový popis datového spojení, třída do které byl přenos zařazen a klíče do asociativních polí *h* a *flow_chars*.

Těchto informací je následně využíváno ke kontrole, zda-li nedošlo ke změně typu datového přenosu, případně jestli nevypršel čas, kdy je spojení považováno za vypršené a má dojít k odstranění záznamů z příslušných datových struktur a odebrání pravidel z paketového filtru. Odstraňování vypršelých spojení se provádí periodicky po uplynutí určené doby. Perioda lze měnit testováním proměnné clear ve funkci *signal_handler()* oproti větší hodnotě. Hodnota clear se zvětšuje o 1 s každým příchodem signálu SIGALRM, který určuje periodu zpracování zachycených paketů.

Práce tohoto modulu končí zavoláním funkce *exec_iptables()*, které se předají informace potřebné k nastavení paketového filtru. Touto funkcí začíná poslední fáze programu.

6.5 Nastavení priorit paketům pomocí iptables

Funkce *exec_iptables()* je místo, kde se předávají veškeré zjištěné informace o probíhajících datových přenosech jádru operačního systému prostřednictvím nástroje *iptables*.

```
int exec_iptables(int mode, char *flowdesc, int class, int oldclass);
```

První parametr funkce určuje, zda se má přidat, odebrat, či změnit klasifikace datového přenosu. Druhý parametr je ukazatel na textový popis, ze kterého jsou vyseparovány potřebné údaje pro iptables. Třetí a čtvrtý parametr obsahují současnou, respektive minulou třídu, do které byl přenos klasifikován. Pokud se třetí a čtvrtý parametr nerovnají, tak je zřejmé, že došlo ke změně charakteristiky provozu a je nutné jej přeřadit do jiných prioritních tříd.

Zařazení do tříd je provedeno pomocí modulu *MARK* paketového filtru iptables. Sestavené pravidlo může mít například tvar:

```
iptables -t mangle -A MYSHAPER-OUT -p tcp -s 192.168.2.183 --sport 37273 \
-d 209.85.137.83 --dport 443 -j MARK --set-mark 22
```

Toto pravidlo říka, že všechny pakety, které mají zdrojovou ip adresu 192.168.2.183, zdrojový port 37273, cílovou adresu 209.85.137.83 a cílový port 443 náleží do prioritní fronty 22. Tyto fronty jsou nad síťovým zařízením pomocí samostatného skriptu, jehož spuštění musí předcházet použití programu.

6.6 Schéma řadících disciplín

Abychom dosáhli možnosti prioritizovat různé typy provozu procházející přes síťová rozhraní počítače, je nutné nahradit jednoduchou výchozí frontu FIFO na výstupních síťových zařízeních hierarchickým plánovačem, který je schopen frontu navázanou na jedno rozhraní spravovat tak, aby provoz, který je klasifikován jako prioritní byl přednostně odeslán. K tomuto účelu byl použit plánovač *HTB*, který je popsán v kapitole 4.

Nastavení síťových rozhraní je implementováno ve skriptech *shaper.sh* a *shaper_desktop.sh*. Skript shaper.sh je nutno použít v případě, že počítač, na kterém jej pouštíme funguje jako router a obsahuje tedy alespoň dvě síťová rozhraní. Skript shaper_desktop.sh použijeme v případě koncového počítače, kterým je například pracovní stanice.

Před samotným spuštěním skriptu je nutné jej upravit. Je potřeba nastavit správá jména síťových karet a rychlost sítě v příchozím i odchozím směru. Rychlostí se rozumí ve většině případů zakoupená konektivita do sítě Internet.

Spuštěním skriptu se nad síťovými rozhraními definuje pevné schéma prioritních front, do kterých bude spadat klasifikovaný provoz. Tyto fronty sdílí kapacitu linky tak, aby byla maximálně využita dostupná konektivita a aby neprioritní provoz neobsadil celou kapacitu linky, čímž by omezil použití některých dalších služeb, kterým by se nedostalo požadované propustnosti. Zároveň jsou vytvořeny řetězce paketového filtru iptables, kam je prostřednictvím iptables směrován klasifikovaný provoz. V případě routeru je řetězec *MYSHAPER-IN* pro příchozí směr umístěn do řetězce *FORWARD* a řetězec *MYSHAPER-OUT* je připojen k řetězci *POSTROUTING*. Pokud nepracujeme s routerem, ale koncovou stanicí, je vytvořen pouze řetězec *MYSHAPER-OUT* je připojen k řetězci *OUTPUT*. Postup vytvoření schématu použitého v těchto skriptech je popsán v kapitole 4.

Schéma prioritních front je vždy vybudováno v bodě, kdy data opouštějí síťové rozhraní, protože efektivně lze řídit rozřazování jednotlivých paketů do front pouze pokud čekají na odeslání. V případě, kdy je na síťové rozhraní přijat paket z vnějšku, tak už vlastně vygeneruje určitý provoz a musí být co nejrychleji odeslán. Nelze tedy efektivně a přesně řídit provoz v příchozím směru, a to je také důvodem, proč jsou v případě skriptu shaper_desktop.sh definovány prioritní fronty pouze pro odchozí provoz.

Pro omezování příchozího provozu, které by bylo vhodné implementovat v případě použití koncového počítače je možné využít například modulu *IMQ - Intermediate Que-*

ueing Device.[9] Tento modul ale není součastí oficiální distribuce jádra operačního systému, a pokud jej chceme použít je nutno aplikovat příslušný patch a modul si zkompilovat. Existují ovšem problémy se stabilitou tohoto řešení a to zejména ve spojeni s IPsec a GRE tunely.

[9] http://wiki.nix.hu/cgi-bin/twiki/view/IMQ/ImqFaq

Kapitola 7

Použití a testy systému

V této kapitole je vysvětleno jak nainstalovat a s jakými parametry spouštět výsledný program *flows*. Jsou zde provedeny zatěžové testy a dále je provedena diskuze paměťové a výpočerní náročnost. Pro vygenerování testovacího provozu je použit program *ttcp*.

7.1 instalace a použití

Program je distribuován ve zdrojovém tvaru a je tedy nutné jej přeložit. Před samotným překladem zdrojových kódů je třeba nainstalovat knihovnu *fann*, která je uložena v adresáři fann. Postup instalace je uveden v souboru README. Pokud není v systému přítomna knihovna pcap, je nutno ji doinstalovat. Tato knihovna je standardní součástí většiny distribucí GNU Linuxu a neměl by být problém ji pomocí příslušných distribučních nástrojů nainstalovat. Pokud jsou tedy v systému obsaženy všechny potřebné knihovny, stačí v adresáři se zdrojovými kódy spustit nástroj *make*, který přeloží a sestaví spustitelný program *flows*.

Program lze spouštět s následujícími parametry:

```
[-i inner interface [-o outer interface]] [-D download] [-U upload] [-L 1|0]
```

První parametr je jméno síťové karty, která je na routeru připojena do vnitřní síťě. Pokud zároveň s tímto rozhraním není druhým parametren zadáno rozhraní vnější, tak program automaticky pracuje v tzv. „desktop módu“ a prioritizace provozu probíhá pouze v odchozím směru na zadané síťové kartě. Jak již bylo řečeno, druhý parametr označuje ethernetové rozhraní, které je v případě routeru připojeno do vnější sítě. Pokud je zadáno jméno vnějšího rozhraní, musí být zadáno také jméno vnitřního rozhraní, jinak program považuje zadání parametrů za chybné. Není-li zadáno žádné rozhraní, program se pokusí nalézt první dostupné zařízení a s ním pak bude pracovat v dektop módu.

Třetí a čtvrtý parametr nastavují rychlost v kilobitech za sekundu, na kterou je nastavena linka v příchozím, respektive odchozím směru. Není-li některý z těchto parametrů zadán, použije se implicitní hodnota 1024kbps.

Poslední parametr říká programu, zda-li má generovat soubory obsahující vygenerované vektory z klasifikovaných toků. Tyto soubory jsou následně ukládány do aktuálního adresáře a mohou sloužit k dalšímu natrénování neuronové sítě. Implicitní hodnotou je nula, tedy negenerovat žádné soubory.

Příklad použití:

```
./flows -i eth0 -o eth1 -D 2048
```

Těmito parametry je programu sděleno, že vnitřní síť routeru je připojena na rozhraní eth0, vnější na eth1, propustnost linky v příchozím směru z pohledu klientů ve vnitřní síti je 2048kbps a propustnost v odchozím směru je 1024kbps. Program nebude generovat žádné vypočtené vektory.

Před použitím programu musí být nad síťovými rozhraními definován systém prioritních front. K tomuto účelu jsou v adresáři *scripts* uloženy dva skripty. Skript *shaper.sh* pro použití na routeru a soubor *shaper_desktop.sh* pro desktop. Před jeho spuštěním je vhodné jej upravit tak, aby odpovídal nastavení počítače, na kterém bude spuštěn. Tato úprava spočíva v nastavení proměnných na počátku skriptu, které definují rychlost downloadu a uploadu a jména síťových karet.

7.2 Testy paměťové složitosti a výpočetní náročnosti

Z návrhu systému je zřejmé, že výsledný program musí udržovat v paměti velké množství datových struktur, které reprezentují aktuální stav síťového provozu procházejícího routerem, nebo generovaného desktopem. Vysoké nároky na paměť navíc rostou s počtem TCP nebo UDP spojení, a proto byl vypracován následující test, který zdůvodní paměťové nároky a poskytne představu, jak velký provoz je systém schopen obsloužit a kolik k tomu potřebuje paměti.

Paměťově nejnáročnější je uložení probíhajících spojení, kdy pro každý směr je potřeba jednoho kruhového bufferu, který dokáže pojmout až 16384 paketů. Datová struktura reprezentující jeden paket má velikost 32 bytů. Dva kruhové buffery potřebné pro reprezentaci jednoho spojení zaberou tedy přibližně jeden megabyte paměti. Další datové struktury již zdaleka nejsou tolik paměťově náročné. Pro uložení průběhu závislosti rychlosti přenosu na čase jednoho probíhajícího spojení je potřeba přibližně 20 kilobytů. Tento rozdíl je způsoben tím, že program je schopen uchovat maximálně 360 posledních sekund takové charakteristiky. Další datové struktury již v celkovém měřítku nehrají podstatnou roli. Pro deset spojení tedy dojdeme k číslu přibližně 10.6MB. K této hodnotě je potřeba přičíst velikost přilinkovaných knihoven, která činí přibližně dva megabyty. Celkem dostáváma 12.6 megabytů pro deset spojení.

Test byl proveden tak, že bylo pomocí skriptů a programu ttcp vytvořeno n spojení a následně bylo programem *pmap* zjištěno, kolik paměti zabírají všechny datové struktury a slinkované knihovny. Naměřené hodnoty byly pak zapsány do tabulky a zaneseny do grafu.

Skript, který je spuštěn na počítači, který bude přijímat data:

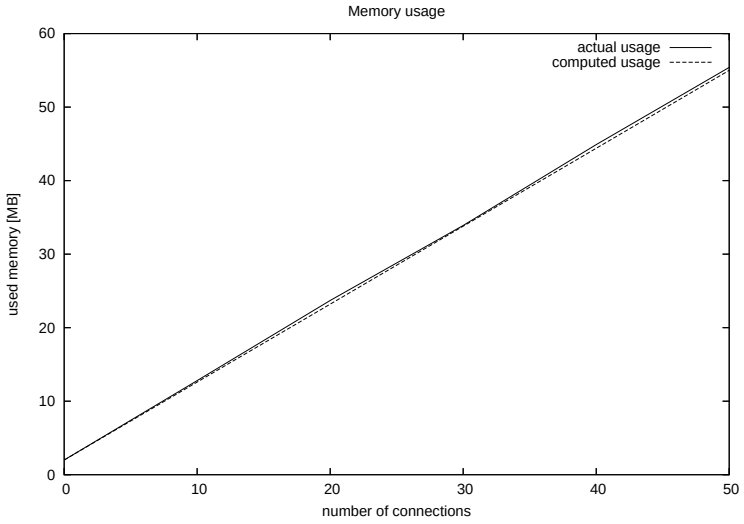
```
#! /bin/bash
for i in ‘seq 5001 5010‘;
do
    ttcp -r -p $i &
done
```

Skript, který je suštěn na počítači, který odesílá data:

```
#! /bin/bash
for i in ‘seq 5001 5010‘;
do
    cat /dev/zero | ttcp -t -p $i 192.168.2.109 &
done
```

počet spojení	skutečné obsazení paměti	vypočtené obsazení paměti
10	12.8	12.6
20	23.7	23.2
30	33.9	33.8
40	44.9	44.4
50	55.4	55

Tabulka 7.1: Závislost obsazené paměti na počtu spojení



Obrázek 7.1: Závislost obsazené paměti na počtu spojení

Z tabulky a grafu je zřejmé, že se zvyšujícím se počtem spojení lineárně roste množství potřebné paměti. Značná paměťová náročnost je daň za možnost posuzovat právě probíhající síťový provoz v širším kontextu a možnost jej relativně rychle klasifikovat.

Paměťovou náročnost je možné redukovat nalezením kompromisu mezi prostorem dostupným k uložení kontextu síťového provozu, délkou periody se kterou probíhá výpočet charakteristik provozu a velikostí vektoru potřebného pro klasifikování. Tato optimalizace by se musela týkat především velikosti kruhového bufferu pro uložení jednotlivých paketů. Pokud by byl zkrácen interval výpočtu charakteristik, tak by bylo možné zmenšit i tento buffer. Buffer ale musí zůstat dostatečně velký, aby byl program schopen fungovat i na linkách o větší kapacitě, kde se může pochopitelně rychleji zaplnit.

Program neprovádí nad uloženými datovými strukturami mnoho složitých výpočtů, z čehož plyne, že na rozdíl od nároků na množství paměti systém nespotřebuje příliš výpočetní kapacity stroje. Byly provedeny testy, kdy bylo vygenerováno až 150 spojení s co nejvyšším množstvím odeslaných paketů za jednotku času, ale program nikdy nespotřeboval na základě pozorování v nástroji *top* více než dvě procenta výkonu procesoru. Vzhledem k těmto výsledkům je podrobnější analýza výpočetní složitosti programu nedůležitá, protože aby nastala situace, kdy je zatížení procesoru objektivně měřitelné, tak by byla vyčerpána dostupná paměť.

Kapitola 8

Závěr

Operační systém GNU Linux poskytuje velké množství nástrojů pro zajištění kvality síťových služeb. S pomocí těchto nástrojů jsme schopni implementovat širokou škálu různých modelů a nasadit je v produkčním prostředí.

V současných systémech pro zajištění kvality služeb je využíváno informací uložených v hlavičce jednotlivých paketů nebo informací získaných analýzou samotných dat v paketu přenášených. Převládá použití prvního přístupu, který nevyžaduje velký výpočetní výkon a lze tedy nasadit i v prostředí větších sítí. Druhý přístup, tedy analýza paketu na úrovni aplikační vrstvy, přináší možnost klasifikovat provoz přesněji podle typu použité aplikace, ale nároky na výpočetní výkon jsou v takovém případě značné.

Ani jeden z těchto přístupů nedokáže klasifikovat všechen provoz. V případě analýzy paketu na úrovni síťové vrstvy stačí, aby použité aplikace používaly nestandardní čísla portů a nebo aby nevhodným způsobem nastavovaly pole Type of Service v hlavičce paketu a klasifikátor není schopen provoz zařadit. Při analýze na úrovni aplikační vrstvy je problémem velké množství různých aplikací generujících různě charakteristické pakety. Pro oba přístupy je z principu nemožné klasifikovat šifrovaný provoz.

Systém navržený v této práci se snaží přistoupit k problematice klasifikace síťového provozu úplně odlišným způsobem, než který je v současných systémech používán. Místo často složitého a výpočetně náročného zkoumání dat každého paketu je předložen k posouzení vektor dat reprezentující závislost rychlosti přenášených dat na čase a to zároveň v obou směrech spojení. Podle „tvaru“ takového vektoru lze často posoudit o jaký typ provozu se jedná. Tento způsob klasifikace nevyžaduje žádnou výpočetně náročnou analýzu samotných dat paketu a není závislý na žádných pevně smluvených číslech portů běžných služeb. Ke své činnosti vyžaduje pouze informace o čase přijetí paketu, jeho délce a samozřejmě zdrojovou a cílovou adresu spolu s čísly portů, podle kterých seřazuje pakety do jednotlivých toků. Tyto informace obsahuje každý paket a program založený na tomto principu tedy dokáže klasifikovat i šifrovaný provoz.

Vzhledem ke způsobu navržení systému je nutné aby si program udržoval v paměti velké množství datových struktur, které reprezentují aktuální stav síťového provozu na daném zařízení. Paměťové nároky lineárně stoupají s množstvím aktivních síťových spojení. Díky velkému množství uložených informací ale není následná klasifikace výpočetně náročná a velké paměťové nároky jsou tak částečně vykoupeny malými nároky na výpočetní výkon.

Problém klasifikace jednotlivých vektorů je řešen použitím schopností neuronových sítí, jejichž použití je málo výpočetně náročné a pokud je poskytnuto dostatečné množství kvalitních referenčních vzorků, tak i přesné. Tvar charakteristiky předkládaný neuronové síti k posouzení se může lišit a pokud bude program pracovat s řádově rychlejšími, nebo po-

malejšími linkami, kde může určitý typ provozu znamenat téměř nulovou zátěž, respektive může linku úplně saturovat. Tuto situace lze řešit opětovným natrénováním neuronové sítě.

Při vývoji a používání výsledného programu vyšlo najevo, že paměťové nároky v současné době brání použití tohoto způsobu klasifikace síťového provozu v produkčním prostředí, kde je často generován provoz čítající stovky spojení. Námětem pro další práci by mohlo být nalezení optimálních hodnot velikosti datových struktur, velikosti vektoru pro klasifikaci neuronovými sítěmi a četnosti periodické klasifikace. Tato optimalizace by mohla přinést úsporu paměti a posunula by tak takový systém blíže produkčnímu prostředí.

Pokud bychom šli v představách o využití tohoto způsobu klasifikace ještě dále, je možno uvažovat o implementaci mechanismů do síťové vrstvy operačního systému na úrovni jádra, které by poskytovaly uživatelské aplikaci charakteristiky potřebné ke klasifikaci jednotlivých datových toků. Tímto způsobem by šlo omezit paměťové nároky a dramaticky zvýšit výkonnost.

Literatura

[1] T. Aura. RFC 3972: Cryptographically generated addresses (cga). http://www.ietf.org/rfc/rfc3972.txt, March 2005.

[2] Martin A. Brown. Traffic control howto. http://www.linux.com/howtos/Traffic-Control-HOWTO/index.shtml, October 2006.

[3] Tim Carstens. Programming with pcap. http://www.tcpdump.org/pcap.htm.

[4] L. Roberts. Qos signaling for ip qos support. http://www.packet.cc/TIA1039ID.htm, July 2005.

[5] Richard W. Stevens. *Unix Network Programming*. Adison Wesley, 2004. ISBN 0-13-141155-1.

[6] WWW stránky. Icmpv6. http://en.wikipedia.org/wiki/ICMPv6.

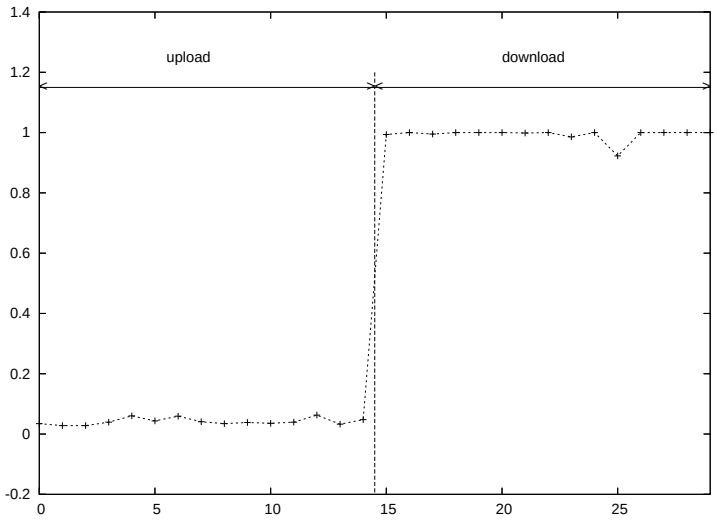
[7] WWW stránky. Quality of service. http://en.wikipedia.org/wiki/Qos.

[8] S. Thomson and T. Narten. RFC 2462: IPv6 stateless address autoconfiguration. http://www.ietf.org/rfc/rfc2462.txt, dec 1998.

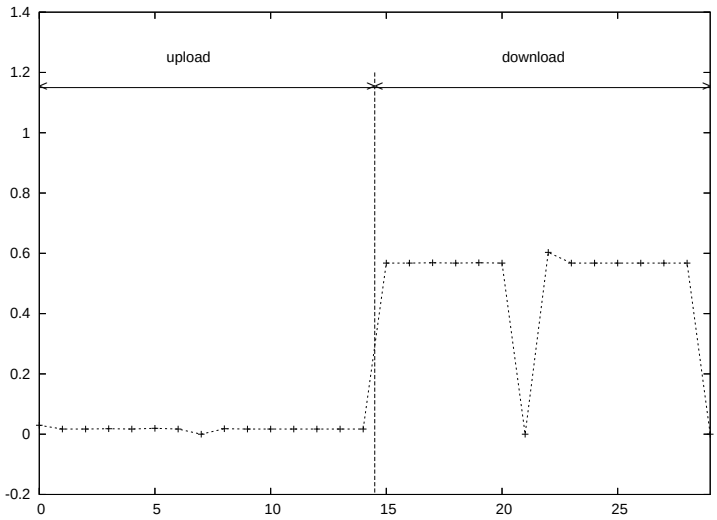
Index

Dodatek A

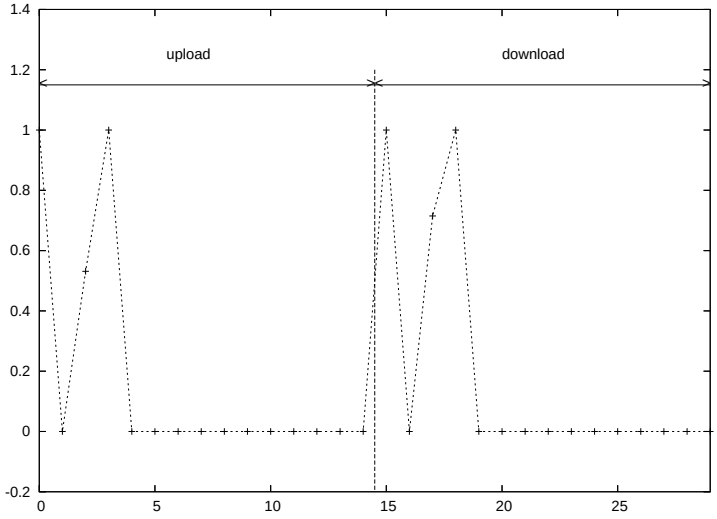
Grafické znázornění typů provozu



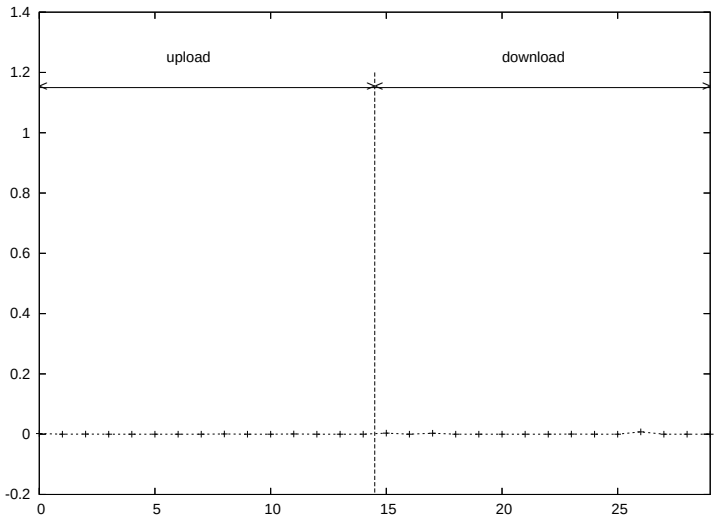
Obrázek A.1: Typ 1 - Download



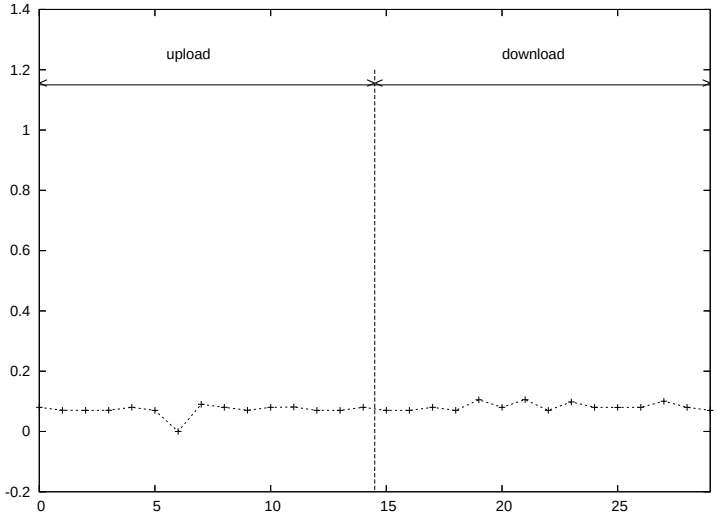
Obrázek A.2: Typ 2 - Streamované video



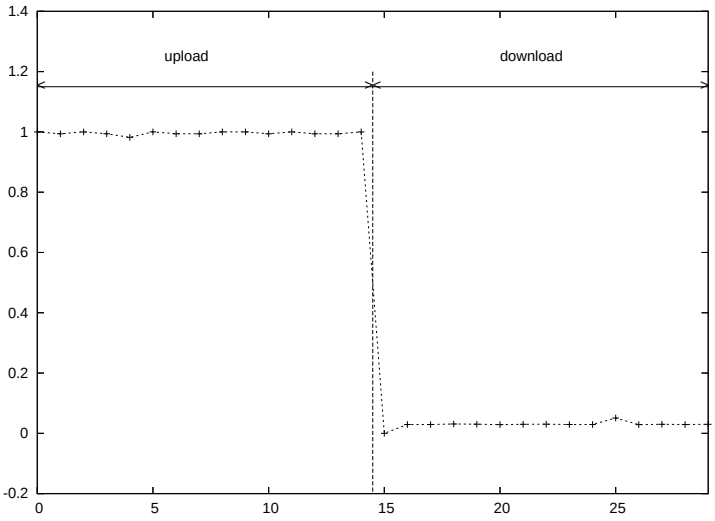
Obrázek A.3: Typ 3 - Interaktivní traffic



Obrázek A.4: Typ 4 - Režie spojení



Obrázek A.5: Typ 5 - Voice over IP



Obrázek A.6: Typ 6 - Upload

Dodatek B

Popis technického vybavení

B.1 Testovací počítače

B.1.1 Server

Operační systém	GNU/Linux Fedora Core 6
Procesor	1 x Intel(R) Core 2 Duo(R) E6400 CPU 2.13GHz, L2 cache 2MB
RAM	2GB DDR2
Swap	2GB
Ethernet	2x Realtek 8139

B.1.2 Pracovní stanice

Operační systém	GNU/Linux Fedora Core 6
Procesor	1 x Intel(R) Pentium M(R) CPU 1.7GHz bez HT, L2 cache 2 MB
RAM	768MB DDR
Swap	1.5GB
Ethernet	Broadcom BCM4401 100Base-T