VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

DIPLOMOVÁ PRÁCE

2004 Radim Paseka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Omezování nežádoucího provozu na bezdrátových sítích

2004 Radim Paseka

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

Ve Frýdku-Místku, ...
dne 10.5. 2004 Radim Paseka

Chtěl bych poděkovat vedoucímu diplomové práce, ing. Martinu Němcovi, za pomoc při řešení této diplomové práce. Dále panu Bc. Lumíru Jasiokovi za odborné konzultace.

Abstrakt

Obsahem diplomové práce je návrh administračního systému, který komunikuje se síťovými směrovači a umožňuje jejich bezpečnou vzdálenou správu. Směrovače jsou softwarového typu, fungují na platformě UNIX a jsou rozmístěny v síti, která kombinuje architektury přenosových prostředků, ethernetu a bezdrátových sítí.

Základní požadavek je kladen na správné přidělení šířky pásma jednotlivým klientům sítě a kontrolu, zda tato šířka pásma není překračována. Přidělení pásma probíhá po komunikaci s databází, ve které jsou uloženy příslušné informace, následně jsou na směrovačích spouštěny kontrolní mechanizmy, které v případě překročení šířky pásma provedou korekci a upozorní administrátora na provedenou změnu. Administrátor má tedy možnost využít automatické korekce nebo provést změny manuálně z uživatelského prostředí programu. Systém dále monitoruje síť a předává reálné průtoky dat klientů ke grafickému zpracování.

Klíčová slova

Algoritmus, bezdrátová síť, bezpečnost, CBQ, Ethernet, HTB, IP adresa, iptables, Java, JSP, MySQL, omezování provozu, PRIO, protokol, QoS, Rrdtool, RSA, SFQ, směrovač, SSL, šířka pásma, TBF, tc, TCP/IP, tvarování provozu, u32.

Abstract

The diploma work contents an proposal of an administrative system, which communicates with network routers and makes possible their safe distant repair. The functions of routers are based on UNIX platform and they are placed in a net, which combines an architecture of cause-nasal transmission substance, the ethernet and wireless LANs.

The pre-requisite is laying on a single network client right bandwidth delegation and verification, whether this bandwidth isn't overload. The zone delegation is proceed up the communication with a database, in that are storaged pertinent information, subsequently they are starting the check contrivances on the router, which are in case of the overfullfilmented bandwidth making corrections and calling the attention to the administrator on fulfilmented changes. The administrator has then the possibility to take an advantage of automatic corrections or to lead manualy through changes from an administrator's environment programme. The system further monitors the network and hands down the real flow data client to the graphic processing.

Keywords used

Algorithm, wireless network, safeness, CBQ, Ethernet, HTB, IP address, iptables, Java, JSP, MySQL, restraint operation, PRIO, record, QoS, Rrdtool, RSA, SFQ, router, SSL, bandwidth, TBF, tc, TCP/IP, shaping operation, u32.

Obsah

1. Úvod

1.1 Motivace a cíle této práce

Mnoho organizací, které se zabývají zprostředkováním připojení k Internetu, dnes řeší otázku, jak co nejvýhodněji zvládnout velké množství klientů, ať už z veřejného či komerčního sektoru.

Na straně jedné se snaží poskytnou co nejvýhodnější služby, reagují na zvyšující se nároky provozovaných aplikací a kvalitu připojení, na straně druhé jsou omezováni fyzikálními vlastnostmi přenosových médií a pro každý druh média dále technologií pro vysílání a přijímání signálu. Dochází ke sdílení média a poskytovatel musí zajistit rozdělení šířky přenosového pásma mezi více klientů s různými prioritami.

Jelikož ne vždy můžeme na zvyšující se nároky reagovat zvýšením šířky přenosového pásma, snažíme se najít jiné cesty. Jedna z nich je použít disciplíny zabývající se omezováním a tvarováním provozu[1] a kvalitou služby [2].

Ukážeme, že některé typy služeb na Internetu si vystačí s menším přenosovým pásmem, aniž by uživatel pozoroval nějaký zásadní problém v použití služby. Jiné služby vyžadují vysokou interaktivitu a tedy musí dostávat vysokou prioritu pro své přenosy. To vše v přiděleném přenosovém pásmu. Bohužel praxe ukazuje, že nelze spoléhat na počáteční konfigurace služeb, jelikož se vyskytují aplikace, které jsou schopny nastavení překračovat a různými způsoby obcházet.

Práce si klade za cíl vytvořit síťový systém se vzdálenou správou, který s využitím dostupných nástrojů pro tvarování provozu a kvalitu služeb bude řídit a kontrolovat provoz na síti a bude automaticky detekovat a korigovat nežádoucí aktivity[3].

1.2 Organizace dokumentu

V první části dokumentu je rozebrán provoz na Internetu (kapitola 2) a možnosti jeho omezování a priorizace (sekce 2.5). Dále se zaměříme na nástroje UNIXu, které byly vyvinuty právě za účelem monitorování a tvarování provozu (sekce 2.9). V druhé části (kapitola 3) bude popsán návrh a implementace administračního systému.

1.3 Poznámky k odborným termínům a zkratkám

Problematika české terminologie je často diskutovaným tématem snad ve všech dynamicky se rozvíjejících vědních oborech. Informatika, jako věda relativně mladá, se navíc nemůže opírat ani o tradiční názvosloví latinské či řecké a téměř výhradně odvozuje své názvosloví z angličtiny, která se stala univerzálním referenčním jazykem tohoto oboru. Vzhledem k rychlosti a překotnosti vzniku nových technologií na poli informatiky české názvosloví v mnoha případech není ustálené či vůbec neexistuje.

[1] z anglického „traffic shaping“

[2] z anglického „quality of service“ (QoS)

[3] jde především o tzv.„download akcelerátory“, které svým provozem narušují činnost jiných aplikací

Ani v této práci se proto nevyhneme termínům, které nemají daleko k anglicismům. V některých případech bylo nutné uchýlit se k extrémnímu řešení a anglický termín použít v jeho originální podobě. U odborných termínů typu „Hierarchical Token Bucket“ by používání českého překladu pravděpodobně vedlo pouze ke zmatení čtenáře.

Pokud jde o jednotky a jejich zkratky, je v případě jednotek fyzikálních dodržena soustava SI, u bezrozměrných veličin používaných v informatice dáváme přednost českým tvarům. Bity za sekundu jsou označeny b/s, kilobity za sekundu kb/s. Ve výjimečných případech jsou také použity kilobajty za sekundu, kde ctíme úzus použití kB/s, přestože zkratka „B“ je v soustavě SI vyhrazena pro bell. Výjimku z tohoto pravidla tvoří výpisy standardního výstupu programů, které používají anglický styl zkratek, kde místo lomítka najdeme písmeno „p“ ve významu „per“. Jednotka kb/s je tedy v takových případech značena jako kbps a analogicky jsou tvořeny jednotky další. Tento přístup umožňuje zachovat naprostou autenticitu strojově generovaných výstupů.

2. Omezování provozu a kvalita služby

2.1 Motivace

Celkový provoz na Internetu za posledních několik let velmi vzrostl a často se stává, že jednotlivé přenosové trasy nejsou dostatečně dimenzovány. Uživatelé (přesněji řečeno zákazníci), uzavírají se svým poskytovatelem smlouvu o připojení k Internetu. Pro nás je důležitým prvkem smlouvy šířka přenosového pásma, kterou dostávají k dispozici. Není těžké domyslet, že zákazníci vygenerují mnohem více požadavků na přenos, než kolik jim určuje smlouva a velmi často je to více, než kolik je schopna zajistit celá trasa k ISP.

Hodně k tomu také přispívá rodina protokolů TCP/IP, kterou využívá většina aplikací Internetu. Protokoly jsou konstruovány k maximálnímu využití síťových zdrojů. Snaží se přenést data co nejrychleji k cíli, to funguje do chvíle, než je z nějakého důvodu některý z paketů ztracen nebo příjemce není schopen zvládnout množství dat, které se na něj valí.

Zákazníci dnes disponují nejčastěji síťovými kartami o kapacitě 10/100 Mb/s a těmi jsou připojeni po vnitřní síti k příslušnému směrovači, který tuto kapacitu prostě nezvládne. Je proto důležitější mít kontrolu nad odchozím než příchozím provozem, máme k tomu několik důvodů.

1. Typicky „nejužší“ místo sítě, skrz které k nám přicházejí data, je připojení k ISP
2. Jestliže jsme schopni více či méně kontrolovat odchozí tok dat, opačnou stranu neovlivníme vůbec

Hledáme proto možnosti, jak co nejlépe optimalizovat provoz a to právě na směrovačích. Nabízejí se nám možnosti provoz tvarovat a omezovat (traffic shaping) a dále priorizovat určité služby nejvhodnějším způsobem tak, abychom dodrželi šířku pásma a zákazník se cítil co nejméně poškozen. Priorizací paketů samozřejmě nedosáhneme celkové vyšší přenosové kapacity, avšak „urychlíme“ přenos těch paketů, které příslušejí ke službám nejvíce citlivým na přenosovou rychlost a odezvu a naopak „přibrzdíme“ pakety, které patří službám, u kterých odezva není tak kritickou vlastností. Abychom mohli priorizaci paketů zajistit, je nutné zavést mechanismus kontroly a řízení odesílání paketů. Na teoretické úrovni se těmto mechanismům říká „kvality služby“ (Quality of Service, často zkracováno na „QoS“).

2.2 Omezování provozu

Jak již bylo naznačeno v sekci 2.1, nejčastěji používanými protokoly na Internetu jsou protokoly TCP/IP, jelikož se chovají jako „greedy“ protokoly, snaží se získat pro přenos co nejvíce z přenosového pásma. Data jsou posílány na směrovače, které je přeposílají na místo určení, co nejdříve to jde.

Prakticky jsou příchozí data ukládány do front (jsou bufferovány), buffery jsou limitovány množstvím vyrovnávací paměti, při přeplnění front (vyčerpání paměti) se příchozí

pakety rovnou zahazují (tail drop), což má za důsledek nepříjemnou ztrátu dat.[1] Nemluvě o tom, že po síti mezi sebou komunikuje velké množství aplikací a při zahlcení sítě to vypadá, že „nic nejede“.

Této situaci se snažíme vyhnout kontrolou provozu. Nejčastěji na směrovačích, ale i na přepínačích, které tuto službu podporují. Tedy, ovlivňování provozu vhodným zahazováním paketů na vstupních či výstupních rozhraních (input a output drops) se nazývá *omezování nebo také tvarování provozu*. V anglické literatuře nacházíme pojmenování „*traffic shaping*“, které budeme používat dále v textu. Je založen na spojení koncepcí klasifikace provozu, frontovacích mechanismů, vynucovacích politik, férovosti a kvality služby.

Důsledně a účinně můžeme provádět traffic shaping v jeho pravém slova smyslu pouze na výstupním zařízení, protože jsme schopni opravdu ovlivnit jak a v jakém množství budou od nás pakety odcházet. Výstupní provoz je specifikován dvěma parametry a to:

1. **Committed Information Rate (CIR)** - propustnost sítě zajištěná na podporu uživatele při běžném provozu
2. **Burst Commited (Bc)** - zajištěný objem dat, odpovídající předplacené hodnotě CIR

Provoz, který tvarujeme, může být posílán rychlostí o velikosti maximální hodnoty CIR za určitý čas. Této době se říká „shaping interval“ a je roven nebo vetší než podíl Bc/CIR. Z toho plyne, že pošleme-li v tomto intervalu právě Bc bitů, dodržíme CIR. Kdybychom nebrali v potaz tento interval, odchozí rychlost by byla větší, často až na hranici kapacity linky.

Traffic shaping nám zajišťuje:

- Kontrolu granularity na síti.
- Efektivní využití omezených nebo sdílených zdrojů.
- Snížení doby odezvy a zvýšení dostupnosti služeb.
- Garantovanou kvalitu služby.

2.3 Rozdělení provozu

Nekontrolovaný provoz na síti zabírá šířku pásma různými způsoby. Abychom mohli navrhnout efektivní řešení řízení provozu, musíme pochopit, jak jsou jednotlivé typy aplikací závislé na šířce pásma a jak traffic shaping ovlivní jejich chod.

2.3.1 Objemný provoz

Objemným provozem[2] rozumíme přenos velkých bloků dat. Přenosy dat přes FTP často znamenají generování provozu na úrovni, která překračuje kapacitu linky. Servery posílají obrovskou vlnu dat, pak čekají na potvrzení, než pošlou další vlnu. Bez vhodného

[1] Klasickým příkladem je spojení dvou sítí s různými přenosovými rychlostmi, kde klient na „rychlejší“ lince posílá data klientovi na „pomalejší“ linku, směrovač není schopen přeposílat pakety adekvátní rychlostí dál, ukládá je do vyrovnávací paměti a po přeplnění je zahazuje.

[2] v anglické literatuře nalezneme termín Bursty Traffic nebo také Bulk Traffic

managementu by každá z vln mohla způsobit saturaci sítě, nehledě na omezení provozu dalších aplikací, včetně jiných FTP přenosů. FTP, multimediální přenosy (avi, mpeg, mov, wav, mp3, ...), grafické přenosy (jpeg, gif, bmp, ...) včetně HTTP provozu můžou být „uhlazeny“ vymezením určité dostupné šířky pásma. Jestliže totiž tyto objekty zabírají prostor interaktivním aplikacím, je lepší jejich rozpínání eliminovat. Podobného efektu dosáhneme, přidělíme-li těmto přenosům nízkou prioritu, čímž zvýhodníme důležitější a časově citlivé aplikace.

2.3.2 Interaktivní provoz

Relace Telnetu, HTTP nebo SSL přenosy, to jsou příklady interaktivního provozu na Internetu. Tyto aplikace vytvářejí po spuštění relaci založenou na poměrně krátkých zprávách dotaz/odpověď. Interaktivní provoz[3] se týká časově citlivých aplikací, které komunikují s koncovým uživatelem (on-line nakupování nebo „surfování“ na Internetu), pro svůj provoz nepotřebují příliš z přenosového pásma, ale jakmile dojde ke konkurenci a boj o přenosové pásmo, díky špatné odezvě ztrácejí na efektivitě. Do této oblasti také patří aplikace, které plní důležité úkoly[4]. Interaktivní provoz můžeme upřednostnit priorizací, zajistíme jim tak výhodu před méně důležitými a časově nekritickými aplikacemi. Pro tzv. „mission critical“ aplikace by bylo vhodné rezervovat určitou šířku pásma.

2.3.3 Časově nekritický provoz

Tento provoz neprovází nutnost interaktivní komunikace, samozřejmě není jedno, kdy bude příslušný paket doručen, ale můžeme si dovolit malé zpoždění. Klasickými zástupci této provozní skupiny jsou SMTP[5] nebo NNTP[6]. Jelikož i tady dochází k přenosům větších bloků dat, které můžou způsobovat zahlcení sítě, zařadíme je pod interaktivní aplikace, ale svou prioritou budou převyšovat objemný provoz.

2.3.4 Časově kritický provoz

Jde o nejcitlivější provoz na Internetu, každé zpoždění, které vznikne se negativně promítne na kvalitě. Do této kategorie patří např. IP telefonie (Voice Over IP) nebo real-time audio/video. Těmto službám by se mělo dostávat nejvyšších priorit.

2.3.5 Zábavný provoz

Chceme-li klasifikovat provoz na Internetu, musíme brát v úvahu pracovní nebo obchodní stránku provozu. Tato úvaha nás přivádí k zábavnému provozu (on-line hry, Napster,...). V některých případech je z důvodu bezpečnosti a velkého zatížení sítě zakázán.

Jak je z výše uvedeného členění vidět, v určitých situacích nejsme schopni objektivně rozhodnout, kterou službu priorizovat a neexistuje jednotné priorizační schéma, kterým

[3] v anglické literatuře Interactive Traffic

[4] tzv. mission-critical applications

[5] Simple Mail Transfer Protocol - jednoduchý protokol pro přenos elektronické pošty mezi stanicemi [RFC 821]

[6] Network News Transfer Protocol - protokol transferu zpráv zpravodajských skupin [RFC 977]

bychom se každému zavděčili. Ale je bezesporu důležité rozlišovat mezi aplikacemi, které pro svůj efektivní provoz potřebují krátkou dobu odezvy a mezi těmi, co ji nepotřebují.

2.4 Klasifikace provozu

Abychom vůbec byli schopni řídit provoz, musíme dodržet následující postup:

1. Identifikujeme provoz
2. Klasifikujeme provoz
3. Na základě klasifikace vybereme vhodné řešení řízení provozu

Není naším cílem klasifikovat všechen provoz na Internetu, pouze ten pro tuto problematiku podstatný. Tabulka 2.1 zachycuje současné chování vybraných aplikací na Internetu. Snažíme-li se o traffic shaping, měli bychom být schopni detekovat provoz na síti Internet

Typ	**Příklady**	**Provoz**	**Současný stav**	**Požadovaný stav**
Grafické soubory	jpg, gif, swf	Interaktivní aplikace, objemné soubory	Zmrazování interaktivní grafiky, stop-and-go zobrazování	Plynulé zobrazování
HTML soubory	htm, html	Interaktivní aplikace, malé textové soubory	Nepředvídatelné zobrazování, náhodné zpoždění	Rychlé zobrazování, vysoká priorita přístupu
Real-time audio, video	ra, rstp	Vysoce interaktivní aplikace	Nespojitý tok dat, trhané video, nízká kvalita audia	Konstantní bitová rychlost v předvídatelné šířce
Reprodukované soubory	avi, mov, mpeg, mp3	Interaktivní stahování, objemné soubory	Vytváření relací pro stahování	Eliminace těchto relací
Přenos souborů	TFTP, FTP	Interaktivní stahování, objemné soubory	Ukazatele trendu pro stahování	Eliminace velkého množství relací
Vzdálené přihlašování	telnet, ssh	Interaktivní, malé pakety	Soupeření o šířku pásma	Vysoká priorita přístupu

Tabulka 2.1: Klasifikace vybraného provozu na Internetu

a pro příslušné typy provozu definovat třídy provozu(traffic class).

Třídou provozu rozumíme mechanismus detekce atributů provozu. Velmi záleží na úhlu pohledu. Např. chceme-li řídit provoz generovaný web serverem, máme několik možností, jak vytvořit třídy:

- IP adresa web serveru, tj. všechen provoz, který server generuje.

- Pouze webové služby.
- URL, tj. pouze část služeb, které server nabízí (gif, mpeg).

Obecně tedy klasifikujeme provoz podle:

- IP adresa, MAC adresa nebo maska sítě.
- Typ aplikace vztažené k číslu portu, které používá.
- Transportní protokol.
- URL.

2.4.1 IP adresa, MAC adresa, maska sítě

Hostem definujeme kterékoliv koncové zařízení (uzel) sítě, pro naše potřeby s podporou protokolu IP. Každý host má přidělenu svoji MAC adresu, IP adresu a masku sítě. Traffic Shaper [7] je podle těchto informací schopen klasifikovat jednotlivé toky. IP paket navíc obsahuje zdrojovou a cílovou adresu účastníků komunikace, čímž je nám dána možnost zaměřit se na zdroj, cíl nebo taky na obě adresy.

2.4.2 Čísla portů

Protokoly vrchních vrstev typicky používají pro svou komunikaci čísla portů, které jim byly přiděleny společností Internet Assign Numbers Authority (IANA) . Podle tohoto čísla jsme schopni protokol identifikovat a poskytnout mu určitý servis. IANA pro ně definuje tzv. „well-known port numbers“. V tabulce 2.2 najdeme výběr ze standartních služeb.

Služba	**Port**	**Služba**	**Port**	**Služba**	**Port**
ECHO	7	DNS	53	HTTP	80
FTP-data	20	VIA-FTP	63	KERBEROS	88
FTP	21	BOOTTPS	67	POP3	110
Telnet	23	BOOTTPC	68	IMAP	143
SMTP	25	TFTP	69	SNMP	161
NTP	37	GOPHER	70	BGP	179
LOGIN	49	FINGER	79	SERVLET	8080

Tabulka 2.2: Standardní služby a čísla jejich portů

2.4.3 Transportní protokoly

Klasifikace a tvarování provozu na úrovni transportních protokolů je jedním z nejčastěji používaným mechanismem řízení provozu. Jelikož transportní protokoly poskytují služby protokolům vyšších vrstev a jsou zodpovědné za přenos dat, nabízí se nám řešení omezovat aplikace již na této úrovni.

[7] výraz pro zařízení nebo aplikaci, která provádí tvarování provozu

2.4.4 URL

Identifikátorem objektů ve webovém světě jsou tzv. URI (Uniform Resource Identifiers). URI mohou být:

- Uniform Resource Names (URN)
- Uniform Resource Locators (URL)
- Uniform Resource Characteristics (URC)

V praxi se ale setkáváme pouze s URL. Jednotlivé aplikační protokoly mají své schéma, které je podle [RFC1738] definováno jako

<schéma>:<závislá část>

kde <schéma> může mj. být:

http	Protokol HTTP	ftp	Protokol FTP
mailto	Protokol SMTP	nntp	Protokol NNTP (diskusní skupiny)
Telnet	Odkaz na relaci protokolem Telnet	file	Soubor (např. z disku)
imap	Protokol IMAP	ldap	Protokol LDAP
pop	Protokol POP3		

Pro ilustraci si rozeberme schéma http:

http://<uživatel:heslo><server><port>/<cesta>?<dotaz># <fragment>

Uživatel a heslo slouží pro autentizaci klienta, dále následuje jméno serveru uvozené znakem @ a číslo portu. Cesta obsahuje identifikaci souboru skládající se z adresářů oddělených lomítkem. Za otazníkem následuje řetězec dotazu, dvojitý kříž specifikuje odkaz na fragment web-stránky.

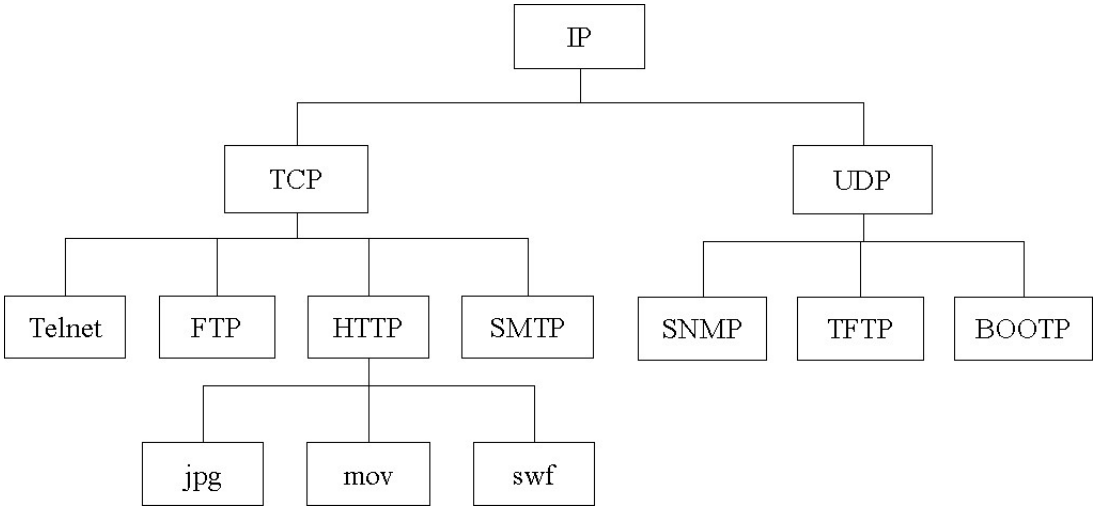
Příklady:
http://novak@server.firma.cz
http://server.firma.cz/cgi-bin/formular.exe?pole1=%20&pole2=hod2
http://server.firma.cz/addr1/addr2/dokument.html#odstavec5

Traffic shaper se podle tohoto schématu může zaměřit na konkrétní druh provozu. Třídu provozu odvozuje podle URL adresy nebo se zaměřuje na jednotlivé typy protokolů (gif, mov, mpeg).

2.4.5 Hierarchický strom tříd provozu

Jakmile jsme definovali třídy provozu, uspořádáme je ve vztahu potomek - rodič. Tímto hierarchickým uspořádáním dostaneme strom tříd provozu[8]. Třídy potomků mají mnohem specifičtější kritéria, než jejich rodiče. Snažíme-li se klasifikovat provoz, procházíme strom

[8] tento strom se v anglické terminologii nazývá „Traffic Class Tree“



Obrázek 2.1: Příklad stromu tříd provozu pro HTTP.

od kořene k listům a hledáme v něm stejnou charakteristiku. Obsahuje-li již rodičovská třída shodu v charakteristice provozu, proces pokračuje hledáním bližších informací v třídách potomků. Prohledávání končí v případě, že jsme nalezli třídu provozu odpovídající charakteristice toku dat, ale její potomci už nenesou žádné další informace, které by ji upřesnily.

Na obrázku 2.2 vidíme příklad stromu provozu pro HTTP. Procházíme přes protokol IP, dále TCP a HTTP až ke konkrétním protokolům, které definují konkrétní URL. V této chvíli můžeme aplikovat příslušné politiky pro účinný traffic shaping.

2.4.6 Politiky

Jakmile provoz na síti překročí šířku pásma dané linky, „greedy“ aplikace mohou linku zahltit a omezit provoz ostatních aplikací, zvláště citlivé jsou pak aplikace typu mission-critical, které nemusí dokončit svoji úlohu, samozřejmě se také zvyšuje doba odezvy.

Metody řízení provozu nám umožňují alokovat šířku pásma v závislosti na prioritě ve stromu provozu. Dochází k vytváření politik pro jednotlivé třídy provozu. Politiky nastavují kvalitu služby pro tyto třídy(*viz* kapitola 2.5).
Politiky pro traffic shaping jsou založeny na

- přenosová rychlost
- priorita
- zahození
- ignorování
- odmítnutí přístupu

Politiky jsou založeny na typu a důležitosti provozu. Politika přenosových rychlostí spočívá v rezervaci šířky pásma pro jednotlivé přenosy, je potřebná zvláště u objemných provozů,

které zahlcují síť. Je úzce spjata s priorizační politikou, která upřednostňuje určitý druh provozu a dovoluje definovat kvalitu služby pro tento provoz. Další dva přístupy určují způsob počáteční reakce na typ provozu. Příchozí pakety můžeme jednoduše zahazovat s potvrzením, že byly zahozeny nebo ignorovat a potvrzení vůbec nevystavovat. Poslední politika nám umožňuje odmítnout směrovat provoz dále sítí, např. podle IP adresy.

2.5 Kvalita služby

Kvalita služby (Quality of Service,QoS)[RFC2212] je pojem, kterým označujeme zabezpečení potřebných komplexních vlastností sítě, zaručujících doručení dat uživateli v potřebné kvalitě.

Tento pojem se začal široce využívat především v souvislosti s rozvojem služeb, jejichž úspěšnost z pohledu uživatele významně závisí na časových charakteristikách komunikace přes počítačovou síť. Obecně se kvalita služby uplatňuje v přenosech multimediálních dat, v IP telefonii, ale čím dál tím častěji se s tímto pojmem setkáváme i na poli běžných TCP/IP sítí. Důvodem je především skutečnost, že dochází ke konvergenci provozu klasických telekomunikačních sítí a sítí datových.[9] Utility, kterými zajišťujeme kvalitu služby, jsou nesmírně silnými nástroji i pro traffic shaping.

Abychom dosáhli efektivního řešení kvality služby, je nezbytné, aby mechanismy QoS byly implementovány po celé trase přenosu od zdroje k cíli. Nejčastější nasazení QoS je na druhé a třetí vrstvě modelu OSI pomocí technologie ATM a IP protokolu. V souvislosti ústupu od technologie ATM se zaměříme na implementaci na úrovni protokolu IP.

- Best Efford - úplně první služba Internetu, bezpriorizační přenášení paketů, v případě zahlcení se paket zahazuje.
- Differentiated Services - poskytují škálovatelné omezování služeb bez potřeby znalosti stavových informací o přenosu a signalizaci mezi jednotlivými uzly. Využívají nastavení bitů v poli paketu pro typ služby (Type of Service, ToS), směrovač je informován o tom, jak má ve vnitřní síti s daným paketem zacházet v souladu s požadavky na konkrétní službu, dochází k upřednostnění dat, nikoliv ke garanci doručení do předvídatelné doby [RFC2430][RFC2638].
- Integrated Services - poskytují diferenci služeb na základě explicitního vyjádření požadavku na kvalitu služby a rezervaci síťových zdrojů (kapacita linek, paměti ve frontách, CPU přepínacích prvků). K rezervaci dochází proti směru toku dat, je nutno rezervovat celou trasu, využívá se protokol RSVP (Resource Reservation Protocol) [RFC1633].
- MPLS - Multiprotocol Level Switching, multiprotokolové přepínání na základě speciálních návěstí, služba je zabezpečená prostředky pro agregovaný přenos dat. MPLS kombinuje technologie přepínaných okruhů a IP protokolu [RFC2702].

Kvalita služby je ovlivněna všemi komponentami sítě:

- stanicemi (WS, servery)

[9] Prostřednictvím jedné datové sítě jsou informace přenášeny v jakékoli multimediální formě, tj. nejen jako data, ale i jako hlas a video.

- směrovači, přepínači
- linkami (mezi routery, LAN)

2.5.1 Parametry QoS

Parametry QoS lze prezentovat jako charakteristiky datových toků a síťových linek, které vyjadřují jejich výkonnost a kvalitu (dostupnost (minimalizace času výpadku spojení), chybovost, latence, propustnost, ztrátovost paketů, doba vytvoření spojení, rychlost, detekce a oprava chyb).

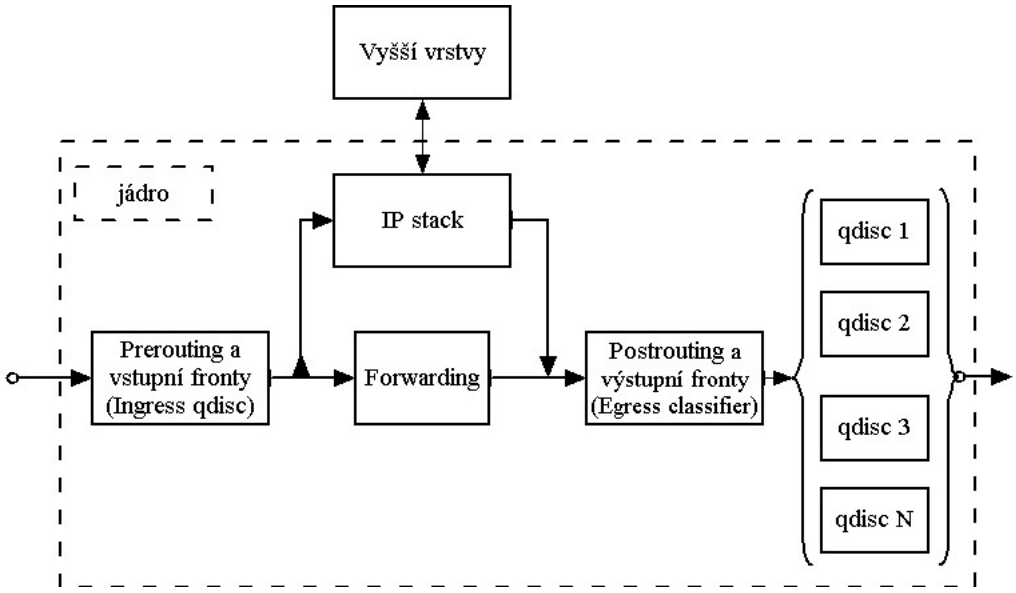
- **Šířka pásma (bandwidth)** - udává množství bitů přenesených za jednotku času, kritická je především u multimediálních aplikací, ale někdy třeba i u nejběžnějšího přenosu WWW stránek - zde záleží především na hlavním účelu použití dané přenosové trasy a příslušných prioritách uživatelů. [RFC3148]
- **Jednosměrné zpoždění (one-way delay)** - je čas potřebný k odeslání paketů a jeho přijetí v cíli[RFC2330]. Skládá se z dvou částí:
  - času potřebného na přenesení paketu přes fyzické médium, což je funkce přenosové rychlosti linky
  - času, který představuje zpoždění způsobené řazením do front, zpracováním v síťových zařízeních a přetížením linek
- **Rozptyl zpoždění (jitter)** - je definované pro dva pakety jako rozdíl mezi jednosměrným zpožděním jednoho paketu a jednosměrným zpožděním druhého paketu [RFC3393].
- **Ztrátovost paketů (packet loss)** - je poměr poslaných paketů, které nebyly přijaty v cíli nebo byly přijaty s chybou [RFC2680], často v důsledku fyzických chyb, přetížení procesoru přepínacích prvků nebo přeplnění výstupních front na těchto prvcích.

Komponenty zpoždění a jejich příčiny
Ke zpoždění může docházet v důsledku bufferování paketů do front ve směrovačích a přepínačích, tato složka zpoždění je *proměnná*, jelikož nikdy nemůžeme přesně určit dobu tohoto zpoždění. *Pevnou* složku zastupuje serializační zpoždění a doba šíření paketu, které se dají odvodit a spočítat. Dále dochází k rozptylu zpoždění a to v důsledku nejednotného zacházení s jednotlivými pakety téhož toku při bufferingu, z toho samozřejmě plyne proměnné pozdržení ve frontách.

2.6 Kontrola provozu

Kontrola provozu (angl. traffic control) muže probíhat na libovolné síťovém uzlu a je základním obecným nástrojem k ovlivnění a dosažení kvality služby. Jedná se vlastně o celou cestu mezi vstupním a výstupním síťovým zařízením daného uzlu. Základní rozhodnutí, které musí učinit uzel provádějící kontrolu provozu je:

- Paket je klasifikován na vstupním zařízení (zahozen, odmítnut, ignorován, přijat).
- Paket je určen pro toto zařízení a je poslán vyšším vrstvám.
- Paket je určen pro odeslání na jiný síťový uzel a vybere se příslušné výstupní zařízení.
- Podle nastavených politik je paket buď zahozen nebo odeslán.



Obrázek 2.2: Průchod paketu směrovačem.

Celý proces průchodu paketu směrovačem je naznačen na obrázku 2.2. Paket přichází na vstupní zařízení, kde je zařazen do vstupního řetězce (INPUT) a podle vstupní politiky klasifikován a dále zařazen do vstupních front. Často v této fázi dochází k tzv. „preroutingu“.

Po opuštění vstupních front je provedeno zásadní rozhodnutí. Pokud je paket určen lokálnímu stroji, je předán do zásobníku IP paketů a odtud dál vyšším vrstvám (např. aplikační programy), pokud je určen jinému stroji, pokračuje do předávacího řetězce (FORWARD). Provoz, který je vygenerován lokálním strojem je stejně jako výstup z předávajícího řetězce podroben filtraci nebo značkování a je tady také navázán vlastní algoritmus směrování, paket je dále předáván do výstupních front, kde je prováděn tzv. „postrouting“. V poslední době je postrouting masově využíván v technologii Network Address Translation (zkráceně NAT, lze se také setkat s pojmem Masquerading, či SNAT - Source Address Translation). Je-li definitivně rozhodnuto o odeslání paketu, bude paket zařazen do výstupního řetězce (OUTPUT).

Pomocí výstupních front definovaných v operačním systému můžeme ovlivnit způsob, jakým jsou data posílána. Zde tedy probíhá vlastní kontrola síťového provozu, která má za úkol zajistit kvalitu služby. Pakety se zde řadí do front dle stanovených pravidel (politik), která můžeme ovlivnit a dosáhnout tak kontroly přenosové rychlosti a priorizace paketů.

2.7 Ingress Traffic Shaping

Ingress Traffic Shaping (tvarování provozu na vstupním zařízení) nám neposkytuje příliš prostoru pro manipulaci s pakety, to co nám přijde neovlivníme a můžeme to buď poslat dál nebo zahodit.

2.7.1 TCP

Provoz spojově orientovaného protokolu transportní vrstvy (TCP) lze ovlivnit služebním protokolem síťové vrstvy modelu OSI - protokolem ICMP [RFC792]. ICMP standard definuje zprávu „*ICMP Source Quench*“[10]. Směrovač tuto zprávu generuje, pokud není schopen došlé datagramy zpracovat tak rychle, jak na něj přichází, ať už kvůli nedostatku paměti nebo malému množství interních zdrojů. Zahodí-li směrovač datagram, pošle odesílateli ICMP Source Quench, ten na něj reaguje snížením velikosti vysílacího okna a to do té doby, nezmění-li směrovač velikost okna sám nebo nepošle potvrzení (ACK) nějakého předešlého datagramu. Komunikace probíhá dál, ale již za snížené rychlosti, což je efekt, kterého jsme chtěli dosáhnou.

2.7.2 UDP

Bohužel, toto neplatí pro nespojově orientovaný protokol (UDP), ten tento typ zprávy ignoruje. Chceme-li nějak ovlivnit jeho provoz, nezbývá nám nic, než tyto pakety zahazovat.

Existují i další možnosti, jak ovlivňovat provoz na vstupu. Např. můžeme použít tzv. Intermediate Queueing Device (IMQ). O této metodě a mnoha dalších, včetně disciplín front, pojednává dokument HOWTO [LARTC].

2.8 Egress Traffic Shaping

Egress Traffic Shaping (tvarování provozu na výstupním zařízení) poskytuje širokou škálu metod, které pro manipulaci s pakety můžeme použít. Je to skutečně jedna z efektivních metod, protože než paket pošleme na výstupní zařízení, můžeme s ním nakládat podle libosti. V následujících sekcích se podíváme, jakými prostředky disponuje operační systém UNIX.

2.8.1 Podpora v jádře

V současné době ukončila firma RedHat freewarovou distribuci Linuxu RedHat. Na trhu ji zastoupila distribuce Fedora Core [FCP03], která je opět silně zaměřená na síťové aplikace a jejich podporu. V samotném jádře Fedora Core najdeme následující moduly pro podporu QoS a traffic shapingu.

- **CBQ** - Class-Based Queuing
- **HTB** - Hierarchical Token Bucket

[10] Jde o žádost o snížení velikosti vysílacího okna (třeba i na nulu), paket nese v hlavičce kód 0 a typ 4

- **CSZ** - Clark-Shenker-Zhang
- **ATM** - Asynchronous Transfer Mode
- **PRIO** - Priority Queueing
- **RED** - Random Early Drop (Discard)
- **SFQ** - Stochastic Fairness Queuing
- **TEQL** - Traffic Equalizer
- **TBF** - Token Bucket Filter
- **GRED** - Generalized RED
- **DSMARK** - Diff-Serv Marker

Máme tedy docela širokou škálu prostředků, které můžeme použít, některé si blíže popíšeme a ukážeme si jejich konfiguraci s využitím nástrojů UNIXu.

2.9 Nástroje UNIXu pro řízení provozu

Přestože nástroje pro kontrolu síťového provozu jsou k dispozici již poměrně dlouhou dobu (několik let), stále neexistuje oficiální dokumentace těchto vlastností. Střípky pokusů o zdokumentování alespoň nejdůležitějších funkcí nalezneme roztroušené po Internetu, avšak jediným skutečně relevantním zdrojem informací jsou vlastní zdrojové kódy příslušných subsystémů. Tento neutěšený stav způsobil, že většina administrátorů, natož potom běžných uživatelů, o možnostech kontroly síťového provozu vůbec netuší a pokud už tuší, jsou odrazeni od jejího používání komplexností a komplikovaností problematiky a až neuvěřitelnou složitostí stávajících konfiguračních nástrojů.

2.9.1 iptables

Prvním prostředkem, který umožňuje manipulaci s pakety a o kterém se zmíníme, je nástroj IPTABLES. Dovoluje zacházet s pakety podle předem definovaných pravidel. Ke své práci používá tzv. „řetězců“. Řetězce byly rozebrány v sekci 2.6.

Jelikož je použití velmi rozsáhlé, nebudeme zacházet do širších podrobností a ukážeme si pouze, jak tímto nástrojem lze měřit průtoky vztažené k IP adresám a portům, jelikož je tento způsob použit v praktické části práce. Bližší informace o programu iptables nalezneme v [IPT03] nebo manuálových stránkách.

Standardní pravidla

Standardně jsou definovány tři druhy pravidel:

- cílem je naše PC - obsahuje dva druhy řetězců s politikou ACCEPT
    1. PREROUTING - slouží pro změnu polí TOS nebo pro přípravu na DNAT

2. INPUT - řetězec paket zpracovává předtím, než je poslán ke zpracování pro směrování nebo ho filtruje příchozími pravidly, tímto řetězcem prochází veškerý příchozí provoz

- zdrojem je naše PC - obsahuje opět dva druhy řetězců s politikou ACCEPT

  1. POSTROUTING - využívá se při SNAT a také ho používá firewall, než je paket definitivně odeslán
  2. OUTPUT - většinou se používá pro filtrování odchozího provozu

- paket je přeposílán - paket je pouze forwardován a je zařazen do řetězce FORWARD, může také procházet řetězci PREROUTING a POSTROUTING

Abychom mohli rozhodnout o průtoku každé IP adresy, musíme přidat evidenci IP adresy řetězci, který ji bude zpracovávat.

```
iptables -A OUTPUT -o eth0 -p all -d 158.196.135.3
iptables -A OUTPUT -o eth1 -p all -s 158.196.135.3
```

Tento příklad ukazuje, jakým způsobem můžeme evidovat provoz pro IP adresu 158.196.135.3. Do výstupního řetězce OUTPUT přidáme nové pravidlo, které říká, eviduj provoz na výstupním rozhraní eth0/eth1, kontroluj všechen provoz, jehož zdrojem nebo cílem je IP adresa 158.196.135.3. Výpis pouze pro tuto adresu provedeme následovně:

```
iptables -n -L -x -v | grep 158.196.135.3
```

kde přepínače postupně nastavují číselné výstupy IP adres a portů, výpis všech řetězců, výpis počtu paketů a počtu prošlých bytů.

2.9.2 Traffic Control

Traffic Control (tc)[11], jak již název napovídá, jedná se o nástroj pro řízení provozu. Pomocí tohoto programu lze na uživatelské úrovni vytvářet a asociovat fronty s výstupními zařízeními. Na tuto utilitu se podíváme podrobněji. Traffic Control využívá následujících komponent:

- qdisc - z anglického termínu „queuing disciplines“, které bychom mohli přeložit jako disciplíny front, ale přidržíme se anglického termínu a dále v textu budeme používat jeho zkrácenou podobu, tedy tyto qdisc jsou zodpovědné za přeposílání dat.

- třídy provozu - do těchto tříd je rozdělen provoz, který chceme tvarovat, jestliže se třída dále nedělí na další podtřídy, je jí přiřazena jedna qdisc, která se stará o zpracování provozu a přenos dat, který touto třídou teče.

- filtry - často také označovány jako klasifikátory, jsou obecně spjaty s třídou provozu a příslušnou qdisc a rozdělují provoz do podtříd dané třídy provozu.

- kontrolor - z anglického „policer“, provádí kontrolu, zda filtry skutečně třídí provoz správně, mohou být sdíleny mezi filtry i mezi rozhraními.

[11] v textu budeme používat tuto zkratku

Každému rozhraní (síťová karta) je dále přiřazena tzv. *root qdisc*, přes kterou musí projít všechen odchozí provoz. Počáteční nastavení ji přiřazuje klasickou frontu FIFO, ale může být nahrazena férovější disciplínou.

Hierarchické uspořádání

Elementy uspořádání, které můžeme programem vytvořit jsou třídy a qdisc. Vztahy v této hierarchii jsou rodič - potomek, každá třída nebo qdisc má jednoho rodiče a může mít více potomků. Známe-li vztahy mezi rodiči a potomky, můžeme sestavit hierarchické uspořádání. Představíme-li si toto uspořádání jako strom, provoz teče od listů ke kořenu. Kmen představuje maximální šířku pásma, kterou máme k dispozici, každá větev reprezentuje třídu provozu. Rozdělení do tříd zajišťují filtry, které značkují pakety. Vstupuje-li paket do nějaké větve, je kontrolován a teprve potom je puštěn k cíli. Podrobnější popis najdeme v sekci 2.11 - classfull qdisc, která toto uspořádání vyžadují.

Filtry

- **fw** - filtr umí přesměrovat provoz do příslušné větve
- **u32** - filtr umí označit paket podle zdrojové nebo cílové adresy a portu a přiřadit jej do příslušné třídy
- **route** - filtr směruje pakety podle směrovací tabulky
- **rsvp** - filtr klasifikuje pakety na základě jejich RSVP požadavků

Syntaxe programu Traffic Control

Základní úroveň syntaxe programu tc:

```
 tc [ OPTIONS ] OBJECT { COMMAND | help }
           kde  OBJECT := { qdisc | class | filter }
                OPTIONS := { -s[statistics] | -d[details] | -r[raw] }
```

Na této úrovni je nejdůležitějším parametrem OBJECT, který může být qdisc, třída, či filtr. Práce s programem tc začíná zpravidla vytvořením queueing disciplines. Zde se uplatní tato syntaxe:

```
tc qdisc [ add | del | replace | change | get ] dev STRING
       [ handle QHANDLE ] [ root | parent CLASSID ]
       [ estimator INTERVAL TIME_CONSTANT ]
       [ [ QDISC_KIND ] [ help | OPTIONS ] ]

       tc qdisc show [ dev STRING ]
kde:

QDISC_KIND := { [p|b]fifo | tbf | prio | cbq | red | etc. }
```

Interpretace jednotlivých částí:

- *handle* reprezentuje jedinečné označení, které je přiřazeno queueing discipline. Dvě různé qdisc nemohu mít toto označení shodné.
- *root* označuje frontu, která je na vrcholu hierarchie tříd a qdisc pro dané zařízení.
- *parent* obsahuje odkaz na označení (handle) nadřazené qdisc či třídy.
- *estimator* je používán pro zjištění, zda byly uspokojeny požadavky fronty.

Syntaxe pro vytváření tříd:

```
tc class [ add | del | change | get ] dev STRING
       [ classid CLASSID ] [ root | parent CLASSID ]
       [ [ QDISC_KIND ] [ help | OPTIONS ] ]

       tc class show [ dev STRING ] [ root | parent CLASSID ]
kde:
QDISC_KIND := { prio | cbq | etc. }
```

Hodnota dosazená za QDISC_KIND musí být qdisc, která podporuje dělení do tříd jako například CBQ nebo HTB a nikoliv tedy TBF. Interpretace dalších polí:

- *classid* reprezentuje označení (handle), které je přiřazeno třídě. Skládá se ze dvou částí oddělených dvojtečkou. První část je označením úrovně, na které se třída v hierarchickém stromu nalézá, druhá část je jedinečné označení této třídy na příslušné úrovni.
- *root* označuje třídu na vrcholu hierarchie tříd pro dané zařízení.
- *parent* obsahuje označení (handle) na nadřazenou třídu.

A ještě jeden výpis syntaxe - tentokrát pro definici filtrů:

```
tc filter [ add | del | change | get ] dev STRING
       [ prio PRIO ] [ protocol PROTO ]
       [ root | classid CLASSID ] [ handle FILTERID ]
       [ [ FILTER_TYPE ] [ help | OPTIONS ] ]

       tc filter show [ dev STRING ] [ root | parent CLASSID ]
kde:
FILTER_TYPE := { rsvp | u32 | fw | route | etc. }
FILTERID := ... formát se různí dle FILTER_TYPE
```

Rozbor parametrů:

- *prio* označuje prioritu, která je filtru přiřazena. Pokud dva filtry mají stejnou hodnotu *protocol*, poté má přednost ten, který má vyšší prioritu.
- *protocol* je hodnota, která se použije, pokud identifikujeme pouze pakety patřící jistému protokolu.
- *root* označuje, že filtr je na vrcholu hierarchie pro dané zařízení.

- *classid* reprezentuje označení (handle) třídy, na kterou je filtr aplikován.
- *handle* je jednoznačné označení filtru. Formát tohoto označení je závislý na FILTER_TYPE.

2.10 Classless queuing disciplines

Classless qdisc [12] dokáže pakety ve výstupní frontě „přeházet“, zdržet či zahodit. Tento způsob řízení provozu může být použit pro celkové nastavení přenosového pásma, bez možnosti aplikace složitých pravidel pro detekci zajišťující komplexní priorizaci vybraných paketů. Tento přístup je standardně použit ve většině operačních systémů a je založen na typu výstupní fronty označované jako pfifo qdisc.

2.10.1 pfifo

Fronta pfifo (packet first in first out) qdisc se skládá ze tří pásem, či kanálů (band), které mají interní označení 0, 1 a 2. Každý paket, který dorazí do výstupní fronty je zařazen do jednoho ze tří kanálů. Jediný příznak, který je zde brán v úvahu a který o zařazení rozhoduje je tzv.„Type Of Service“ (často známější ve formě zkratky ToS). Informace, které může nést záhlaví ToS v IP datagramu ukazuje tabulka 2.3.

Binárně	**Desítkově**	**Význam**
1000	8	Minimalizuj zpoždění
0100	4	Maximalizuj propustnost
0010	2	Maximalizuj spolehlivost
0001	1	Minimalizuj finanční náklady
0000	0	Normální služba

Tabulka 2.3: Význam bitů v poli ToS hlavičky IP datagramu.

Jednotlivé označení můžeme kombinovat, kombinace vidíme v tabulce 2.4. Sloupec Linux udává, jak pole ToS interpretuje jádro OS UNIX a poslední sloupec ukazuje zařazení do příslušného kanálu a vytváří tzv. „mapu priorit“. V zásadě můžeme říci, že pakety s nevyšší prioritou jsou vždy zařazeny do kanálu 0, pakety s nižší prioritou do kanálu 1 a pakety s nejnižší prioritou do kanálu 2. Algoritmus FIFO je potom aplikován na každý kanál zvlášť s tím, že dokud jsou nějaké neodeslané pakety kanálu 0, neodesílají se pakety z kanálu 1 a 2 a analogicky to platí pro kanál 1, který dostane přednost před kanálem 2.

Parametry a použití utility tc

- *priomap* - zobrazí mapu priorit. Níže je uvedena implicitní mapa priorit nastavená v Linuxu. Jednotlivé čísla znamenají do jaké fronty budou řazeny pakety pro jednotlivé kombinace ToS bitu zleva číslovaných od 0 do 15.

[12] dalo by se přeložit jako „Beztřídové disciplíny front“, nicméně přidržíme se zde opět odborného názvu v angličtině

TOS	Bity	Význam	Linux	Provoz	Kanál
0x0	0	Normální služba	0	Best Effort	1
0x2	1	Minimalizuj finanční náklady (mfn)	1	Filler	2
0x4	2	Maximalizuj spolehlivost (ms)	0	Best Effort	1
0x6	3	mfn+ms	0	Best Effort	1
0x8	4	Maximalizuj propustnost (mp)	2	Bulk	2
0xa	5	mfn+mp	2	Bulk	2
0xc	6	ms+mp	2	Bulk	2
0xe	7	mfn+ms+mp	2	Bulk	2
0x10	8	Minimalizuj zpoždění (mz)	6	Interactive	0
0x12	9	mfn+mz	6	Interactive	0
0x14	10	ms+mz	6	Interactive	0
0x16	11	mfn+ms+mz	6	Interactive	0
0x18	12	mp+mz	4	Int. Bulk	1
0x1a	13	mfn+mp+mz	4	Int. Bulk	1
0x1c	14	ms+mp+mz	4	Int. Bulk	1
0x1e	15	mfn+ms+mp+mz	4	Int. Bulk	1

Tabulka 2.4: Kombinace vlastností ToS.

```
1, 2, 2, 2, 1, 2, 0, 0 , 1, 1, 1, 1, 1, 1, 1, 1
```

- *txqueuelen* - délka fronty je získaná z nastavení rozhraní, které můžeme vidět a nastavit pomocí *ifconfig* a *ip*. Tento parametr nelze nastavit pomocí tc.

2.10.2 Token Bucket Filter (TBF)

Nejznámějším algoritmem, který používá pro svou práci principy pfifo je Token Bucket Filter. Tento algoritmus má své použití v případě, že chceme pouze omezit celkovou šířku přenosového pásma na daném síťovém rozhraní - lapidárně řečeno, TBF umožňuje zpomalit datový tok dle přání uživatele. Vzhledem k designu pfifo není použitelný pro selektivní prioritizaci dle cílového TCP portu či IP adresy a podobně.

Parametry a použití utility tc

- *limit/latency* - limit je počet bytů, které mohou ve frontě čekat na dostupný token. Toto můžeme nastavit i jinou cestou, a to nastavením parametru latence, který nastavuje maximální množství času, kdy může paket zůstat v TBF.
- *burst/buffer/maxburst* - parametry rozhodují o velikosti paměti, která může být odeslána nad rámec pravidel.
- *mpu* - minimální velikost paketu.
- *rate* - představuje požadovanou bitovou rychlost, která má být přidělena dané třídě (to je tedy pravděpodobně nejdůležitější parametr).

Ukázková konfigurace

Ukázka jednoduché, ale velmi používané konfigurace:

```
# tc qdisc add dev ppp0 root tbf rate 220kbit latency 50ms burst 1540
```

Uvedená konfigurace nám říká že k zařízení ppp0 přiřadíme qdisc tbf která bude kořenovou (root) qdisc, maximální bitová rychlost bude 220kbit, latence 50ms a velikost kbelíku(bucket) 1540 bytů.

2.10.3 Stochastic Fairness Queueing (SFQ)

Stochastic Fairness Queueing je jednoduchá implementace z rodiny fair queueing algoritmů. Klíčovým slovem v SFQ je komunikace , která odpovídá TCP session nebo UDP streamu. Provoz je rozdělován do poměrně velkého počtu FIFO front, zjednodušeně řečeno pro každou komunikaci existuje jedna fronta. Pakety jsou rovnoměrně odebírány z každé fronty a předávány nižším vrstvám, tím pádem každý přenos má rovnocennou šanci na přístup k přenosovému pásmu a není možné, aby existovalo jedno spojení, které znemožní přístup ostatním. SFQ má již v názvu slovo „stochastický“, protože ve skutečnosti nevytváří jednu frontu pro každý jednotlivý datový tok, to by bylo příliš paměťově náročné, místo toho je zde implementován algoritmus, který rozděluje provoz do určeného omezeného (byť relativně velkého) počtu front.

Parametry a použití utility tc

- *perturb* - překonfigurování hashingu (funkce, která mapuje datové toky na vytvořené fronty) proběhne jednou za nastavený počet sekund. Pokud tato hodnota není nastavena k rekonfiguraci nedojde nikdy. Dobrou hodnotou je 10 sekund.
- *quantum* - množství bytů, které se odebírají z kanálu do té doby, než bude kanál přidělen jiné frontě. Minimum je MTU a větší hodnoty.

Ukázková konfigurace

Jestliže máte zařízení, které má stejnou rychlost linky a aktuální dostupnou rychlost, jako telefonní modem, tato konfigurace bude pomáhat podporovat férovost, to znamená že nemůže dojit k zablokování některé z front:

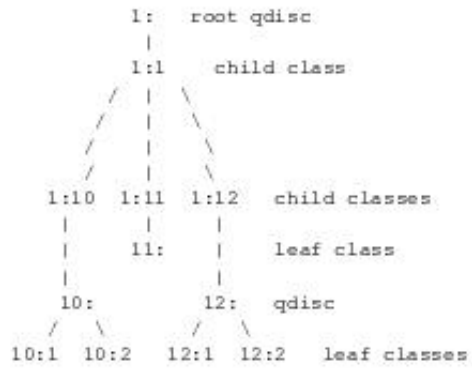
```
# tc qdisc add dev ppp0 root sfq perturb 10
# tc -s -d qdisc ls
 qdisc sfq 800c: dev ppp0 quantum 1514b limit 128p flows 128/
 1024 perturb 10sec
 Sent 4812 bytes 62 pkts (dropped 0, overlimits 0)
```

Výše uvedená konfigurace přiřadí na zařízení ppp0 sfq qdisc a ta zde bude jako root, hodnota perturb byla nastavena na 10s. Druhý použitý příkaz již neslouží ke konfiguraci, ale jen k výpisu právě provedené konfigurace.

2.11 Classful queuing disciplines

Provoz, který přichází do výstupní fronty, je rozdělen do jednotlivých tříd (classes). Mluvíme tady o klasifikaci provozu. Klasifikace se děje na základě filtrů, které jsou definovány uživatelem. Zde je tedy, narozdíl od metody classless qdisc, prostor pro konfiguraci řízení síťového provozu - právě pomocí classful qdisc je tedy možné dosáhnout definice kvality služby. Na základě rozhodnutí filtrů jsou pakety rozřazeny do tříd. Každá třída může mít dále vlastní filtry, které pakety rozřadí do podtříd. Pokud třída neobsahuje podtřídy, obsahuje vlastní frontu paketů. Navíc ještě může každá třída obsahovat vlastní metodu qdics, a to jak classful, tak classless. Takto mohou vzniknout i velmi složité stromové struktury hierarchie tříd.

Každé rozhraní má jednu výstupní kořenovou qdisc („root qdisc“), implicitně je nastavena výše zmiňovaná pfifo qdisc. Ke každé qdisc je přiřazen handle, který může být použit v pozdějších konfiguračních příkazech týkajících se této qdisc. Handle těchto qdiscs obsahuje dvě části: majoritní číslo a minoritní číslo. Obvykle je kořenová třída pojmenována "1:" , to je rovno "1:0". Minoritní číslo qdisc je většinou 0. Třídy potřebují mít stejné majoritní číslo jako jejich rodiče. Typickou hierarchii můžeme vidět na obrázku 2.3.



Obrázek 2.3: Typická hierarchie tříd.

Jádro systému komunikuje pouze s root qdisc a dává mu příkazy na zařazení paketu do hierarchie stromu (*enqueue*) nebo na odebrání paketu a odeslání jej na příslušné rozhraní (*dequeue*). Předpokládejme, že byla vytvořena následující klasifikace pro právě příchozí paket:

```
1: -> 1:1 -> 1:12 -> 12: -> 12:2
```

Jeho cílová třída je 12:2, může do ní být přiřazen postupně průchodem celého stromu nebo o zařazení rozhodne kořenová třída a bude použit filtr, který paket zařadí přímo:

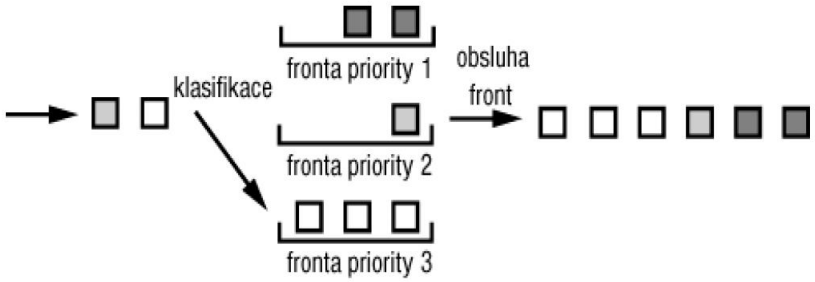
```
1: -> 12:2
```

Jakmile jádro rozhodne, že možno odeslat nějaký paket výstupním rozhraním, pošle kořenové třídě požadavek *dequeue*. Třída ho rozešle všem potomkům v pořadí priorit, které mají. To je pro nás důležitý efekt, kterého jsme chtěli dosáhnout, jelikož pokud rodič

nepožádá o odebrání paketu, paket zůstane ve frontě vnořené třídy. Tímto zajistíme spravedlivý scheduling mezi třídami, o tvarování provozu (traffic shaping) se starají příslušné qdisc, které jsou ke třídám přiřazeny.

2.11.1 Priority Queuing (PRIO)

PRIO qdisc ve skutečnosti netvaruje, pouze rozděluje provoz podle toho, jak jsou nakonfigurovány filtry. Můžeme PRIO qdisc přirovnávat k pfifo, kde je každý kanál samostatnou třídou namísto jednoduché FIFO. Když je paket zařazen do fronty s PRIO qdisc, třída již není vybírána pouze na základě zadaných ToS bitů, jako tomu bylo u classless qdisc, ale využívá plné síly, kterou nám poskytuje použití filtrů. Implicitně jsou vytvořeny tři třídy. Tyto třídy obsahují jednoduché pfifo qdisc bez vnitřní struktury, toto nastavení lze změnit na jakoukoliv jinou disciplínu. Jak je vidět z obrázku 2.4, fronty s vyšší prioritou mají absolutní přednost, z toho plyne, že pokud existuje paket ve frontě s vyšší prioritou, nebude odeslán žádný paket z fronty s nižší prioritou.



Obrázek 2.4: Ukázka fungování Priority Queuing.

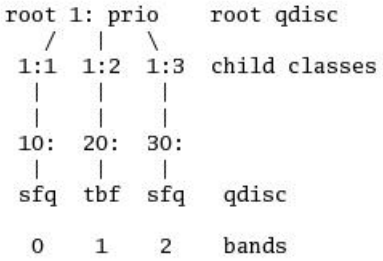
Parametry a použití utility tc

- *bands* - počet kanálů, které se mají vytvořit. Každý kanál je v podstatě třída. Jestliže změníte toto číslo, musíte také změnit priomap.
- *priomap* - jestliže nepoužijeme tc filtry pro klasifikaci provozu, PRIO qdisc se podívá na TC_PRIO priority a podle nich rozhoduje, jak se bude provoz řadit do front.

Ukázková konfigurace

V následující ukázkové konfiguraci budeme vytvářet hierarchickou strukturu, která je znázorněna na obrázku 2.5 ve formě stromu.

Celkový datový tok půjde do kořene stromu (root 1: prio), kterému je přiřazena disciplína priority queueing. V tomto bodě se datový tok dělí do tří tříd 1:1, 1:2 a 1:3, ke každé této větvi je připojena jedna qdisc tak, jak je vidět na obrázku 2.5. V tomto příkladě není



Obrázek 2.5: Hierarchie pro ukázkovou konfiguraci.

nastaven žádný filtr a proto jsou jednotlivé fronty plněny podle implicitní prioritní mapy. To znamená že data s nejvyšší prioritou se budou řadit do fronty 10: a data s nejméně významnou prioritou do fronty 30:. Následující posloupnost konfiguračních příkazů nám zajistí vytvoření stromové struktury, kterou jsme právě popsali. S tím, že u fronty s označením 20: jsme přiřadili qdisc TBF, která má nastavenou maximální bitovou rychlost na 200kb, limit byl nastaven na 3000 bytů a buffer má velikost 1600 bytů.

```
# tc qdisc add dev eth1 root handle 1: prio
# tc qdisc add dev eth1 parent 1:1 handle 10: sfq
# tc qdisc add dev eth1 parent 1:2 handle 20: tbf \
  rate 200kbit buffer 1600 limit 3000
# tc qdisc add dev eth1 parent 1:3 handle 30: sfq
```

Výhodou utility tc je, že si celou konfiguraci můžeme okamžitě zobrazit, jak je vidět na výstupu níže. Pro náš příklad byl generován interaktivní HTTP provoz, který byl zařazen do středně prioritní třídy 20:.

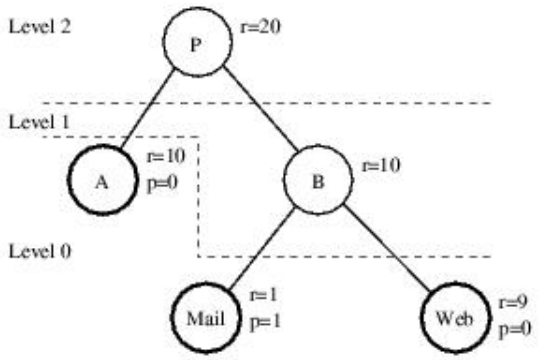
```
# tc -s qdisc ls dev eth1
 qdisc sfq 30: quantum 1514b
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

 qdisc tbf 20: rate 200Kbit burst 1599b lat 66.8ms
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

 qdisc sfq 10: quantum 1514b
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

 qdisc prio 1: bands 3 priomap 1 2 2 2 1 2 0 0 1 1 1 1 1 1 1 1
 Sent 0 bytes 0 pkts (dropped 0, overlimits 0)
```

2.11.2 Class-Based Queuing (CBQ)

CBQ je nejstarším classful omezovacím algoritmem a také tím nejsložitějším a možná i nejkomplikovanějším, pokud jde o korektní konfiguraci[13]. Tyto nepříjemné vlastnosti nejsou CBQ dány proto, že by autoři byli zlomyslní či dokonce nekompetentní, je to proto, že CBQ algoritmus je sám o sobě od základu poměrně nepřesný a především neodpovídá příliš způsobu, jakým dnes pracují moderní operační systémy - je na ně poněkud násilně „napasován".

CBQ je algoritmus, který umí klasifikovat provoz do tříd a který navíc umí omezit šířku přenosového pásma. Bohužel právě na tomto poli nedosahuje zcela přesných výsledků. Základní myšlenka algoritmu vypadá takto: Pokud se snažíme omezit linku 10 Mb/s na přenosovou kapacitu 1 Mb/s, pak by linka měla být „prázdná" po 90% času. Pokud není, musíme frontováním paketů zajistit, aby byla 90% času prázdná. Problém je v tom, že je poměrně těžké měřit toto procento času, po které je linka prázdná a tak CBQ tuto hodnotu raději odvozuje od počtu mikrosekund, které uběhnou mezi jednotlivými požadavky z hardwarové vrstvy na předání dalších paketů.

Hierarchické sdílení



Obrázek 2.6: Sdílení kapacity linky.

Běžný problém, který administrátor řeší, je znázorněn na obrázku 2.6. Poskytovatel internetového připojení potřebuje rozdělit internetové pásmo mezi firmy označené **A** a **B**. Každá firma má specifický požadavek na garantovaný datový tok r a maximální odezvy. Velikost odezvy přitom záleží na prioritě třídy p. Firma B si navíc přeje oddělit datové toky pro mailové a webové služby. Zavedeme si základní terminologii:

- **třída** - je uzlem hierarchického stromu, jak jsme ji definovali již dříve, dále pro ni definuje požadovaný tok r.
- **list** - označuje uzel, který již nemá žádné další následovníky, pouze listy mohou obsahovat pakety ve své interní formě.

[13]tento algoritmus byl dlouhou dobu „horkým kandidátem" pro použití v praktické části, proto mu budeme věnovat více pozornosti

- **plný list** - je list, který obsahuje nějaké pakety.
- **plný vnitřní uzel** - je uzel s potomkem typu plný list.
- **podlimitní uzel** - tímto rozumíme uzel, jehož aktuální tok je menší než jeho požadovaný tok r, o podlimitnosti rozhoduje *estimátor*.
- **nespokojený uzel** - uzel, který je současně plný a podlimitní, tj. tento uzel má nárok, aby byl obsloužen.
- **úroveň** - z anglického *level*, definuje hloubku uzlu, pro listy je hloubka definována jako 0. Způsob číslování je dále zřejmý z obrázku 2.6.

V následujícím textu budeme sledovat, jak se tato disciplína staví k problému „půjčování toku“ a jakým způsobem určit nadlimitní nebo podlimitní stavy.

CBQ a půjčování

Autor CBQ ukázal, že je nutno respektovat následující pravidlo

Pravidlo 1 *Je-li X úroveň nejvyššího nespokojeného uzlu , není možné si půjčit od uzlu s úrovní větší než X.*

Pokud toto pravidlo nebude algoritmus respektovat, nastane situace, kde listy s vyšší prioritou (*A,Web*) zahltí celou kapacitu linky a *Mail* nedostane ani svůj garantovaný tok. Při respektování tohoto pravidla tato situace nenastane, neboť *Mail* je nespokojeným uzlem a ostatní si nemohou půjčit z jiné úrovně než 0 (neboli si nemohou půjčit vůbec).

Algoritmus CBQ implementuje toto pravidlo proměnnou *toplevel*. Proměnná je přepočítávána vždy při zafrontování (enequeue) nového paketu a při odebírání paketu (dequeue). Obě akce totiž ovlivní, zda je list plný či ne a definice nespokojeného listu je odvozena právě od plnosti listu.

Nespokojenost uzlu záleží také na aktuálním toku uzlu (zda je podlimitní). Uzel, který je nad limitem, se ovšem může stát podlimitním nejen během zafrontování či odebrání paketu, ale i po uplynutí dostatečné doby, kdy uzel nebyl využíván. Tato asynchronní událost není v *toplevel* algoritmu zohledněna a proto CBQ implementace, které *toplevel* využívají[14], jsou zatíženy jistou chybou, což potvrzují i praktická měření.

CBQ estimátor

Originální práce [LRMM95] nedefinuje, jaký estimátor se má použít. Avšak stejně jako každá implementace používá *toplevel*, tak i při volbě estimátoru je použit originální návrh autorů CBQ.

Tento algoritmus detekuje limitní stavy tříd měřením doby, která uplyne mezi odesláním dvou za sebou následujících paketů téže třídy. Čas se porovná s očekávanou dobou (ta je odvozená z požadované rychlosti) a znaménko odchylky určuje, zda je třída nad či pod limitem.

Pomineme-li složitost měření mezipaketové mezery s dostatečnou přesností (řádově mikrosekundy), zjistíme, že tento algoritmus potřebuje ke své funkci znát přesnou rychlost

[14] jedná se vpodstatě o všechny známé implementace

fyzického rozhraní. Není problém určit rychlost Ethernetového rozhraní, ale je to nemožné u zařízení jako Wireless LAN nebo softwarových zařízení (tunely). Podobná zařízení totiž nemají pevnou rychlost odeslání dat a jejich aktuální rychlost kolísá a závisí na mnoha podmínkách (kvalita signálu, zatížení směrovače,...). Navíc nastavení takového estimátoru není vůbec jednoduchá záležitost a autoři CBQ sepsali dokument [NCBQ96], který pojednává právě o problematice nastavení estimátoru[15].

Parametry a použití utility tc

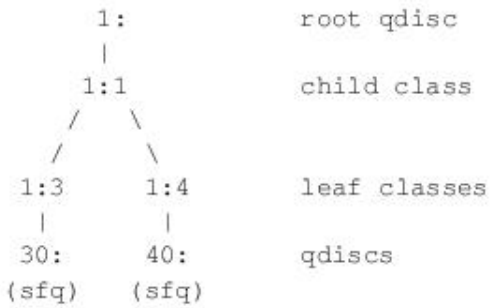
- *avpkt* - je velikost průměrného paketu a CBQ ji potřebuje znát, aby mohlo stanovit čas, který je potřeba pro odeslání jednoho paketu. Tento čas je stanoven jako podíl AVPKT a BANDWIDTH.
- *bandwidth* - představuje maximální přenosovou rychlost, která je k dispozici pro zařízení, na které je fronta navázána. Pokud je výstupním zařízením ethernetová síťová karta 100 Mb/s, pak bude hodnota 100Mbit (jednotka se zadává právě v tomto tvaru).
- *cell* - reprezentuje granularitu výstupního zařízení. Odeslání paketu o velikosti 800 či 806 bytů může totiž z technických důvodů trvat naprosto stejný čas. Hodnota CELL nastavuje právě velikost kroku, po kterém se bude doba potřebná k odeslání paketu zvětšovat. Běžná hodnota je 8 a důležité je, že CELL musí být celá mocnina dvou.
- *maxburst* - maximální počet bytů, které lze poslat při krátkodobém rychlém nárůstu požadavků.
- *minburst* - představuje nejmenší možný počet bytů, které je nutno poslat.
- *mpu* - je minimální počet bytů, které budou poslány v jednom paketu. Pokud je požadavek na odeslání paketu o menším počtu bytů, bude paket dorovnán právě na velikost MPU. Tato hodnota existuje z důvodu technických omezení jednotlivých fyzických sítí, u Ethernetu je MPU rovno 64.
- *rate* - představuje požadovanou rychlost, která má být přidělena dané třídě (to je tedy pravděpodobně nejdůležitější parametr).
- *allot* - omezený objem dat, který může být poslán třídou v jednom kole odesílání paketů na rozhraní.
- *prio* - reprezentuje prioritu dané třídy.
- *weight* - pomáhá algoritmu pro scheduling (Weighted Round Robin) rozhodnout, kolik paketů poslat v závislosti na šířce pásma, které bylo třídě přiděleno(časté použití je RATE/10).
- *bounded* - označuje, že třída si nemůže půjčit nevyužitou kapacitu od svých rodičů. Pokud není specifikováno, třída si takto půjčovat kapacitu může - pokud jsou k dispozici třídy, které kapacitu nabízejí, *viz* další parametr

[15] toto byly docela pádné důvody, proč tuto disciplínu nepoužít v praktické části práce

- *isolated* - označuje, že třída nebude nabízet svou nevyužitou kapacitu jiným třídám. Pokud není uvedeno jinak, kapacita je nabízena.

Ukázková konfigurace

Předpokládejme síťovou kartu 100Mbit, na které vytvoříme třídy pro provoz FTP a webových aplikací. FTP dostane 3Mbit a web(HTTP) 5Mbit. Dohromady jim je přiděleno maximálně 6Mbit. Třídy si mohou jedna od druhé půjčit. Hierarchii vidíme na obrázku 2.7.



Obrázek 2.7: Hiearchie ukázkové konfigurace.

V první části vytvoříme kořenovou disciplínu označenou 1:0, která bude používat cbq. Bandwidth v této disciplíně bude nastaven na 100Mbit podle typu síťové karty, velikost průměrného paketu je nastavena na 1000 bytů a hodnota cell je nastavena na 8. K této qdisc je připojena třída označená 1:1, která také používá cbq v této třídě je nastaveno spoustu parametrů, jejich význam je vysvětlen výše. Co stojí za zmínku je, že třída 1:1 má nastaven argument bounded, což znamená, že si třída nemůže půjčit nevyužitou kapacitu od svých rodičů a celková bitová rychlost nemůže přesáhnout 6Mbit/s, to plyne z hodnoty rate která je nastavena na 6Mbit/s.

```
# tc qdisc add dev eth1 root handle 1:0 cbq bandwidth \
  100Mbit avpkt 1000 cell 8
# tc class add dev eth1 parent 1:0 classid 1:1 cbq bandwidth \
  100Mbit rate 6Mbit weight 0.6Mbit prio 8 allot 1514 \
  cell 8 maxburst 20 avpkt 1000 bounded
```

Následující příkazy vytváří další dvě třídy se společnou rodičovskou třídou 1:1. První třída označená 1:3 bude použita pro HTTP provoz, tuto vlastnost zajistíme dále, použitím filtru, použitou disciplínou bude opět cbq a v tomto případě bude maximální bitová rychlost nastavena na 5 Mbit/s. Druhá třída označená 1:4 bude použita pro FTP provoz, to bude opět zajišťovat správné nastavení filtru. Pro FTP bude maximální bitová rychlost nastavena na 3 Mbit/s. Můžeme podotknout, že obě třídy nejsou bounded, tudíž si mohou vypůjčit nevyužitou kapacit od své rodičovské třídy do jejího maxima - 6 Mbit/s.

```
# tc class add dev eth1 parent 1:1 classid 1:3 cbq bandwidth \
```

```
  100Mbit rate 5Mbit weight 0.5Mbit prio 5 allot 1514 \
  cell 8 maxburst 20 avpkt 1000
# tc class add dev eth1 parent 1:1 classid 1:4 cbq bandwidth \
  100Mbit rate 3Mbit weight 0.3Mbit prio 5 allot 1514 \
  cell 8 maxburst 20 avpkt 1000
```

Následující dva příkazy přiřadí výše vytvořeným třídám disciplíny front, ty budou označeny 30: s rodičovskou třídou 1:3 a 40: s rodičovskou třídou 1:4. Oběma bude přiřazena sfq qdisc.

```
# tc qdisc add dev eth1 parent 1:3 handle 30: sfq
# tc qdisc add dev eth1 parent 1:4 handle 40: sfq
```

Na závěr cele konfigurace musíme nastavit používání filtrů, pro jednotlivé typy provozu. Z konfiguračních příkazů lze vyčíst, že filtry budou přiřazeny přímo v kořenové třídě. Je použit filtr u32. V prvním příkaze je provoz se zdrojovým portem 80 zařazen do třídy 1:3. V druhém příkaze je provoz se zdrojovým portem 21 zařazen do třídy 1:4.

```
# tc filter add dev eth1 parent 1:0 protocol ip prio 1 \
  u32 match ip sport 80 0xffff flowid 1:3
# tc filter add dev eth1 parent 1:0 protocol ip prio 1 \
  u32 match ip sport 21 0xffff flowid 1:4
```

2.11.3 Hierarchical Token Bucket (HTB)

Nesnáze, s kterými se potýkala disciplína CBQ, byly prvotním impulsem pro vývoj nové disciplíny, která nevykazuje žádný podobný problém. Autorem této disciplíny je Martin Devera. Podrobnosti o vývoji nalezneme na [HTB03].

HTB estimátor

HTB estimátor dělí třídy nejen na podlimitní a nadlimitní, ale navíc rozlišuje stavy z hlediska aktuální velikosti toku. Tyto stavy si v textu rozlišíme barvami semaforu. Každá třída má krom garantovaného toku r také definovaný maximální tok c[16].

Nadefinujeme si barvu třídy pro daný časový okamžik t a aktuální tok *a(t)* jako

- zelená - pokud je třída pod limitem a platí $a(t) < r$
- oranžovou - pokud $c \geq a(t) \geq r$, taková třída je sice nad limitem, ale nedosáhla svého stropu, může si tedy ještě půjčit od rodiče nebo sourozenců
- červenou - $a(t) > c$, třída přesáhla strop a proto nemůže odeslat žádný další paket

Implementace estimátoru využívá tzv. Leaky Bucket (dále jen LB). Výhodou je tolerance vůči rozlišení systémového časovače a zejména nezávislost na fyzické rychlosti rozhraní. Použití LB řeší oba problémy z sekce 2.11.2 o CBQ. LB je definován dvěma parametry, datovým tokem r (v bajtech za sekundu) a *burstem*(v bajtech). Běžně má takový datový tok rychlost r. Pokud je ovšem $a(t) < r$, pak si LB zapamatuje, kolik bajtů bylo

[16] z anglického ceil - strop

nevyužito. Maximální velikost této paměti je právě *burst*. Pokud je požadavek na přenesení dat nad limit r, LB v každou chvíli umožní přenést až tolik bajtů bez omezení rychlosti, kolik jich bylo zapamatováno.

Je dobré si uvědomit, že právě *burst* je klíčem k nezávislosti LB na rozlišení systémového časovače. Uvedeme pouze, že pokud je rozlišení časovače Δt (10ms v Linuxu), musí pro *burst* platit:

$$burst \geq \Delta t \times r \tag{2.1}$$

Utilita *tc* pro HTB s tím počítá a pokud *burst* není uveden, nastaví automaticky nejmenší možnou hodnotu pro daný systém podle 2.1.

Pro implementaci stropu (c) musí mít každá třída přiřazena dva LB, jeden pro garantovaný tok r a druhý pro c. Abychom se v textu lépe orientovali, budeme *burst* vztažený ke stropu označovat *cburst*.

HTB hierarchie a půjčování

Nejprve definujeme algoritmus pro výběr dalšího paketu k odeslání:

Algoritmus 1 *Ze všech plných listů volme takový, který by si při odesílání paketu mohl půjčit tok od rodiče na nejnižší úrovni. Pokud je takových listů více, vybereme ten s nejvyšší prioritou. Pokud máme stále více listů, střídáme je pravidelně podle poměrů jejich r.*

Tento algoritmus splňuje následující očekávání:

- toky tříd se stejnou prioritou si půjčují od společného rodiče v poměru jejich r
- třída s vyšší prioritou získá tok od společného rodiče přednostně
- třída s vyšší prioritou má kratší odezvy
- garantované toky jsou splněny

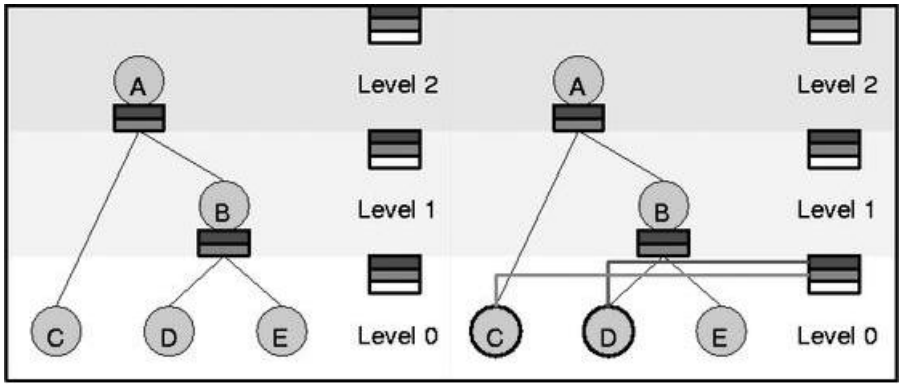
Simulace výběru paketu

Během výběru paketu musíme najít zelenou třídu s nejnižší úrovní, která má jako potomka plný list a k potomkovi vede cesta s oranžovými třídami (včetně listu).

Pro účel vysvětlení algoritmu jsme pozměnili označení tříd. List je označen tučnou kružnicí, pokud je plný, vnitřek kružnice je světle šedý, pokud reprezentuje zelenou třídu, bílý, pokud je třída oranžová a tmavě šedá, pokud je třída červená. Podívejme se na obrázek 2.8.

Každý vnitřní uzel si udržuje seznam potomků, kteří by si od něj rádi půjčili nějaký tok. Podle algoritmu 1 to mohou být pouze oranžové třídy. Tento seznam musí být veden zvlášť pro jednotlivé priority, neboť uzel může být předkem listu s libovolnou prioritou. Naše příklady mají priority pouze dvě, vysoká (tmavě šedý obdélník) a nízká (světlejší obdélník o něco níže). Oba reprezentují interní půjčovací seznamy.

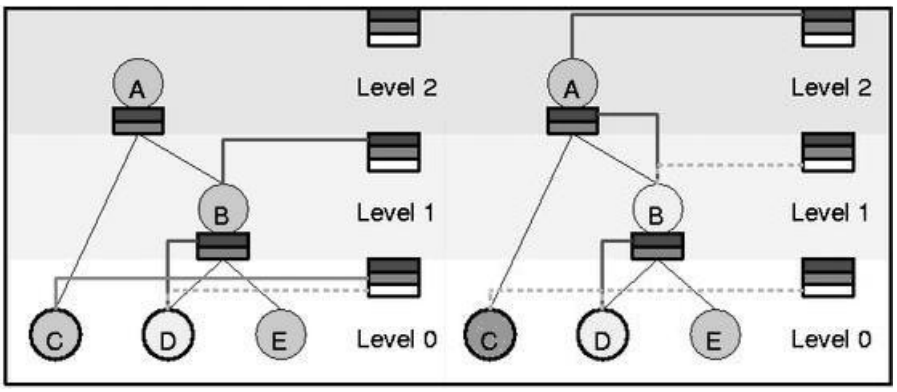
V pravé části obrázku vidíme podobné obdélníčky, dva horní reprezentují globální půjčovací seznam, o posledním bílém budeme mluvit později. Seznam obsahuje třídy určité úrovně a priority, které jsou plné a zároveň zelené (nespokojené). Jinými slovy, třídy, které jsou na globálním seznamu jsou připraveny odeslat paket. Nyní tedy v souladu s



Obrázek 2.8: **Vlevo:** žádné pakety, třídy jsou zelené, pouze D má vyšší prioritu.**Vpravo:** přišly pakety pro C a D.

algoritmem 1 stačí vybrat nejspodnější úroveň s neprázdným globálním seznamem, zde seznamem s nejvyšší prioritou, a sledovat interní seznamy, které nás dovedou k listu, který má odeslat paket.

Obrázek 2.8 vlevo ukazuje stav, kdy v HTB není žádný paket a všechny třídy jsou zelené. Pokud přijdou pakety určené listům C a D, listy se změní na plné a protože jsou zatím zelené (pod limitem), můžeme je podle priorit připojit ke globálním seznamům (*viz* obrázek 2.8 vpravo). Pokud by si síťová karta nyní řekla o paket, velmi rychle zjistíme, že máme poslat paket z listu D a případně další z C, pokud D bude prázdný. Podívejme se, co by se stalo, pokud tedy pošleme paket z D a estimátor rozhodne, že je D oranžová třída (došlo k překročení r). Tento stav vidíme na obrázku 2.9 vlevo.



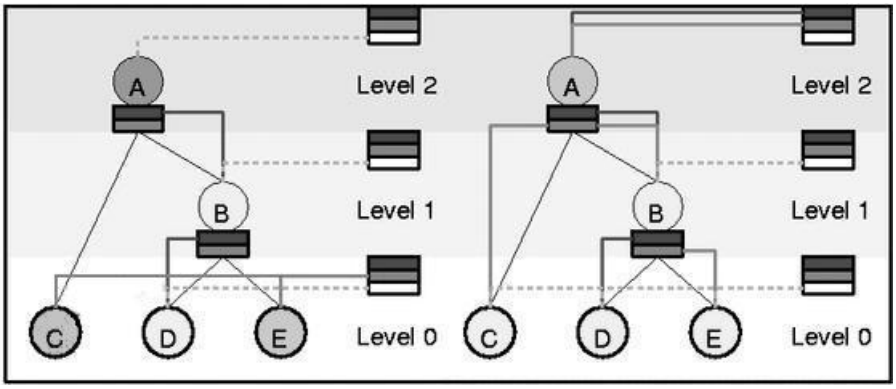
Obrázek 2.9: **Vlevo:**D je nyní oranžová.**Vpravo:** C je červená a zároveň B je oranžová.

List D byl odpojen od globálního seznamu, neboť už nemůže odesílat bez půjčky a zároveň byl zařazen na vysokoprioritní interní seznam třídy B. Samotná třída je zařazena na globální seznam úrovně 1, do vysoké priority. To proto, aby třída D mohla nadále posílat pakety s výpůjčkou od své rodičovské třídy B. Dále, pokud D nějakou dobu neodešle paket, tak se její barva změní po určité době opět na zelenou, zařadíme D na speciální čekací

seznam (reprezentuje ho bílý obdélník, o kterém byla řeč v úvodu). Každá úroveň má svůj čekací seznam, třídám je po uplynutí předem spočítaného času změněna barva.

Při žádosti o další paket bude vybrána třída C a nikoliv D a to i přesto, že D má vyšší prioritu. Je to proto, že v globálním seznamu začíná cesta k C na nejnižší úrovni (jde o splnění pravidla 1, proč dávat přednost třídě D, když C má nárok na svůj garantovaný tok).

Obrázek 2.9 vpravo zobrazuje situaci po odeslání paketu z C. Řekněme, že tato třída překročí svůj strop c a je tudíž červená. B je navíc oranžová. Vidíme, jak se B připojí na interní seznam třídy A a místo B je nyní v globálním seznamu třída A. C je pouze na čekacím seznamu(nemůže si ani půjčit) a jediný, kdo může posílat pakety je D.



Obrázek 2.10: **Vlevo:** A je červená, C i E jsou plné a zelené.**Vpravo:** všechny listy si půjčují od A.

Na obrázku 2.10 vlevo vidíme situaci, kdy A je červená, a C i E jsou plné a zelené. Zde lze pozorovat, že cesta po interních seznamech existuje, i když nejvyšší třída nemá žádný spoj na globální seznam. Navíc jsou C i E připojeny ke společnému globálnímu seznamu. V tomto případě je nutno pravidelně střídat odebírání paketů mezi C a E v poměru jejich r. Obrázek 2.10 vpravo poukazuje na možnost napojení více tříd na společné interní seznamy. Nebýt vyšší priority D, byli bychom nuceni střídat odebírání paketu mezi všemi listy. Neboť má ale D nejvyšší prioritu, bude z globálního seznamu vybrán pouze D.

Parametry a použití utility tc

- *default* - parametr přiřadí třídu pro neklasifikovaný provoz.
- *mtu* - maximální velikost paketu, který může být odeslán.
- *rate* - představuje požadovanou rychlost, která má být přidělena dané třídě (třída si stále může půjčit).
- *ceil* - definitivní strop, vyšší rychlosti pro danou třídu nelze dosáhnout.
- *prio* - reprezentuje prioritu dané třídy.
- *burst* - maximální velikost paměti, kde budou ukládány pakety pro garantovanou rychlost.

- *cburst* - maximální velikost paměti pro strop.
- *quantum* - kolik bytů můžeme najednou poslat z listu, byl-li vyzván odeslat pakety.

Ukázková konfigurace

Funkčnost je identická s příkladem na CBQ, který je uvedený výše. Jak je vidět v následujících příkazech, všechny třídy používají HTB. Třídy označené jako 1:10 a 1:20 mají stejný význam jako v CBQ příkladě, to znamená ze do 1:10 bude řazen HTTP provoz a do 1:20 FTP provoz. Tentokrát nám zde přibude třída označena 1:30, která bude sloužit pro zbývající blíže nespecifikovaný provoz, je zajímavé že v této třídě je nastavena maximální bitová rychlost na 10kbit/s ale hodnotou ceil je řečeno, že může dosáhnout až rychlosti 6Mbit/s.

```
# tc qdisc add dev eth1 root handle 1: htb default 30
# tc class add dev eth1 parent 1: classid 1:1 htb \
  rate 6Mbit burst 15k
# tc class add dev eth1 parent 1:1 classid 1:10 htb \
  rate 5Mbit burst 15k
# tc class add dev eth1 parent 1:1 classid 1:20 htb \
  rate 3Mbit ceil 6Mbit burst 15k
# tc class add dev eth1 parent 1:1 classid 1:30 htb \
  rate 10kbit ceil 6Mbit burst 15k
```

V následující sekvencí příkazů, přiřadíme každé třídě jednu disciplinu front, u všech tříd je shodně použito sfq.

```
# tc qdisc add dev eth1 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth1 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth1 parent 1:30 handle 30: sfq perturb 10
```

Opět je nutné v závěrečné fázi nakonfigurovat filtry.

```
# U32="tc filter add dev eth1 protocol ip parent 1:0 prio 1 u32"
# $U32 match ip sport 80 0xffff flowid 1:10
# $U32 match ip sport 21 0xffff flowid 1:20
```

Nezávislost disciplíny HTB na přenosovém médiu, velmi dobrá škálovatelnost a větší rychlost zpracování než CBQ, to vše byly důvody, proč HTB použít v praktické implementaci v kombinaci s utilitou tc.

3. Administrační systém

Tato kapitola popisuje praktickou část diplomové práce - administrační systém pro vzdálenou správu směrovačů na síti s kombinovanou architekturou přenosových prostředků, ethernetu a bezdrátových sítí. Systém umožňuje správci ovládat nastavení služeb na jednotlivých směrovačích. Tímto rozumíme přidělovat příslušnou šířku pásma koncovým uživatelům sítě. Systém obsahuje kontrolní a korekční mechanizmy, které hlídají, aby tato šířka nebyla překračována. Zároveň dovoluje správci tyto mechanizmy ovlivňovat a měnit a tím zasahovat do chodu systému.

3.1 Analýza

Analýza vychází z požadavků, které jsou na systém kladeny. Tyto požadavky jsou blíže popsány ve specifikaci požadavků *viz* sekce 3.1.1 Z požadavků je vytvořen model požadavků a z něj pak dále model analýzy.

3.1.1 Specifikace požadavků

Na administrační systém jsou kladeny následující požadavky:

- Systém lze ovládat z grafického rozhraní, které je dostupné přes Internet

  Grafické rozhraní musí umožňovat zobrazení informací o jednotlivých klientech, směrovačích a dodatečně i informace o zvláštním nastavení[1]. Informace jsou uloženy v databázi. Rozhraní obsahuje administrační část, ze které lze ovládat nastavení jednotlivých směrovačů. Rozhraní podporuje bezpečnostní přihlašovací a komunikační mechanizmy.

- Systém zajišťuje nastavení šířky pásma jednotlivým uživatelům sítě

  Systém podle informací z databáze generuje skripty pro nastavení šířky pásma klientům, kteří spadají jako koncoví uživatelé pod příslušný směrovač. Dále zajišťuje spuštění tohoto skriptu na směrovači, v případě překročení přidělené šířky pásma je zahájen korekční mechanizmus a vygenerován a spuštěn nový skript.

- Systém podporuje monitoring průtoku a spouští korekční mechanizmy v případě překročení

  Na straně směrovače musí být hlídány průtoky klientů, které jsou v pravidelných intervalech kontrolovány a v případě překročení povolené tolerance šířky pásma musí být spuštěn korekční mechanizmus, který informuje systém o potřebné změně nastavení. Jednotlivé průtoky jsou pravidelně ukládány ve formátu pro nástroj Rrdtool [Rrd03].

- Systém bude nasazen v prostředí operačního systému UNIX

[1] jedná se o dodatečné změny nastavení šířky pásma klientům, které jsou mimo dohodnuté podmínky

- Systém musí spolupracovat s bezpečnostním systémem, který povoluje uživatelů přístup do sítě

3.1.2 Rozbor požadavků

Ze specifikace požadavků vidíme, že administrátorské rozhraní ukazuje na webovou aplikaci, která ovládá chování serveru. Ten komunikuje se směrovači a řídí jejich chování. Vše postaveno na platformě UNIX. Systém je napojen na databázi, ze které čerpá potřebné informace.

Celý systém můžeme rozdělit na tři logické části:

1. **Hlavní server** - nejdůležitější článek systému, je napojen na databázi, z které čerpá informace pro generování skriptů, které jsou spouštěny na směrovačích a obsahují nastavení šířky pásma jednotlivým klientům, dále zpracovává a reaguje na požadavky jak správce systému tak jednotlivých směrovačů.

2. **Klient na směrovači** - pravidelně kontroluje průtoky jednotlivým uživatelům a v případě překročení limitu informuje server, klient je napsán v jazyce C z důvodu rychlosti a omezené kapacity místa, pro kontrolní mechanizmy jsou použity prostředky operačního systému UNIX

3. **Grafické uživatelské rozhraní** - dovoluje správci systému pohodlně kontrolovat dění na síti, zobrazuje nejdůležitější informace o klientech a jejich nastaveních, umožňuje evokovat generování nových skriptů nastavení, dovoluje komunikovat s databází.

Volby technologií

- **Databáze** - jako databázového severu použijeme MySQL, výhody tohoto systému jsou jak finanční nenáročnost, tak jeho dobrá dostupnost a jednoduchá správa. Současné MySQL podporují téměř všechny podstatné funkce důležité pro efektivní vývoj (cizí klíče, indexování, granty, apod.).

- **Grafické uživatelské rozhraní** - z důvodu potřeby rychlého přístupu k systému jsem zvolil dynamickou webovou aplikaci realizovanou technologií JSP, nasazenou na serveru JBoss s využitím kontejneru Tomcat. Kombinace těchto technologií je nezávislá na platformě a poskytuje všechny prvky bezpečnosti, které jsou požadovány. Další velkou výhodou je možnost vkládání Java kódu do JSP stránek a tedy využití velkého množství API pro práci s databází a sítí.

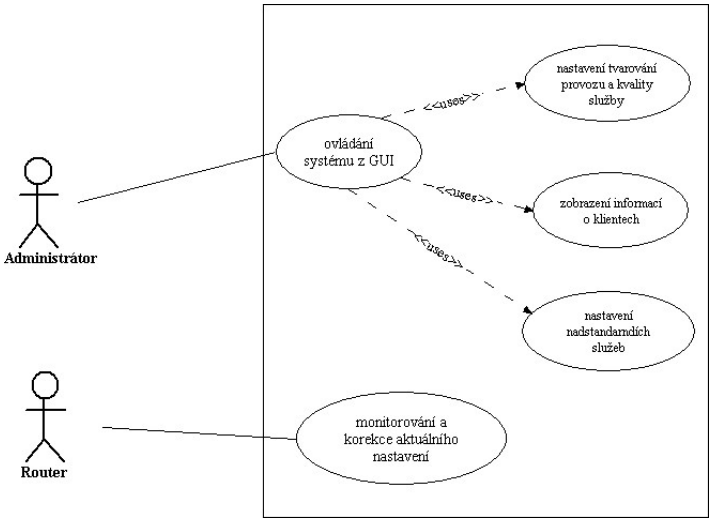
- **Server** - jako hlavní článek, který bude neustále v pohotovosti, volím sever napsaný v jazyce Java, opět kvůli rozsáhlým API pro síťovou komunikaci a možnostem využívat příkazů nativního operačního systému. Verze prostředí programovacího jazyku Java je 1.4.1.

- **Klient** - klientský program, který bude umístěn přímo na směrovači, je napsán v jazyce C, dále jsou využívány programy a příkazy operačního systému UNIX. Vše z důvodu nedostatku místa na směrovači, nelze doinstalovat virtuální stroj jazyka Java.

Systém při přihlášení do grafického uživatelského rozhraní vyžaduje autentifikaci pomocí formulářů přes HTTPS, skripty jsou spouštěny přes ssh s požadavkem na RSA autentifikaci.

3.1.3 Model požadavků

Činnost systému lze tedy rozdělit na **ovládání systému z GUI**, kde dochází ke generování a spouštění příslušných skriptů, dále zobrazování a editaci informací, při které zobrazujeme data z databáze, nastavujeme nadstandardní služby a na **monitorování a korekci aktuálního nastavení**, kde se monitorují průtoky dat a spouští se žádosti o korekce v případě překročení limitu. Systém lze používat podle obrázku 3.1.



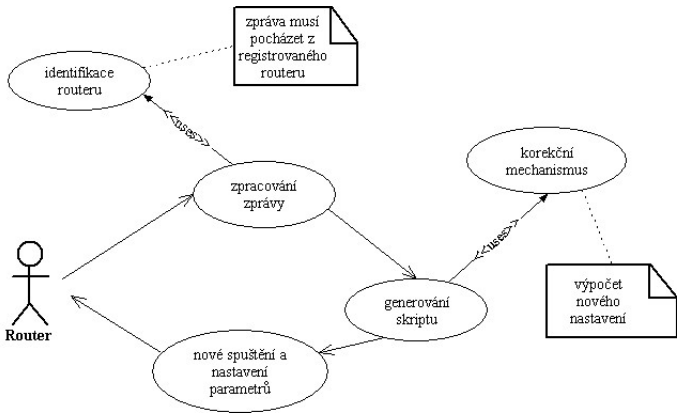
Obrázek 3.1: Případy použití.

Ovládání systému z grafického uživatelského rozhraní lze rozdělit na následující logické celky:

- nastavení tvarování provozu a kvality služby, tj. systém musí být schopen zpracovat, nastavit a spustit skript na příslušném směrovači tak, aby byla všem uživatelům koncové sítě přidělena správná šířka pásma včetně nastavení maximální šířky pásma směrovače, které nesmí být překročeno.
- zobrazení informací o klientech, tj. systém musí mít přístup do centrální databáze klientů, ze které získá potřebné informace, tato databáze je pro administrátora sítě přístupná pouze pro čtení
- nastavení nadstandardních služeb, tj. přístup a zápis do zvláštní tabulky

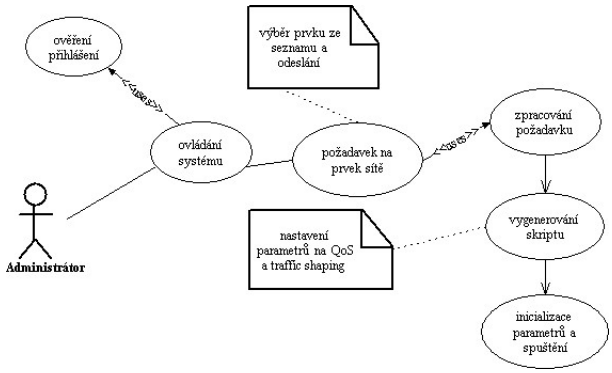
Monitoring a korekce aktuálního nastavení *viz* obrázek 3.2 lze rozdělit na následující celky:

- pravidelné monitorování průtoků koncových klientů, uložení ve formátu pro RrdTool



Obrázek 3.2: Případy použití korekce.

- kontrola průtoku, při překročení se odesílají data se žádostí o korekci
- korekce nastavení a spuštění nově vygenerovaného skriptu



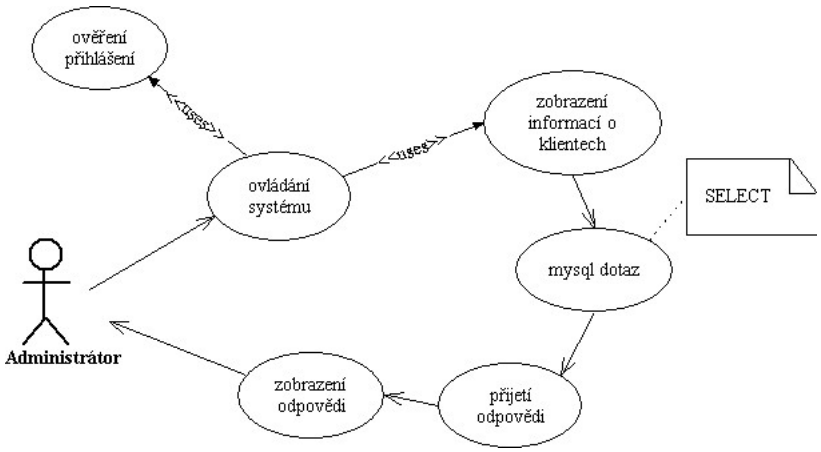
Obrázek 3.3: Případy použití nastavení tvarování provozu a kvality služby.

Nastavení tvarování provozu a kvality služby rozdělíme do níže popsaných částí, grafické znázornění je obrázku 3.3.

- požadavek na nastavení příslušného směrovače
- zpracování požadavku a vygenerování skriptu podle informací z databáze
- uložení nastavení pro daný směrovač a spuštění skriptu

Zobrazení informací o klientech, tento případ lze rozdělit na:

- žádost o zobrazení dat o klientech pomocí SQL dotazu
- přijmutí odpovědi z databáze

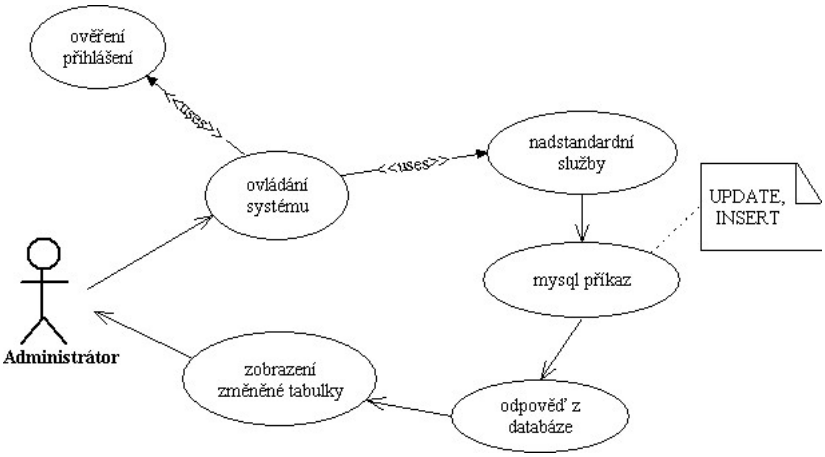


Obrázek 3.4: Případy použití zobrazování dat.

- zobrazení odpovědi administrátorovi

Grafické znázornění nalezneme na obrázku 3.4

Nastavení nadstandardních služeb je poslední případ užití, kterým se budeme zabývat, celou situaci lze vidět na obrázku 4.5.

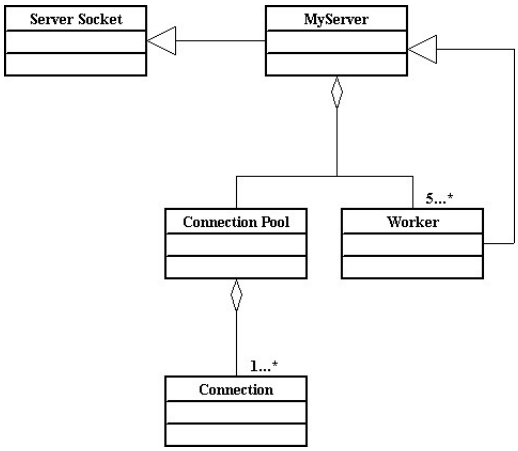


Obrázek 3.5: Případy použití nastavení nadstandardních služeb

- zaslání změn pomocí SQL příkazu databázi
- přijmutí odpovědi z databáze
- zobrazení odpovědi administrátorovi

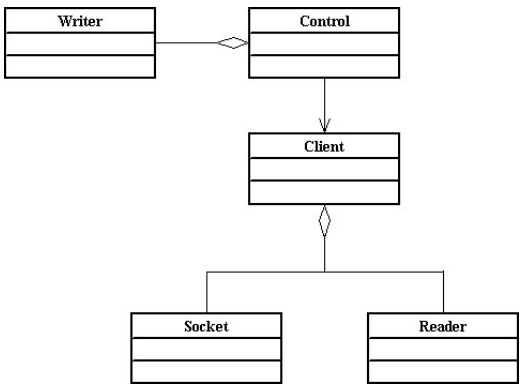
3.1.4 Model analýzy

Tento model dává systému objektovou strukturu. Typy objektů popisují třídní diagramy a vztahy mezi nimi. Sekce obsahuje třídní diagramy základních částí systému tak, jak byly navrženy v analýze. Pro zjednodušení nejsou popsány metody, ale pouze účel a vlastnosti tříd.



Obrázek 3.6: Třídní diagram hlavního serveru (analýza)

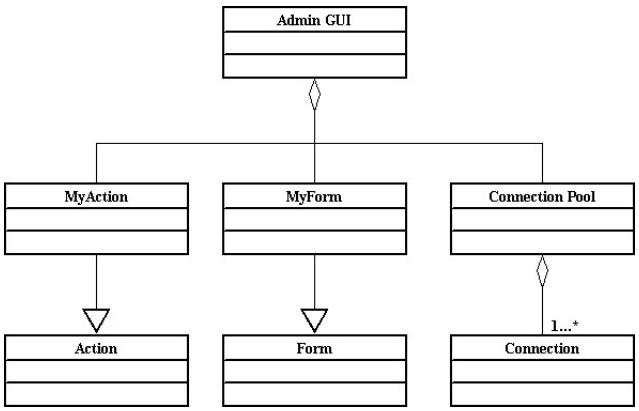
Hlavní server
Server při spuštění vytváří instanci třídy Server Socket, která je vázána na IP adresu a poslouchá na daném portu. Dále je vytvořeno spojení s databází přes tzv. Connection Pool, ten sdružuje, udržuje a ruší aktivní spojení s databází. Třída Worker rozšiřuje hlavní třídu Server. Pokaždé, je-li akceptováno nové spojení, je předáno ke zpracování právě instanci této třídy. Třídní diagram hlavního serveru je na obrázku 3.6.



Obrázek 3.7: Třídní diagram klienta na směrovači (analýza)

Klient
Jelikož je na straně směrovače velmi málo místa, volíme prostředky operačního systému UNIX. Třída Control v pravidelných intervalech kontroluje průtoky dat jednotlivým kon-

covým uživatelům, ukládá je ve formátu pro RrdTool [2] a jednou za hodinu dává pokyn k přenesení těchto dat do databáze. V případě překročení nastavené šířky pásma u kteréhokoliv uživatele (z jakýchkoliv důvodů) spouští Clienta, který vytvoří soketovou komunikaci s hlavním serverem a předá mu žádost o korekci nastavení. Třídní diagram klientské strany je na obrázku 3.7.



Obrázek 3.8: Třídní diagram grafického rozhraní administrátora (analýza)

Grafické uživatelské rozhraní administrátora
Prostředí je složeno z formulářů, které vyvolávají příslušné akce, vše podle pravidel Model View Controller, které JSP samozřejmě podporují. Přístup do databáze řídí opět Connection Pool. Třídní diagram vidíme na obrázku 3.8.

3.2 Návrh

Návrh ve svém modelu, oproti analýze, bere v úvahu konkrétní implementační prostředí. Rozšiřuje a zpestřuje model analýzy a vytváří model návrhu. I v této sekci je zachováno členění aplikace z modelu analýzy, části však obsahují detailní popis chování komponent. K popisu použijeme nástroj UML [3].

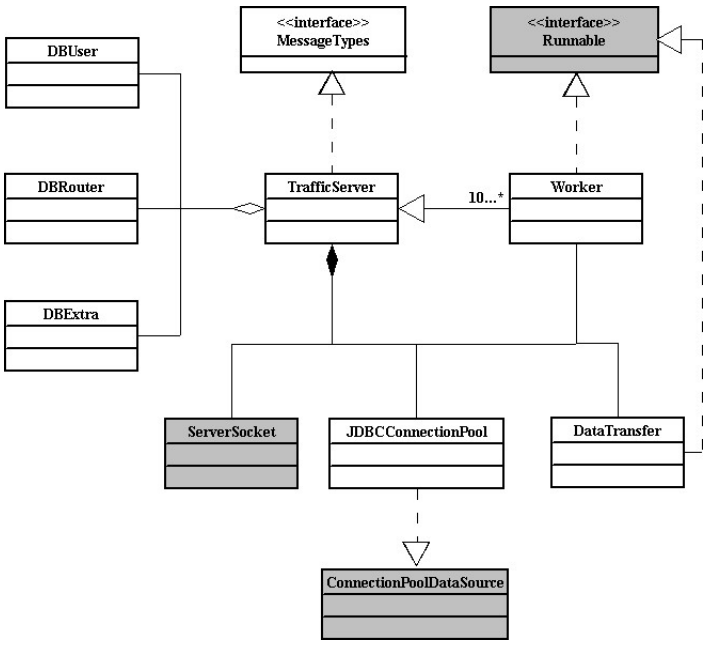
3.2.1 Hlavní server

Princip fungování severu se od analýzy neliší. Došlo k rozšíření tříd o třídy implementující typy zpráv, na které server reaguje a dále o třídy, které zapouzdřují informace přístupu do databáze. Poslední změna se týká postavení třídy Server Socket v diagramu, sever z ní nedědí, ale vlastnosti agreguje. Na obrázku 3.9 vidíme diagram tříd, který tuto situaci popisuje.

Popis tříd:

- *TrafficServer* - třída implementuje rozhraní *MessageTypes*, je to množina všech zpráv, na které server reaguje. Načítá ze souboru konfiguraci serveru a parametry předává dál vytvářeným instancím. Třída je pevně svázána podle konfiguračních

[2] jedná se o průměrné rychlosti (bitrates) uživatelů měřené každých pět minut
[3] v třídních diagramech jsou standardní třídy zvýrazněny šedou barvou



Obrázek 3.9: Třídní diagram grafického hlavního serveru (návrh)

parametrů s IP adresou a portem, na kterém poslouchá. Akceptuje-li spojení, je předáno ke zpracování instanci třídy Worker. TrafficServer vytváří těmto instancím správce (pool), který řídí předání zpracování.

- *JDBCConnectionPool* - tato třída se stará o spojení s databází, vytváří tzv. *PooledConnection*, které sdružuje a spravuje připojení k databázi. Udržuje aktivní spojení s databází a na požádání (*getConnection()*) předá jedno ze svých existujících spojení nebo vytvoří spojení nové, jsou-li všechna aktivní. Jakmile aplikace dané spojení nepotřebuje, metoda *close()* nezavře spojení, jak by se dalo čekat, ale předá ho zpět správci připojení. Díky tomu může správce zabránit zbytečnému zatížení systému v důsledku tvorby stále nových připojení.

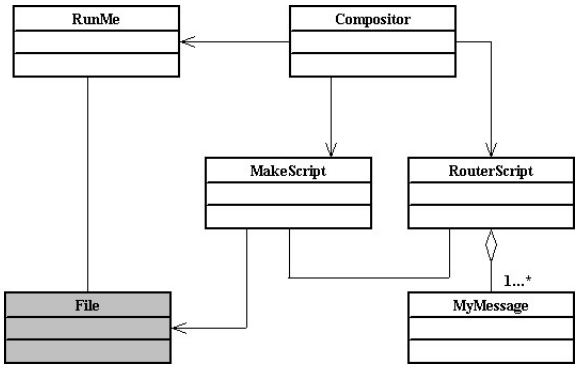
- *ServerSocket* - třída je importována z balíku *java.net.ServerSocket* a stará se o soketovou komunikaci. Je svázána s IP adresou a portem, na kterém poslouchá.

- *DBUser,DBRouter,DBExtra* - do těchto tříd je načtena struktura tabulek databáze, obsahují atributy tabulek, se kterými aplikace pracuje a potřebuje je znát. Zůstává tak nezávislá na ostatních změnách v atributech, se kterými nepracuje.

- *DataTransfer* - tato třída byla vytvořena pro přesun dat ze směrovače na hlavní server. Toto vlákno bude buzeno každou hodinu a posbírá ze směrovačů naměřené průtoky klientů.

- *Worker* - třída implementuje rozhraní *Runnable*, vlákna jsou spravována v poolu třídy TrafficServer, počet vláken, které jsou při startu vytvořeny, záleží na konfiguračním nastavení. Ihned po startu jsou uspány, vzbudí je až spojení, které akceptuje

TrafficServer. Jsou-li všechna vlákna zaměstnána a přijde-li požadavek na další obsloužení spojení, vytvoří se jen „additional worker“, jehož činnost je po obsloužení spojení ukončena. Jelikož je funkce této třídy rozsáhlá, budeme se jejím popisem zabývat více.



Obrázek 3.10: Třídní diagram kompozice skriptu (návrh)

Worker

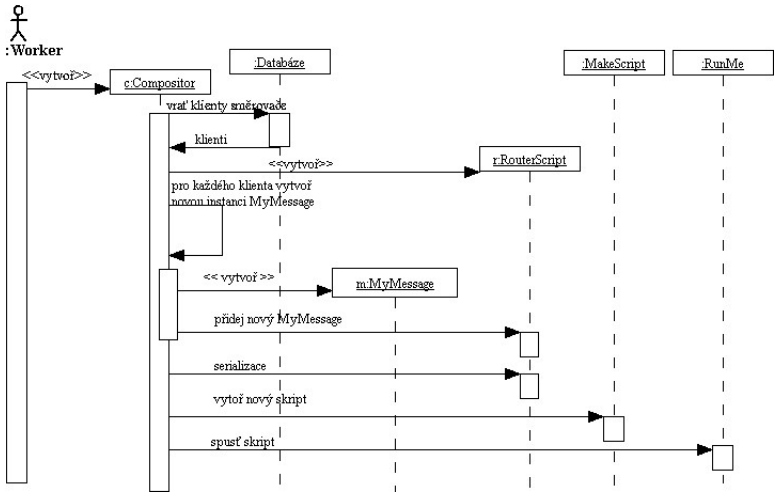
Jak již bylo uvedeno, třída Worker obsluhuje veškeré příchozí požadavky. V první fázi reaguje pouze na dva typy zpráv. Jsou to zprávy LOGIN z administračního systému a zpráva ROUTER REQUEST ze směrovače. Jakékoliv jiné zprávy ignoruje a okamžitě uzavírá spojení. Spojení je také uzavřeno, vyprší-li nastavený čas od připojení k doručení první zprávy.

Je-li detekována zpráva z administračního systému, nejprve probíhá ověření přístupu, systém využívá správce zabezpečení jazyka Java třídy Permission. Pokud je přístup povolen, Worker plní příkazy administrátora, až do jeho odhlášení (zpráva QUIT).

Zprávy, na které Worker reaguje jsou implementovány třídou *MessageTypes*. Jedním z hlavních úkolů je ale sestavování skriptů. Na obrázku 3.10 vidíme diagram tříd, které spolupracují při této činnosti.

- *Compositor* - třída řídí činnost generování skriptu. Implementuje rozhranní Runnable, jelikož je velmi pravděpodobné, že při spuštění aplikace budou vzneseny mnohonásobné požadavky na generování skriptů a tento způsob zpracování velmi urychlí. Třída vybere z databáze data, které se týkají konkrétního směrovače a vytvoří objektové zapouzdření budoucího skriptu, které se serializuje z důvodu dalšího použití nebo oprav. Kompozice obsahuje IP klienta a povolenou šířku pásma spolu s třídou příslušnosti, do které byl zařazen(*viz* skce 3.2.4). Nakonec vytvoří instance tříd *MakeScript* a *RunMe*.

- *MakeScript* - třída zpracuje vygenerovaný objekt a vytvoří z něj spustitelný skript. Ten je tvořen příkazy utility *tc*, které přiřazují koncovým uživatelům sítě deklarovanou šířku pásma. Objekty jsou seskupovány podle příslušnosti k dané třídě, je

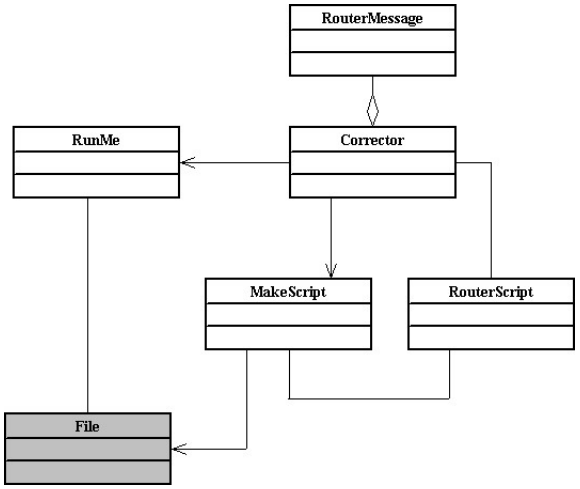


Obrázek 3.11: Sekvenční diagram kompozice skriptu (návrh)

vytvořen hierarchický strom, kde každému uzlu a listu jsou přiřazeny povolené průtoky. Následně je každá IP adresa označkována a přiřazena do příslušné části stromu. Celý skript je uložen do souboru.

- *RunMe* - zajišťuje vzdálené spuštění právě vygenerovaného skriptu. Třída *RunMe* využívá vlastností programů *scp* a *ssh*. Programem scp skript nakopíruje do paměti směrovače a pak ho spustí přes ssh. Oba programy podporují RSA autentifikaci, čímž obejdeme zadávání přístupového hesla.

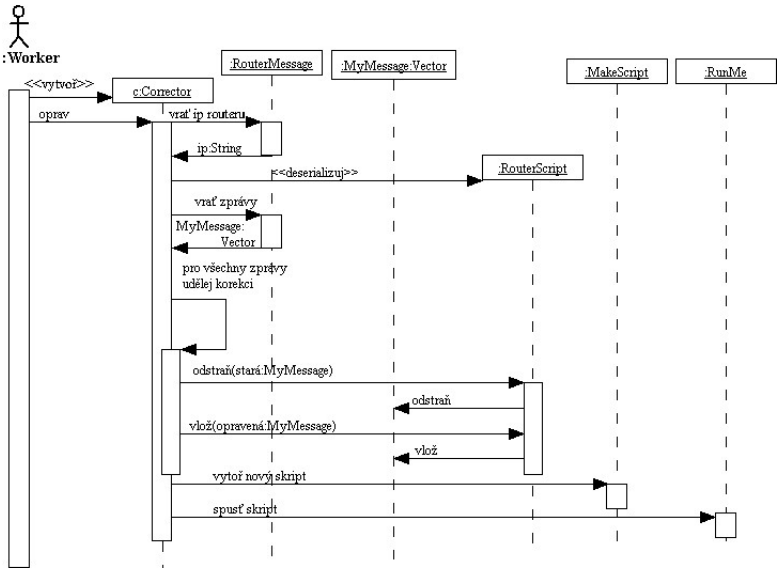
Sekvenční diagram kompozice skriptu je na obrázku 3.11. Jako iniciátora uvažujeme třídu Worker, která rozpoznala zprávu pro kompozici.



Obrázek 3.12: Třídní diagram korekce (návrh)

Je-li detekována zpráva ze směrovače, opět dochází k ověření IP adresy v databázi. Jednou za hodinu posílá každý směrovač historii průtoků jednotlivých klientů, ta je ukládána do databáze. Dalším druhem zprávy může být žádost o korekci. Ta je zaslána směrovačem v případě, že byly překročeny povolené limity průtoků. Systém na ně reaguje tak, že procentuálně upraví šířku pásma, tj. překročil-li uživatel svůj limit o 20%, bude mu o 20% snížen. Diagram tříd pro tuto situaci vidíme na obrázku 3.12.

- *Corrector* - řídí celý proces korekce skriptu. V konstruktoru je mu předán objekt třídy *RouterMessage*, který v sobě nese informace o nepovolených průtocích. Podle IP směrovače si Corrector deserializuje příslušný objektový skript směrovače, provede korekci šířky pásma a nový objekt předá třídě *MakeScript*. Dále následuje již známá procedura generování spustitelného skriptu a jeho spuštění.



Obrázek 3.13: Sekvenční diagram korekce skriptu (návrh)

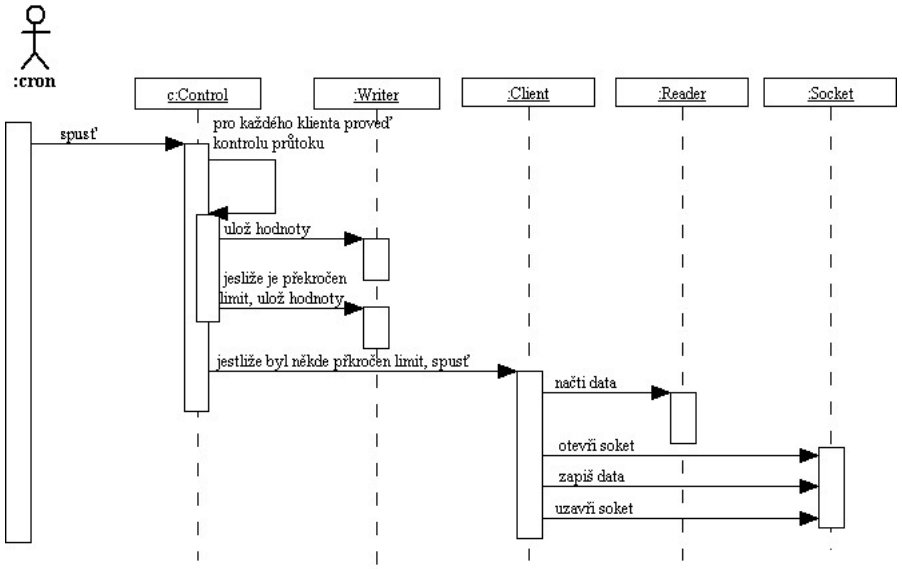
Důvod, proč jsou objektové skripty s originálním nastavením serializovány, je jednoduchý. Umožňují jednoduše změnit obsah, aniž by bylo nutno složitě procházet generovaný skript, zvláště, byl-li by hierarchizován. Vygenerované skripty navíc udržují informace o aktuálním nastavení směrovače. Sekvenční diagram korekce nastavení je na obrázku 3.13. Iniciátorem je opět třída Worker.

3.2.2 Klient - směrovač

Při návrhu systému na straně směrovače se budeme držet analýzy. Omezení, které je nám kladeno ze strany volné kapacity na serveru, je striktní. Jelikož jsou směrovače postaveny na platformě UNIX, využijeme možnosti tohoto operačního systému a v návrhu komponent budeme brát v úvahu tuto skutečnost.

- *Control* - zodpovídá za pravidelné kontroly průtoků jednotlivých klientů, spouštění v daných časových intervalech ukazuje na nasazení v kombinaci s daemonem *cron*. Průtoky uživatelů budou měřeny každých pět minut nástrojem *iptables* a budou ukládány do souboru. Pokud *Control* při kontrole průtoků narazí na překročení limitu, volá program *Client* a předá mu parametry překročení.
- *Client* - program otevře soketovou komunikaci s hlavním serverem a předá mu parametry překročení limitů. Vše, jak bylo deklarováno v analýze.

Sekvenční diagram jednoho kontrolního cyklu na směrovači je na obrázku 3.14.



Obrázek 3.14: Sekvenční diagram jednoho kontrolního cyklu na směrovači (návrh)

3.2.3 Grafické uživatelské rozhraní

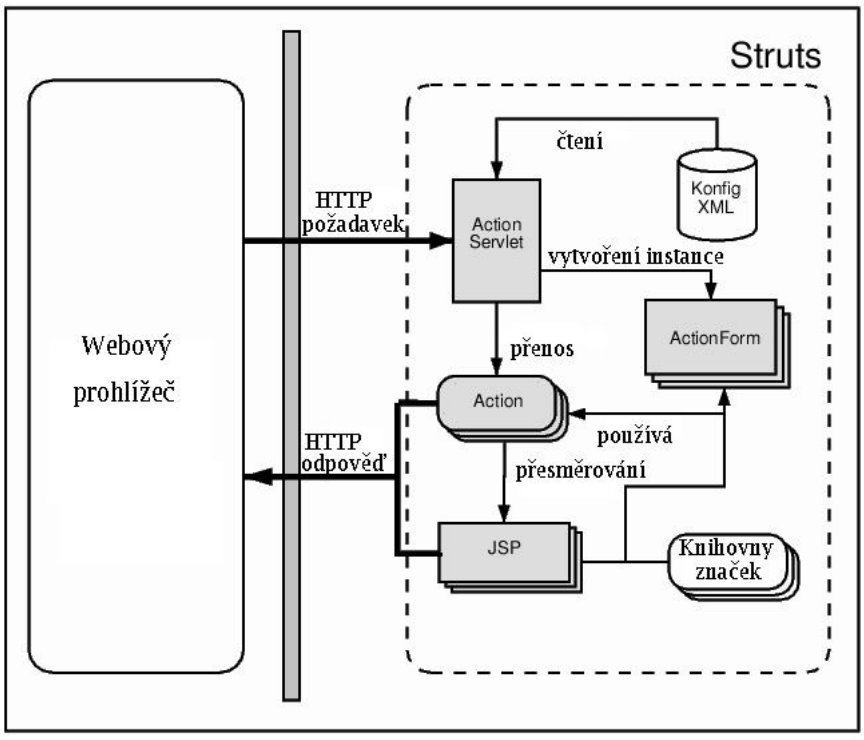
Grafické uživatelské rozhraní je navrženo jako dynamická webová aplikace. Využijeme technologie *Struts* a *JSP* stránek. Struts je systém, který implementuje návrhový vzor *Model View Controller* (MVC). Celý systém můžeme rozdělit na čtyři základní oblasti, které s tímto modelem korespondují.

- třída **Action** - reprezentuje **model**. Řídí logiku aplikace a manipuluje s daty.
- JSP stránky a knihovny značek (custom tags) reprezentují **view**. Slouží pro zobrazování dat uživateli.

- třída **ActionServlet** - reprezentuje **controller**. Zpracovává požadavky uživatele a předává je příslušné třídě Action.
- množství podpůrných tříd, které umožňují parsování XML, automatické nastavování proměnných, lokalizaci atd.

Struts má ve svém středu tzv. ActionServlet, který naslouchá požadavkům uživatele, resp. webového prohlížeče, a přeposílá je dále speciálním komponentám, zvaným Action. Každá Action reprezentuje jednu specifickou funkci webové aplikace. Je obvykle spojena s množinou dalších komponent, kterým říkáme *ActionForm* (jsou to tzv. javabeans, které spojují proměnné s daty, získané ze stránky). Pokud uživatel vyplní na stránce nějaká data, obvykle nechceme, aby se ukládaly do databáze ihned, udržujeme je proto v paměti například kvůli kontroly správnosti aj.

Vlastní spojení mezi *Action* a *ActionForm* se nazývá *ActionMapping*. Komponenta Action zpracuje HTTP požadavek prohlížeče a generuje odpověď. Přesměrování provede pomocí komponenty *ActionForward*. Architektura Struts je znázorněna na obrázku 3.15.

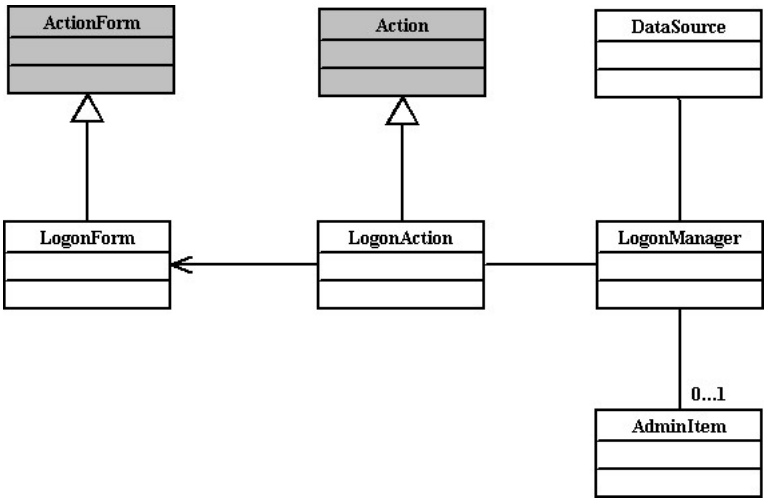


Obrázek 3.15: Schéma technologie struts.

Webové rozhraní dodržuje spojení příslušných akcí a formulářů, jak bylo deklarováno v analýze. Změna byla provedena v přístupové metodě k databázi. Nebude použit ConnectionPool, ale DataSource. DataSource bude definován v xml souboru a jeho bližší nastavení si uvedeme v implementační části (sekce 3.3.3). V následujících sekcích si rozebereme jednotlivé funkce webové aplikace.

Přihlášení do systému

Aplikace požaduje při přihlášení uživatelské jméno a heslo, pokud jsou správně zadány, je uživatel přesměrován na hlavní stránku. Zároveň je při této akci navázáno spojení s hlavním serverem. Tj. bude otevřen soket a výstupní stream, kterým budeme se serverem komunikovat. V této části musíme provést autentifikaci LOGIN příkazem. Komunikace je nešifrovaná, protože hlavní server a webový server, na který se bude uživatel hlásit, budou nasazeny na jeden stroj. Třídní diagram vidíme na obrázku 3.16.



Obrázek 3.16: Třídní diagram přihlášení do systému.

Popis tříd:

- *LogonForm* - potomek standardní třídy ActionForm, reprezentuje javabean logon a je přímo napojen na JSP stránku, odkud získá přihlašovací údaje.
- *LogonAction* - potomek třídy Action, tato akce je vyvolána stiskem tlačítka pro odeslání jména a hesla. Podle výsledku přihlášení je provedeno přesměrování na související stránky. Pokud bylo přihlášení správné, bude uživatel přesměrován „do systému“, jinak bude vrácen zpět na přihlášení.
- *LogonManager* - třída provádí kontrolu přístupového jména a hesla.
- *AdminItem* - třída zapouzdří informace o přihlášení.

Zobrazování informací o klientech

Administrátor má možnost zobrazit si informace o klientech, které jsou pro něj podstatné, struktura informací je rozebrána v sekci 3.2.4. Dále je mu dána možnost informace řadit podle různých kritérií. Tyto informace budou pro nás pouze pro čtení.

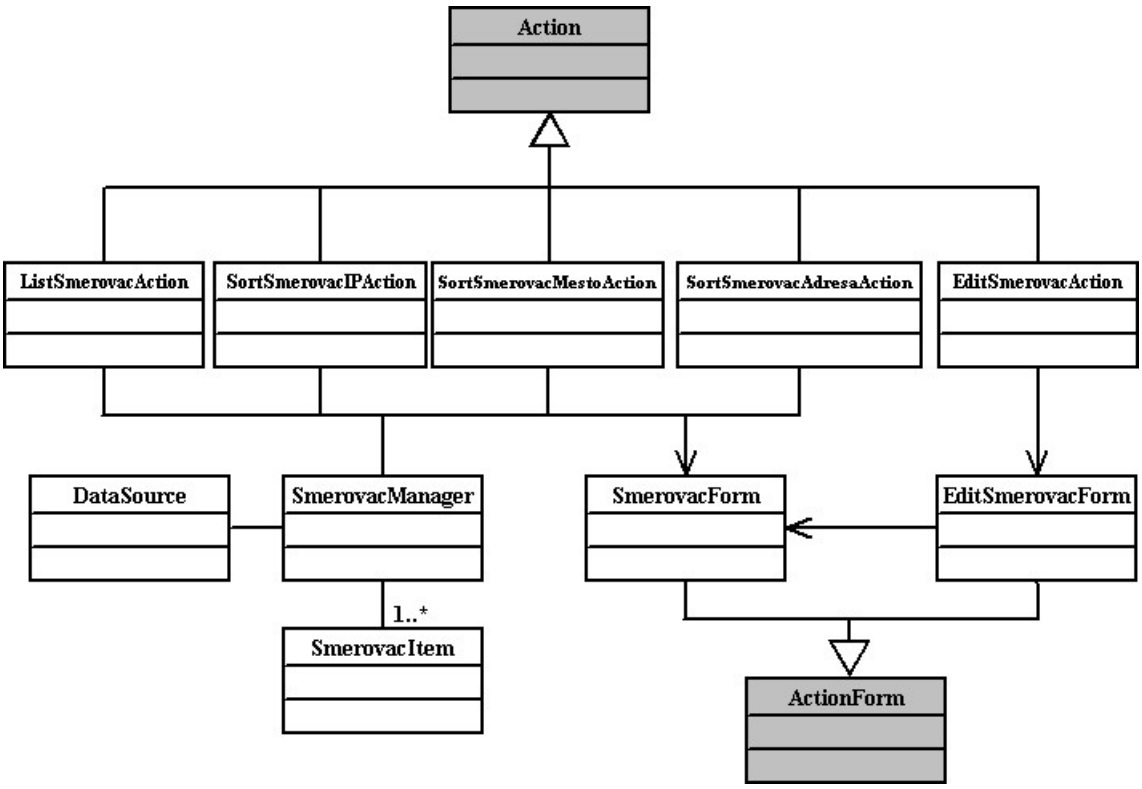
- *ListUzivatelForm* - potomek standardní třídy ActionForm, je přímo napojen na JSP stránku a reaguje na požadavky administrátora.

- *ListUzivatelAction* - potomek třídy Action, akce je vyvolána z hlavní stránky a zajišťuje přesměrování na stránku, kde budou zobrazeny informace o klientech.
- *SortUzivatelIP* - potomek třídy Action, akce seřadí klienty podle IP adresy.
- *SortUzivatelPrijmeni* - potomek třídy Action, akce seřadí klienty podle Příjmení.
- *SortUzivatelSmerovac* - potomek třídy Action, akce seřadí klienty podle čísla směrovače.
- *UzivatelManager* - třída získává informace o klientech z databáze.
- *UzivatelItem* - třída zapouzdří informace o klientovi.

Evidence směrovačů

Dále bylo důležité zobrazit informace o příslušných směrovačích, opět s možností řazení podle kritérií. Navíc ale administrátor může informace editovat a dávat příkazy pro rekonfiguraci nastavení v důsledku provedených změn. Třídní diagram je zobrazen na obrázku 3.17.

- *ListSmerovacForm* - potomek standardní třídy ActionForm, je přímo napojen na JSP stránku a reaguje na požadavky administrátora.
- *ListSmerovacAction* - potomek třídy Action, akce je vyvolána z hlavní stránky a zajišťuje přesměrování na stránku, kde budou zobrazeny informace o směrovačích.
- *SortSmerovacIPAction* - potomek třídy Action, akce seřadí směrovače podle IP adresy.
- *SortSmerovacMestoAction* - potomek třídy Action, akce seřadí směrovače podle města, ve kterém jsou umístěny.
- *SortSmerovacAddressAction* - potomek třídy Action, akce seřadí směrovače podle adresy v konkrétním městě.
- *SmerovacManager* - třída získává informace o klientech z databáze, umí provést i zápis.
- *EditSmerovacForm* - potomek třídy ActionForm, je napojen na JSP stránku, kde bude probíhat editace.
- *EditSmerovacAction* - potomek třídy Action, předá data k zápisu do databáze, přesměruje administrátora zpět na stránku se směrovači.
- *RestartSmerovacAction* - potomek třídy Action, zadá příkaz hlavnímu serveru k rekonfiguraci příslušného směrovače.
- *RestartAllAction* - potomek třídy Action, zadá příkaz k rekonfiguraci všech směrovačů, třeba při startu systému.
- *UzivatelItem* - třída zapouzdří informace o klientovi.



Obrázek 3.17: Třídní diagram pro evidenci směrovačů.

Evidence nadstandardních služeb

V případě, že chce klient vyzkoušet vyšší rychlost připojení, vloží administrátor tyto údaje právě do tohoto formuláře. Opět je mu dána možnost informace zobrazovat a třídit podle kritérií.

- *ListExtraForm* - potomek standardní třídy ActionForm, je přímo napojen na JSP stránku s nadstandardními službami a reaguje na požadavky administrátora.
- *ListExtraAction* - potomek třídy Action, akce je vyvolána z hlavní stránky a zajišťuje přesměrování na stránku, kde budou zobrazeny informace o nadstandardních službách.
- *SortExtraIPAction* - potomek třídy Action, akce seřadí informace podle IP adresy klienta.
- *SortExtraPrijmeniAction* - potomek třídy Action, akce seřadí směrovače podle příjmení klienta.
- *ExtraManager* - třída získává informace o nastavení z databáze, vkládá do ní nové nastavení.
- *EditExtraForm* - potomek třídy ActionForm, je napojen na JSP stránku, kde bude probíhat editace.

- *EditExtraAction* - potomek třídy Action, předá data k zápisu do databáze, přesměruje administrátora zpět na stránku s nadstandardními službami.
- *InsertExtraAction* - potomek třídy Action, vloží do databáze nová data.
- *ExtraItem* - třída zapouzdří informace o těchto službách.

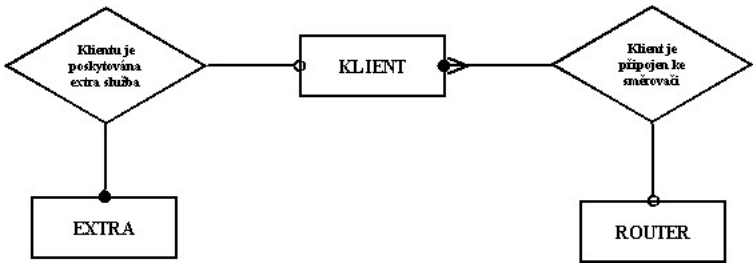
Příkazy pro hlavní server

Poslední formulář, který je třeba navrhnout, je formulář s příkazy pro hlavní server s následující sadou akcí.

- *NewLogAction* - potomek třídy Action, příkaz na vytvoření nového log souboru.
- *ServerDownAction* - potomek třídy Action, příkaz na zastavení činnosti serveru.

3.2.4 Datový model

Struktura tabulek, se kterými systém pracuje, je popsána v lineárním zápise. Tabulka KLIENT obsahuje data klientů, kteří využívají internetové připojení. Tato tabulka je ze strany našeho administračního systému pouze ke čtení. K tabulce ROUTER má systém plné práva, ta udržuje většinou technické informace o jednotlivých směrovačích. Poslední tabulka EXTRA slouží k ukládání dodatečných nastavení. Např. klient žádá o přidělení větší šířky pásma na zkoušku na tři dny, tyto údaje jsou pak zaneseny do této tabulky. Systém má zde opět plná přístupová práva. ER diagram je znázorněn na obrázku 3.18.



Obrázek 3.18: ER diagram (návrh)

Lineární zápis struktury jednotlivých tabulek

KLIENT(id_user, id_router, my_ip, mask, share, rate)

id_user -	primární klíč tabulky, jednoznačně identifikuje klienta
id_router -	identifikace směrovače, který směruje klienta
my_ip -	IP adresa klienta
mask -	síťová maska, která se vztahuje k IP adrese
share -	číslo, které určuje sdílenou nebo nesdílenou větev stromu provozu. Např. třída 0 je sdílená větev a budou do ní zařazeni běžní uživatelé, třída 1,2... sdružuje připojení firem, které mohou disponovat několika počítači, každé firmě bude přiděleno jedno číslo a počítače budou seskupovány podle něj
rate -	povolená šířka pásma

ROUTER(id_router, ip_router, mask, limit_in, eth_in,limit_out, eth_out, mesto, adresa, active)

id_router -	primární klíč, jednoznačně identifikuje směrovač
ip_router -	IP adresa směrovače
mask -	síťová maska směrovače, která se vztahuje k jeho IP adrese
limit_in -	maximální šířka pásma, kterou směrovač zvládne na eth_in
eth_in -	označení rozhraní, které je „vzdálenější“ od uživatelů
limit_out -	maximální šířka pásma, kterou směrovač zvládne na eth_out
eth_out -	označení rozhraní, které je „blíže“ k uživatelům
mesto	město, ve kterém se směrovač nachází
adresa	adresa směrovače v rámci města
active -	atribut, který informuje o tom, zda je směrovač v provozu nebo je jen zaveden do databáze a bude teprve připojen

EXTRA(id_user, new_rate, days)

id_user -	cizí klíč, identifikace uživatele, který žádá tuto službu
new_rate -	nová šířka pásma
days -	doba, po kterou bude služba aktivní, pak dojde k nastavení starých hodnot

Tabulky neobsahují kompletní seznam atributů, jsou uvedeny pouze ty, se kterými systém pracuje. Primární klíče jsou podtrženy.

3.3 Implementace

3.3.1 Hlavní server

Jak již bylo řečeno v sekci 3.2.1, hlavní server řídí provoz na síti tím, že generuje a spouští na směrovačích skripty, které obsahují příkazy pro vhodný traffic shaping a QoS.

Server je napsán v jazyce Java a plní následující logické funkce, jejichž příklady si ukážeme:

1. **Převádí objektové skripty na spustitelné skripty**

```
public MakeScript(RouterScript rs) throws IOException{
  if(rs == null) throw new IOException("Router is not active");
  this.myScript = rs;
  File file = new File("TrafficServer/scripts/"+myScript.getIP());
  FileWriter filewriter = new FileWriter(file);
this.writer = new BufferedWriter(filewriter);
makeRoot();
makeChild();
writer.close();
}
```

Třída *Compositor* předává informace o jednotlivých klientech a nastaveních právě třídě *MakeScript*. Ta z objektu vytvoří spustitelný skript pomocí příkazů utility tc. Nejprve jsou nastaveny kořenové třídy a jim pak přiřazeny třídy s potomky.

2. **Spouští nové skripty**

```
...
Runtime r = Runtime.getRuntime();
Process p1= r.exec("scp -i "+user_home+
     "/.ssh/"+TrafficServer.getIdentity()+
     " " + app_home +"/TrafficServer/scripts/"+name+
     "  root@" + name + ":TrafficServer/scripts/"+name);
     p1.destroy();
Process p2 = r.exec("ssh -i "+user_home+
     "/.ssh/"+TrafficServer.getIdentity()+ "
     root@"+name + "  ./TrafficServer/scripts/" + name);
     p2.destroy();
...
```

Skripty jsou spouštěny s využitím nativních programů operačního systému UNIX scp[4] a ssh[5]. Oba zmíněné programy podporují autentizační metodu RSA (*viz* Bezpečnostní mechanizmy serveru). Program vytvoří samostatné procesy, nejprve nakopíruje nový skript na směrovač a pak ho spustí, které jsou po provedení úkolu ukončeny.

3. **Generuje nové skripty**

```
...
ResultSet s = stat.executeQuery("SELECT <<ip>>,<<bandwidth>>  ...");
while(s.next()){
MyMessage m =
new MyMessage(<<ip>>, <<bandwidth>>);
```

[4] *secure copy* - kopíruje soubory mezi počítači za použití šifrované komunikace
[5] *secure shell* - klient pro bezpečné spouštění příkazů a programů na vzdálených počítačích

```
script.add(m);
}
...
FileOutputStream fos =
new FileOutputStream("TrafficServer/store/"+script.getIP());
ObjectOutputStream oos = new ObjectOutputStream(fos);
oos.writeObject(script);
...
MakeScript ms = new MakeScript(script);
RunMe rm = new RunMe(script.getIP());
...
```

Nejprve jsou z databáze získány informace o IP adresách a jejich šířkách pásma, které patří ke směrovači. Zprávy jsou přidány do objektového skriptu a ten je následně serializován kvůli dalšímu použití. Z objektu je vygenerován spustitelný skript a ten je spuštěn.

4. **Provádí korekci nastavení**

```
...
message = routerMessage.getMessages();
RouterScript script = (RouterScript) (ois.readObject());
Vector child = script.getChild();
for(int i=0;i<message.size();i++){
   MyMessage corMes = (MyMessage) message.elementAt(i);
   for(int j=0;j<child.size();j++){
       MyMessage serMes = (MyMessage) child.elementAt(j);
          if(corMes.getIP().compareTo(serMes.getIP())==0){
          int old_rate = serMes.getRate();
          int curr_rate = corMes.getRate();
          float reduct = (float)old_rate/(float)curr_rate;
          if(reduct<1){
            serMes.setRate(new Float(reduct*old_rate).intValue());
            script.remove(j);
            script.add(serMes);
            break;
          }
          }
     }
}
MakeScript ms = new MakeScript(script);
RunMe rm = new RunMe(script.getIP());
...
```

Po přijetí zprávy ze směrovače (*routerMessage*) je deserializován objektový skript, je vyhledána zpráva podle IP adresy, která překročila svůj limit, dále je proveden výpočet nového nastavení šířky pásma a je provedeno nahrazení, vygenerování nového skriptu a spuštění.

5. **Stahuje ze směrovačů informace**

```
private void getData(String ip, Runtime r) throws IOException{
Process p = r.exec("ssh -i " + System.getProperty("user.home")
+ "/.ssh/" + TrafficServer.getIdentity() + "  root@"+ip +
" zip -rm TrafficServer/store "+ip);
...
p.destroy();
p = r.exec("scp -i "+System.getProperty("user.home") + "/.ssh/"
+TrafficServer.getIdentity() + "  root@"+ip +":"+ip
+" TrafficServer/history/");
...
p.destroy();
}
```

Informace jsou zabaleny s celým adresářem a překopírovány na server do adresáře s historií.

Bezpečnostní mechanizmy serveru

Jelikož server běží na veřejné síti, bylo nutné provést některá bezpečnostní opatření.

- **RSA** - server pro spouštění a kopírování skriptů používá autentizaci pomocí RSA veřejných a soukromých klíčů. Jsou mu nastaveny cesty k *.ssh* adresáři, kde jsou tyto klíče uloženy. Na směrovačích je veřejný klíč serveru uložen v souboru *authorized_keys* a při přihlášení nevyžaduje heslo. Provede jen kontrolu veřejného klíče, zašifruje jím náhodné číslo a pošle jej serveru, ten číslo může dešifrovat pouze svým soukromým klíčem, tak směrovači ukáže, že zná soukromý klíč, aniž by jej prozradil a směrovač mu povolí přístup.
- **Kontrola IP adres** - sever kontroluje IP adresy, ze kterých požadavky přicházejí a neznámé adresy ignoruje. Tato metoda je dostačující, jelikož je podpořena bezpečnostními opatřeními sítě (k síti jsou připojeni jen registrovaní uživatelé a jejich IP adresy jsou známy, při změně IP adresy bude uživateli odepřen přístup). Také požadavky, které posílá administrátor z webového rozhraní pocházejí se serveru, jehož IP adresa je známá.
- **Typy zpráv** - sever reaguje jen na určitý počet zpráv, jakmile dostane neznámou zprávu, ukončí spojení, spojení je také ukončeno, není-li do určité doby po přijetí spojení detekována známá zpráva.

Informační soubory

Server udržuje základní informace o chování systému prostřednictvím logového a chybového souboru.

- *log* - v tomto souboru uchovává informace o startu serveru, přijatých spojeních či zadaných příkazech, údaje jsou opatřeny časovým razítkem

- *error* - v tomto souboru můžeme najít veškeré chybné operace, zachycené chyby během chodu programu či neoprávněné pokusy o přístup, údaje jsou opět opatřeny časovým razítkem

Důležitým souborem, ve kterém se nacházejí počáteční konfigurační nastavení serveru, je tzv. *ini* soubor, který se zadává jako parametr při spuštění serveru. Bližší parametry nastavení nalezneme v uživatelské dokumentaci *viz* příloha 5.

Adresářová struktura serveru

Nakonec si popíšeme strukturu adresářů, které jsou vytvořeny v aplikaci.

/	v kořenovém adresáři je uložen konfigurační soubor serveru, chybové a logovací soubory
/src	obsahuje všechny zdrojové soubory serveru
/TrafficServer	obsahuje stěžejní *.class soubory, které obsluhují příchozí spojení
/TrafficServer/db	soubory pro práci s databází
/TrafficServer/generator	soubory, které generují a opravují skripty
/TrafficServer/messages	soubory, které zapouzdří různé typy zpráv
/TrafficServer/scripts	do tohoto adresáře se ukládají vygenerované skripty
/TrafficServer/store	adresář slouží jako úložní prostor pro serializované objekty

Tabulka 3.1: Adresářová struktura hlavního severu.

3.3.2 Klient - směrovač

Klient systému je nasazen na směrovači. Spolupracuje s bezpečnostním systémem, který již spravuje povolení přístupu k síti. Bezpečnostní systém nejprve vyzve klienta k zadání uživatelského jména a hesla, po správném zadání povolí přístup dané IP adrese do sítě pomocí programu iptables. Dále měří a ukládá průtoky klientů způsobem, který jsme popsali v sekci 2.9.1. Klient je spuštěn s parametrem IP adresy směrovače. Vytvoří spojení s hlavním serverem, načítá postupně dvojice IP adresa - překročený limit a odesílá je ke zpracování a korekci.

```
...
s=socket( AF_INET,SOCK_STREAM,IPPROTO_TCP);
if (s==SOCKET_ERROR) {
printf("Can't create a socket, err: (%3d)\n",errno); exit(-1); }
saMy.sin_family=AF_INET;
saMy.sin_port=htons(MY_PORT);
addr=inet_addr(MY_ADDR);
if (addr==-1)
{ printf("Cannot obtain own host IP address\n"); close(s); exit(-1); }
  memcpy(&(saMy.sin_addr),&addr,sizeof(long));
if (bind(s,&saMy,sizeof(saMy))==SOCKET_ERROR)
```

```
{ printf("Bind failed (%3d)\n",errno); close(s); exit(-1); }
saHis.sin_family=AF_INET;
saHis.sin_port=htons(HIS_PORT);
addr=inet_addr(HIS_ADDR);
if (addr==-1)
{ printf("Cannot obtain remote host IP address\n"); close(s); exit(-1); }
memcpy(&(saHis.sin_addr),&addr,sizeof(long));
if (connect(s,&saHis,sizeof(saHis))==SOCKET_ERROR)
{ printf("Cannot connect to the remote host\n"); close(s); exit(-1); }
...
while(fgets(message,15,file)){
strcpy(buf,message);
strcat(buf,"\n");
write(s,buf,strlen(buf));
i++;
if(i%2){
strcpy(buf,"1");
strcat(buf,"\n");
write(s,buf,strlen(buf));
}
strcpy(buf,"0");
strcat(buf,"\n");
write(s,buf,strlen(buf));
}
close(s);
...
```

Ukázka klienta. Kód ukazuje navázání lokální IP adresy a portu (parametr MY_PORT, MY_ADDR) a pokus o připojení k hlavnímu serveru (parametr HIS_PORT, HIS_ADDR). Následuje čtení dat ze souboru a zápis do streamu. Data jsou párovány, jestliže následuje další pár, oznamujeme to parametrem „1“, pokud již klient nebude nic posílat, zapíše do streamu „0“.

3.3.3 Grafické uživatelské rozhraní

Pro ovládání a správu hlavního serveru bylo nutné vytvořit rozhraní, přes které jej budeme ovládat. Volba padla na dynamickou webou aplikaci napsanou technologií JSP a nasazenou na server JBoss verze 3.0.6 v kombinaci se servletovým kontejnerem Tomcat verze 4.1.x. Servlety jsou hlavním prostředkem rozšiřování a zkvalitňování webových aplikací pomocí platformy Java. Servlet je mapován k jednomu nebo více URL, je-li zachycen požadavek na URL, které servlet obsluhuje, volá se jeho metoda, která požadavek vyřídí. Že se servlety píší v Javě je nespornou výhodou i z toho důvodu, že mohou používat rozsáhlých API (programová rozhraní Javy) včetně Java DataBase Connectivity (JDBC) a Enterprise JavaBeans (EJB). JSP jsou přirozeným rozšířením technologie Java Servlet, obsahují kombinaci statických HTML tagů, tagů ve stylu XML a scriptletů. Webová aplikace dále využívá podporu rámců Jacarta Struts a architekturu Model View Controler 2.

Konfigurace a struktura Struts

Rámec Struts používá dva typy oddělených, avšak v jistém smyslu navzájem souvisejících konfiguračních souborů, které je třeba vhodně nakonfigurovat, aby aplikace pracovala správně. Oba typy konfiguračních souborů jsou postaveny na rozšířeném, flexibilním a názorném XML.

- *web.xml* - popisovač nasazení je podrobně popsán ve specifikaci Java servletů, obsahuje inicializační parametry, konfigurační spojení, deklarování servletů, mapování servletů, atd.
- *struts-config.xml* - umožňuje deklaratorně nastavit chování naší aplikace, tj. nastavení datového zdroje, výjimky, přesměrování toku nebo mapování akcí či zdroje zpráv.

V následujících příkladech si ukážeme části konfigurací xml souborů, příklady provází komentáře.

Příklad 1 *ukázka konfigurace web.xml*

```
<!-- Umístění lokalizovaných textů -->
    <init-param>
      <param-name>application</param-name>
      <param-value>netadmin.resources.application</param-value>
    </init-param>

<!-- Umístění konfiguračního souboru Struts -->
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>

<!-- Mapování standardního řídícího servletu -->
    <servlet-mapping>
      <servlet-name>action</servlet-name>
      <url-pattern>*.do</url-pattern>
    </servlet-mapping>
```

Ukázka obsahuje mapování souboru s vlastnostmi aplikace, nastavení konfiguračního souboru pro Jakarta Struts a nastavení kontejneru, kterému tím říkáme, že má obsluhovat všechny požadavky, které mají příponu *.do*.
Příklad 2 *ukázka konfigurace struts-config.xml*

```
<!-- ================ Data Source Configuration ============ -->
<data-sources>
   <data-source>
```

```
    <set-property property="autoCommit" value="true" />
    <set-property property="description"
     value="MySQL Data Source Configuration" />
    <set-property property="driverClass"
     value="org.gjt.mm.mysql.Driver" />
    <set-property property="maxCount" value="4" />
    <set-property property="minCount" value="2" />
    <set-property property="url value="jdbc:mysql://localhost/test
     ?useUnicode=true&amp;characterEncoding=iso-8859-2" />
    <set-property property="user" value="radim" />
    <set-property property="password" value="heslo" />
  </data-source>
</data-sources>
```

Konfigurace datového zdroje, nastavení ovladače databáze, mapování databáze, uživatelské jméno a heslo

```
<!-- ================= Global Forward Definitions =========== -->
<global-forwards>
        <forward name="main" path="/Main.do"/>
        <forward name="logoff" path="/Logoff.do"/>
        <forward name="logon" path="/Logon.do"/>
        <forward name="uzivatele" path="/Uzivatele.do"/>
        <forward name="smerovace" path="/Smerovace.do"/>
        <forward name="extra" path="/Extra.do"/>
        <forward name="dokumentace" path="/Dokumentace.do"/>
        <forward name="kontakt" path="/Kontakt.do"/>
        <forward name="edit" path="/Edit.do"/>
 </global-forwards>
```

Konfigurace přesměrování, příkaz *forward* přiřazuje logický název k relativní URI aplikace, která pak může provést přeposlání pomocí logického jména. Tento postup pomáhá oddělit model a řadič od pohledu.
Adresářová struktura webové aplikace je rozebrána v tabulce 3.2.

JSP stránky

Webová aplikace obsahuje následující hlavní JSP stránky:

- *logon.jsp* - stránka s přihlašovacím formulářem, na tuto stránku je uživatel přesměrován z *index.jsp* stránky, po vyplnění správných přihlašovacích údajů bude uživatel přesměrován na *main.jsp*, v případě, že byly vyplněny špatné údaje, aplikace vypíše chyby a uživatel zůstane v přihlašovací sekci

- *main.jsp* - úvodní strana webové aplikace, v této chvíli bude vytvořeno spojení s hlavním serverem a uživatel má možnost využít přesměrování na stránky, které zobrazí informace o klientech (*uzivatele.jsp*), které zobrazí informace o směrovačích a

/classes	obsahuje *.class soubory aplikace
/config	obsahuje *deployment descriptor*, který je pojmenován jako *netadmin.xml*
/lib	obsahuje knihovny *jar*, které aplikace ke svému běhu potřebuje (jde například o JDBC ovladače)
/src	zdrojové soubory aplikace rozdělené do příslušných balíků
/web/css	soubory, které definují styly
/web/image	v tomto adresáři jsou uloženy obrázky, které aplikace používá
/web/WEB-INF	obsahuje konfigurační soubor *struts-config.xml* a tag knihovny
/web/WEB-INF/jsp	všechny jsp stránky aplikace

Tabulka 3.2: Adresářová struktura webové aplikace

dovolí jejich nastavení měnit (*smerovace.jsp*) nebo které umožní měnit zvláštní nastavení klientů (*extra.jsp*), dále obsahuje proklik pro odhlášení a stránku s základními příkazy serveru (*server.jsp*)

- *uzivatel.jsp* - zobrazí informace o jméně, příjmení, adrese, IP adrese, IP směrovače, pod který spadá, přidělené šířce pásma, atd., tato stránka má pouze informativní charakter a nelze na ní nic měnit
- *smerovace.jsp* - zobrazí informace o směrovačích, jejich IP adresu, maximální šířky pásma síťových kartách, označení síťových karet a dialogy pro znovuspuštění nastavení
- *extra.jsp* - zobrazí zvláštní nastavení všem, kteří požadovali tuto službu, obsahuje dialogy pro editaci položek, přidávání a odebírání těchto klientů z tabulky
- *server.jsp* - stránka obsahuje základní příkazy pro hlavní server (nové logovací a chybové soubory, vypnutí serveru)

JSP stránky využívají akcí, se kterými jsou spojeny v konfiguračním souboru aplikace (struts-config.xml). Např. přihlašovací stránka volá akci, která vyhledá uživatele v databázi a buď povolí nebo zakáže přístup pro zadané přihlašovací jméno a heslo, které bylo zadáno do přihlašovacího formuláře(*LogonForm*).

```
public ActionForward perform(ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response)
        throws IOException, ServletException {
        String username = ((LogonForm) form).getUsername();
        String password = ((LogonForm) form).getPassword();
        boolean validated = false;
        UzivatelItem uzivatelItem;
```

```
        try {
         DataSource ds = getDataSource(request);
         UzivatelManager uzivatelManager = new UzivatelManager(ds);
         uzivatelItem= uzivatelManager.findUzivatel(username);
         if(uzivatelItem == null)
         {
         validated = false;
         } else{
         if(password.equals(uzivatelItem.getPass()))
         validated = true;
         else
         validated = false;
         }
        }
        catch (Exception ude) {
            ActionErrors errors = new ActionErrors();
            errors.add(ActionErrors.GLOBAL_ERROR,
            new ActionError("error.logon.connect"));
            saveErrors(request,errors);
            return (new ActionForward(mapping.getInput()));
}
```

Bezpečnost aplikace

Vzhledem k tomu, že celý systém je ovládán právě z webového rozhraní, byla jednou z nejdůležitějších otázek při implementaci otázka bezpečnosti. Hlavními prvky bezpečnosti jsou **autentizace** a **autorizace**. Autentizace je akt ověření, že uživatel, který přistupuje do systému, je znám. Autorizace je další stupeň ochrany, který určuje oprávnění přístupu ke zdrojům aplikace. V našem případě dostává přihlášený uživatel (administrátor) plné práva.

Autentizace

Webové aplikace založené na J2EE poskytují tři typy autentizace.

- *basic* - základní autentifikace, zobrazí se okno pro zadání uživatelského jména a hesla
- *form-based* - přihlášení probíhá prostřednictvím formuláře, tato metoda se používá nejčastěji, dovoluje měnit vzhled přihlašovacího okna
- *mutual* - kombinace obou dvou metod

Tyto typy autentifikace nejsou bezpečné, pokud komunikace neprobíhá prostřednictvím šifrovacího protokolu. V naší aplikaci použijeme bezpečnou SSL komunikaci mezi klientem a webovým serverem a jako autentifikační metodu vybereme autentifikaci pomocí formulářů.Vše ale předchází správné nakonfigurování serveru JBoss a kontejneru Tomcat. Aby kontejner Tomcat používal protokol SSL, musíme pro něj vygenerovat certifikát a RSA soukromý a veřejný klíč. Použijeme k tomu prostředky Javy.

```
keytool -genkey -alias tomcat -keyalg RSA
```

Jako heslo zvolíme „changeit“, kontejner nyní bude schopen přijímat SSL spojení. Dále musíme změnit konfigurační nastavení v souboru kontejneru*server.xml* i serveru JBoss (jedná se o soubor *tomcat41-service.xml*).

```
<Connector className="org.apache.coyote.tomcat4.CoyoteConnector"
        port="8443" minProcessors="5" maxProcessors="75"
        enableLookups="true"
        acceptCount="100" debug="0" scheme="https" secure="true"
        useURIValidationHack="false" disableUploadTimeout="true">
<Factory className="org.apache.coyote.tomcat4.CoyoteServerSocketFactory"
        clientAuth="false" protocol="TLS" />
</Connector>
```

Značka *Connector* definuje typ připojení k serveru, atribut *className* určí třídu, která vytváří připojení k serveru. Dále definujeme port, na kterém bude server očekávat tento druh připojení a nakonec řekneme, které rozhraní bude toto připojení zpracovávat a jaký protokol bude použit. Jakmile aplikaci nasadíme na server, dostaneme se k ní, zadáme-li do prohlížeče:

```
https://<IP adresa počítače>:8443/netadmin/
```

Autentifikace pomocí formuláře probíhá následovně:

- kontejner vrátí přihlašovací formulář a uloží si příslušné URL
- klient vyplní formulář a odešle ho na server
- kontejner autentifikuje uživatele s využitím položek formuláře
- pokud je autentifikace špatná, vrátí mu přihlašovací formulář spolu s popisem chyby, jinak ho přesměruje na hlavní stránku

3.3.4 Balení aplikace

Je-li webová aplikace hotová, musíme ji nasadit na server. Existuje několik způsobů, jak to udělat. Nejčastěji je aplikace zabalena do souboru s příponou *.war, .jar* nebo *.ear*. První způsob, distribuce balíku s příponou war, umožní nasadit aplikaci na server bez jeho restartu, bohužel jakákoliv změna vyžaduje nové balení aplikace a nové nasazení. Druhý přístup je vhodný v testovací fázi, jelikož je aplikace rozbalena do určitého adresáře a provedené změny se aktualizují pouhým kopírováním změněných souborů, tedy bez balení celé aplikace.

Naše aplikace bude balena a nasazena prostřednictvím programu *Ant 1.5*. Ant je sestavovací nástroj nezávislý na platformě, který lze nastavit tak, aby kompiloval soubory zdrojového kódu Javy. Umí vytvořit nasazovací soubory JAR a WAR. Obsahuje mnoho dalších funkcí a lze ho rozšířit tak, aby plnil úkoly, které si stanovíme sami.

Konfigurační soubor pro balení naší aplikace se jmenuje *build.xml* a dodatečné nastavení, jako mapování serveru a servletů, jsou uloženy v souboru *build.properties*. Tyto soubory by měly být uloženy v kořenovém adresáři naší aplikace, z kterého také Ant spouštíme.

3.3.5 Optinalizace výkonu a správa paměti

Aplikace byla v konečné fázi testována a byla navržena vhodná optimalizace paměti a kódu v místech, kterým se říká kritická místa (*Hot Spots*). Pro odhalování problematických míst byl použit profilovací program HPROF. Výstup obsahuje níže uvedené položky:

- informace o vláknech
- údaje z trasování
- výpisy dynamické paměti (haldy)
- vzorkování CPU a čas CPU

Příkazem

```
java -Xrunhprof:file=mytest.txt,thread=y <název programu>
```

spustíme monitoring s výsledným výpisem procesů, trasování, vláken a dynamické paměti. Níže vidíme neoptimalizovaný výstup.

```
SITES BEGIN (ordered by live bytes) Mon May 10 20:51:01 2004
          percent          live       alloc'ed  stack class
 rank   self  accum    bytes objs   bytes objs trace name
    1 10.35% 10.35%   235888   44  235888   44     0 [B
    2  4.98% 15.33%   113632 1050  113632 1050     0 [C
    3  3.56% 18.89%    81056    5  327048   36   290 [B
    4  3.35% 22.24%    76472  167   76472  167     0 [I
    5  3.00% 25.24%    68488  540   70128  571     1 [C
    6  2.88% 28.12%    65552    1   65552    1  1604 [B
    7  2.88% 30.99%    65552    1   65552    1  1612 [B
    8  2.86% 33.85%    65128    6   65128    6  1727 [B
    9  2.25% 36.10%    51360  240   51424  242     1 [B
   10  1.47% 37.58%    33616  818   33616  818   827 [C
   11  1.46% 39.04%    33200   12   33200   12    77 [B
   12  1.44% 40.47%    32800    2   32800    2   426 [C
   13  1.28% 41.75%    29136  488   29136  488   153 [C
   14  1.15% 42.90%    26184 1091   26184 1091     0 java.lang.String
   ...
```

Sloupec označený „percent“ udává hodnocení položky z hlediska spotřebované paměti, jeho podsloupce „self“ - množství paměti obsazené danou položkou a „accum“ - úhrnná hodnota všech doposud uvedených položek. Další dva sloupce udávají původní alokovaný počet objektů a místo v paměti, které zabírají (sloupec „alloc'ed“) a pak skutečně použité množství (sloupec „live“). Z výpisu vidíme, že téměř 230KB zabírá položka *trace 0* (systémová počáteční alokace při spuštění programu) a dále 110KB zase ona. Je to proces, který již nebude použit, ale pouze zabírá paměť. První položka programu je *trace 290* a zabírá asi 80KB.

```
TRACE 290: (thread=1)
sun.misc.Resource.getBytes(Resource.java:62)
java.net.URLClassLoader.defineClass(URLClassLoader.java:247)
java.net.URLClassLoader.access$100(URLClassLoader.java:54)
java.net.URLClassLoader$1.run(URLClassLoader.java:193)
```

Z důvodu nešetrného hospodaření s pamětí jsme do kódu na vhodná místa[6] umístili systémové volání garbage collectoru. Výsledek je zřejmý na první pohled. Došlo k mnohem lepšímu hospodaření s pamětí.

```
rank   self  accum    bytes objs   bytes objs trace name
    1  5.88%  5.88%    68488  540   70128  571     1 [C
    2  5.63% 11.51%    65552    1   65552    1  1605 [B
    3  5.63% 17.14%    65552    1   65552    1  1612 [B
    4  4.58% 21.72%    53328   43   53328   43     0 [I
    5  4.41% 26.13%    51344  239   51424  242     1 [B
    6  2.89% 29.01%    33616  818   33616  818   827 [C
    7  2.85% 31.87%    33200   12   33200   12    77 [B
    8  2.82% 34.68%    32800    2   32800    2   426 [C
    9  2.50% 37.18%    29136  488   29136  488   153 [C
   10  2.13% 39.31%    24752  312   24752  312   145 [C
   11  2.12% 41.43%    24744  323   24744  323     1 [S
   12  1.69% 43.12%    19632  818   19632  818   827 java.lang.String
   13  1.60% 44.72%    18632  348   18768  351     1 java.lang.Object
   14  1.53% 46.25%    17808  729   17808  729   747 [C
   15  1.50% 47.75%    17496  729   17496  729   747 java.lang.String
   16  1.48% 49.23%    17224    9   17224    9     0 [C
   17  1.43% 50.66%    16640  520   16640  520   834 java.lang.String
```

Nakonec se podíváme na vzorkování procesorového času. V případě, že narazíme na proces, který zabírá nejvíce procesorového času (jedná se nejčastěji o poměr 80/20 nebo dokonce 90/10), našli jsme místo, které je slabým článkem systému a mělo by být optimalizováno, pokud to implementace dovolí.
Příkazem

```
java -Xrunhprof:cpu=samples,file=mytest.txt,thread=y <název programu>
```

bude výstup produkovat také vzorkování CPU.

```
CPU SAMPLES BEGIN (total = 518) Mon May 10 21:39:32 2004
rank   self  accum   count trace method
   1 83.78% 83.78%   434    43 java.net.PlainSocketImpl.socketAccept
   2  7.72% 91.51%    40    51 java.net.PlainSocketImpl.socketAvailable
   3  0.97% 92.47%     5    48 TrafficServer.Worker.handleAdmin
   4  0.77% 93.24%     4    49 java.io.InputStreamReader.ready
```

[6] jedná se většinou o koncové části větších celků programu, např. před ukončením nebo uspáním nějakého vlákna, které provedlo větší paměťovou operaci

```
   5  0.58% 93.82%     3    50 java.net.PlainSocketImpl.available
   6  0.58% 94.40%     3    64 java.lang.UNIXProcess.waitForProcessExit
   7  0.58% 94.98%     3    63 java.io.FileInputStream.readBytes
   8  0.39% 95.37%     2    53 sun.nio.cs.StreamDecoder$CharsetSD.implReady
   9  0.39% 95.75%     2    66 java.net.PlainSocketImpl.available
  10  0.39% 96.14%     2    55 java.net.SocketInputStream.available
  11  0.39% 96.53%     2    54 java.net.PlainSocketImpl.available
  12  0.19% 96.72%     1    32 com.mysql.jdbc.StringUtils.toAsciiString
  13  0.19% 96.91%     1    52 java.io.BufferedReader.ready
  14  0.19% 97.10%     1    68 java.lang.UNIXProcess.waitForProcessExit
  15  0.19% 97.30%     1     5 java.lang.StringBuffer.append
  16  0.19% 97.49%     1    44 java.lang.Throwable.fillInStackTrace
  17  0.19% 97.68%     1    35 java.lang.Thread.setPriority0
  18  0.19% 97.88%     1    34 java.lang.Runtime.gc
  19  0.19% 98.07%     1     1 java.util.jar.JarFile.getEntry
  20  0.19% 98.26%     1    65 java.lang.Thread.start
  21  0.19% 98.46%     1    33 TrafficServer.TrafficServer.main
  22  0.19% 98.65%     1    67 java.io.FileInputStream.readBytes
  23  0.19% 98.84%     1    62 java.lang.UNIXProcess.forkAndExec
  24  0.19% 99.03%     1    27 sun.nio.cs.UTF_8$Decoder.decodeArrayLoop
  25  0.19% 99.23%     1    61 java.lang.System.arraycopy
  26  0.19% 99.42%     1    21 java.lang.ClassLoader.defineClass0
  27  0.19% 99.61%     1    38 java.lang.Thread.start
  28  0.19% 99.81%     1    39 java.lang.Runtime.gc
  29  0.19% 100.00%    1    36 java.lang.Runtime.gc
CPU SAMPLES END
```

V našem případě je toto slabé místo v oblasti příjmu soketového spojení, bohužel toto nelze nijak optimalizovat. Jak ale jinak vidíme, ostatní procesy si velmi rovnoměrně rozebraly procesorový čas. Jednotlivé sloupce znamenají:*self* - zabraný procesorový čas, *accum* - součet procesorového času všech dosavadních procesů, *count* - počet označení, které byly provedeny během celé doby značkování procesů, *trace* - číslo procesu a název.

4. Nasazení v praxi

Tato kapitola nás provede konkrétními stránkami webového rozhraní a ukáže nám, jak celý systém funguje v praxi. Pro tuto potřebu byly vytvořeny ukázkové tabulky v databázi MySQL, které nalezneme v příloze 5. Podrobnější informace o nastavení hlavního serveru nebo serveru JBoss nalezneme v uživatelské příručce tamtéž.

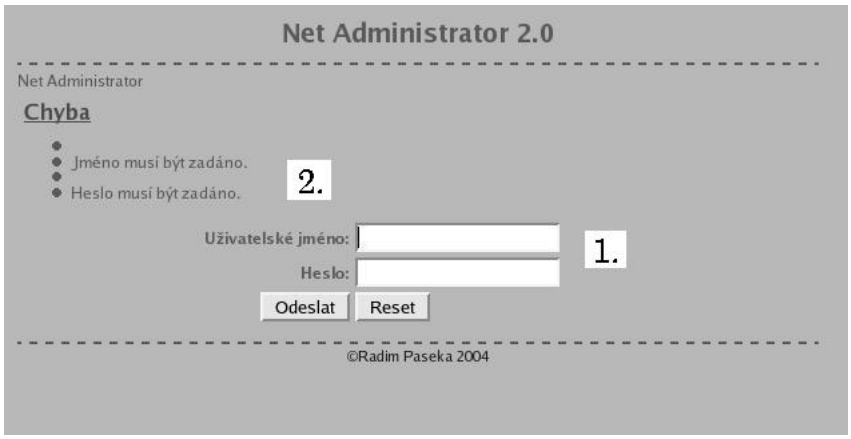
4.1 Spuštění serveru

Kontaktním článkem systému je hlavní server, musí být spuštěn jako první. K jeho konfiguraci použijeme *ini* soubor, který bude zapsán jako parametr spuštění.

```
java TrafficServer/TrafficServer server.ini
```

4.2 Přihlášení do systému

Zadáme-li do prohlížeče URI adresu počítače, na kterém nám běží webový server s naší aplikací, zobrazí se nám přihlašovací stránka (obrázek 4.1). V oblasti označené číslem 1.



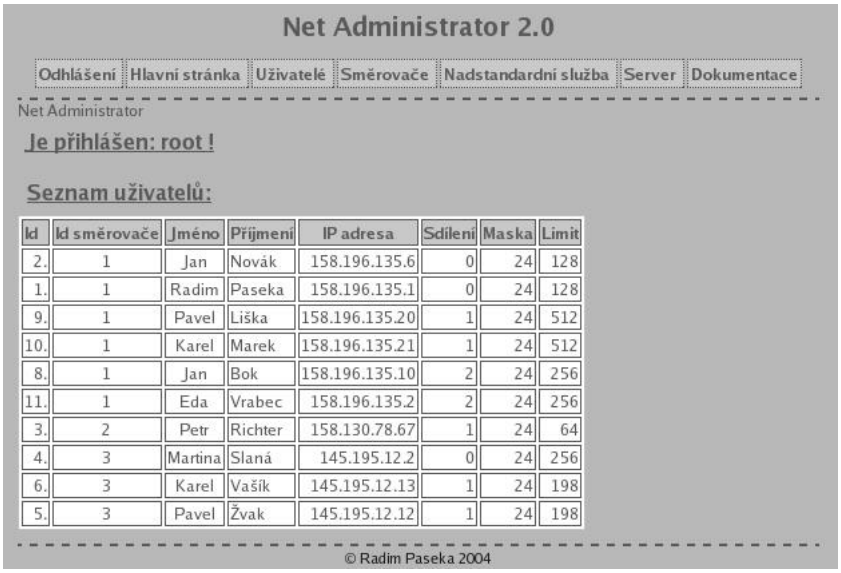
Obrázek 4.1: Přihlášení do systému.

zadáme přihlašovací jméno a heslo, pokud bylo zadáno špatné jméno nebo heslo nebo se vyskytla jiná chyba, nalezneme hlášení o chybě v sektoru 2. Jinak budeme přesměrováni na hlavní stranu. Odtud můžeme přistupovat do ostatních sekcí.

4.3 Uživatelé

Proklik na tuto stránku nám zobrazí informace o jednotlivých klientech systému. Informace můžeme ředit podle následujících kritérií. Identifikační číslo uživatele, identifikační číslo

směrovače, příjmení nebo IP adresy klienta. Informace slouží pouze ke čtení. Obrazovku vidíme na obrázku 4.2.



Id	Id směrovače	Jméno	Příjmení	IP adresa	Sdílení	Maska	Limit
2.	1	Jan	Novák	158.196.135.6	0	24	128
1.	1	Radim	Paseka	158.196.135.1	0	24	128
9.	1	Pavel	Liška	158.196.135.20	1	24	512
10.	1	Karel	Marek	158.196.135.21	1	24	512
8.	1	Jan	Bok	158.196.135.10	2	24	256
11.	1	Eda	Vrabec	158.196.135.2	2	24	256
3.	2	Petr	Richter	158.130.78.67	1	24	64
4.	3	Martina	Slaná	145.195.12.2	0	24	256
6.	3	Karel	Vašík	145.195.12.13	1	24	198
5.	3	Pavel	Žvak	145.195.12.12	1	24	198

Obrázek 4.2: Klienti systému.

4.4 Směrovače

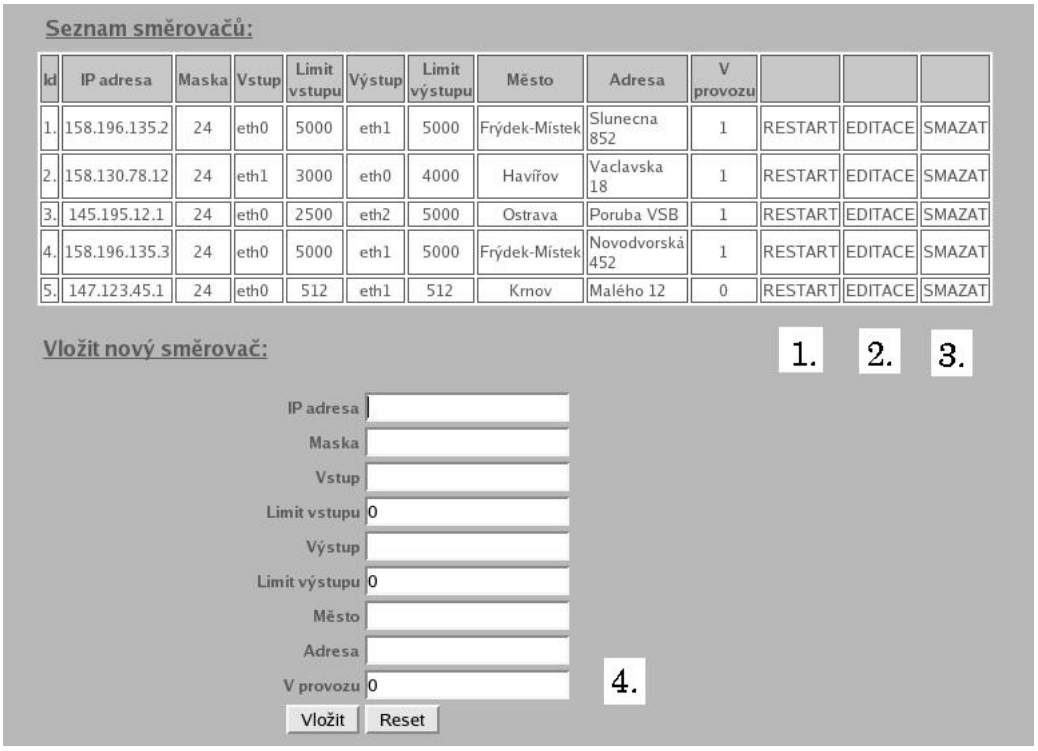
Stránka se směrovači dovoluje správci jednoduchou orientaci v jejich nastavení. Opět je implementováno řazení podle identifikačního čísla, IP adresy, města nebo konkrétní adresy ve městě. Jsou přidány tlačítka pro odstranění směrovače (číslo 3.), dále formulář pro přidání nového směrovače do tabulky (číslo 4.) a nakonec tlačítka pro editaci (číslo 2.) a restart směrovače (číslo 1.), vše vidíme na obrázku 4.3.

V horní části je oblast, která slouží pro chybové hlášení. Na obrázku je prázdná, bez chyb. Podívejme se nyní blíže na možnost editace a restartu směrovače.

4.4.1 Editace

Volíme-li volbu editace, jsme přesměrováni na editační stránku. Do formuláře jsou načteny původní hodnoty směrovače a položky, které smíme upravit jsou umístěny v editačních polích (číslo 2.). V horní oblasti (číslo 1.) je opět chybový výstup formuláře, jelikož se pokaždé spustí kontrola zadaných hodnot. Do formuláře se načtou zase původní hodnoty. Situace je na obrázku 4.4.

Správce může změnou nastavení směrovače výrazně ovlivnit jeho koncové nastavení. Změna, která byla učiněna v polích s limitem či rozhraním, vyžaduje jeho restart.



Obrázek 4.3: Správa směrovačů.

4.4.2 Restart

Restart směrovače způsobí znovunačtení všech hodnot z databáze a vygenerování a spuštění nového skriptu. Tento příkaz je zaslán s parametrem IP adresy hlavnímu serveru, který si příslušné informace načte z databáze, vygeneruje nový objektový skript. Ten pak převede do spustitelné podoby s příkazy utility tc. Máme následující seznam klientů (tabulka 4.1) pro směrovač číslo 3. s IP adresou 145.195.12.1. Další informace jsou na obrázku 4.3. Sdílení určuje příslušnost k třídě běžných uživatelů (0) nebo firmám (1 a výše). Jeli-

IP adresa	**Limit**	**Sdílení**
145.195.12.2	256kbps	0
145.195.12.12	198kbps	1
145.195.12.13	198kbps	1

Tabulka 4.1: Klienti připojeni ke směrovači

kož najdeme IP adresu 145.195.12.2 mezi adresami nadstandardních služeb s nastavením 512kbps, projeví se to i ve spustitelném skriptu, který server vygeneroval. Nastavení je ukázáno pro rozhraní eth0.

```
tc qdisc del dev eth0 root
tc qdisc add dev eth0 root handle 1: htb
tc class add dev eth0 parent 1: classid 1:1 htb rate 2500kbps \
```

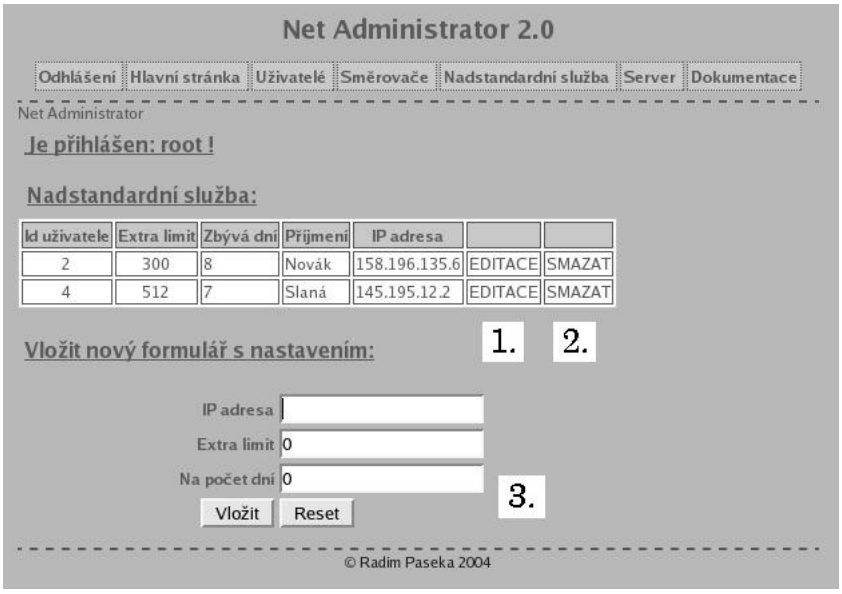
Obrázek 4.4: Editace směrovače.

```
ceil 2500kbps
tc class add dev eth0 parent 1:1 classid 1:11 htb rate 198kbps \
ceil 2500kbps prio 1
tc class add dev eth0 parent 1:11 classid 1:110 htb rate 198kbps \
ceil 2500kbps
tc qdisc add dev eth0 parent 1:110 handle 110: htb
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip \
src 145.195.12.12 flowid 1:110
tc class add dev eth0 parent 1:11 classid 1:111 htb rate 198kbps \
ceil 2500kbps
tc qdisc add dev eth0 parent 1:111 handle 111: htb
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip \
src 145.195.12.13 flowid 1:111
tc class add dev eth0 parent 1:1 classid 1:10 htb rate 2302kbps \
ceil 2500kbps prio 2
tc class add dev eth0 parent 1:10 classid 1:100 htb rate 512kbps \
ceil 2500kbps
tc qdisc add dev eth0 parent 1:100 handle 100: htb
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 match ip \
src 145.195.12.2 flowid 1:100
```

Nejprve musíme odstranit původní nastavení (první řádek) a dále je vytvářena hierarchie pro jednu firmu s dvěma IP adresami a jednoho běžného uživatele. Firma má garantovaný tok 198kbps pro všechny své počítače, jejich priorita je vyšší než pro běžného uživatele.

4.5 Nadstandardní služby

Jak jsme viděli v předchozím příkladě, do výsledné konfigurace byl započítán limit z této tabulky. Udává přechodné nastavení pro klienta, které je platné na definovanou dobu. Formulář vidíme na obrázku 4.5. Správci je umožněno třídit informace podle identifikace klienta, jeho IP adresy nebo příjmení. Údaje můžeme mazat (číslo 2.) editovat (číslo 1.) nebo přidávat (číslo 3.).



Obrázek 4.5: Zvláštní služby.

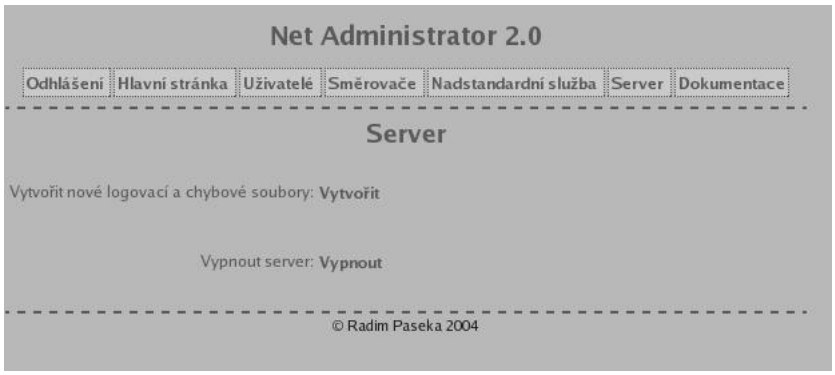
4.5.1 Editace

Prostředí editace je podobné jako u směrovačů (obrázek 4.6), editovatelné položky jsou v textových boxech a po odeslání formuláře k zápisu je proveden kontrola. Chyby se zobrazují v horní části (na obrázku nejsou).

4.6 Server

Na této stránce najdeme příkazy pro vytvoření nových logovacích a chybových souborů. Dále je zde tlačítko pro vypnutí hlavního serveru. Stránka je zobrazena na obrázku 4.7. Všechny ovládací tlačítka jsou během pobytu v systému dostupná v horní části obrazovky. Dojde-li k neaktivní činnosti (správce nepoužívá systém) po dobu 15 minut, bude automa-

Obrázek 4.6: Editace zvláštních služeb.



Obrázek 4.7: Ovládání serveru.

ticky odpojen. Pokud je činnost v systému ukončena, odhlásíme se tlačítkem „Odhlášení“. Jednoduchý popis ovládání systému nalezneme v sekci „Dokumentace“.

5. Závěr

Hlavním cílem diplomové práce bylo popsat problematiku omezování provozu a kvalitu služby v závislosti na přenosovém médiu a operačním systému.

Důraz byl kladen na analýzu internetového provozu a na nalezení vhodné disciplíny, která by umožnila provoz klasifikovat a omezovat. Nejlepších parametrů dosahuje disciplína HTB, která svými možnostmi, univerzálností a kvalitou převyšuje ostatní.

Součástí práce je administrační systém pro vzdálenou správu směrovačů. Vývoj aplikace prochází všemi fázemi softwarového procesu. Kopíruje vodopádový model, který lze v různých modifikacích nalézt ve většině současných přístupů. Model vychází z rozdělení životního cyklu vývoje na čtyři části: zadání a specifikaci požadavků, návrh systému, implementace a testování.

Celá aplikace je postavena na různorodých technologiích, které dohromady tvoří komplexní systém. Jedná se o síťové prvky v aplikaci s využitím již zmiňované disciplíny HTB, dále bezpečnostní opatření během komunikace (HTTPS, RSA), podpora databázového serveru MySQL a konečně webové služby založené na technologiích JSP a Struts.

Aplikace prošla laděním a testováním v lokálních podmínkách za umělého provozu. Nyní ji čeká zkušební test za reálných podmínek.

Bude přínosem pro poskytovatele internetových připojení, jejichž sítě obsahují unixové směrovače. Systém se určitě nevyhne různým úpravám a dalšímu rozšiřování. Jedním z možných rozšíření je prohloubení hierarchického stromu provozu z úrovně uživatele na úroveň aplikace či služby nebo zvětšení množiny příkazů, na které server reaguje.

Literatura

[PJS03] Cavaness Chuck: Programujeme Jakarta Struts, První vydání, Praha, Grada Publishing a.s., 2003. Autorizovaný překlad z anglického originálu „*Programming Jakarta Struts*“, ISBN 80-247-0667-9

[NPPV03] Gamma Erich, Helm Richard, Jonson Ralph, Vlissides John: Návrh programů pomocí vzorů - Stavební kameny objektově orientovaných programů, První vydání, Praha, Grada Publishing a.s.,2003. Autorizovaný překlad z anglického originálu„*Design Patterns - Elements of Reusable Object-Oriented Software*“, ISBN 80-247-0302-5

[JSP03] Bollinger Gary, Natarajan Bharathi: JSP - Java Server Pages, Podrobný průvodce začínajícího tvůrce webu, První vydání, Praha, Grada Publishing a.s., 2003.Autorizovaný překlad z anglického originálu „*JSP - A Beginner's Guide*“, ISBN 80-247-0340-8

[JAVA01] Schildt Herbert, Java2 - Příručka programátora, První vydání, Brno, SoftPress s.r.o., 2001.Autorizovaný překlad z anglického originálu „*Java2 - A Beginner's Guide*“, ISBN 80-86497-04-6

[JPP02] Spell Brett, Java Programujeme profesionálně, První vydání, Praha, Computer Press, 2002. Autorizovaný překlad z anglického originálu „*Proffesional Java Programming*“, ISBN 80-7226-667-5

[LPP01] Matthew Neil, Stones Richard a další: Linux Programujeme profesionálně, První vydání, Praha, Computer Press, 2001. Autorizovaný překlad z anglického originálu „*Professional Linux Programming*“, ISBN 80-7226-532-6

[BUIP98] Garfinkel Simson, Spafford Gene: Bezpečnost v UNIXu a Internetu v praxi, První vydání, Computer Press, Zlín, 1998. Autorizovaný překlad z anglického originálu „*Practical UNIX & Internet Security*“, ISBN 80-7226-082-0

[MKS98] Pužmanová Rita: Moderní komunikační sítě od A do Z, První vydání, Computer Press, Praha, 1998. ISBN 80-7226-098-7

[TKP03] Osterloh Heather: TCP/IP - Kompletní průvodce, První vydání, SoftPress s.r.o., Praha, 2003. Autorizovaný překlad z anglického originálu „*TCP/IP Primer Plus*“, ISBN 80-86497-34-8

[RFC792] Postel J.: Internet Control Message Protocol, RFC 792, září 1981

[RFC1738] Berners-Lee T.: Uniform Resource Locators, RFC 1738, prosinec 1994

[RFC1633] Braden R.,Clark D., Shenker S.: Integrated Services in the Internet Architecture: An Overview, RFC 1633, červenec 1994

[RFC2430] Blake S.,Black D.,Carlson M.: An Architecture for Differentiated Services, RFC 2475, prosinec 1998

[RFC2638] Nichols K., Jacobson V., Zhang L.: A Two-bit Differentiated Services Architecture for the Internet, RFC 2638, červenec 1999

[RFC2212] Shenker S.,Partridge C., Guerin R.: Specification of Guaranteed Quality of Service, RFC 2212, září 1997

[RFC2702] Awduche D., Agogbua J., O'Dell M., Malcolm J., McManus J.: Requirements for Traffic Engineering Over MPLS, RFC 2702, září 1999

[RFC3148] Mathis M., Allman M., A Framework for Definning Empirical Bulk Transfer Capacity Metrics, RFC 3148, červernec 2001

[RFC2330] Paxson V., Almes G., Mahdavi J., Mathis M.:Framework for IP Performance Metrics, RFC 2330, květen 1998

[RFC3393] Demichelis C., Chimento P., IP Packet Delay Variation Metric for IP Performance Metrics, RFC 3393, listopad 2002

[RFC2680] Almes G., Kalidindi S., Zekauskas M., A One-way Packet Loss Metrics for IPPM, RFC 2680, září 1999

[IPT03] Oskar Andreasson, Iptables Tutorial 1.1.19, 2001-2003, dostupné na Internetu - URI: `http://www.boingworld.com/workshops/linux/iptables-tutorial/`

[LRMM95] Floyd S., Jacobson V., Link-sharing Resource management Models for Packet Network, IEEE/ACM Transaction on Networking, Vol. 3 No.4, srpen 1995

[NCBQ96] Floyd S., Notes on Class-Based Queuing: Setting Parameters, LBNL, únor 1996

[HTB03] Devera Martin, HTB Linux queuing discipline manual, dostupné na Internetu - URI: `http://luxik.cdi.cz/~devik/qos/htb/htbman.htm`

[LARTC] Bert H., Linux Advanced Routing & Traffic Control HOWTO, dostupné na Internetu - URI: `http://lartc.org/HOWTO//cvs/2.4routing/2.4routing-howto.html`

[Rrd03] Oetiker Tobi, `http://www.rrdtool.org/`

[FCP03] Fedora Core Project, `http://fedora.redhat.com/`

Rejstřík

Přílohy

CD médim, které obsahuje:

- server JBoss 3.0.6 se servletovým kontejnerem Tomcat 4.1.x
- sestavovací nástroj Ant 1.5
- text diplomové práce v elektronické podobě
- zdrojové texty diplomové práce
- uživatelská a programátorská příručka v elektronické podobě
- webová aplikace se zdrojovými texty
- hlavní server se zdrojovými texty
- klient se zdrojovými texty
- soubor s testovacími daty ve formátu sql

Seznam obrázků

Seznam tabulek