

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

ZAJIŠTĚNÍ QOS V ROZSÁHLÝCH SÍTÍCH

QUALITY OF SERVICES IN WAN

BAKALÁŘSKÁ PRÁCE
BACHELOR THESIS

AUTOR PRÁCE **MICHAL KOBLÍŽEK**

AUTHOR

VEDOUCÍ PRÁCE **DOC. RNDR. ING. JIŘÍ ŠŤASTNÝ, CSC.**

SUPERVISOR

BRNO 2008

Licenční ujednání

Licenční ujednání

Poděkování

Na tomto místě bych rád poděkoval Doc. RNDr. Ing. Jiřímu Šťastnému, Csc., vedoucímu mé bakalářské práce, za vedení, konzultace a cenné rady a připomínky.

ABSTRAKT

Bakalářská práce se zabývá řízením provozu v počítačových sítích založených na protokolu TCP/IP. Cílem je řídit pakety tak, aby se síťový provoz provoz choval předvídatelně podle našich pravidel. K tomuto řízení použijeme pouze volně dostupný software – vybrané řešení je založeno na operačním systému FreeBSD, paketovém firewallu PF a frontovacím frameworkem ALTQ. Součástí práce jsou příklady nastavení tohoto frontování, včetně plných verzí skriptů, které jsou v elektronické příloze.

Tato závěrečná práce byla zpracována v rámci vědecko-výzkumného záměru MSM0021630529 Inteligentní systémy v automatizaci.

ABSTRACT

This bachelor thesis is inquire into direct traffic in computer network based on TCP/IP protocol. In the finish we can direct packets, that network traffic will behave by our rules. We will use only free software for this solution – this solution is based on FreeBSD system, PF packet firewall and queue framework ALTQ. Part of this thesis make examples queuing include fullversion scripts that are in elektronic appendix.

The thesis was developed in research project MSM 0021630529 Intelligent Systems in Automation.

Klíčová slova: HFSC, BSD, ALTQ, shaping, řízení provozu, QoS

Keywords: HFSC, BSD, ALTQ, shaping, traffic control, QoS

Obsah

1 Úvod....13
2 Přehled současného stavu....15
2.1 Co je QoS?....15
2.2 Rozdělení QoS podle způsobu implementace....16
2.3 Nekomerční softwarově zajištěné QoS....16
2.3.1 FreeBSD....17
2.3.2 Paketový firewall PF....17
2.3.3 ALTQ....17
3 Rozdělení typů softwarového řízení provozu....20
3.1 Fronty....20
3.2 Shedulery....20
3.2.8.1 HFSC - Hierarchie návrhu třídy....23
3.2.8.2 HFSC - Vnořené třídy....23
3.2.6.3 HFSC – priorizace provozu....24
4 Použité řešení....26
4.1 Shaping pro hraniční router....26
4.2 Shaping pro tranzitní router....27
5 Aplikace vytvořeného řešení....30
5.1 Rozřazení skupin....30
5.2 Příprava FreeBSD - instalace systému a příprava jádra....31
5.3 Konfigurace systému....32
5.4 Nastavení queuingu pomocí ALTQ....34
5.5 Filtrovací pravidla....38
5.2 Nastavení shapingu pro tranzitní router....42
6 Závěr....46
7 Seznam použité literatury....48
7.1 Monografie....48
7.2 WWW stránky....48
8 Přílohy....50
8.1 Seznam příloh....50
8.2 Výpis skriptu pro shaping na hraničním routeru....50
8.3 Plný výpis skriptu pro tranzitní shaping....50

1 ÚVOD

Snahy o zajištění Quality of Service (dále jen QoS) v počítačových sítích jsou v podstatě od samotného počátku rozvoje počítačových sítí. V úplných počátcích nebylo nutné zajišťovat žádné QoS, protože plně dostačovalo alespoň fungující propojení několika počítačů. Postupem času, když se propojování počítačů začalo rozšiřovat a sítě se zvětšovali, bylo nutné vzrůstající počet kolizí v síťovém provozu a tím pádem také zmenšující se propustnost sítě.

Tyto požadavky se ovšem ve vetší míře začaly objevovat až v době rozvoje Internetu a protokolu TCP/IP, neboť v lokálních počítačových sítích (LAN) byla zajištěna dostatečná (až naddimenzovaná) propustnost dat, která bránila nastání situací vyžadující externí řešení pro zachování propustnosti dat. V případě kdy daná propustnost linky již nebyla dostatečná, začalo docházet k přetečení síťového provozu a celkovému snížení propustnosti linky. Proto se na aktivních prvcích na klíčových místech sítě využívající protokol TCP/IP začal různými metodami řídit řídit provoz. Tyto prvky v drtivé většině případů zahrnují routery, aktivní switche s pokročilým managementem a hraniční routery/firewally. Tyto klíčové prvky mohly začít řídit či omezovat provoz díky svému umístění, kdy v podstatě veškerý datový tok prochází právě přes tyto prvky.

První snahy o zajištění QoS byly založeny na jednoduché podobě bufferování síťového provozu. Jednalo se nejčastěji o metodu FIFO (First In First Out), která zajistila postupné doručení paketů podle principu první přišel první odešel. Tato metoda přežila i do dnešní doby, ovšem pouze v podobě kdy se jedná o jednu z částí komplexního řízení toku. V současnosti je vyvinuto mnoho variant řízení toku, které v této bakalářské práci popíši.

Požadavky na řízení toku se měnily v průběhu let od počátků síťového provozu. V síťovém pravěku bylo důležité, aby vůbec nějaký síťový provoz fungoval. Po prvních pomalých dial-upových připojeních (v začátcích s s rychlostí 9600b až závratných 14.4kb) se přenášely jen nezbytně nutná data a nikdo neměl tendence využívat jednu linku k několika různým činnostem zároveň. Postupem doby se zvyšovala kapacita linek, kterým při využití zamýšlených služeb přebývala kapacita. Tato nevyužitá kapacita samozřejmě začala být využívána dalšími službami, které se používali zároveň. Čím byly tyto služby datově náročnější začínalo se objevovat vzájemné ovlivňování těchto služeb v různé míře. Velcí internetoví provideři na tyto trendy samozřejmě reagovali a jako přidanou hodnotu ke svým připojením začali nějakým způsobem nasazovat řízení provozu tak, aby z pohledu uživatele nebyl vůbec zaznamenán, ale zároveň poskytl pro různé služby požadované vlastnosti linky. Od původního FIFO řízení toku se vyvinulo několik dalších algoritmů, jejichž kombinováním s priorizací paketů se daří dosáhnout stavu, kdy v dnešní době uživatel takto ošetřené linky využívá různých služeb zároveň a jejich kvalita se realtimově přizpůsobuje možnostem linky. U některých typů připojení, či menších providerů kde nejsou tyto služby poskytnuty, dochází často k rozčarování uživatelů, protože při ovlivňování služeb se degraduje použitelnost celé linky.

V této bakalářské práci se pokusím popsat řešení řízení síťového provozu, používaném ve firmě ITSolutions, která se zabývá poskytováním internetové konektivity pro koncové zákazníky. Toto řešení bylo vyvinuto právě podle připomínek jednotlivých uživatelů a nyní funguje k plné spokojenosti uživatelů. Bez tohoto řešení by bylo možné poskytovat služby v dostatečné kvalitě mnohem menšímu počtu uživatelů, než kterých má firma nyní.

2 PŘEHLED SOUČASNÉHO STAVU

2.1 Co je QoS?

Quality of Service je základní zajištění kvality služby pro uživatele či zákazníky poskytované pomocí počítačových sítí. V této bakalářské práci se budu zabývat pouze QoS založené na IP vrstvě. Hlavní složky QoS můžeme rozdělit do následujících skupin [14]:

- **Přenosové pásmo** – jedná se v podstatě o rychlost kterou je schopna linka přenášet data, případně rychlosti jednotlivých skupin priorizovaného provozu – různé druhy provozu vyžadují (nebo jim stačí) různá rychlost, kterou my můžeme ovlivnit.
- **Latence** – jedná se o čas, za který je možno odeslat packet a přijmout ho na cílovém uzlu. Čas kdy přijde odpověď od koncového uzlu je v podstatě 2x dlouhá trasa odpovědi a takto je známá díky protokolu ICMP–ECHO. Jedná se o velice důležité hledisko pro IP telefonii.
- **Jitter** – jedná se v podstatě o variabilitu latence. V ideálním případě by měla být tato variabilita nulová a požadavky na odezvu by měli mít pokaždé stejnou hodnotu. Jitter je ale rozdíl mezi reálnými hodnotami různých odezev. Jde opět o jeden z velice důležitých faktorů pro IP telefonii, ale i pro videokonference a další real-timeové aplikace.
- **Ztrátovost** – procento ztracených packetů při přenosu. Největší problémy dělají opět real-time aplikacím – naopak u stabilnějších datových přenosů (např. FTP) nehrají (kromě určitého snížení rychlosti kvůli vynucení opětovného odeslání packetu) téměř roli.

Pokud některý z těchto požadavků není splněn v námi požadovaných mezích, degraduje se tím kvalita linky a služeb které na ní lze provozovat. Pro různé typy provozu jsou samozřejmě potřebné různé parametry. Základní rozčlenění můžeme vidět v následující tabulce:

Typ služby	Požadavky na QoS			
	Šířka pásma	Latence	Jitter	Ztrátovost
Email	malá	velká	velký	velká
FTP	velká	velká	velký	velká
SSH (Telnet)	velmi malá	malá	malý	střední
Streaming	velká	malá	střední	malá
Videokonference	velká	velmi malá	velmi malý	malá
VoIP	malá	velmi malá	velmi malý	malá

Z tohoto orientačního rozčlenění vidíme, že datové služby typu FTP (a jím podobné) mají ze současného pohledu nízké nároky na kvalitu linky. Pro provoz těchto služeb nejsou vůbec podstatné faktory jako latence a jitter, ovlivnit jej může samozřejmě šířka pásma a také ztrátovost. Ovšem samotný provoz tohoto typu služeb bude fungovat i s malou šířkou pásma (tedy linka bude pomalá), ale dokáže se vypořádat i s poměrně velkou ztrátovostí.

Naproti tomu realtime služby typu VoIP (internetová telefonie), jsou velice náchylné na latenci, jitter i ztrátovost. Pokud libovolný z těchto faktorů není splněn, služba se stane nepoužitelnou [14].

Z tohoto důvodu se v současné době téměř na všech typech datových přenosů používá nějaká verze QoS. Nejčastěji se s ní setkávají uživatelé internetového připojení i když si jej v drtivé většině případů vůbec

neuvědomují. Podstatou této služby je také transparentnost, tudíž nezávislost na samotném nastavení uživatele. V současné době stoupají nároky na on-line interaktivní provoz jako je IP telefonie, online hry, streamované video či začínající širší použití videokonferencí. Všechny tyto služby jsou z pohledu uživatele poskytovány na jediné lince. V zájmu každého internetového providera je zajistit provoz těchto služeb uživateli tak, aby uživatel stahující si velký soubor z FTP, který během čekání na konec stahování ještě zavolá pomocí IP telefonu svému známému a u toho si ještě na webových stránkách prohlíží streamované video neměl šanci poznat degradaci těchto služeb. Tomuto lze zabránit dvěma způsoby – buď dostatečně dimenzovanou linkou, nebo nasazením QoS založeném na priorizaci provozu. Samozřejmě dostatečně dimenzovaná linka je dražší pro uživatele i providery, takže se velmi často využívá QoS doplněné ještě FUP (Fair User Policy). Pomocí správně fungujícího QoS nám umožní předchozí modelovou situaci elegantně vyřešit za pomoci priorizace provozu. Díky ní by služba VoIP měla nejvyšší prioritu, webové prohlížení, nebo streamování střední prioritu a FTP přenos nejnižší prioritu. Pakety VoIP by byly odbavovány přednostně a teprve po jejich odbavení by přišly na řadu pakety streamingu a následně jako poslední by byly vyřizovány požadavky FTP. Po tomto přeuspořádání provozu, by byla zajištěna dobrá funkčnost všech služeb zároveň tak, aby například nadměrný FTP provoz neovlivňoval negativně IP telefonii. Právě o takto fungující model síťového provozu se pokusím vytvořit v této bakalářské práci [14].

2.2 Rozdělení QoS podle způsobu implementace

V současné době je možno rozdělit způsoby QoS na hardwarové a softwarové. Hardwarové prostředky řízení toku a priorizace provozu bývají implementovány formou podpory služeb přímo chipsetem hardwaru.

Softwarové prostředky řízení provozu jsou v drtivé většině založeny na vlastnostech protokolu TCP/IP. Vzhledem k naprosté převaze protokolu TCP/IP v počítačových sítích je největší výběr způsobu řízení dostupná právě pro tento protokol.

2.2.1 Řízení hardwarové

Zajištění QoS pomocí hardwarových prostředků jsou v drtivé vetšině případů zajištěny proprietárním firmwarem, který spolupracuje a využívá funkce podporované hardwarem. Nejznámější zástupce je zřejmě proprietární technologie Motorola Canopy. V této bakalářské práci se hardwarovou podporou nebudeme zabývat, neboť softwarové řízení nám dává větší rozptyl nasazení.

2.2.2 Řízení softwarové

Softwarové prostředky řízení toku je možno rozdělit na 2 velké skupiny – komerční a nekomerční. V komerční sféře se používá ve vetšině případů v high-end sféře a nejznámější je podpora v systému IOS firmy Cisco v celém spektru jejich produktů. Jejich značnou výhodou je okamžitá funkčnost daná jednoduchým uživatelským softwarem (samozřejmě v profesionálním použití se značným množstvím voleb a parametrů).

V této bakalářské práci se však budu zabývat nekomerčními volně dostupnými prostředky. Jejich výhoda je viditelná na první pohled – tyto řešení bývají dostupné pro každého a ve velké většině případů je možné je zprovoznit na standartním počítačovém hardwaru. Jejich největší nevýhoda není tak zřejmá – pro zprovoznění kvalitního a funkčního řešení je zapotřebí problematice hlouběji porozumět a z toho vyplývající nutnost více času na nastudování a zprovoznění.

2.3 Nekomerční softwarově zajištěné QoS

Světu nekomerčního softwaru nyní vévodí platformy založené na systémech typu GNU/Linux a free Unixové odvozeniny – nejčastěji platforma BSD (FreeBSD, OpenBSD, NetBSD, ...). Díky dodržování norem POSIX je možné software vyvíjený na jednotlivých platformách jednoduše portovat i na ostatní – což se také velice často děje – a tak je možné v dnešní době na všech těchto platformách provozovat téměř totožná řešení. Při mém předchozím snažení, jsem používal jako základ svých routerů různé distribuce GNU/Linux. Vedli mne k tomu důvody uživatele a správce systému. Systém GNU/Linux má vetší uživatelskou základnu, která

poskytuje velice dobrou podporu na webových diskuzních fórech. Samotné zprovoznění jednoduchého routeru či firewallu je proto (obzvláště dnes) již jednoduchou záležitostí. Bohužel pro profesionální použití s velkou zátěží přestává tato základní konfigurace stačit. Při velkém síťovém vytížení pro které stačí i pouhých pár uživatelů, začne celková síťová propustnost klesat, zejména díky saturaci a souvisejícím zahlcením sítě. Linuxové nástroje na řízení QoS se opírají hlavně o dvojici IPTABLES+HTB (či CBQ). Firewall IPTABLES je díky defaultní přítomnosti součástí téměř každé distribuce Linuxu, HTB je rozšíření jádra pro řízení datového toku. Vzhledem ke stáří IPTABLES, nebylo v době jeho vzniku uvažováno o těchto doplňkových funkcích. Proto byly doplněny dodatečně, což se podepsalo na jejich určité kostrbatosti. I díky tomu, je pro řízení používáno 2 způsobů – jednak markování pomocí nástroje TC a markování přímo pomocí IPTABLES. Oba způsoby se používají hlavně z důvodu, že ani jeden ze způsobů není vhodný pro všechny typy použití (nejčastější problém tvoří NAT) [13].

Naproti tomu firewall PF, který byl vyvíjen pro použití v operačním systému OpenBSD jako náhrada za původní používaný firewall IPFW, byl vyvíjen již se záměrem využívat všech prostředků řízení provozu QoS a zároveň se snahou zjednodušit syntaxi zápisu těchto pravidel. Proto již na první pohled uvidíme, že zápis pravidel PF je jednodušší při zachování všech funkcí původního firewallu. Díky použití nové funkce tabulek IP adres, můžeme původní rozsáhlé firewally - ve kterých musí být použity pravidla i duplicitně kvůli řazení provozu do front (typicky IPTABLES) – přepsat na čtvrtinu původní velikosti, což má samozřejmě i kladný vliv na výkon. Navíc díky snadné portaci je PF firewall portován i na ostatní operační systémy z rodiny BSD. V mém případě jsem se rozhodl pro systém FreeBSD, který je z těchto BSD systémů nejdál v oblasti nových vlastností a podpory HW na platofrmě i386. I z tohoto důvodu jsem se rozhodl ve své bakalářské práci zabývat nasazením řešení pomocí FreeBSD, PF a ALTQ [7, 9, 13].

2.3.1 FreeBSD

Kořeny FreeBSD sahají až k verzím BSD od Computer Systems Research Group na University of California v Berkeley. Přes deset let práce bylo věnováno vylepšování, bylo přidáno v současných počítačích stále běžnější SMP, vícevláknování a zlepšena síťová výkonnost. Dále byly přidány nové administrátorské nástroje, souborové systémy a bezpečnostní vlastnosti. Výsledkem je, že s FreeBSD je možné se setkat všude po internetu, jako s operačním systémem páteřních směrovačů, kořenových doménových serverů, významných webových sítí a jako se základem pro široce používané operační systémy stolních počítačů. Mezi nejznámější a největší uživatele FreeBSD patří např. společnost Yahoo [6].

2.3.2 Paketový firewall PF

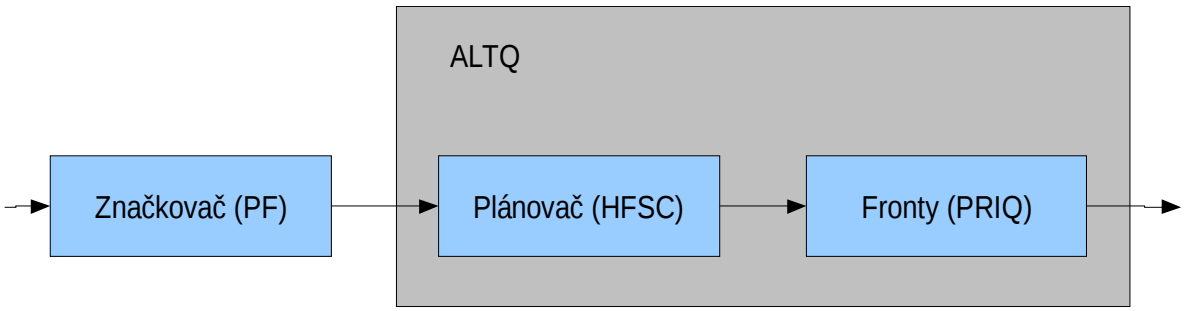
Paketový firewall PF (Packet Firewall), je moderní paketový firewall, který je od začátku vyvíjen pro systém OpenBSD, který je vyvíjen s největším důrazem na bezpečnost. PF poskytuje možnosti NATování, normalizace a úpravy TCP/IP provozu a pomocí nástroje ALTQ také řízení toku a priority. V systémech BSD má navíc ještě podporu redudantního provozu pomocí protokolu CARP a nástroje pfsync. Další zajímavou vlastností je podpora rozdělení zátěže pomocí několika způsobů od jednoduchého statického nastavení, po round-robin pravidla. Vzhledem k těmto vlastnostem je PF portován i na ostatní operační systémy. Samozřejmostí jsou ostatní systémy z rodiny BSD, ale také systémy GNU/Linux a dokonce probíhá i portace pro systém MS Windows [7, 9].

2.3.3 ALTQ

Nástroj ALTQ (ALTernate Queuing) je systémový framework, který řadí veškeré obsluhované pakety do front, se kterými dokáže dále pracovat podle námi zvolených algoritmů (HFSC, CBQ, PRIO, ...). Jedná se o mezivrstvu mezi systémem (kernelem) a interfacem, která je vyvolána pouze pokud tuto podporu zapneme. V případě že tuto podporu nezakompilujeme do kernelu, probíhá komunikace mezi jádrem a interfacem pomocí standardních nástrojů jádra. Ve FreeBSD je zahrnuta podpora pro ALTQ od verze 5.2. Pro starší verze je možné patchovat jádro [6, 7].

3 ROZDĚLENÍ TYPŮ SOFTWAROVÉHO ŘÍZENÍ PROVOZU

Pro rozsáhlé sítě s velkým počtem uživatelů není vhodné aby byl na všech routerech použit stejný typ řízení provozu. Jiné požadavky jsou kladeny na tranzitní routery a jiné zase na routery hraniční (brány, či gateway). Proto je v této aplikaci použito dvou druhů omezování provozu a to pro tranzitní router a gateway. V obou případech ale samozřejmě platí stejné schéma datového toku paketů. Paket který je řízen, vstupuje na rozhraní do rozhodovací logiky firewallu, který dále určí cestu paketu (povolí, zahodí, atd...) a *zároveň* ještě označí paket podle námi zadaných pravidel. Takto označený paket, putuje do plánovače (sheduler), což je algoritmus který dále řídí provoz. Složitost těchto pravidel je dána nejen nastavením, ale i použitím konkrétního plánovače. Plánovač již rozřadí pakety do front. Z těchto front paket opouští řízení provozu metodou FIFO, protože vlastní řazení do front řídí plánovač. Schéma průchodu paketem přes plánovač zajištěný PF+ALTQ je na obrázku č. 1



Obr. č. 1 – Schéma průchodu paketu při zařazení do scheduleru

3.1 Fronty

Do fronty řadíme pakety které čekají na zpracování podle pořadí určeného splánovačem. Pořadí, ve kterém plánovač vybírá pakety ke zpracování, může ovlivnit výkon sítě. Situace na které si můžeme nejlépe přiblížit rozdíl ve zpracování paketu je dobře vidět na protokolech SSH a FTP. Protokol SSH díky svému použití je citlivý na zpoždění, protokol FTP na zpoždění citlivý relativně není. Je to dáno tím, že SSH protokol používáme v drtivé většině pro interaktivní přístup – typicky jako vzdálená konzole – v tomto případě tedy při zmáčknutí klávesy na lokálním PC, chceme vidět okamžitou změnu i na vzdálené konzoli. Naproti tomu stahování dat pomocí FTP obslouží FTP klient, takže si nevšimneme ani dlouhé prodlevy. Tato prodleva může pouze vyvodit ve svém důsledku snížení rychlosti. Pokud nepoužijeme žádné další řízení front je možné, že při vetším zatížení FTP, budou tyto pakety obslouženy na úkor SSH, které se začne zpožďovat. V případě většího nárůstu provozu, se po překročení velikosti fronty začnou pakety zahazovat a SSH si díky svým vlastnostem “nevybojuje“ přenosové pásmo a spojení může úplně vypadávat. Pokud začneme tyto fronty řídit, můžeme citlivým nastavením zajistit, aby se tyto fronty odbavovali spravedlivěji a zvýšit tím celkovou použitelnost různých typů provozu.

Uvědomme si, že queueing má smysl pouze u paketů v *odchozím* směru. Jakmile paket dorazí na rozhraní v příchozím směru, je už příliš pozdě řadit ho do fronty - přenosové pásmo (bandwith) už spotřeboval na to, aby se dostal na rozhraní, které ho právě přijalo. Jediné řešení je zapnout queueing na předcházejícím routeru nebo, pokud počítač, který paket přijal funguje jako router, zapnout queueing na vnitřním rozhraní, kde pakety opouštějí router [7].

3.2 Shedulery

Scheduler je to, co rozhoduje, které fronty zpracovávat a v jakém pořadí. FreeBSD defaultně používá First In

First Out (FIFO) scheduler. FIFO fronta funguje jako fronta v supermarketu nebo bance - první položka ve frontě je zpracována jako první. Když dorazí nový paket, je zařazen na konec fronty. Pokud se fronta zaplní, jsou nově příchozí pakety zahazovány. Tomu se říká tail-drop.

FreeBSD podporuje další schedulery:

- Class Based Queueing
- Hierarchical Fair Service Curve
- Priority Queueing

3.2.1 FIFO (First In First Out)

FIFO je nejobyčejnější typ fronty, která se používá při řízení toku. Nemá možnost žádné regulace a je to defaultní chování na každém síťovém rozhraní. Pakety které přijdou na rozhraní jsou zařazeny do fronty a odbaveny ve stejném pořadí v jakém na rozhraní přišli. Jedná se v podstatě o datový buffer.

3.2.2 PRIQ

Priority Queueing (PRIQ) přiřazuje více front jednomu rozhraní s tím, že každá fronta má přiřazenu jedinečnou úroveň priority. Fronta s vyšší prioritou je *vždy* zpracována dřív než fronta s nižší prioritou.

Struktura front PRIQ je plochá - nemůžete vytvořit fronty uvnitř front. Je stanovena root fronta, která specifikuje celkovou dostupnou šířku pásma a pod ní jsou stanoveny podfronty. Uvažujme následující příklad:

Root Queue (10Mbps)
    Queue A (priority 1)
    Queue B (priority 2)
    Queue C (priority 3)

Root fronta je určena tak, že má k dipozici pásmo o šířce 10Mbps a jsou stanoveny tři podfronty. Fronta s nejvyšší prioritou (nejvyšší číslo priority) je obsluhována jako první. Když jsou všechny pakety z této fronty zpracovány nebo je fronta prázdná, PRIQ pokračuje frontou s další nejvyšší prioritou. V rámci jedné fronty jsou pakety zpracovány způsobem First In First Out (FIFO). Je důležité uvědomit si, že při použitá PRIQ musíte naplánovat své fronty velmi pečlivě. Protože PRIQ *vždy* zpracovává fronty s vyšší prioritou před těmi s nižší prioritou, je možné, aby fronta s vyšší prioritou způsobovala zpoždění nebo zahazování paketů fronty o nižší prioritě, pokud ta s vyšší prioritou přijímá stálý proud paketu [7].

3.2.3 TBF (Tocken Bucket Filter)

TBF je regulační prvek, který dokáže omezit maximální rychlost buď na jednotlivých podtřídách, nebo na celém rozhraní. Umožňuje pomocí direktivy burst nastavit krátkodobé překročení maximální rychlosti. Tento scheduler FreeBSD nepodporuje.

3.2.4 SFQ

SFQ je algoritmus rozřazování toku, který podle hashovací funkce určí ke kterému toku daný paket patří. Jednotlivé toky jsou obsluhovány pravidlem round-robin, takže každý paket je obsloužen férově podle času který určí hashovací funkce. Navíc se tato hashovací funkce po určité době přepočítává (řádově jde o několik sekund). Pro každý tok umožní tok MAX/celkový tok a v případě že nějaký tok nevyužije celé pásmo, přebytek rozdělí mezi ostatní. Tento scheduler FreeBSD nepodporuje.

3.2.5 RED (Random Early Detection)

Jde o algoritmus zabraňující zahlcení. Jeho účelem je zabránit aby se fronta nezaplnila na 100%. Toho je dosahováno průběžným počítáním průměrné délky (velikosti) fronty a porovnáváním se dvěma prahovými hodnotami, minimální a maximální. Pokud je průměrná délka fronty pod minimálním prahem, žádný paket nebude zahozen. Pokud průměrná délka fronty překračuje maximální práh, potom *všechny* nově příchozí pakety budou zahozeny. Pokud je průměr mezi prahovými hodnotami, pakety budou zahazovány na základě pravděpodobnosti počítané z průměrné velikosti fronty. Jinak řečeno, čím více se velikost fronty blíží maximálnímu prahu, tím více paketů bude zahazováno. Při zahazovaní paketů RED náhodně vybírá, ze kterého spojení paket zahodit. Spojení využívající velkou část pásma mají větší pravděpodobnost, že jejich pakety budou zahozeny.

RED je užitečný, protože zabraňuje situaci známé jako globální synchronizace a je schopný zvládnout nápor provozu. Globální synchronizací je nazývána ztráta celkové propustnosti kvůli tomu, že pakety jsou zahazovány z několika spojení současně. Pokud například dojde k zahlcení routeru zpracovávajícího data deseti FTP spojení a pakety ze všech (nebo většiny) budou zahozeny (což se děje u FIFO front), dojde k prudkému poklesu celkové propustnosti. To není moc ideální, protože u všech FTP spojení poklesne propustnost a zároveň nebude využit maximální potenciál sítě. RED tomu zabraňuje náhodným výběrem ze kterého spojení zahodit pakety, místo zahození všech. Spojení využívající více pásma mají větší pravděpodobnost zahození paketu. Tím jsou spojení náročnější na pásmo přiškrcena a zahlcení je zabráněno. Navíc je RED schopen zvládnout nárazový provoz, protože začne zahazovat pakety *dříve*, než se fronta zaplní. Když tedy nárazový provoz prochází, ve frontě bude dost místa pro nové pakety.

RED by měl být používán pouze pokud je transportní protokol schopen reagovat na příznaky zahlcení sítě. Ve většině případů to znamená, že RED by měl být používán u TCP spojení a ne u UDP a ICMP [7].

3.2.6 CBQ (Class Based Queueing)

CBQ je první algoritmus, který nabídl stromové členění toku a dokáže tak data rozdělovat do tříd a podtříd. V současné době se jedná o standard v oblasti řízení toku na platformě Linux kde jej postupně nahrazuje HTB. Jeho konfigurace a vlastnosti jsou podobné HFSC který jej překonává modernějším pojetím řízení toku a několika funkcemi navíc.

3.2.7 HTB (Hierarchichal Token Bucket)

HTB je dostupný pouze na platformě Linux kde je vyvinut jako náhrada CBQ se snazší konfigurací. Zachovává vlastnosti CBQ pro které přidává konfiguraci `rate` a `ceil`. Parametr `rate` nám v podstatě udává garantovanou rychlost toku, parametr `ceil` udává maximální rychlost třídy. V BSD systémech není HTB dostupný a jako modernější náhrada se používá HFSC.

3.2.8 HFSC (Hierarchical Fair Service Curve)

Hierarchical Fair Service Curve je první algoritmus, který se snaží obsáhnout tři důležité služby: garanci realtime provozu, adaptivní metodu best-effort a hierarchické sdílení linky. Jedná se o pokračovatele algoritmu HPFQ. V současnosti se jedná o nejmodernější shapovací algoritmus, který poskytuje podporu priority provozu a v návaznosti dokáže přerozdělovat tok linky realtimově podle aktuálního vytížení [8].

Hierarchie návrhu tříd je obdobná jako u CBQ – používá stromovou strukturu s hlavní ROOT třídou, která obsahuje další podtřídy a fronty. Tyto podtřídy se mohou libovolně dále větvit. HFSC rozděluje pásmo pro síťová spojení mezi několik front nebo tříd. Ke každé frontě jsou potom přiřazena spojení na základě zdrojové nebo cílové adresy, čísla portu, protokolu atd. Fronta může být volitelně nakonfigurována tak, aby si půjčovala pásmo od své rodičovské fronty, pokud je ta nevytížená. Frontám jsou také přiřazeny priority, takže pakety interaktivních spojení, jako je SSH, mohou být zpracovány dříve než fronty obsahující běžný provoz, jako třeba FTP. HFSC fronty jsou uspořádány hierarchicky.

3.2.8.1 HFSC - Hierarchie návrhu třídy

Největší možnou je root fronta, která definuje celkovou dostupnou šířku pásma. Z ní jsou vytvářeny dceřiné fronty, z nichž každé může být přiřazena část pásma root fronty. Fronty mohou mít uspořádání podobné tomuto:

```
Root Queue (50Mbps)
     Queue A (25Mbps)
     Queue B (15Mbps)
     Queue C (10Mbps)
```

V tomto případě je celková dostupná šířka pásma nastavena na 50 megabitů za sekundu (Mbps). Toto pásmo je rozděleno mezi tři dceřiné fronty. Každá fronta má přiřazenu velikost kterou nemůže překročit – díky tomu se fronty nemohou vzájemně ovlivňovat. [7]

3.2.8.2 HFSC - Vnořené třídy

Hierarchie může být dále rozšířena definováním front uvnitř front. Pro rozdělení pásma mezi různé uživatele rovným dílem a klasifikovaní jejich provozu tak, aby nějaký protokol netrpěl na úkor ostatních nedostatkem pásma, může být definována následující struktura front:

```
Root Queue (25Mbps)
   UzivatelA (10Mbps)
         ssh (1Mbps)
         bulk (9Mbps)
   UzivatelB (15Mbps)
         audio (3Mbps)
         data (7Mbps)
               http (1Mbps)
               ftp (6Mbps)
         bulk(5Mbps)
```

Součet šířky pásma přiřazeného podfrontám nesmí přesahovat šířku pásma přiděleného rodičovské (root) frontě.

Fronta může být nastavena tak, aby si půjčovala pásmo od svého rodiče, pokud ten má přebytečnou dostupnou kapacitu, protože není naplno využívána ostatními dceřinými frontami. Uvažujme následující schema front:

```
Root Queue (20Mbps)
     UzivatelA (10Mbps)
          ssh (1Mbps)
          ftp (realtime 9Mbps, upperlimit 12Mbps)
     UzivatelB (10Mbps)
```

Pokud provoz ve frontě `ftp` překročí 9Mbps a provoz ve frontě `UzivatelA` je maximálně vytížen na 1Mbps, fronta `ftp` si vypůjčí přebytečné pásmo od `UzivatelB`. Tímto způsobem může `ftp` využít více než přiřazené pásmo, pokud je přetíženo. Když se ve frontě `UzivatelB` zvýší zátěž, vypůjčené pásmo bude vráceno. Jak jsem již uvedl, hodnota realtime vyhrazuje pásmo, tudíž i v případě že ftp nevyužívá celé

pásmo, jeho nevyužitá kapacita nebude dostupná pro ostatní fronty. Naopak, pokud nadefinujeme upperlimit větší než je rychlost fronty, algoritmus HFSC se postará o půjčení pásma z jiných front pokud taková pásmo je k dispozici. Pokud není, rozdělí kapacitu mezi ostatní definované fronty. Z tohoto také vyplývá, že není vhodné pokud chceme mít dostatečnou kapacitu pro půjčování pásma, aby všechny fronty byly definovány na plnou rychlost parametrem realtime. U parametru realtime také není možné, aby součet definovaných rychlostí byl vyšší, než je celková rychlost nadřazené třídy. Naproti tomu parametr upperlimit nám zajistí "přesah" rychlostí do ostatních front a umožnění poskytnutí této kapacity dalším frontám. [7]

3.2.6.3 HFSC – priorizace provozu

HFSC navíc může přiřadit každé frontě prioritu. Fronty s vyšší prioritou jsou při zahlcení preferovány oproti frontám s nižší prioritou, pokud obě mají společnou nadřazenou třídu. Fronty se stejnou prioritou jsou zpracovávány round-robin mechanismem. Pokud předchozí příklad doplníme nastavením priorit, může vypadat následovně:

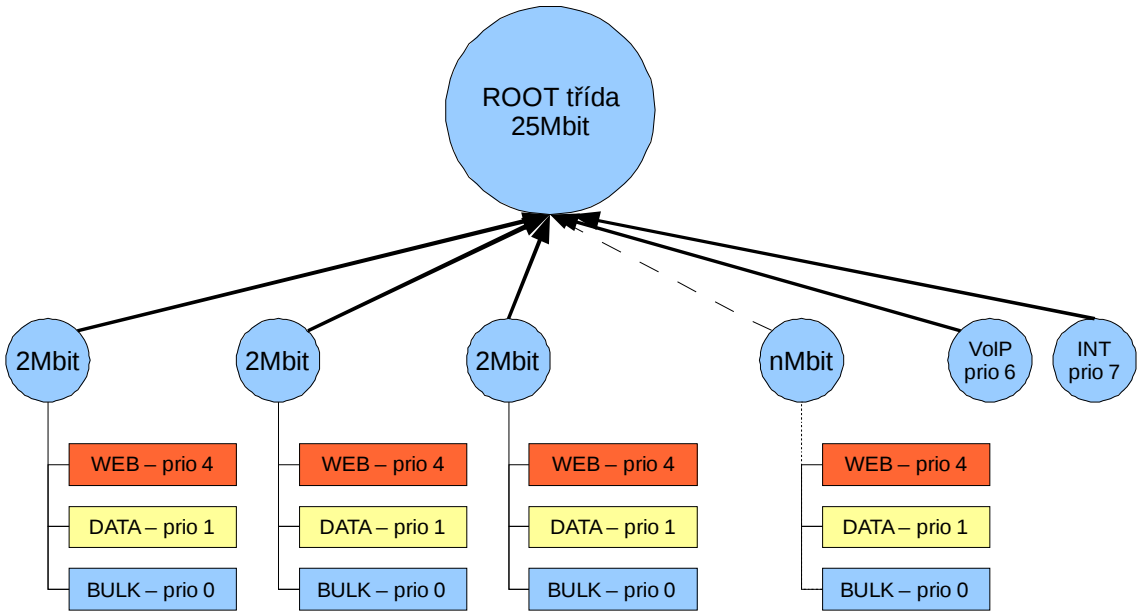
```
Root Queue (20Mbps)
    UzivatelA (10Mbps, priority 1)
        ssh (1Mbps, priority 7)
        ftp (realtime 9Mbps, upperlimit 12, priority 3)
    UzivatelB (10Mbps priority 1)
```

HFSC bude zpracovávat fronty `UzivatelA` a `UzivatelB` round-robin mechanismem – žádná fronta nebude upřednostňovaná. Během zpracovávání fronty `UzivatelA`, HFSC také zpracuje její dceřiné fronty. V tomto případě má fronta `ssh` vyšší prioritu a dostane se jí přednostnějšího zacházení než frontě `ftp`, pokud dojde k zahlcení. Všimněte si, že priority front `ssh` a `ftp` se neporovnávají s frontami `UzivatelA` a `UzivatelB`, protože nejsou všechny ve stejné větvi hierarchie [7].

4 POUŽITÉ ŘEŠENÍ

4.1 Shaping pro hraniční router

Hraniční router je vetšinou router, přes který prochází veškerý síťový provoz vedoucí z a do naší sítě. V drtivé většině případů je připojení do internetu je realizováno jedním připojením, které může být zálohováno připojením dalším (třeba i úplně jinou technologií) na které se v případě poruchy přepne. Toto přepnutí je nutné realizovat na tomto routeru. Díky tomuto faktu (že veškerý externí provoz nám může procházet jedním routerem), můžeme na tomto routeru rozdělit veškerý externí provoz. Můžeme na tomto místě rozdělit jednotlivým uživatelům rychlosti externí linky i prioritu provozu. Pro gateway (jakožto centrálního routeru sítě) je potřebné, aby požadavkům přicházejícím od uživatelů dokázala garantovat určitou rychlost linky, kterou je navíc schopna dále priorizovat. Tento tok je ve většině případů potřeba rozdělit pro uživatele s různými rychlostmi připojení, proto na tomto routeru vytvoříme třídy s různými rychlostmi. V našem případě to budou třídy s rychlostmi 4Mbit, 2Mbit, 1Mbit a 256Kbit. Tříd s těmito rychlostmi může být samozřejmě více podle počtu uživatelů. V těchto jednotlivých třídách dále rozdělíme provoz do dalších podtříd které budou mít nastaveny různé úrovně priorit. Nastavení tříd bude mít stromovou strukturu podobnou té, která je vyobrazena na obrázku č.2.



Obr. č. 2 – Schéma rozdělení toku v případě hraničního routeru

Z obrázku je patrné schéma rozdělení skupin. Jsou zde vyobrazeny 3 základní skupiny 2Mbit, VoIP a INT. Skupiny 2Mbit jsou ve schématu naznačeny také 3, nicméně jejich počet může být v podstatě libovolný a mohou mít i různou rychlost. Podrobný popis skupin a jejich nastavení zde bude popsáno později.

Je vhodné aby hraniční router měl pouze 2 síťová rozhraní. Jedno rozhraní bude pro externí provoz (vetšinou tedy provoz do internetu – upload) a druhé rozhraní pro interní provoz (tedy ten, který směřuje dovnitř naší lokální sítě). Vzhledem k vlastnosti, že ovlivňovat můžeme pouze odchozí provoz, na externím rozhraní budeme řídit kompletní odchozí provoz, tj. upload, na interním rozhraní budeme řídit z pohledu klientů download. V Linuxu je možné využít „berličku“ v podobě patche jádra IMQ. Tento patch vytvoří v jádře virtuální rozhraní, do kterého přesměruje provoz ze všech fyzických rozhraní. Díky tomu, nám stačí řídit provoz na tomto jediném rozhraní a zároveň tím budeme řídit datový tok všech rozhraní na routeru. Bohužel tento patch zřejmě nebude nikdy součástí oficiálního linuxového jádra, protože ve svém běhu

používá techniky, které nejsou z pohledu jaderných programátorů “čisté“. BSD systémy mají obdobu v podobě pipe, které ovšem umožní jen základní rozdělení rychlosti – bez vnořených podtříd a bez půjčování nevyužitého pásma.

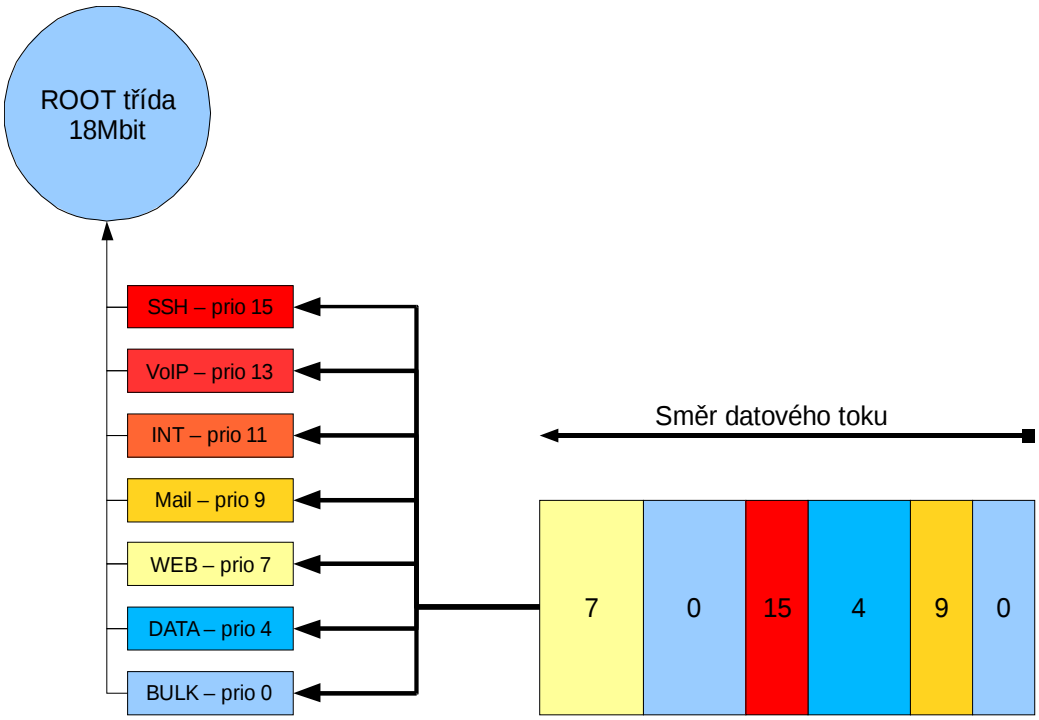
4.2 Shaping pro tranzitní router

Shaping pro tranzitní router je v tomto řešení mnohem jednodušší. I tak se ale dá provést dvěma hlavními shedulery – buď čistě pomocí sheduleru PRIQ, nebo pomocí sheduleru HFSC+PRIQ.

Použití samotného sheduleru PRIQ má výhodu hlavně v jednoduchosti a plném využití pásma které můžeme na interface použít. Naproti tomu kombinace HFSC+PRIQ nám dává možnosti mnohem jemnějšího nastavení a omezení toku i díky vlastnosti classfull disciplíny, protože tuto kombinaci můžeme dále větvit.

V této práci popíši hlavně první verzi – tj. verzi čistě pomocí sheduleru PRIQ, protože v reálném provozu dosáhla velice dobrých výsledků propustnosti. Toto nastavení je také méně náročné na procesorový výkon než varianta s HFSC a proto se dá použít téměř na jakýkoliv typ routeru, včetně speciálních platforem typu WRAP a podobných.

Pro základní rozvržení tranzitního provozu potřebujeme vědět celkovou velikost toku, kterou je schopen interface převést. Díky tomu, můžeme root třídu navrhnout podle této rychlosti a provoz v ní priorizovat až do maximální rychlosti root třídy. Na následujícím obrázku můžeme vidět schéma základního rozdělení této třídy.



Obr. č. 3 – schéma rozdělení toku v případě tranzitního routeru

Z obrázku je patrný rozdíl mezi rozdělením toku u hraničního routeru a tranzitního routeru. Zatímco v případě rozdělení toku na hraničním shaperu je použito stromové schéma rozdělení toku, kdy se třídy dělí na další podtřídy, v případě tranzitního provozu se využije spíše schéma roury, kdy provoz tekoucí do této roury pouze „přeskládáváme“ a pouštíme dál v námi určeném pořadí, kdy provoz s největší prioritou bude dál puštěn jako první a provoz s nejnižší bude poslán jako poslední až po odbavení veškerých předchozích dat.

Důležitý faktor, která je důležitá pro plné využití rychlosti linky je reálná rychlost rozhraní. Toto se týká nejčastěji bezdrátových síťových karet. Rychlost bezdrátových síťových karet je totiž halfduplexní, takže

maximální rychlost kterou výrobce udává může přenést v jednom okamžiku pouze v jednom směru. Pro příklad uvedu odhad rychlosti pro kartu XI-626, která používá oblíbený chipset PRISM 2.5. Tato karta je standartu 802.11b a má tedy od výrobce označení maximální rychlosti 11Mbit. V reálném provozu za relativně ideálních podmínek (dostatečně silný signál, malé nebo žádné rušení) tato karta dokáže přenášet data rychlostí 550kB/s, což odpovídá rychlosti 4,4Mbit/s. Toto je tedy reálná propustnost, ale pouze v jednom směru. Pokud by jsme tedy nastavily tuto rychlost na tranzitních routerech na obou stranách jednoho spoje, mohlo by se při plném zatížení tohoto spoje lehce stát, že v součtu rychlostí překročíme reálnou rychlost linky, spoj začne saturovat a QoS které jsme nasadili, přestane fungovat. Proto je vhodné pokud požadujeme symetrickou rychlost linky, zvolit tuto maximální rychlost poloviční – pro každé rozhraní tedy rychlost kolem 2Mbit. Ztratíme tím sice maximální možnou rychlost, budeme však schopni garantovat na této lince QoS. Pokud nepotřebujeme symetrickou rychlost, můžeme nastavit na jednom spoji různé rychlosti na routerech, v poměru jaký požadujeme, tj. např na jednom routeru rychlost rozhraní 3Mbit, na protější straně však už pouze 1Mbit, atp. Při použití těchto zásad se nám nemůže stát, že překročíme celkovou rychlost linky a proto nám bude QoS spolehlivě fungovat i při maximálním vytížení spoje.

Jiná situace je při použití tranzitního QoS na interfacech, které jsou fullduplexní – tedy ty, které dokážou přenášet data v obou směrech zároveň stejnou rychlostí. Jedná se hlavně o fast a giga ethernetové síťové prvky. U těchto „spojů“ je možné použít téměř maximální rychlost rozhraní, nepatrně sníženou pouze s ohledem na režii samotného QoS.

5 APLIKACE VYTVOŘENÉHO ŘEŠENÍ

Praktické řešení které je použito v této bakalářské práci je použito v ostrém provozu ve firmě ITSolutions. Tato firma se zabývá poskytováním nejen bezdrátového internetu – ten však tvoří největší část zákazníků. Cílem QoS v této síti, je poskytnout maximální možný komfort při užívání internetu, který standart Wi-Fi 802.11b/g nepodporuje v požadované míře. Na úrovni linkové vrstvy je částečná podporu multimediálního provozu pouze u standartu 802.11g – bohužel do značné míry je odlišná podle výrobce používaného zařízení. Tato podpora ovšem není dostatečná pro komerční nasazení, kdy je potřeba aby se síťový provoz choval v co největší míře předvídavě, byl odolný vůči přetížení a bylo možné jej co nejlépe řídit. Pro tuto aplikace jsem proto po dlouhotrvajícím testování různých druhů systémů a obslužných programů vybral kombinaci operačního systému FreeBSD a v něm obsaženého firewalu PF se shapovacím algoritmem ALTQ + HFSC na úkor hojně používané kombinace Linux/HTB.

Pro tuto bakalářskou práci budeme používat zjednodušené verze skriptů použitých v reálném nasazení. Nebude to samozřejmě na úkor funkčnosti, ale kvůli zkrácení zápisu vybraných skriptů. V současné době má firma ITSolutions okolo 500 klientů a proto se jedná již o značný počet shapovacích skupin – díky různě nastavené agregaci se jedná o cca 30 skupin jejichž nastavení je úplně stejné a liší se pouze drobnými úpravami názvu (číslo skupiny). Proto jsem se v této bakalářské práci rozhodl ukázat použití na 3 uživatelských skupinách (skupiny s rychlostí provozu 1Mbit) a 2 systémových skupin pro VoIP a ostatní realtime provoz a původní skripty budou umístěny v dodatcích.

5.1 Rozřazení skupin

Všechny uživatele, které chceme rozřadit do různých rychlostních skupin, máme uloženy v souboru uzivatele. Tento soubor má následující syntaxi:

```
IP_adresa1:rychlost1    #Jméno uživatele1
IP_adresa2:rychlost2    #Jméno uživatele2
IP_adresa3:rychlost3    #Jméno uživatele3
```

Pro příklad v reálném použití je to pro 3 různé uživatele a 3 různé rychlostní skupiny takto:

```
192.168.27.43:2048       #Uzivatel1
192.168.27.51:1024       #Uzivatel2
192.168.27.59:256       #Uzivatel3
```

Pro samotné rozřazení se používá skript sorting.sh. Jedná se o shellový skript, který dělá 3 základní operace. Tento skript načítá ze souboru uzivatele IP adresy a k nim zadané rychlosti. Podle rychlostí řadí IP adresy do požadovaných skupin, které ukládá do souborů. Ve skriptu je nastaven maximální možný počet IP adres v souboru, podle kterého dokáže skript třídit IP adresy do dalších souborů po překročení námi zadaného limitu. Navíc toto generování roztřiďuje IP adresy v těchto souborech náhodně a s rozptylem tak, aby ve všech skupinách byl přibližně stejný počet IP adres. V součinosti s podporou tabulek firewallu PF, jsme schopni tímto jednoduchým způsobem definovat námi požadovanou agregaci pro jednotlivé linky.

Samozřejmě je více typů těchto vygenerovaných souborů a proto je pro IP adresy s rychlostí 2Mbit použita sada souborů pojmenovaných sk2048_1 až 2048_N, kde N je číslo, které vyjde po rozřazení všech IP adres. Pokud například nastavíme agregaci 1:30 pro skupiny s rychlostí 2048kbit a v souboru uživatele budeme mít 31 IP adres, pak tento skript zabrání nerovnoměrnému rozdělení tak, že rozdělí do jednoho souboru sk2048_1 16 adres a v souboru sk2048_2 bude 15 adres.

Pro ostatní námi definované rychlosti jsou samozřejmě další soubory pojmenovány jinak, ale sémantika

pojmenování většího počtu souborů téže rychlosti zůstává stejná. Výsledek rozdělení je například adresář se soubory sk2048_1 až sk2048_12.

Pro ukázku nastavení hraničního shaperu bude použito 3 různých rychlostních skupin (1Mbit, 512Kbit a 256Kbit). Skupina s rychlostí 1Mbit bude však použita vícekrát. Těchto skupin bude více díky použití vlastnosti firewallu PF, kdy umožňuje načíst více IP adres do tabulek a tyto tabulky dále používat pro rozřazování paketů. Proto budou vytvořeny tabulky obsahující maximálně 20 IP adres, což v našem řešení bude odpovídat také maximální agregaci. Navíc budou tyto 1Mbit skupiny obsahovat ještě další podskupiny které budou použity pro priorizaci provozu v rámci hlavní 1Mbit třídy. Skupiny s rychlostí 512Kbit a 256Kbit nebudou obsahovat další podskupiny, budou však mít nastaveny různé priority, které budou vyšší než standardní 1Mbit třída. Tyto skupiny budou zajišťovat interaktivní provoz nezávisle na jednotlivých rychlostních třídách z důvodu, aby tento provoz nebyl závislý na rychlosti a vytížení defaultních tříd (1Mbitových).

Tyto tabulky budou naplněny z vygenerovaných souborů IP adres. Počet IP adres v tabulce bude dán právě počtem adres ve vygenerovaném souboru.

5.2 Příprava FreeBSD - instalace systému a příprava jádra

Stáhneme CD ISO image s nejnovější verzí FreeBSD. Odkaz pro stáhnutí je například zde: ftp://ftp.freebsd.org/pub/FreeBSD/. Po stažení a vypálení nainstalujeme systém v konfiguraci kterou potřebujeme. Budeme však navíc potřebovat zdrojové kódy jádra, protože podpora ALTQ, není v distribučním jádře defaultně zapnuta. Tyto zdrojové kódy můžeme vybrat již během instalace, nebo je stáhnout samostatně pomocí CVS.

Zkopírujeme tedy distribuční konfiguraci jádra ze souboru GENERIC, do souboru ALTQ.

```
brana# cp /usr/src/sys/i386/conf/GENERIC /usr/src/sys/i386/conf/ALTQ
```

Poté do tohoto nového souboru přidáme následující řádky:

```
options         ALTQ
options         ALTQ_CBQ
options         ALTQ_RED
options         ALTQ_RIO
options         ALTQ_HFSC
options         ALTQ_PRIQ
options         ALTQ_NOPCC

device          pf
device          pflog
device          carp
device          pfsync
```

Touto direktivou zapneme v kernelu podporu pro schedulery CBQ, HFSC, RED, RIO a PRIQ a také samotný firewall PF. V tomto příkladu je navíc zapnuta podpora protokolu CARP, který je úzce spojen právě s firewallem PF a umožňuje vytvořit zcela redudantní firewall.

Po editaci a uložení konfiguračního souboru je potřeba zkompilovat nový kernel:

```
brana# make buildkernel KERNCONF=ALTQ
brana# make installkernel KERNCONF=ALTQ
```

Tyto příkazy nám zajistí kompilaci nového kernelu s podporou ALTQ, kterou jsme zapnuli v konfiguračním souboru a také zkopírování souborů do adresáře /boot/kernel. Původní kernel je přesunut do adresáře /boot/kernel.old. V případě jakýchkoliv problémů při bootování nového jádra pak můžeme nabootovat původní distribuční jádro. Poté stačí systém FreeBSD restartovat a podpora pro ALTQ bude v novém jádře zapnuta.

5.3 Konfigurace systému

V základní instalaci je již PF předkonfigurován na co největší bezpečnost s defaultní politikou DENY. Není ovšem zapnut. Samotné zapnutí firewallu PF je třeba zapnout v souboru /etc/default/rc.conf. Najdeme a zeditujeme řádek `pf_enable="NO"` na `pf_enable="YES"`. Touto volbou zapneme firewall při každém bootování. Ručně je možno zapnout PF pomocí pfctl -e , vypnout pomocí pfctl -d . Toto zapnutí ale nenahraje žádná pravidla firewallu. Pouze zapne firewall, který má prázdnou sadu pravidel, tudíž nefiltruje žádný provoz. Samotné načtení pravidel je při tomto ručím spuštění potřeba nahrát ze souboru pf.conf pomocí příkazu pfctl -f /etc/pf.conf. V případě zapnutí firewallu pomocí rc.conf za nás toto udělá systém sám – je třeba jej jen zrestartovat, aby se načetla nová konfigurace systému. Pro samotné ovládání PF pomocí příkazové řádky slouží program pfctl.

Jeho základní příkazy jsou:

```
# pfctl -f /etc/pf.conf     Nahraje soubor pf.conf
# pfctl -nf /etc/pf.conf    Zpracuje soubor, ale nenahraje ho – užitečné zejména
při nastavování a ladění nových pravidel
# pfctl -Nf /etc/pf.conf    Načte ze souboru pouze pravidla NAT
# pfctl -Rf /etc/pf.conf    Načte ze souboru pouze pravidla filtru

# pfctl -sn                 Ukáže aktuální pravidla NAT
# pfctl -sr                 Ukáže aktuální pravidla filtru
# pfctl -ss                 Ukáže aktuální tabulky stavů
# pfctl -si                 Ukáže statistiky filtru a počítadla
# pfctl -sa                 Ukáže vše co může být ukázáno
```

Defaultní cesta ke konfiguračnímu souboru PF, je /etc/pf.conf.

Základní schéma rozvržení konfiguračního souboru PF, musí dodržovat pořadí jednotlivých nastavení. Tyto nastavení jsou:

- Makra – uživatelsky definované proměnné –jsou definovány jako první tak, aby s nimi mohl PF pracovat v jakékoliv následující části
- Tabulky – seznam IP adres – důležitý pomocník pro zjednodušení syntaxe pravidel
- Volby (options) – nastavení vlastností PF – timeouts, limits, optimalization, atd...
- Scrubing – zpracování paketů – defragmentace a normalizace
- Fronty – nastavení využití pásma a priorizace paketů

- Překlad adres – nastavení překladu adres (NAT) a také přesměrování paketů
- Filtrovací pravidla – filtrování a blokování paketů

5.3.1 Makra

Mezi základní vlastnosti firewallu PF, je možnost použití uživatelských proměnných v konfiguračním souboru pf.conf. Tyto proměnné jsou pojmenovány jako makra, nicméně jejich definice je obdobná definici proměnných u různých programovacích jazyků. Makro musí začínat písmenem a může obsahovat písmena, čísla a podtržítka. Název makra nemůže být rezervované slovo jako `pass`, `out`, nebo `queue`.

Direktiva je následující `makro = "parametr1, parametr2, ... parametrN"` Vlastní vyvolání provedeme znakem $.

V našem případě pro nadefinování typů služeb je nastavíme takto:

```
data= "{ smtp smtps pop3 pop3s ftp ftps ftp-data ftps-data imap imaps }"
voip= "{ 1722 5004 5060:5064 }"
int="{ 27000:27039 16567 10000:10020 3784 3788 50050 6112:6119 }"
```

Jak z této ukázkové definice můžeme vidět, je možný zápis dvěma způsoby. V případě standardních služeb definovaných RFC, můžeme použít jméno služby (typicky, http, https, ftp, ftps a ostatní služby definované v souboru `/etc/services`). V případě standardních i nestandardních služeb můžeme použít přímo číslo portu.

5.3.2 Tabulky

Jsou používány jako uložení skupiny IP adres. Vyhledávání v tabulkách je mnohem rychlejší než v seznamech, či přímo v zápisech pravidel, proto se výborně hodí pro naše použití, při kterém dochází k prohledávání několika souborů IP adres. Další vlastností tabulek totiž je, že mohou být naplněny ze souboru, kde každá IP adresa zabírá právě jeden řádek. Pro zavedení tabulky se použije direktiva table doplněná závorkami s názvem tabulky, následovaná složenými závorkami se seznamem adres.

```
table <ukazka> {192.168.1.0/24, 192.168.2.0/24 }
```

Pro vyvolání tabulky v pravidle `pass in on rl0 from <ukazka> to any` způsobí virtuální rozložení tohoto pravidla pro všechny uvedené IP adresy. Virtuálně proto, že jak jsem již uvedl, seznam IP adres je pomalejší na prohledávání – proto toto pravidlo zůstává jedno, pouze se k němu v tabulce vyhledá potřebná IP adresa. Díky tomuto způsobu, je značně usnadněn zápis pravidel firewallu a také takto zapsaný firewall je méně náročný na procesorový výkon, než je jeho obdoba například pomocí IPTABLES v GNU/Linuxu. V našem případě ovšem využijeme vlastnosti naplnění tabulky ze souboru. Syntaxe potom bude následující:

```
table <1M_1> persist file "/etc/shaping/sk1024_1"
```

Vytvoří tabulku s názvem 1M_1, kterou naplní seznamem IP adres ze souboru /etc/shaping/sk1024_1.

5.3.3 Volby

Umožňují nastavit globální vlastnosti chování PF.

5.3.4 Scrubing

Jedná se o normalizaci paketů tak, aby se odstranili nejednoznačnosti v interpretaci na cílové destinace paketů. Zároveň slouží ke spojení fragmentovaných paketů a zahodí TCP pakety s neplatnou kombinací flagů.

5.3.5 Fronty (queuing)

Fronty jsou v PF velice dobře integrovány, protože firewall PF je moderní a poměrně mladý typ firewallu, takže již při jeho navrhování se počítalo s podobnými funkčnostmi. Na rozdíl od již zmiňovaných IPTABLES, do kterých byly funkce na markování a řazení paketů do front přidány až při pozdějším vývoji. Z tohoto důvodu není podle mého názoru zprovoznění shapování v IPTABLES tak jednoduché jako u PF. Zapnutí rozřazování do front provedeme pomocí nástroje ALTQ, který je opět přímo integrován v PF. Tímto nástrojem nadefinujeme veškeré vlastnosti rozřazování do front, avšak samotné rozřazení provozu do těchto nadefinovaných front provádíme pomocí filtrovacích pravidel (povolený provoz) ke kterým přidáme direktivu queuing.

5.4 Nastavení queuingu pomocí ALTQ

Syntaxe nastavení shapingu rozhraní v souboru /etc/pf.conf [7]:

```
altq on interface scheduler bandwidth bw qlimit qlim tbrsize velikost queue { queue_list }
```

- `atlq` – zapíná řazení do front na rozhraní, definuje použití plánovače a vytváří root frontu
- `queue` – definuje fronty
  - *`interface`* – definuje rozhraní na kterém zapínáme řazení do front. V současné době jsou pro ALTQ podporovány síťové karty využívající tyto ovladače: an, ath, awi, bfe, bge, dc, de, ed, em, fxp, hme, le, lnc, my, nve, re, rl, sf, sis, sk, ste, stge, vr, wi a xl
- *`scheduler`* – zapíná typ plánovače na rozhraní. Možné hodnoty jsou HSFC, CBQ a PRIQ. Pouze jeden plánovač může být použit na root rozhraní v jednu chvíli.

`bw` – celkové množství datového toku dostupné na rozhraní. Toto může být specifikováno jako absolutní hodnota pomocí přídavku b, kb, mb a gb které reprezentují bity, kilobity, megabity a gigabity za sekundu, nebo procentuálně jako šířku pásma dostupného na rozhraní.

- `qlim` – maximální počet paketů ve frontě. Jedná se o nepoviný parametr, jehož defaultní hodnota je 50
- `queue_list` – seznam front vytvořených pod hlavní root třídou. Tyto fronty mohou obsahovat další podtřídy pokud použijeme classfull disciplínu

5.4.1 Nastavení hlavní třídy - upload

Nastavením hlavní třídy myslíme vytvoření root třídy na požadovaném rozhraní. V tomto případě vycházím z funkčního nastavení, při kterém je hlavní třída nastavena na velikost 25Mbit a dělí se na další podtřídy.

V našem příkladě tedy:

```
altq on rl0 hfsc bandwidth 25Mb qlimit 50 queue { voip_net, int_net, 1M1_net,
1M2_net,1M3_net }
```

V tomto příkladě na rozhraní rl0 definujeme hlavní HFSC root třídu o velikosti 25Mbit. Tato hlavní třída obsahuje podtřídy voip_net, int_net, 1M1_net, 1M2_net a 1M3_net. Přídavek _net je tam z důvodu rozlišení skupin provozu který směřuje do internetu – pro provoz směřující do sítě používám u všech skupin nastavení _sit.

5.4.2 Nastavení podtříd

Definice jednotlivých podtříd je podobná, pouze není potřebné definovat rozhraní na kterém tato podtřída běží. Je to dáno definicí hlavní root třídy, pod kterou nám tato podtřída spadá.

Syntaxe pro zápis definice front je následující je opět uvedena v příručce PF na stránkách www.openbsd.org [7]:

```
queue name bandwidth bw [priority pri] [qlimit qlim] \
   scheduler ( sched_options ) { queue_list }
```

- *name* – název fronty. Musí se shodovat s jedním ze jmen uvedeném v *queue_list*. Jména front můžou být maximálně 15 znaků dlouhé.
- *bw* – šířka pásma (dostupná rychlost) pro tuto frontu. Opět může být specifikována pomocí přídavku `b`, `Kb`, `Mb` a `Gb` který reprezentuje bity, kilobity, megabity a gigabity za vteřinu, nebo procentuálně z šířky pásma. Tento parametr je dostupný pouze s použitím classfull disciplín. Když není zadán, použije se defaultně celá dostupná šířka pásma.
- *pri* – určuje prioritu fronty. Pro CBQ a HFSC je rozmezí priorit 0 až 7 a pro `priq` je rozmezí 0 až 15. Priorita 0 je nejnižší stupeň. Pokud není určena, je defaultně použita priorita 1.
- *qlim* – maximální počet paketů ve frontě. Pokud není nastaven, je defaultně použito 50.
- *scheduler* – typ plánovače který je použit (HFSC, CBQ, PRIQ,...). Musí být použit stejný jako v nadřazené root třídě.
- *sched_options* – další volby plánovače:
  - `default` – definuje defaultní frontu, do které se budou řadit veškeré jinak nezařazené pakety. Musí být uvedena pouze jedna defaultní fronta na rozhraní.
  - `red` - zapíná Random Early Detection (RED) na této frontě.
  - `rio` – zapíná RED s IN/OUT. Toto rošíření RED udržuje vícenásobně počítanou hodnotu fronty v závislosti na jednotlivých IP spojeních.
  - `ecn` - zapíná Explicit Congestion Notification (ECN) na této frontě. `Ecn` implikuje `red`.
  - `borrow` – určuje že tato fronta si může půjčit kapacitu z nadřazené fronty. Tato volba může být použita pouze s plánovačem CBQ.
  - *queue_list* – seznam podtříd pro které je nadřazena tato třída. Tuto volbu je možné použít pouze s plánovačem CBQ a HFSC.
  - `upperlimit` – (HFSC) – definuje nejvyšší možnou rychlost fronty
  - `realtime` – definuje vždy vyhrazenou šířku pásma fronty. V podstatě se jedná o minimální

garantovanou rychlost. Pouze u HFSC

- `linkshare` – poměr mezi upperlimit a realtime. Tato volba nám může v určitých případech nahradit předchozí dvě. Určuje jaká velikost fronty je vyhrazená/může být poskytnuta ostatním.

V našem případě nastavení podtříd pro skupinu 1Mbit:

```
queue 1M1_net bandwidth 1Mb hfsc { 1M1_n_web, 1M1_n_data, 1M1_n_bulk }
      queue 1M1_n_web bandwidth 1% priority 4 hfsc(red upperlimit 99%)
      queue 1M1_n_data bandwidth 1% priority 1 hfsc(red upperlimit 80%)
      queue 1M1_n_bulk bandwidth 1% priority 0 hfsc(red upperlimit 50%)

queue 1M2_net bandwidth 1Mb hfsc { 1M2_n_web, 1M2_n_data, 1M2_n_bulk }
      queue 1M2_n_web bandwidth 1% priority 4 hfsc(red upperlimit 99%)
      queue 1M2_n_data bandwidth 1% priority 1 hfsc(red upperlimit 80%)
      queue 1M2_n_bulk bandwidth 1% priority 0 hfsc(red upperlimit 50%)

queue 1M3_net bandwidth 1Mb hfsc { 1M3_n_web, 1M3_n_data, 1M3_n_bulk }
      queue 1M3_n_web bandwidth 1% priority 4 hfsc(red upperlimit 99%)
      queue 1M3_n_data bandwidth 1% priority 1 hfsc(red upperlimit 80%)
      queue 1M3_n_bulk bandwidth 1% priority 0 hfsc(red upperlimit 50%)
```

Z tohoto zápisu je patrné, že nastavujeme podtřídy pouze pro jednu 1Mbit skupinu do tří podtříd. Jedná se o podtřídy pojmenované 1M1_n_web, 1M1_n_data a 1M1_n_bulk. Přídavek _n je opět použit pro rozlišení provozu směřujícího do internetu. Podtřída 1M1_n_web je určena převážně pro webový provoz, který je nadefinován v jiné části skriptu pf.conf. Bandwidth je u všech podskupin nastaven na 1%, protože v definici podtříd má význam vyhrazení pásma. V tomto případě je největší část pásma sdílena i s ostatními třídami až do výše hodnoty toku definovanou velikostí upperlimit. Při nastavení upperlimit 100% zajistí nejvyšší možnou rychlost 1Mbit, podle nastavení zděděném hlavní třídou. Tato rychlost bude ale dostupná pouze v případě, že další podtřídy budou prázdné a nepůjde přs ně žádný datový tok. V opačném případě budou sdílet tento provoz. Z tohoto důvodu je direktivou priority, přiřazena jednotlivým podtřídám priorita zpracování požadavků. Web provoz s prioritou 4 bude mít v této hlavní třídě nejvyšší prioritu zpracování. Požadavky přicházející na provoz z ostatních podtříd, budou vyřízeny až po obsloužení této třídy.

Obdobně budou vyřizovány požadavky z dalších tříd – vždy podle úrovně priorit. Třída data má nižší prioritu a je do ní řazen datový provoz typu FTP, kde nám nezáleží tolik na rychlosti vyřízení požadavku. Třída bulk je třída s nastavenou nejnižší prioritou. Do této třídy řadíme veškerý ostatní provoz, který nevyhoví předchozím pravidlům a nadefinovaným skupinám provozu. Z velké části se bude jednat o provoz různých P2P sítí, a služeb které nevyužívají pevně dané porty. Tyto služby proto budou obsluhovány s nejnižší prioritou. Při bližším prozkoumání těchto skriptů si můžeme všimnout rozdílného nastavení parametru upperlimit. Je to z důvodu maximální rychlosti kterou přiřadíme jednotlivým podtřídám. Pokud se specializujeme na provoz a poskytování webu, je zřejmé, že ostatní provoz bude mít nižší prioritu a provoz typu P2P, který je všeobecně u poskytovatelů internetu považován za škodlivý, budeme chtít omezit.

Někteří poskytovatelé jdou cestou úplného zákazu P2P provozu, nicméně já jsem šel cestou snížení rychlosti tohoto provozu na akceptovatelnou rychlost tak, aby ovlivňovala „hlavní“ služby co nejméně a zároveň byla pro klienty použitelná.

5.4.3 Nastavení třídy voip_net

```
queue voip_net bandwidth 512Kb qlimit 15 priority 7 hfsc(realtime 512Kb upperlimit 1Mb)
```

Tato třída nebude obsahovat další podtřídy. Jedná se o podtřídu hlavní root třídy, která má nastavenu nejvyšší prioritu provozu. Bude do ní řazen veškerý námi zvolený VoIP provoz i ze všech ostatních tříd IP adres z důvodu, aby byla co nejvíce nezávislá na přenosovém pásmu jednotlivých podtříd, protože i v případě maximálního vytížení některé podtřídy půjde tento provoz přes třídu, která je vyčleněná jen pro telefonii. Při použití parametru realtime, navíc garantujeme nastavenou rychlost při jakémkoliv spojení.

5.4.4 Nastavení třídy int_net

queue int_net bandwidth 256Kb qlimit 20 priority 6 hfsc(realtime 128Kb upperlimit 256Kb)

Tato třída opět nebude obsahovat další podtřídy a její určení je obdobné jako u předchozí voip třídy. Má nastavenu nižší prioritu než voip třída a slouží k řazení dalšího interaktivního provozu, u kterého nechceme aby nám ovlivňoval voip třídu.

5.4.5 Nastavení tříd pro download

Pro nastavení příchozího provozu využijeme všech předchozích poznatků a nastavení platná pro upload. Jedinou změnou bude přejmenování těchto tříd a jejich nastavení na jiné síťové rozhraní. V našem případě tedy podle předchozího nastavení bude skript vypadat následovně:

```
altq on $int_if hfsc bandwidth 25Mb qlimit 50 queue { voip_sit, int_sit, 1M1_sit, 1M2_sit, 1M3_sit }

queue voip_sit bandwidth 512Kb qlimit 15 priority 7 hfsc(realtime 512Kb upperlimit 1Mb)

queue int_sit bandwidth 256Kb qlimit 20 priority 6 hfsc(realtime 128Kb upperlimit 256Kb)

    queue 1M1_sit bandwidth 2Mb hfsc { 1M1_s_web, 1M1_s_bulk, 1M1_s_data }
      queue 1M1_s_web bandwidth 1% priority 4 hfsc(red upperlimit 2Mb)
      queue 1M1_s_data bandwidth 10% priority 1 hfsc(red upperlimit 80%)
      queue 1M1_s_bulk bandwidth 1% priority 0 hfsc(red upperlimit 50%)

    queue 1M2_sit bandwidth 2Mb hfsc { 1M2_s_web, 1M2_s_data, 1M2_s_bulk }
      queue 1M2_s_web bandwidth 1% priority 4 hfsc(red upperlimit 2Mb)
      queue 1M2_s_data bandwidth 1% priority 1 hfsc(red upperlimit 80%)
      queue 1M2_s_bulk bandwidth 1% priority 0 hfsc(red upperlimit 50%)

    queue 1M3_sit bandwidth 2Mb hfsc { 1M3_s_web, 1M3_s_data, 1M3_s_bulk }
      queue 1M3_s_web bandwidth 1% priority 4 hfsc(red upperlimit 2Mb)
      queue 1M3_s_data bandwidth 1% priority 1 hfsc(red upperlimit 80%)
      queue 1M3_s_bulk bandwidth 1% priority 0 hfsc(red upperlimit 50%)
```

5.4.6 Překlad adres

V této části konfiguračního souboru PF nadefinujeme NAT, BINAT a redirecting. NAT (Network Address Translation) je síťová technika, která ve výsledku umožňuje použití jedné IP adresy pro několik, které používáme za rozhraním na kterém máme spuštěný NAT.

5.5 Filtrovací pravidla

V této sekci konfiguračního souboru PF provádíme samotné rozřazení paketů. Jde o selektivní výběr, který umožní průchod nebo blokování paketů podle námi nastavených pravidel. Pomocí direktivy queue umožňuje také rozřazení do jednotlivých front. Má smysl pouze ve spojení s pravidlem pass – tedy pravidlem, které povoluje provoz – a také podle důvodů které jsou uvedeny v předchozí části musí být aplikován pouze na odchozí provoz. V tomto případě povolí provoz, ale zároveň jej zařadí do požadované fronty. Syntaxe pravidel pro firewall PF, jak jej uvádí stránky www.openbsd.org [5]:

Obecná, *velmi zjednodušená* syntax pro filtrovací pravidla je:

```
action direction [log] [quick] on int [af] [proto protocol] \
   from src_addr [port src_port] to dst_addr [port dst_port] \
   [tcp_flags] [state]
```

action
: Akce, která bude provedena na pakety splňující pravidlo, buď *pass* nebo *block*. Akce *pass* nechá projít paket zpátky do kernelu pro další zpracování, zatímco akce *block* bude reagovat v závislosti na volbě block-policy. Defaultní rekace může být přepsána buď specifikováním *block drop* nebo *block return*.

direction
: Směr paketu jakým jde na interface, buď *in* nebo *out*.

`log`
: Specifikuje že by paket měl být zalogován pomocí pflogd. Pokud pravidlo specifikuje volbu *keep state* nebo *modulate state* pak pouze pokud paket nastaví stav, je stav zalogován. K logování všech paketů, nehledě na to jestli nastaví stav nebo ne, použijte *log-all*.

`quick`
: Pokud paket matchuje pravidlo obsahující `quick` pak je toto pravidlo považováno za poslední splňující a je provedena akce specifikovaná v tomto pravidle.

int
: Jméno síťového interface, kterým prochází paket.

af
: Address family paketu, buď `inet` pro IPv4 nebo `inet6` pro IPv6. PF je obvykle schopen zjistit tento parametr v závislosti na zdrojové a cílové adrese (adresách).

protocol
: Layer 4 protokol:
  - tcp
  - udp
  - icmp
  - icmp6
  - Platné jméno protokolu z /etc/protocols
  - Číslo protokolu mezi 0 a 255
  - Množina protokolů použitím list.

src_addr, *dst_addr*
: Zdrojová/cílová adresa v IP hlavičce. Adresy mohou být specifkovány jako:
  - Jedna IPv4 nebo IPv6 adresa.
  - Síťový blok v CIDR formátu.
  - Plně kvalifikované doménové jméno které se resolvuje via DNS ve chvíli,, kdy je množina pravidel nahrána. Všechny výsledné IP adresy budou substituovány do tohoto pravidla.
  - Jméno síťového interface. Všechny IP adresy přiřazené tomuto síťovému interface budou substituované do tohoto pravidla.
  - Jméno síťového inteface následovaného */netmask* (např., `/24`). Každá IP adresa na tomto zařízení kombinovaná s maskou vytvoří CIDR síťový blok, který jje substituován do tohoto pravidla.

- Jméno síťového interface v závorkách ( ). Toto řekne PF aby obnovil pravidlo pokud se IP adresa (adresy) na takto pojmenovaném interface změní. Toto je užitečné na interface, které dostane svoji IP adresu via DHCP nebo dial-up, protože množina pravidel nemusí být znovu nahrána pokaždé, když se adresa změní.
- Jméno síťového interface následované klíčovým slovem `:network` nebo `:broadcast`. Výsledná CIDR síť (např., 192.168.0.0/24) nebo broadcast adresa (např., 192.168.0.255) bude substituována do pravidla když je množina pravidel nahrána.
- table.
- Cokoliv z výše uvedeného, ale negováno pomocí modifiktoru `!` ("not").
- Množina adres použitím list.
- Klíčové slovo *any* představující všechny adresy
- Klíčové slovo *all* což je zkratka za `from any to any`.

src_port, dst_port

Zdrojový/cílový port v Layer 4 hlavičce paketu. Porty mohou být specifikovány jako:

- Číslo mezi 1 a 65535
- Platná jméno služby z /etc/services
- Množina portů použitím list
- Rozmezí:
  - != (nerovnost)
  - < (ostře menší než)
  - > (ostře větší než)
  - <= (menší nebo rovno)
  - >= (větší nebo rovno)
  - >< (rozmezí)
  - <> (inverzní rozmezí)

    Poslední dva jsou binární operátory (berou dva argumenty) a nezahrnují argumenty v rozmezí.

tcp_flags

Specifikuje flagy, které musí být nastaveny v TCP hlavičce při použití `proto tcp`. Flagy jsou specifikovány jako `flags check/mask`. Např. : `flags S/SA` - to řekne PF aby se díval pouze na flagy S a A (SYN a ACK) a matchoval pokud je "nahozen" SYN flag.

state

Specifikuje zda je udržována informace o stavu paketů splňujících toto pravidlo.

- *keep state* - pracuje s TCP, UDP, a ICMP.
- *modulate state* - pracuje pouze s TCP. PF bude generovat silná Initial Sequence Numbers (ISNs) pro paketu splňující toto pravidlo.

5.1.1 Filtrovani upload

Při použití v našem příkladě, vypadá zápis povolení a řazení provozu následovně:

```
#Povoleni upload provozu a zaroven rozrazovani do trid pro prvni skupinu 1Mbit

pass in log quick on $int_if proto icmp from <1M_1> to any queue int_net

pass in log quick on $int_if proto tcp from <1M_1> to any port ssh  queue int_net

pass in log quick on $int_if proto { tcp,udp } from <1M_1> to any port $int queue int_net

pass in log quick on $int_if proto { tcp } from <1M_1> to any port { http https } queue 1M1_n_web

pass in log quick on $int_if proto { tcp } from <1M_1> to any port $data queue 1M1_n_data
```

```
pass in log quick on $int_if proto { tcp udp } from <1M_1> to any port $voip
queue voip_net
pass in log quick on $int_if from <1M_1> to any queue 1M1_n_bulk
```

Těmito pravidly roztřídíme datový tok pouze pro jeden směr (interní interface – download). Prvními třemi pravidly `pass` povolíme provoz na interním interface pro protokol ICMP, pro SSH a pro IP provoz využívající porty, které jsme nadefinovali makrem int ze skupiny IP adres <1M_1> kamkoliv a zároveň jej zařadíme do fronty int_net. Tuto frontu máme již definovánu v ALTQ s prioritou 7, tedy nejvyšší prioritou. Zvláště u protokolu SSH je to důležité kvůli obsluze routerů po trase, aby i v případě maximálního vytížení skupiny měl stále dostupnou kapacitu.

V případě protokolu http a https (tedy nejčastěji webový provoz) jej rozřadíme do fronty 1M1_n_web, který je podtřídou skupiny 1M1. V tomto nastavení má prioritu 4. Podtřída web má díky tomu nejvyšší prioritu v třídě 1M1 a zaručí, že webový provoz bude mít přednost před ostatním provozem, zároveň však maximální rychlost nepřekročí rychlost podtřídy 1M1 (tedy 1Mbit).

Pro rozřazení do podtřídy 1M1_n_data použijeme makro $data, ve kterém máme nadefinované požadované typy provozu. V tomto případě tedy provoz typu FTP, SMTP, POP3, atd.

Třída voip se rozřazuje do jedné z hlavních tříd, opět podle nastavení v makru $voip.

Jako poslední pravidlo platné pro skupinu 1M1 je zařazení provozu do podtřídy bulk. Tato třída má v ALTQ nastavenu nejnižší prioritu a podle syntaxe zápisu se do ní bude řadit veškerý zbývající provoz, který není nadefinován v ostatních skupinách. Zajistí nám to předvídatelné chování, neboť v případě kdy se jedná o problémové služby typu P2P sítí, které naráz otvírají několik tisíc spojení pokaždé na jiném portu, tak všechen tento tok spadne do třídy s nejnižší prioritou a bude obsloužen až po odbavení ostatních přednějších paketů. Zároveň díky vlastnosti HFSC algoritmu, který umožňuje půjčování nevyužitého pásma, při nízkém využití předchozích tříd, bude mít i tato třída dostupné zbylé volné pásmo.

Všechna tato uvedená nastavení jsou obdobná i pro ostatní třídy. V našem případě máme k dispozici 3 skupiny s rychlostí 1Mbit. Pokud chceme rozřazovat provoz i do těchto skupin, musíme tuto soupravu pravidel použít tolikrát, kolik máme skupin. V těchto sadách pravidel pouze upravíme názvy tříd a tabulek z 1M1 na 1M2 a 1M3.

Výsledný skript pro další dvě skupiny bude vypadat takto:

```
#Povoleni upload provozu a zaroven rozrazovani do trid pro druhou skupinu 1Mbit
pass in log quick on $int_if proto icmp from <1M_2> to any queue int_net
pass in log quick on $int_if proto tcp from <1M_2> to any port ssh  queue
int_net
pass in log quick on $int_if proto { tcp,udp } from <1M_2> to any port $int
queue int_net
pass in log quick on $int_if proto { tcp } from <1M_2> to any port { http
https } queue 1M2_n_web
pass in log quick on $int_if proto { tcp } from <1M_2> to any port $data queue
1M2_n_data
pass in log quick on $int_if proto { tcp udp } from <1M_2> to any port $voip
queue voip_net
pass in log quick on $int_if from <1M_2> to any queue 1M2_n_bulk

#Povoleni upload provozu a zaroven rozrazovani do trid pro treti skupinu 1Mbit
pass in log quick on $int_if proto icmp from <1M_3> to any queue int_net
pass in log quick on $int_if proto tcp from <1M_3> to any port ssh  queue
```

```
int_net
pass in log quick on $int_if proto { tcp,udp } from <1M_3> to any port $int
queue int_net
pass in log quick on $int_if proto { tcp } from <1M_3> to any port { http
https } queue 1M3_n_web
pass in log quick on $int_if proto { tcp } from <1M_3> to any port $data queue
1M3_n_data
pass in log quick on $int_if proto { tcp udp } from <1M_3> to any port $voip
queue voip_net
pass in log quick on $int_if from <1M_3> to any queue 1M3_n_bulk
```

5.1.2 Filtrování download

Tímto základním zápisem rozřadíme provoz pro 3 skupiny o jedné rychlosti, ale pouze na interním rozhraní a pouze provoz který jde od uživatelů směrem do internetu – upload. V případě že chceme takto řídit i provoz který jde směrem k uživatelům – download – musíme tuto sadu skriptů upravit. Na jednom rozhraní můžeme mít totiž pouze jednu třídu pomocí ALTQ a v případě že by jsme i provoz jdoucí jiným směrem řadily do těchto tříd, snížili by jsme si celkovou propustnost na polovinu. Naneštěstí díky použití NATu, nemůžeme použít pro markování provozu externí rozhraní. Použijeme k tomu drobného triku, kterým provoz řazený do front na externím rozhraní, budeme markovat na rozhraním interním. Pro řazení downloadu použijeme dalšího rozhraní, na kterém budeme mít pomocí ALTQ navěšeny další třídy se stejnou strukturou. Tato struktura bude stejná základním členění tříd, musí však mít rozdílné pojmenování. Já jsem zvolil 2 řídící pojmenování – výše zmíněné net (pro provoz směrem do internetu) a druhé sit (pro provoz směrem do vnitřní sítě).

V našem případě pak bude skript pro rozřazení pro všechny 3 skupiny vypadat následovně:

```
pass out quick on $int_if proto icmp from any to <1M_1> queue int_sit
pass out quick on $int_if proto tcp from any port ssh to <1M_1> queue int_sit
pass out quick on $int_if proto { tcp,udp } from any port $int to <1M_1> queue
int_sit
pass out quick on $int_if proto { tcp } from any port { http https} to <1M_1>
queue 1M1_s_web
pass out quick on $int_if proto { tcp } from any port $data to <1M_1> queue
1M1_s_data
pass out quick on $int_if proto { tcp udp } from any port $voip to <1M_1> queue
voip_sit
pass out quick on $int_if from any to <1M_1> queue 1M1_s_bulk

pass out quick on $int_if proto icmp from any to <1M_2> queue int_sit
pass out quick on $int_if proto tcp from any port ssh to <1M_2> queue int_sit
pass out quick on $int_if proto { tcp,udp } from any port $int to <1M_2> queue
int_sit
pass out quick on $int_if proto { tcp } from any port { http https} to <1M_2>
queue 1M1_s_web
pass out quick on $int_if proto { tcp } from any port $data to <1M_2> queue
1M2_s_data
pass out quick on $int_if proto { tcp udp } from any port $voip to <1M_2> queue
```

```
voip_sit
pass out quick on $int_if from any to <1M_2> queue 1M2_s_bulk


pass out quick on $int_if proto icmp from any to <1M_3> queue int_sit
pass out quick on $int_if proto tcp from any port ssh to <1M_3> queue int_sit
pass out quick on $int_if proto { tcp,udp } from any port $int to <1M_3> queue
int_sit
pass out quick on $int_if proto { tcp } from any port { http https} to <1M_3>
queue 1M1_s_web
pass out quick on $int_if proto { tcp } from any port $data to <1M_3> queue
1M3_s_data
pass out quick on $int_if proto { tcp udp } from any port $voip to <1M_3> queue
voip_sit
pass out quick on $int_if from any to <1M_3> queue 1M3_s_bulk
```

Při bližším prozkoumání těchto skriptů si všimněme hlavního rozdílu pro samotné markování provozu. Markování totiž provádíme na jednom rozhraní, proto pro samotné odlišení příchozího provozu (download) použijeme OUT tok na interním rozhraní. Z pohledu uživatele který má pracovní stanici za routerem je odchozí provoz na interním rozhraní routeru příchozím provozem pro uživatele, proto jej budeme řadit do front které řídí datový tok dovnitř sítě. Naopak u uploadu tento provoz markujeme na rozhraní s direktivou IN.

5.2 Nastavení shapingu pro tranzitní router

Již podle teoretické části této bakalářské práce víme, že pro shaping tranzitního provozu nám stačí jednodušší rozvržení tříd. Toto jednodušší rozvržení se samozřejmě promítne i v praktickém zápisu skriptů.

V našem případě nastavíme shaping pro router u kterého chceme u jednoho rozhraní použít rychlost 18Mbit. V případě použití sheduleru PRIQ nám umožní využít celou dostupnou šířku pásma. Celé využití potom závisí pouze na určení priorit jednotlivých paketů. Bude vytvořeno 7 skupin priorit. Scheduler PRIQ však umožňuje využít 16 priorit – 0-15, kde 15 je priorita nejvyšší, 0 nejnižší.

Rozvržení je následující:

- `Priority 15` – nejvyšší prioritu bude mít provoz SSH. Je to z důvodu vzdálené obsluhy routerů, které musíme spravovat i při maximálním možném provozu a toto nastavení umožní protokolu SSH pracovat téměř za jakékoliv situace.
- `Priority 13` – voip dostatečně vysokou prioritu nastavíme pro provoz internetové telefonie
- `Priority 11` – priorita třídy int – do této třídy zahrneme realtime provoz podle vlastního uvážení – nejčastěji síťové hry a ostaní realtime aplikace, které uživatelé vyžadují, ale také DNS dotazy z důvodu urychlení připojování k internetovým serverům
- `Priority 9` – třída pro mailový provoz – SMTP, POP3, IMAP, atd
- `Priority 8` – třída pro webový provoz – protokoly HTTP a HTTPS
- `Priority 3` – třída pro námi určený datový provoz kterému stačí nižší priorita vyřízení požadavků - typicky jde o provoz typu FTP
- `Priority 1` – bulk třída – do této třídy budeme řadit veškerý provoz, který nevyhovuje předešlým skupinám. Do této třídy spadnou i díky tisícům otevřených spojení na různých portech aplikace typu P2P. Díky tomu, nám tyto „problémové“ aplikace svou podstatou neucpou linku. Budou odbaveny až po vyřízení všech předchozích požadavků. Navíc tuto třídu nastavíme jako

default, tzn. veškerý i jinak nezařazený provoz bude automaticky zařazen do této třídy.

V našem případě tedy nadefinujeme makra, které použijeme pro rozřazení druhů provozu. Veškeré následující ukázky používají stejné příkazy nastavení, které jsem popsal v předchozí kapitole týkající se shapingu pro hraniční router, proto se v této části popisem příkazů nebudu již dále zabývat a ukážeme si již samotné příklady nastavení.

Definice maker

```
mail="{ smtp smtps pop3 pop3s imap imaps }"
data= "{ ftp ftps ftp-data ftps-data  }"
voip="{ 5060 }"
int="{ 27000:27390 16567 10000:10020 3784 3788 50050 dns }"
```

5.2.1 Nastavení shapingu na rozhraní pomocí ALTQ

Vytvoření hlavní root třídy pro scheduler priq:

```
altq on $iface0 priq bandwidth 18Mb queue { iface0_ssh, iface0_voip, iface0_int,
iface0_web, iface0_mail, iface0_data, iface0_bulk }
    queue iface0_ssh priority 15 priq(red ecn)
    queue iface0_voip priority 13 priq
    queue iface0_int priority 11 priq
    queue iface0_mail priority 9 priq(red ecn)
    queue iface0_web priority 8 priq(red ecn)
    queue iface0_data priority 3 priq(red ecn)
    queue iface0_bulk priority 1 priq(red ecn default)
```

Tímto jsme vytvořili základní strukturu jednotlivých tříd. Máme vytvořenu hlavní root třídu o velikosti 18Mbit, která obsahuje fronty `iface0_ssh`, `iface0_voip`, `iface0_int`, `iface0_web`, `iface0_mail`, `iface0_data`, `iface0_bulk` . Tyto fronty již nemají (v případě scheduleru PRIQ ani nemohou) dále nastavenu rychlost, protože využívají celé pásmo nastavené hlavní třídou. Liší se pouze nastavením priorit.

Může proto klidně dojít k situaci, že například skupina bulk, která má nejnižší prioritu, může v případě nulového provozu ostatních skupin využít celé pásmo 18Mbit. K tomuto jevu v reálném provozu také dochází – jedná se hlavně o noční hodiny, kdy má drtivá většina uživatelů zapnuto stahování pomocí P2P sítí a ostatní provoz je využit minimálně.

U všech skupin kromě voip a int se navíc používá algoritmus RED. U skupin voip a int se nepoužívá kvůli předpokládanému převládajícímu UDP provozu, pro který tato volba nemá smysl.

5.2.2 Rozřazování provozu do jednotlivých tříd

Pro markování a rozřazení paketů nám v tomto případě stačí pro každé rozhraní pouze nastavení podobné tomuto:

```
pass out quick on $iface0 inet proto tcp from any port ssh to any queue
iface0_ssh
pass out quick on $iface0 inet proto tcp from any to any port ssh queue
```

```
iface0_ssh

pass out quick on $iface0 inet proto udp from any port $voip to any queue
iface0_voip

pass out quick on $iface0 inet proto udp from any to any port $voip queue
iface0_voip

pass out quick on $iface0 inet proto { tcp,udp } from any port { http,https } to
any queue iface0_web

pass out quick on $iface0 inet proto { tcp,udp } from any to any port
{ http,https } queue iface0_web

pass out quick on $iface0 inet proto { tcp,udp } from any port $mail to any
queue iface0_mail

pass out quick on $iface0 inet proto { tcp,udp } from any to any port $mail
queue iface0_mail

pass out quick on $iface0 inet proto { tcp,udp } from any port $data to any
queue iface0_data

pass out quick on $iface0 inet proto { tcp,udp } from any to any port $data
queue iface0_data

pass out on $iface0 from any to any queue iface0_bulk
```

Princip rozřazení je stejný jako v případě hraničního shaperu. Každá třída se rozřadí do jí odpovídající fronty a opět třída bulk slouží k zachycení veškerého provozu, který není zachycen dříve. V tomto skriptu je navíc použito makro $iface0 díky kterému můžeme jednoduše změnit rozhraní na kterém chceme shaping použít.

Obdobně pro ostatní rozhraní routeru použijeme tento skript. Stačí nám pouze změnit název rozhraní, buď makrem, nebo můžeme použít přímo jméno rozhraní. V tomto případě již není nutné dělit provoz na odchozí a příchozí. Podle pravidla, že síťový provoz můžeme ovlivnit pouze v odchozím směru, na jednotlivých rozhraních routeru zprovozníme frontování pomocí ALTQ a zároveň řadíme pouze odchozí směr do front. Vycházím z předpokladu, že tranzitní router není koncové zařízení na kterém chceme tyto služby zaručit a proto vždy jakýkoliv provoz, který máme na vstupu některého rozhraní, budeme mít vždy i na výstupu jiného rozhraní. Díky tomu, máme tento provoz vždy řízen alespoň na jednom rozhraní, což již zajistí funkčnost tohoto skriptu. Zároveň tímto docílíme pro koncového uživatele dostatečnou transparentnost tohoto řízení toku, protože na svém koncovém zařízení nemusí již nic nastavovat.

Skript pro tranzitní shaping je vhodné používat v LAN – nebo v sítích, kde nepotřebujeme přidělovat pásmo dané velikosti pro uživatele. Z nastavení je patrné, že tento skript přiděluje pásmo pouze na základě protokolu a nejde jej řídit na základě uživatelů.

Výpis plného skriptu pro tranzitní shaping je uveden v dodatku.

6 ZÁVĚR

Nastíněné řešení použité v této bakalářské práci je fungující řešení. Jeho výhodou je kromě samotné funkčnosti také použití čistě nekomerčních nástrojů. Jeho nevýhodou naopak to, že je postaveno na jednom typu operačního systému. Pokud budujeme novou síť a přistoupíme na fakt, aby veškeré routery použité v této síti splňovali požadavky dané touto bakalářskou prací (tj. operační systém z rodiny BSD, PF firewall. ALTQ), potom jednoduchou úpravou předložených skriptů vytvoříme robustní řešení, které se velice dobře vyrovná s danými typy provozů a také velkého zatížení sítě.

Navíc díky použití maker umožňuje jednoduché doplnění priority nových služeb, pouhým doplněním čísla portu. V případě že potřebujeme vytvořit novou třídu priorit, lze to zařídit jednoduchou úpravou stávajících skriptů – buď doplněním požadované služby do makra, nebo definice nové prioritní třídy. Všechny skripty jsou vytvořeny s dostatečnou rezervou, pro definici dalších tříd (týká se hlavně priorit, kde nejsou vyčerpány všechny možné třídy).

Vzhledem k probíhajícím portacím firewallu PF na ostatní operační systémy je reálné, že v průběhu času bude možné toto řešení použít i na síti, která bude více multiplatformní. V současné době je tato multiplatformnost omezena na systémy BSD. Je samozřejmě možná i komunikace s routery na jiných systémech, nicméně nejlepších výsledků jsem v reálném provozu dosáhl za použití BSD a PF na všech routerech. Těchto routerů je v současné době ve firmě ITSolutions cca 30.

Toto řešení snížilo ztrátovost na linkách způsobenou zahlcením současného provozu z několika zdrojů a tím zvýšilo celkovou propustnost. Zároveň s tím díky fungující priorizaci provozu, která byla zvolená na základě zkušeností a požadavků jednotlivých klientů, je zajištěna velice dobrá funkčnost realtime aplikací. Před použitím těchto skriptů se při plném vytížením linky prodleva mohla pohybovat i v řádech sekund, nicméně po nasazení se i při plném vytížení linky pohybuje v řádech desítek milisekund. V tomto případě jde o jednoduše měřitelný rozdíl, jehož zlepšení je po nasazení vidět okamžitě.

Při nasazení tohoto řešení je možné vybudovat v podstatě libovolně rozsáhlou síť. Navíc díky dalším pokročilým funkcím (jako je failover řešení pomocí CARP, rozložení zátěže a dalších) je možné dosáhnout robustnosti a spolehlivosti high-end řešení renomovaných firem.

7 SEZNAM POUŽITÉ LITERATURY

7.1 Monografie

[1] MOCK, Jim. *FreeBSD* . Martin Novák. [s.l.] : Neocortex s.r.o., 2002. 540 s. ISBN 80-86330-07-9.

[2] Kolektiv autorů. *Linux Dokumentační projekt* . 3. aktualiz. vyd. [s.l.] : Computer Press, 2003. 1016 s. ISBN 80-7226-761-2.

[3] LUCAS, Michael. *FreeBSD : Podrobný průvodce síťovým operačním systémem*. [s.l.] : [s.n.], 2003. 642 s. ISBN 80-7226-795-7.

[4] SHAH, Steve, SOYINKA, Wale. *Administrace systému Linux*. 4. aktualiz. vyd. [s.l.] : [s.n.], 2007. 428 s. ISBN 9788024716947.

7.2 WWW stránky

[5] *Www.openbsd.org* [online]. 1996-2008 [cit. 2008-05-03]. Dostupný z WWW: <www.openbsd.org>.

[6] *Www.FreeBSD.cz* [online]. 1995-2008 [cit. 2008-04-27]. # Text v češtině a angličtině. Dostupný z WWW: <www.freebsd.cz>.

[7] *PF: The OpenBSD Packet Filter* [online]. [1996-] , 2003/11/10 [cit. 2008-05-05]. Dostupný z WWW: <http://www.at.openbsd.org/faq/pf/cs/index.html>.

[8] *Hierarchical Packet Schedulers* [online]. 1999 [cit. 2008-05-05]. Anglický. Dostupný z WWW: <http://www.cs.cmu.edu/~hzhang/HFSC/main.html>.

[9] HANSTEEN, Peter N. M.. *Firewalling with OpenBSD\'s PF packet filter* [online]. 2005-2008 [cit. 2008-04-15]. Anglický. Dostupný z WWW: <http://home.nuug.no/~peter/pf/en/>.

[10] *Software : ALTQ* [online]. 2004 [cit. 2008-04-15]. Anglický. Dostupný z WWW: <http://www.sonycsl.co.jp/~kjc/software.html>.

[11] *Technická zpráva - QoS v Linuxu : Technická zpráva CESNETu číslo 20/2001* [online]. 1996-2008 [cit. 2008-04-15]. Český. Dostupný z WWW: <http://www.cesnet.cz/doc/techzpravy/2001/20/>.

[12] BARIENČÍK, Jan. *Klasifikace provozu* [online]. 2006 [cit. 2008-04-15]. Český. Dostupný z WWW: <http://www.fi.muni.cz/~kas/p090/referaty/2006-jaro/ct/xbarienc-qos.html>.

[13] *OpenBSD-Firewall Packet Sceheduling Analysis for IP Telephony QoS : Part 2: Design and Simulation Implementation* [online]. 2008 [cit. 2008-04-15]. Anglický. Dostupný z WWW: <http://my.opera.com/blu3c4t/blog/2008/01/18/openbsd-firewall-packet-sceheduling-analysis-for-ip-telephony-qos-part-2-desig>.

[14] *QoS - voip-info.org* [online]. 2003-2008 [cit. 2008-04-05]. Anglický. Dostupný z WWW: <http://www.voip-info.org/wiki/view/QoS>.

8 PŘÍLOHY

8.1 Seznam příloh

pf.conf.gw – skript nastavení shapování pro hraniční router

pf.conf.tranzit – skript nastavení shapování pro tranzitní router

8.2 Výpis skriptu pro shaping na hraničním routeru

Tento skript je z důvodu jeho velikosti přiložen pouze v elektronické podobě.

8.3 Plný výpis skriptu pro tranzitní shaping

Tento skript je převzat routeru využívající 5 síťových rozhraní. Výpis skriptu pf.conf-tranzit:

```
iface0="ath0"
iface1="wi0"
iface2="wi1"
iface3="wi2"
iface4="wi3"
iface5="rl0"

mail="{ smtp smtps pop3 pop3s imap imaps }"
data= "{ ftp ftps ftp-data ftps-data  }"
voip="{ 5060 }"
int="{ 27000:27390 16567 10000:10020 3784 3788 50050 }"

set optimization aggressive
scrub in all no-df random-id fragment reassemble
scrub out all no-df random-id fragment reassemble

altq on $iface0 priq bandwidth 18Mb queue { iface0_ssh, iface0_voip, iface0_int, iface0_web,
iface0_mail, iface0_data, iface0_bulk }
      queue iface0_ssh priority 15 priq(red ecn)
      queue iface0_voip priority 13 priq
      queue iface0_int priority 11 priq
      queue iface0_mail priority 9 priq(red ecn)
      queue iface0_web priority 8 priq(red ecn)
      queue iface0_data priority 3 priq(red ecn)
      queue iface0_bulk priority 1 priq(red ecn default)

altq on $iface1 priq bandwidth 3Mb queue { iface1_ssh, iface1_voip, iface1_int, iface1_web,
iface1_mail, iface1_data, iface1_bulk }
      queue iface1_ssh priority 15 priq(red ecn)
      queue iface1_voip priority 13 priq
```

```
       queue iface1_int priority 11 priq
       queue iface1_mail priority 9 priq(red ecn)
       queue iface1_web priority 8 priq(red ecn)
       queue iface1_data priority 3 priq(red ecn)
       queue iface1_bulk priority 1 priq(red ecn default)

   altq on $iface2 priq bandwidth 3Mb queue { iface2_ssh, iface2_voip, iface2_int, iface2_web,
iface2_mail, iface2_data, iface2_bulk }
       queue iface2_ssh priority 15 priq(red ecn)
       queue iface2_voip priority 13 priq
       queue iface2_int priority 11 priq
       queue iface2_mail priority 9 priq(red ecn)
       queue iface2_web priority 8 priq(red ecn)
       queue iface2_data priority 3 priq(red ecn)
       queue iface2_bulk priority 1 priq(red ecn default)

   altq on $iface3 priq bandwidth 3Mb queue { iface3_ssh, iface3_voip, iface3_int, iface3_web,
iface3_mail, iface3_data, iface3_bulk }
       queue iface3_ssh priority 15 priq(red)
       queue iface3_voip priority 13 priq
       queue iface3_int priority 11 priq
       queue iface3_mail priority 9 priq(red)
       queue iface3_web priority 8 priq(red)
       queue iface3_data priority 3 priq(red)
       queue iface3_bulk priority 1 priq(red default)

   altq on $iface4 priq bandwidth 3Mb queue { iface4_ssh, iface4_voip, iface4_int, iface4_web,
iface4_mail, iface4_data, iface4_bulk }
       queue iface4_ssh priority 15 priq(red)
       queue iface4_voip priority 13 priq
       queue iface4_int priority 11 priq
       queue iface4_mail priority 9 priq(red)
       queue iface4_web priority 8 priq(red)
       queue iface4_data priority 3 priq(red)
       queue iface4_bulk priority 1 priq(red default)

   altq on $iface5 priq bandwidth 80Mb queue { iface5_ssh, iface5_voip, iface5_int, iface5_web,
iface5_mail, iface5_data, iface5_bulk }
       queue iface5_ssh priority 15 priq(red)
       queue iface5_voip priority 13 priq
       queue iface5_int priority 11 priq
       queue iface5_mail priority 9 priq(red)
       queue iface5_web priority 8 priq(red)
       queue iface5_data priority 3 priq(red)
       queue iface5_bulk priority 1 priq(red default)


   pass out quick on $iface0 inet proto tcp from any port ssh to any queue iface0_ssh
   pass out quick on $iface0 inet proto tcp from any to any port ssh queue iface0_ssh
```

```
   pass out quick on $iface0 inet proto udp from any port $voip to any queue iface0_voip
   pass out quick on $iface0 inet proto udp from any to any port $voip queue iface0_voip
   pass out quick on $iface0 inet proto { tcp,udp } from any port { http,https } to any queue
iface0_web
   pass out quick on $iface0 inet proto { tcp,udp } from any to any port { http,https } queue
iface0_web
   pass out quick on $iface0 inet proto { tcp,udp } from any port $mail to any queue iface0_mail
   pass out quick on $iface0 inet proto { tcp,udp } from any to any port $mail queue iface0_mail
   pass out quick on $iface0 inet proto { tcp,udp } from any port $data to any queue iface0_data
   pass out quick on $iface0 inet proto { tcp,udp } from any to any port $data queue iface0_data
   pass out on $iface0 from any to any queue iface0_bulk

   pass out quick on $iface1 inet proto tcp from any to any port ssh queue iface1_ssh
   pass out quick on $iface1 inet proto tcp from any port ssh to any queue iface1_ssh
   pass out quick on $iface1 inet proto tcp from any to any port $voip queue iface1_voip
   pass out quick on $iface1 inet proto tcp from any port $voip to any queue iface1_voip
   pass out quick on $iface1 inet proto icmp from any to any queue iface1_int
   pass out quick on $iface1 inet proto udp from any to any port $int queue iface1_int
   pass out quick on $iface1 inet proto udp from any port $int to any queue iface1_int
   pass out quick on $iface1 inet proto { tcp,udp } from any to any port { http,https } queue
iface1_web
   pass out quick on $iface1 inet proto { tcp,udp } from any port { http,https } to any  queue
iface1_web
   pass out quick on $iface1 inet proto { tcp,udp } from any to any port $mail queue iface1_mail
   pass out quick on $iface1 inet proto { tcp,udp } from any port $mail to any queue iface1_mail
   pass out quick on $iface1 inet proto { tcp,udp } from any to any port $data queue iface1_data
   pass out quick on $iface1 inet proto { tcp,udp } from any port $data to any queue iface1_data
   pass out on $iface1 from any to any queue iface1_bulk

   pass out quick on $iface2 inet proto tcp from any to any port ssh queue iface2_ssh
   pass out quick on $iface2 inet proto tcp from any port ssh to any queue iface2_ssh
   pass out quick on $iface2 inet proto tcp from any to any port $voip queue iface2_voip
   pass out quick on $iface2 inet proto tcp from any port $voip to any queue iface2_voip
   pass out quick on $iface2 inet proto icmp from any to any queue iface2_int
   pass out quick on $iface2 inet proto udp from any to any port $int queue iface2_int
   pass out quick on $iface2 inet proto udp from any port $int to any queue iface2_int
   pass out quick on $iface2 inet proto { tcp,udp } from any to any port { http,https } queue
iface2_web
   pass out quick on $iface2 inet proto { tcp,udp } from any port { http,https } to any  queue
iface2_web
   pass out quick on $iface2 inet proto { tcp,udp } from any to any port $mail queue iface2_mail
   pass out quick on $iface2 inet proto { tcp,udp } from any port $mail to any queue iface2_mail
   pass out quick on $iface2 inet proto { tcp,udp } from any to any port $data queue iface2_data
   pass out quick on $iface2 inet proto { tcp,udp } from any port $data to any queue iface2_data
   pass out on $iface2 from any to any queue iface2_bulk

   pass out quick on $iface3 inet proto tcp from any to any port ssh queue iface3_ssh
   pass out quick on $iface3 inet proto tcp from any port ssh to any queue iface3_ssh
   pass out quick on $iface3 inet proto tcp from any to any port $voip queue iface3_voip
   pass out quick on $iface3 inet proto tcp from any port $voip to any queue iface3_voip
```

```
   pass out quick on $iface3 inet proto icmp from any to any queue iface3_int
   pass out quick on $iface3 inet proto udp from any to any port $int queue iface3_int
   pass out quick on $iface3 inet proto udp from any port $int to any queue iface3_int
   pass out quick on $iface3 inet proto { tcp,udp } from any to any port { http,https } queue
iface3_web
   pass out quick on $iface3 inet proto { tcp,udp } from any port { http,https } to any  queue
iface3_web
   pass out quick on $iface3 inet proto { tcp,udp } from any to any port $mail queue iface3_mail
   pass out quick on $iface3 inet proto { tcp,udp } from any port $mail to any queue iface3_mail
   pass out quick on $iface3 inet proto { tcp,udp } from any to any port $data queue iface3_data
   pass out quick on $iface3 inet proto { tcp,udp } from any port $data to any queue iface3_data
   pass out on $iface3 from any to any queue iface3_bulk


   pass out quick on $iface4 inet proto tcp from any to any port ssh queue iface4_ssh
   pass out quick on $iface4 inet proto tcp from any port ssh to any queue iface4_ssh
   pass out quick on $iface4 inet proto tcp from any to any port $voip queue iface4_voip
   pass out quick on $iface4 inet proto tcp from any port $voip to any queue iface4_voip
   pass out quick on $iface4 inet proto icmp from any to any queue iface4_int
   pass out quick on $iface4 inet proto udp from any to any port $int queue iface4_int
   pass out quick on $iface4 inet proto udp from any port $int to any queue iface4_int
   pass out quick on $iface4 inet proto { tcp,udp } from any to any port { http,https } queue
iface4_web
   pass out quick on $iface4 inet proto { tcp,udp } from any port { http,https } to any  queue
iface4_web
   pass out quick on $iface4 inet proto { tcp,udp } from any to any port $mail queue iface4_mail
   pass out quick on $iface4 inet proto { tcp,udp } from any port $mail to any queue iface4_mail
   pass out quick on $iface4 inet proto { tcp,udp } from any to any port $data queue iface4_data
   pass out quick on $iface4 inet proto { tcp,udp } from any port $data to any queue iface4_data
   pass out on $iface4 from any to any queue iface4_bulk


   pass out quick on $iface5 inet proto tcp from any to any port ssh queue iface5_ssh
   pass out quick on $iface5 inet proto tcp from any port ssh to any queue iface5_ssh
   pass out quick on $iface5 inet proto tcp from any to any port $voip queue iface5_voip
   pass out quick on $iface5 inet proto tcp from any port $voip to any queue iface5_voip
   pass out quick on $iface5 inet proto icmp from any to any queue iface5_int
   pass out quick on $iface5 inet proto udp from any to any port $int queue iface5_int
   pass out quick on $iface5 inet proto udp from any port $int to any queue iface5_int
   pass out quick on $iface5 inet proto { tcp,udp } from any to any port { http,https } queue
iface5_web
   pass out quick on $iface5 inet proto { tcp,udp } from any port { http,https } to any  queue
iface5_web
   pass out quick on $iface5 inet proto { tcp,udp } from any to any port $mail queue iface5_mail
   pass out quick on $iface5 inet proto { tcp,udp } from any port $mail to any queue iface5_mail
   pass out quick on $iface5 inet proto { tcp,udp } from any to any port $data queue iface5_data
   pass out quick on $iface5 inet proto { tcp,udp } from any port $data to any queue iface5_data
   pass out on $iface5 from any to any queue iface5_bulk
```