MASARYKOVA UNIVERSITA
FAKULTA INFORMATIKY

Diplomová práce

Kontrola síťového provozu

Řešitel: Jakub Mácha, xmacha@fi.muni.cz

Vedoucí: Mgr. Jan Kasprzak, kas@fi.muni.cz

Brno, duben 2000

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Jakub Mácha

Tímto bych chtěl poděkovat svému vedoucímu Mgr. Janu Kasprzakovi za příkladné vedení a mnoho přínosných rad.

Abstrakt

Tato práce přináší přehled možností kontroly síťového provozu. Představeny budou základní přístupy normované jako Integrated a Differentiated service. Dále bude následovat popis algoritmů, které se používají při kontrole provozu, zvaných také algoritmy řazení do front.

Hlavní téma této práce je kontrola provozu v operačním systému Linux. Tady bude věnována maximální pozornost uživatelskému programu `tc`, jehož popis představuje značnou část následujícího textu.

Na konci kapitol o operačním systému Linux je uvedeno několik konkrétních konfigurací síťového rozhraní. Měřením šířky pásma a doby odezvy jsou ukázány některé aspekty kontroly provozu v paketových sítích.

Další část si všímá implementace kontroly provozu ve směrovačích značky Cisco.

Klíčová slova

kontrola provozu, kvalita služby, Integrated Service, Differentiated Service, Linux, řazení do front, klasifikace provozu, omezování provozu, Cisco, CBQ

Zadání

Seznamte se s metodami, které jsou v současných směrovacích systémech dostupné pro řízení a diferencování síťového provozu (omezování šířky pásma, klasifikace datagramů, priority provozu a podobně). Srovnejte existující systémy a podejte detailní popis jednoho z nich.

Navrhněte a vyzkoušejte konkrétní konfiguraci řízení provozu na směrovači s operačním systémem Linux, zaměřenou na zlepšení kvality interaktivního provozu nad velmi zatíženou vytáčenou linkou. Ověřte tuto konfiguraci v praxi a pokuste se měřením prokázat, je-li tato konfigurace výhodnější než konfigurace bez řízení provozu.

Obsah

ABSTRAKT ..... **III**

KLÍČOVÁ SLOVA ..... **III**

ZADÁNÍ ..... **III**

OBSAH **1**

1 ÚVOD **3**

1.1 KVALITA SLUŽBY 3
1.2 KONTROLA PROVOZU 3
1.3 NORMY DIFFSERV A INTSERV 4
1.3.1 DiffServ 4
1.3.2 IntServ 5
1.3.3 RSVP 5
1.4 OMEZOVÁNÍ ŠÍŘKY PÁSMA (TRAFFIC SHAPING) 5
1.4.1 RED (Random Early Detection) 6
1.4.2 TBF (Token Bucket Filter) 7
1.4.3 SFQ (Stochastic Fairness Queuing) 7
1.4.4 CBQ (Class Based Queuing) 7

2 IMPLEMENTACE V OPERAČNÍM SYSTÉMU LINUX **8**

2.1 IMPLEMENTACE V JÁDŘE SYSTÉMU 8
2.1.1 Struktury 9
2.1.2 Class Based Queue 11
2.2 UŽIVATELSKÉ PROGRAMY 12
2.2.1 Traffic shaper 12
2.2.2 Program tc (traffic controller) 13
2.3 PROGRAMOVÉ ROZHRANÍ A DALŠÍ VÝVOJ 24
2.3.1 Program tc 24
2.3.2 Jádro systému 24

3 SMĚROVAČE CISCO **25**

3.1 ŘAZENÍ DO FRONT 25
3.1.1 Priority Queuing 25
3.1.2 Custom Queuing 25
3.1.3 Weighted Round Robin 25
3.1.4 Další algoritmy 26
3.1.5 Předcházení přetížení – WRED 26
3.2 SIGNALIZAČNÍ TECHNIKY 26

4 SROVNÁNÍ POPISOVANÝCH SYSTÉMŮ **27**

4.1 DALŠÍ IMPLEMENTACE KONTROLY PROVOZU 27

5 EXPERIMENTÁLNÍ KONFIGURACE **28**

5.1 PODMÍNKY MĚŘENÍ 28
5.2 MĚŘÍCÍ NÁSTROJE 28
5.3 KONKRÉTNÍ KONFIGURACE 29
5.3.1 Meření 1 – rozdělování šířky pásma 29
5.3.2 Měření 2 – asynchronní toky 30
5.3.3 Měření 3 – více spojení v jedné třídě 31

5.3.4 Měření 4 – omezování provozu na rychlé lince 32
5.3.5 Měření 5 – omezování provozu algoritmem RED 33
5.3.6 Měření 6 – odezva 33
5.4 VÝSLEDKY MĚŘENÍ 34
6 ZÁVĚR 35
7 LITERATURA 36
7.1 NORMY IETF 36
7.2 NORMY DIFFSERV 36
7.3 NORMY INTSERV 37
7.4 SPECIFIKACE RSVP 37
7.5 DISKUZNÍ SKUPINY 37
8 SLOVNÍK POJMŮ 38
9 SEZNAM OBRÁZKŮ 38
10 SEZNAM TABULEK 38
11 PŘÍLOHY 39
11.1 KONFIGURACE 1 39
11.2 KONFIGURACE 2 40
11.3 KONFIGURACE 3 40
11.4 KONFIGURACE 4 41

1 Úvod

Tato práce se zabývá ovládáním a laděním parametrů síťových rozhraní na stanicích v internetu, respektive v paketových sítích, především na směrovačích. Hlavním cílem je zabezpečit určitou kvalitu služeb zásahem do algoritmů, které plánují odesílání paketů. Hlavní část této práce je věnována operačnímu systému Linux, který v této oblasti nabízí značné možnosti a daleko převyšuje všechny ostatní unixové systémy. Více nabízí jen některé komerční směrovače. V práci bude zmínka o produktech firmy Cisco Systems, které jsou v tomto ohledu podle dostupných informací nejdále.

1.1 Kvalita služby

Internet založený na protokolu IP (Internet Protocol) je velmi jednoduchá struktura. Pakety mají svoji zdrojovou a cílovou adresu a na základě toho mohou projít celou sítí nezávisle na směrovačích a dalších zařízeních. Protokol IP zabezpečuje jen adresování a tím vlastně zaručuje každému paketu nezávislost na ostatních paketech. Tím se může stát, že pakety putují jinými cestami nebo se přímo zdvojují. Směrovače mohou zahodit paket bez jakéhokoliv upozornění odesílateli. Aplikace se musí spoléhat na vyšší vrstvy, typicky na protokol TCP.

Pro běžné aplikace toto až donedávna stačilo. Dnes už jsou ale požadavky na přenos paketů výraznější a jedním z těchto mechanizmů je kvalita služby (quality of service). Obecně se uplatňuje v přenosech multimediálních dat, v IP telefonii a sítích ATM.

Kvalitu služeb bychom mohli definovat následovně: *„Jedná se komplexní opatření, snažící se zaručit koncovému uživateli doručení dat v potřebné kvalitě. Kvalita služby může být definována pomocí několika kritérií: dostupnost (minimalizace času výpadku spojení), chybovost, čas odpovědi, propustnost, ztrátovost paketů, doba vytvoření spojení, rychlost, detekce a oprava chyb. Poskytovatel služby by měl vždy zaručovat určitou kvalitu služeb.“* Toliko definice, v dalším textu se zaměřím jen na některé aspekty kvality služby, které považuji za nejdůležitější a především za dnes v praxi nejsnáze prosaditelné. Bude se jednat o šířku pásma (tedy rychlost přenosu dat) a o latenci neboli zpoždění a ještě jinak o dobu odpovědi.

Šířka pásma znamená rychlost přenosu dat a kritická je především u multimediálních aplikací ale třeba i nejběžnějšího přenosu webových stránek. Jak bude zřejmé dále, není ani tak problém šířku pásma zaručit a přidělit, nýbrž spíše omezit v místě nějakého úzkého hrdla (tzn. místa na přenosové cestě, kde je nutné snížit šířku pásma).

Latence, doba odezvy hraje významnou roli v interaktivních službách. Jedná se o dobu mezi odesláním a přijetím paketu. Jedná se o zpoždění na výstupním zařízení a dále na celé přenosové cestě. Latence u síťového zařízení (směrovače) je jeho příspěvek do celkové latence paketu jím procházejícího, tedy doba mezi přijetím a odesláním paketu. U přenosu videa nám vůbec nevadí, že jej dostáváme s několikasekundovým zpožděním. Musíme si ale uvědomit, že interaktivní služby posílají po síti většinou jednotlivě každý znak co nejdříve po stisku klávesy. Jen výjimečně jsou data posílána po celých řádcích. Na většině systémů se problém odezvy řeší přiřazením vyšší priority požadovaným typům dat.

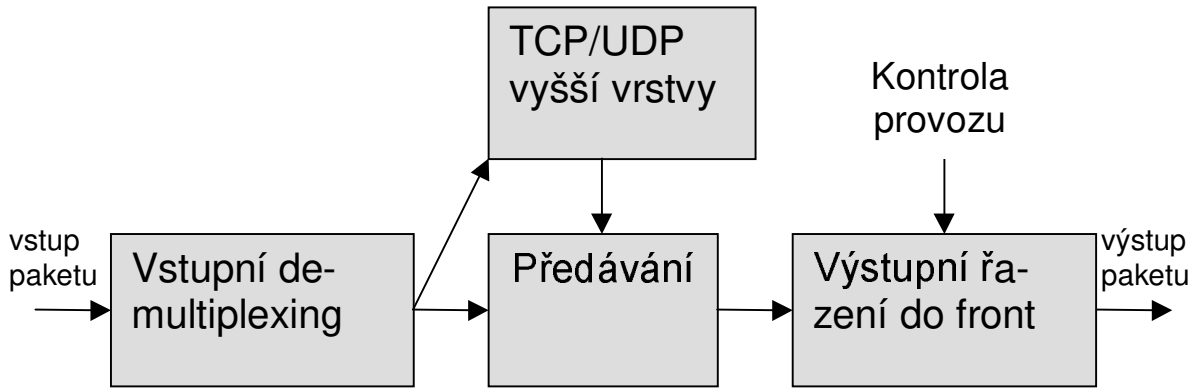
Jako třetí parametr kvality služby je uváděna **změna latence** (angl. jitter), její derivace. Problém nastává v okamžiku, kdy v proudu dat dorazí pakety v špatném pořadí. Cílová stanice to okamžitě pozná a dovede zaručit odevzdání vyšší vrstvě ve správném pořadí, nicméně je nutné některé pakety někde pozdržet. Změna latence je tedy kritická v multimediálních přenosech, typicky u přenosu videa.

1.2 Kontrola provozu

Kontrola provozu (angl. traffic control) je základním obecným nástrojem k ovlivnění a dosažení kvality služby. Jedná se vlastně o celou cestu mezi vstupním a výstupním síťovým zařízením. Základní rozhodnutí jsou:

1. Paket je zahozen na vstupním zařízení.
2. Paket je určen pro tento uzel a je předán vyšším vrstvám.

3. Paket je určen k odeslání, vybere s příslušné výstupní zařízení.
4. Paket je zahozen nebo odeslán dle nějakého plánovacího algoritmu.



obrázek 1 – průchod paketu síťovým uzlem

Ovlivňování paketů na vstupním zařízení se děje jen málokdy. Většina systémů to ani neumožňuje, výjimku tvoří směrovače firmy Cisco. Dle mého názoru to nemá ani smysl, lepší by jistě bylo neodeslat paket z předcházejícího uzlu, čímž by se jistě ušetřily síťové zdroje.

K druhému bodu lze říci, že uzly se rozhodují na základně cílové adresy. Nic jim ale nebrání, aby obsloužily i jiné pakety, či přenechaly paket určený pro sebe dalšímu uzlu. To je výhodné například, když více serverů obsluhuje stejnou službu kvůli rozložení zátěže.

Třetí bod je doménou směrování. Tomuto problému se ve své práci nevěnuji, takže jej pro tuto chvíli přeskočíme.

Čtvrtý bod představuje plánování odesílání paketů. V dalším textu se omezíme jen na tento případ. Paket je již přiřazen konkrétnímu síťovému rozhraní a čeká na odeslání v nějaké obecné frontě. Plánovací algoritmus určí, v jakém pořadí budou pakety odeslány, případně které z nich budou zahozeny.

Existuje několik základních plánovacích algoritmů, od nejjednodušší prosté fronty FIFO až po složité hierarchicky uspořádané struktury CBQ (class bassed queuing). Tuto oblast vymezuje

i několik norem a doporučení, nejznámějšími jsou **Differentiated Service (DiffServ neboli DS)** a **Integrated Services (IntServ)**.

1.3 Normy DiffServ a IntServ

1.3.1 DiffServ

Differentiated Service je IETF standard pro malé, přesně definované bloky paketů, v nichž lze rozlišit více typů služeb. Tím poskytují rámec pro využití QoS v datových sítích. Tato úsilí se převážně soustřeďuje na využití pole ToS (Type of Service) v hlavičce IP (verze 4) jako signalizačního mechanizmu QoS. Dále pak směruje k definici agregovaných toků. RFC 2474 definuje, jak přiřadit třídu služby podle pole ToS. DiffServ je zpětně kompatibilní s nynějším (či starým) používáním pole ToS. Norma DiffServ je budována k použití především v prostředí páteřních IP sítí.

Motivací k budování normy DiffServ bylo, aby mohl poskytovatel zabezpečit svým zákazníkům požadovaný typ služby. Zákazník požaduje určitý typ služby v hlavičce každého paketu. Této informaci by měly rozumět všechny uzly na cestě a tím pádem by měly, pokud to je možné, požadovanou kvalitu zajistit. Normy DiffServ jsou popsány v dokumentech RFC [22]–[29].

1.3.2 IntServ

Integrated Services (IntServ) je množina IETF standardů pokrývající, jak aplikační služba definuje své požadavky, jak je tato informace dostupná pro směrovače, jak se přenáší po síti a nakonec jak je požadovaná kvalita služby dostupná. Podle normy IntServ by měl každý síťový element zveřejnit nebo vystavit své možnosti při ovlivňování QoS a své požadavky na QoS. Způsob zveřejnění a exportu těchto informací norma IntServ dále upravuje. Směrovače by měly podle IntServ klasifikovat pakety a udržovat stavové informace o každém aktuálním toku.

Koncové stanice o kvalitu služby dynamicky žádají dle protokolu RSVP (Resource ReSerVation Protocol) a směrovače by měly požadovanou kvalitu zabezpečit.

1.3.3 RSVP

Resource ReSerVation Protocol (RSVP) je standard IETF jako část architektury IntServ. Dovoluje síťovému rozhraní vyjednat určitou QoS nebo nějakou její charakteristiku. Protokol RSVP rozšiřuje architekturu IntServ o podporu přenosu v reálném čase (hlas a video). Pomocí protokolu RSVP si zdrojová aplikace může rezervovat určité parametry na celé přenosové cestě od zdroje k cíli. RSVP je signalizační protokol (podobný protokolu ICMP), který simuluje internetové přenosy jako virtuální spojení. O tomto spojení pak periodicky informuje uzly na celé cestě. Směrovače podporující RSVP pak pomocí těchto informací plánují další přenos. Rezervace šířky pásma může ale být zakázána, především v závislosti na lokálních podmínkách. Protokol RSVP je navržen pro použití ve velkých lokálních sítích, u sítí WAN mohou nastat problémy.

Protokol RSVP se typicky používá v těchto případech:

1. ve směrovačích, aby mohly udržovat informace o tocích
2. příjemce může kontrolovat kvalitu služby
3. jednoduchá kontrola toků dat a celého média
4. použití v IP multicastingu

1.3.3.1 Jak RSVP funguje

- Odesílatel stanoví hranice parametrů kvality služeb, speciálně šířku pásma, odezvu a změnu odezvy. Tyto parametry vloží do zprávy (PATH message) a pošle jej cílovému uzlu. Na cestě směrovače podporující RSVP přidávají k této zprávě své adresy.
- Poté cílový uzel pak pošle stejnou cestou zpět rezervační požadavek (RESV message). Tento požadavek obsahuje požadovanou kvalitu služby a specifikaci paketů (číslo portu, protokol…). Tyto informace dohromady tvoří identifikátor toku.
- Jesliže směrovač dostane rezervační požadavek, musí proběhnout autentifikace. V kladném případě směrovač provede rezervaci prostředků, v opačném případě (selhala autentifikace nebo nejsou dostupné prostředky) upozorní odesílatele.
- Jestliže i poslední směrovač potvrdí požadavek, jsou oba uzly uvědomeny a prostředky jsou rezervovány.

Obě normy DiffServ a IntServ jsou zabudovány v operačním systému Linux, stejně tak i ve směrovačích Cisco.

1.4 Omezování šířky pásma (Traffic Shaping)

Omezení šířky pásma je na první pohled zcela triviální záležitost. Jakmile je dosažena kritická rychlost, je odesílání paketů zastaveno a řízení je předáno jinému typu provozu. V tomto případě jsou neodeslané pakety ponechány ve frontách na přiřazených výstupnímu zařízení. Jestliže jsou fronty naplněny, jsou další pakety zahazovány. Tady nastává problém, zbytečně se zvyšuje ztrátovost paketů.

Ještě větší problémy nastanou ve spojovaných službách jako je TCP spojení. Jestliže do vypršení časového limitu nedojde k potvrzení paketu (ACK), vysílající strana se jej snaží posílat znova. Paket je pak zdvojen.

Řešením těchto problémů jsou fundované algoritmy odesílání paketů. Podrobněji budou popsány algoritmy RED (Random Early Detection), SFQ (Stochastic Fairness Qeueing) a TBF (Token Bucket Filter).

1.4.1 RED (Random Early Detection)

Algoritmus RED je popsán v [3], nastíním jen jeho základní ideje. Při přijetí paketu je vypočítána průměrná délka fronty a uložena do proměnné *avg* následující formulí:

```
avg = (avg * (1 - 1 / 2^n)) + (aktuální_délka_fronty * 1 /
      2^n)
```

Parametr *n* je konfigurovatelný uživatelem. Jestliže je *avg* větší než *horní limit*, paket je zahozen (fronta je stejně už plná). Na druhé straně, jestliže je *avg* menší, než *dolní limit*, je paket zařazen do fronty (je tam zatím dostatek místa). Jestliže se hodnota *avg* nachází mezi oběma limity, je spočítána pravděpodobnost jeho zahození:

```
prob = max_prob*(avg - dolní_limit)/(horní_limit -
      dolní_limit)
```

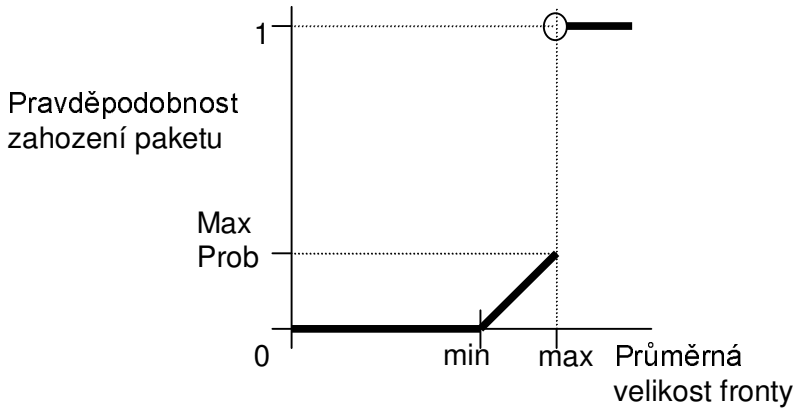
Hodnota *max_prob* je volena malá, obyčejně 0.01 až 0.02.

Algoritmus RED začne pakety zahazovat dříve, než je dosažena maximální kapacita fronty. Pravděpodobnost zahození paketu se zvyšuje úměrně s naplňováním výstupní fronty. Dosažení dolního limitu může být spojeno s upozorněním zdrojové aplikace, aby zpomalila rychlost odesílání dat.

Dolní limit se musí volit velmi uvážlivě. Příliš nízká hodnota snižuje propustnost linky (ztrátovost paketů je příliš vysoká), vysoká hodnota pak zvyšuje latenci. Horní limit by měl být minimálně dvojnásobkem dolního, podle [11] to může býti čtyřnásobek a i více.

Toliko tedy základní verze algoritmu, které je implementována například v Linuxu. Existují
i některé rozšíření. Pod názvem WRED (Weighted RED) se označuje rozšíření RED o podporu priorit. Pakety s vyšší prioritou jsou zahazovány s nižší pravděpodobností. Priorita je určena v hlavičce paketu.

Další možností je DWRED (Distributed WRED), tato verze je implementována ve směrovačích firmy Cisco.



obrázek 2 – algoritmus RED

1.4.2 TBF (Token Bucket Filter)

Idea tohoto algoritmu je jednoduchá. Máme vědro, v jehož dně je díra. Do vědra naléváme vodu a to se může i naplňovat. Dírou vespod totiž voda odtéká konstantní rychlostí bez ohledu na přítok či objem kýblu.

Každých T sekund je generován token (virtuální entita), který je ukádán do fronty. Kapacita fronty je omezená limitem. Příchozímu paketu je přiřazen token. Paket je odeslán v okamžiku odstranění tokenu z fronty. Jestliže je fronta (zásoba tokenů) prázdná, je příchozí paket zahozen. Obecně, při příchodu paketu na síťové rozhraní mohou nastat tyto tři případy:

- Rychlost toku příchozích dat je shodný s danou výstupní rychlostí. To znamená, že počet tokenů ve frontě se nemění, protože stejně rychle tokeny přibývají a ubývají. Jestliže jsou ve frontě nějaké tokeny, je je možné použít v případech chvilkového zvýšení rychlosti.
- Rychlost toku příchozích dat je menší, než výstupní rychlost. Pak se tokeny ve frontě hromadí pro pozdější použití, fronta se muže i úplně naplnit.
- Poslední, rychlost toku příchozích dat je vyšší než výstupní rychlost. V tomto okamžiku tokeny z fronty ubývají. Jakmile je fronta prázdná, je jsou příchozí pakety zahazovány až do okamžiku vygenerování tokenu.

V praxi se algoritmus může nasadit i vícekrát za sebou s různými parametry (hloubka, výstupní rychlost). Tím je možné kontrolovat i další aspekty kvality služeb jako je třeba latence. V komerčních systémech může počet tokenů určovat i platbu za poskytovanou službu či její kvalitu.

Algoritmus TBF je dnes hojně implementován, například v protokolu RSVP a v Linuxu.

1.4.3 SFQ (Stochastic Fairness Queuing)

Tato metoda spočívá v náhledu na přenosy paketů jako na toky. Pakety jsou rozděleny do jednotlivých toků neboli konverzací a ty jsou cyklicky obsluhovány (round robin). Toky mohou být diferencovány například podle zdrojové a cílové adresy, protokolu a portů. Identifikátory toků jsou ukládány do hašovací tabulky a tak se může stát, že dva různé toky budou mít stejný klíč. Kolize se kvůli rychlosti neřeší a tak na tyto dva toky bude nahlíženo jako na jeden. Ke každému toku je pak přiřazena jednoduchá fronta, do které jsou ukládány odchozí pakety.

V tomto systému je možné využít systém priorit, pak jsou některé toky obsluhovány přednostně, algoritmus má pak název **WFQ** (Weighted Fairness Queuing).

Algoritmus SFQ je využíván jako listové řazení do front v hierarchii CBQ. Právě on nejlépe zaručí spravedlivé rozdělení šířky pásma mezi jednotlivé aplikace. Hlavní předností algoritmu SFQ je ale jeho rychlost.

1.4.4 CBQ (Class Based Queuing)

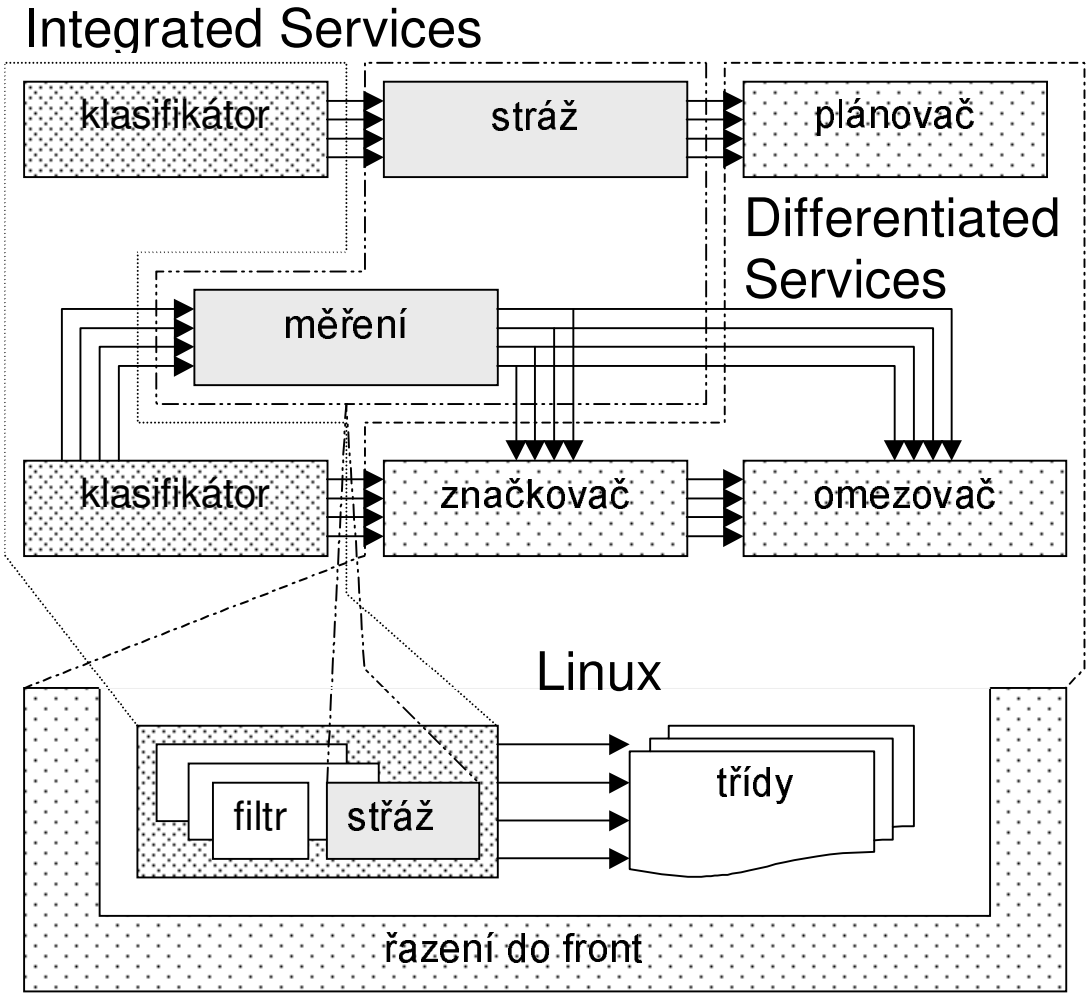
Posledním algoritmem by mohlo být právě řazení do front CBQ. Nejedná se o typický algoritmus omezování šířky pásma, uvádím jej zde jen pro úplnost. Hlavní myšlenka spočívá v třízení (klasifikaci) paketů do tříd. Třídy jsou virtuální entity, které zastupují další algoritmus řazení paketů. Třídy jsou tedy abstrakcemi řazení.

Algoritmus CBQ je detailně popsán v [6] a [7], dále pak v kapitole 2.1.2.

2 Implementace v operačním systému Linux

V této kapitole se na operačním systému Linux pokusím přednést možnosti konfigurace kontroly provozu. Mnoho aspektů je ale obecnějších a shoduje se s teoretickými pracemi v této oblasti. Na případné rozdíly bude upozorněno. Vývoj celé síťové sekce Linuxu řídí Alexej Kuznětcov (`kuznet@ms2.inr.ac.ru`), který je současně autorem většiny programového kódu týkající se kontroly provozu.

Na tomto místě je třeba říci, že části jádra, které se zabývají kontrolou provozu jsou relativně mladé a mají mnoho „dětských chyb“. Některé rutiny dokonce nefungují vůbec. Při jejich kompilaci do jádra je nutné zapnout mimo jiné volbu EXPERIMENTAL OPTIONS.



obrázek 3 – normy IntServ a DiffServ v Linuxu

2.1 Implementace v jádře systému

Implicitně se jádro Linuxu řídí třístupňovým prioritním předáváním založeným na poli TOS (Type Of Service) v hlavičce paketu podle RFC 1349. Tento přístup ale neumožňuje efektivní diferencování služeb, než novější model založený na hierarchické struktuře.

Rozsáhlejší podpora QoS je implementována od verze jádra 2.1.105, dnes jsou už k dispozici jádra daleko novější. Při kompilaci je nutné zahrnout EXPERIMENTAL OPTIONS, dále pak Class Based Quering (CBQ), Token Bucket Flow (TBF), Trafic Shapers, apod. Tři posledně jmenované volby je možné kompilovat jako moduly, což výrazně usnadní práci při ladění a konfiguraci.

2.1.1 Struktury

Nacházíme se v situaci, kdy je paket na základě určitého směrovacího mechanizmu přidělen ke konkrétnímu **síťovému rozhraní**. Každé takové rozhraní je v jádře reprezentováno strukturou `'struct net_device'` (definována v souboru `include/Linux/netdevice.h`). Jednotlivé rozhraní jsou spojena do spojového seznamu, prvním prvkem je loopback neboli zpětná smyčka.

Dalšími základními strukturami jsou řazení do front, třídy a filtry.

2.1.1.1 Řazení do front (queuing discipline)

Každé síťové rozhraní má přiřazenu tzv. „queuing discipline“ (zde použiji termín **řazení do front**). V jádře je tato struktura pojmenovaná `'struct Qdisc'` a definována v souboru `include/net/pkt_sched.h`. Řazení do front obsahuje několik rutin, které jsou uloženy ve struktuře `'struct Qdisc_opt'` (v `include/net/pkt_sched.h`). Tyto struktury jsou opět spojeny do seznamu, který je inicializován při startu systému funkcí `pktsched_init()` (definovanou v `net/sched/sch_api.c`). Další informace o řazení do front jsou v struktuře `'struct cbq_sched_data'`

Rozlišujeme několik základních typů řazení do front:

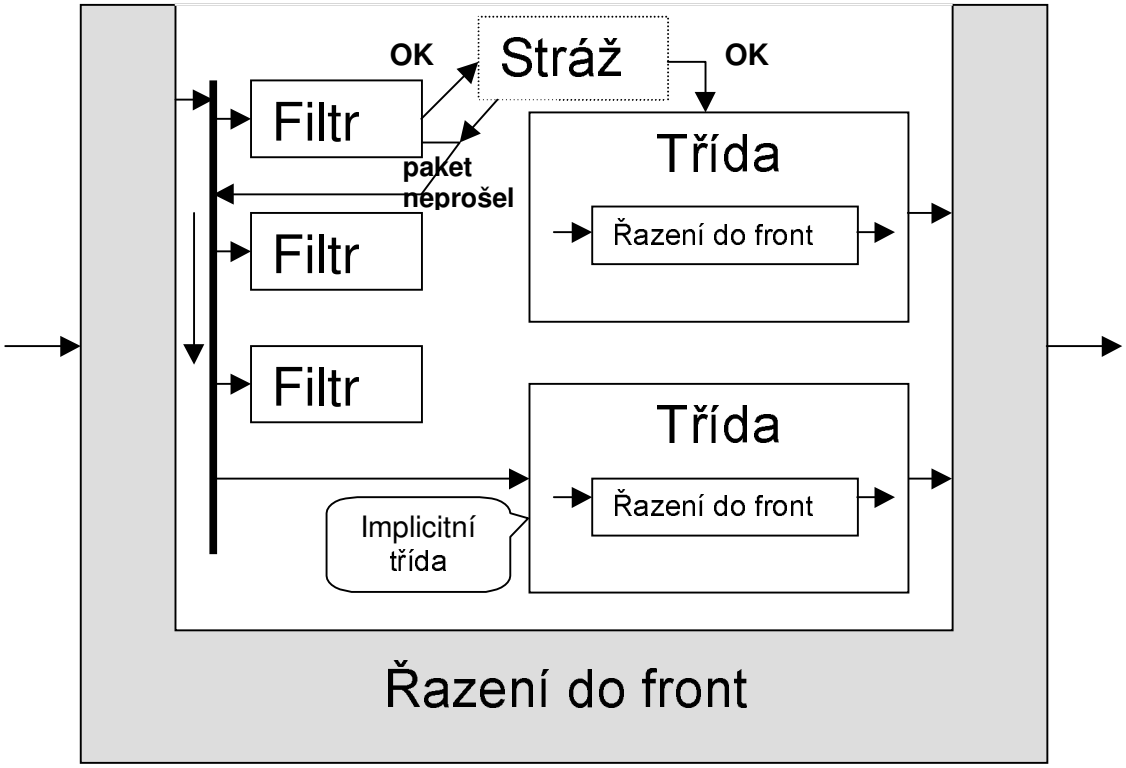
- CBQ, (Class Based Queue)
- FIFO, (First In First Out)
- Prioritní fronta
- TBF (Token Bucket Flow)
- RED (Random Early Detection)
- CSZ (Clark-Shenker-Zhang)
- SFQ (Stochastic Fairness Queuing)
- ATM (Asynchronous Transfer Mode)

a další

V dalším textu bude řeč především o CBQ, což je nejobecnější řazení do front, poněvadž může v sobě zahrnout všechny další, respektive může každému typu služby přiřadit jiný mechanizmus odesílání.

Další informace o už typově rozdělených řazeních do front jsou obsaženy ve strukturách `'struct cbq_sched_data'`, `'struct tbf_sched_data'`, apod. Ty jsou definovány v samostatných souborech `/net/sched/sch_cbq.c`...

Zvenčí funguje řazení do front jako černá skříňka, které přijme paket (funkcí `enqueue`) a později jen může odeslat nebo zahodit. Řazení do front je ale poměrně složitá struktura (především se to týká CBQ), která třídí pakety pomocí různých filtrů do tříd, které mohou být napojeny na další řazení do front na další úrovni. Tím vzniká již dříve zmíněná hierarchická struktura. V Linuxu existuje několik řazení do front, které mohou obsahovat třídy, například CBQ a ATM.



obrázek 4 – obecné schéma řazení do front

Každé řazení do front je identifikováno dvojicí čísel n:0. Nula za dvojtečkou značí, že se jedná o řazení do front. Nenulové vedlejší číslo pak mají třídy, které jsou ke konkrétnímu řazení do front přiřazeny.

2.1.1.2 Třídy

Každý třída je vlastněna nějakým řazením do front (povětšinou typu CBQ) a také vlastní své řazení do front. Třídy vlastně tvoří rozhraní (spoj) v hierarchické struktuře jednotlivých řazení
do front. Třídy jsou již typově rozlišeny podle rodičovského řazení do front. Nejběžnější třídy v CBQ jsou definovány strukturou `'struct cbq_class'`.

Spojení tříd a řazení do front je tak úzké, že při vyvolání funkce `enqueue` v třídě (při zatřiďování paketů) se automaticky volá stejná funkce v přiřazeném řazení do front. Třídy jsou v rámci rodičovského řazení do front uspořádány v hašovací tabulce. Třídy tedy přijaté pakety nijak neuschovávají a okamžitě je předávají svému řazení do front.

2.1.1.3 Filtry

Filtry obecně slouží k klasifikování paketů. Toto klasifikování může sloužit k diferencování, do které třídy příslušný paket přiřadit. Filtrům odpovídá struktura `'struct tcf_proto'` v souboru `include/net/pkt_cls.h`.

Rozlišujeme dva základní typy filtrů:

- **Obecný filtr** (generic filter) se může vyskytovat jen v jediné instanci v rámci řazení do front. Příkladem takového filtru jsou „routovací filtr“, který klasifikuje pakety na základě informací ze směrovací tabulky, resp. tabulek (definován je v souboru `net/sched/cls_route.c`). Dále můžeme zmínit filtr, který využívá informací, které do paketu přidal linuxový firewall `ipchains`. Tento nástroj roztřídí pakety podle svých kritérií a pomocí tohoto filtru jsou pak tyto pakety rozděleny do tříd. Tento filtr najdeme v souboru `net/sched/cls_fw.c`.
- **Specifický filtr** se může vyskytovat ve více instancím v jednom řazení do front. Ke každé třídě je možné vytvořit takový filtr, který do ní bude přiřazovat pakety. Příkladem takového filtru je filtr RSVP. Ten řídí pakety na základě informací poskytnutým protokolem RSVP. Najdeme jej v souboru `net/sched/cls_rsvp.c`. Druhým případem je filtr u32. Toto je asi nejobecnější filtr, umožňuje klasifikovat pakety pomocí jakékoliv informace v hlavičce. Cenou za to je menší rychlost. Na tento typ filtru se zaměříme dále v konkrétních konfiguracích. Najdeme jej v souboru `net/sched/cls_u32.c`.

Filtry jsou označeny podle řazení do front nebo podle třídy, ke které jsou přiřazeny. Tady jsou jednotlivé filtry seřazeny v seznamu podle priority a podle protokolu. Žádné dva filtry nemohou mít stejnou prioritu a zároveň stejný protokol. Tento seznam je procházen jakmile je vyvolána funkce `enqueue` rodiče.

Jestliže filtr klasifikuje paket, základní operací je jeho zařazení do nějaké třídy. Je ale možné použít mnohem sofistikovanější metodu zvanou policing.

2.1.1.4 Policing

Policing je vlastně mechanizmus udržování šířky pásma v určitých hranicích. Vnitřně používá algoritmus TBF a konfiguruje se úplně stejně. Jedná se vlastně o monitorovací systém, který pak zabezpečuje jen povolené chování. Police může vracet čtyři různé návratové hodnoty - OK, RECLASSIFY, SHOT a UNSPEC. Hodnota RECLASSIFY znamená postoupení dalšímu nižšímu mechanizmu (třeba filtru s nižší prioritou), SHOT znamená zahození paketu a UNSPEC značí, že algoritmus se v tomto okamžiku nemůže rozhodnout nebo odpověď není definována. Policing je možné použít ve čtyřech případech:

- Při filtrování
- Při zatřiďování paketů funkcí `enqueue`
- Při zahození paketu z vnitřního řazení do front a jeho předání o úroveň výše
- Při zahození paketu, když dorazí nový

Více informací je možné nalézt v [2], včetně detailního vykreslení jednotlivých použití a jejich schémat.

2.1.2 Class Based Queue

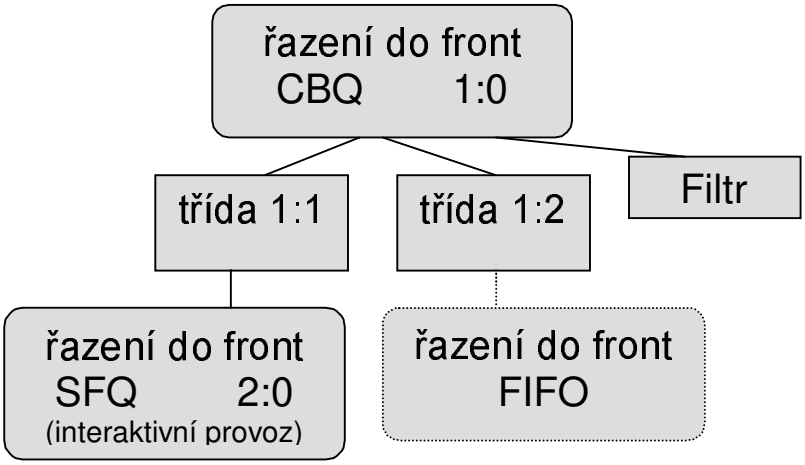
Class Based Queue (CBQ) je patrně nejobecnějším mechanizmem kontroly provozu. CBQ je založena na hierarchickém uspořádání řazení do front, tříd a filtrů. Každé řazení do front stejně tak každá třída má uživatelem přidělenou prioritu. Řazení do front a třídy jsou obsluhovány prioritním cyklickým algoritmem WRR (Weighted Round Robin). Detailní popis tohoto algoritmu je v [7], zde uvedu jen základní myšlenku. Jedná se o prioritní round robin. Kromě priorit má každá třída přiděleno množství dat, které může odeslat, jakmile jí to algoritmus dovolí. Může se ale stát, že tento limit překročí. Pak je v dalším kole patřičně zkrácena.

K síťovému rozhraní je přiřazeno kořenové řazení do front, to pak vlastní další třídy, které pak vlastní zase další řazení do front. Odchozí paket je nejprve přidělen kořenovému řazení do front. Pak jsou postupně aktivovány filtry, čímž je prováděno klasifikování paketu. Ten může být v tomto procesu přidělen do některé třídy, která vlastní již své další řazení do front. Takto tedy paket putuje hierarchií směrem k listům. Na této cestě může být i zahozen či přeřazen do úplně jiné třídy (viz. kapitola 2.1.1.4 Policing).

Každá třída reprezentuje určité síťové prostředky, především šířku pásma. Tuto šířku pásma mohou ale jednotlivé třídy sdílet nebo si ji předávat. Třída může mít atribut *bounded*

(omezená) znamenající, že nemůže překročit přidělenou šířku pásma, i když je aktuálně dostupná. Dalším atributem je *isolated* (izolovaná) značící, že třída nemůže svou šířku pásma poskytnout někomu jinému, i když ji celou aktuálně nevyužívá. Tyto omezení se ale nevztahují na rodiče a potomka, ti si mohou poskytovat šířku pásma bez omezení.

Každé síťové rozhraní i každá třída má přidělené implicitní řazení do front a to FIFO. Přiřazením nového řazení do front se fronta FIFO ruší. Příklad takového hierarchie CBQ je na obrázek 5.



obrázek 5 – příklad hierarchie CBQ

2.2 Uživatelské programy

Bavíme-li se o uživatelských programech, máme na mysli RedHat Linux 6.1 a vyšší. Společným atributem všech uživatelských programů je absence dokumentace a obecně neexistence anglicky psaných informací na toto téma.

Zdrojem mých informací byly především diskusní skupiny (především [43]), dále pak [1], [4]. Ale i v těchto dokumentech není dán vyčerpávající popis všech možností konfigurace kontroly provozu. V některých případech bylo jediným řešením zkoumání zdrojových textů, které jsou v drtivé většině ponechány bez komentářů.

V zásadě existují dva balíky programů pro kontrolu kvality služby. První je velice jednoduchá utilita „Linux traffic shaper" – program `shapecfg`. Druhým balíkem je `iproute2`, což je nesrovnatelně složitější systém. `Iproute2` obsahuje dva základní uživatelské programy – `ip` a `tc`. Program `ip` pokrývá především problematiku směrování a tato práce se jím bude dále zabývat jen okrajově. Naproti tomu program `tc` (traffic controller) je hlavním předmětem dalšího textu.

2.2.1 Traffic shaper

Traffic shaper je v Linuxu představován programem `shapecfg`, jehož autorem je Alan Cox. Slouží výhradně jen k omezení šířky pásma na určitou hodnotu. Program `shapecfg` se nachází v balíku `shapecfg` a je jej možné použít od verze jádra 2.0.36.

Program vytvoří nové síťové rozhraní, které je připojeno k nějakému stávajícímu. Toto nové rozhraní již nese údaj o šířce pásma. Takže konfigurace by mohla vypadat následovně:

```
# /sbin/shapecfg attach shaper0 eth0
# /sbin/shapecfg speed shaper0 64000
```

První řádek vytvoří k rozhraní `eth0` nové `shaper0`. Druhý řádek pak provede omezení šířky pásma na 64000 bitů/s. Tato hodnota se může pohybovat v rozmezí od 9600 bitů/s do 256000 bitů/s, což tuto utilitu staví na vedlejší kolej. Toto omezení spočívá v časování. Jestliže

systém není dlouho v režimu jádra, modul pro kontrolu rychlosti si nemá jak zjistit systémový čas a omezování rychlosti pak nepracuje spolehlivě. Taková situace nastává, když na stejném systému běží náročnější výpočty, které nepotřebují volat služby jádra. Pro běžné použití jsem na dedikovaném počítači vyzkoušel bez problémů i vyšší rychlosti.

Druhou částí konfigurace je směrování:

```
# ifconfig shaper0 host netmask mask broadcast bcast up
# route add -net net netmask mask dev shaper0
```

a dále

```
# route delete default eth0
# route add default gw gateway shaper0
```

Klasifikování může být rozšířeno i o zdrojovou IP adresu. K tomu je potřeba patch o Mika McLagana.

Podrobněji o programu `shapecfq` píše práce [8].

2.2.2 Program tc (traffic controller)

Program `tc` je silný nástroj kontroly provozu, umožňuje přiřadit k síťovému rozhraní řazení do front, třídy a filtry a jejich nesčetné parametry. `Tc` je součástí balíku `iproute2`, který kromě něj obsahuje i program `ip`. Jedná se obecně o nástroj na konfiguraci směrování a dalších síťových služeb (nahrazuje a rozšiřuje uživatelské programy `route`, `ifconfig`, `netstat` a další). K programu `ip` existuje poměrně podrobná dokumentace [9] a proto se mu budu dále věnovat jen okrajově. Program `ip` bude souviset s konfigurací routovacích filtrů.

Naproti tomu k programu `tc` v současné době neexistuje žádná kompletní dokumentace, nejvíce informací je v [1] a [4]. Ale i tyto dokumenty se dotýkají jen základních voleb.

Mnoho parametrů příkazu `tc` vyžaduje číselný parametr vyjadřující rychlost přenosu dat. Tento parametr lze uvést v různých jednotkách (označených klíčovým slovem za číselným údajem). Implicitně jsou rychlosti uváděny v bajtech za sekundu. Další jednotky jsou uvedeny v tabulce 1.

Klíčové slovo	Přesný název	Převod na bajty za sekundu
brak	bity za sekundu	1/8
bps	bajty za sekundu	1
kbps	kilobajty za sekundu	1024
mbps	megabajty za sekundu	1024*1024
kbit	kilobity za sekundu	1024*1/8
mbit	megabity za sekundu	1024*1024*1/8

tabulka 1 – přehled jednotek rychlosti

2.2.2.1 Základní syntax

Základní syntax příkazu `tc` je následující:

```
# tc [VOLBY] OBJEKT { PŘÍKAZ | help }
kde OBJEKT = { qdisc | class | filter }
     VOLBY = {-s[tatistic] | -d[etails] | -r[aw] }
```

Příkaz má tedy tři základní módy týkající se řazení do front, tříd a filtrů. Nejdříve se zaměříme na řazení do front.

2.2.2.2 Řazení do front

Syntax je následující:

```
# tc qdisc [ add | del | replace | change | get ] dev
      ZAŘÍZENÍ [ handle ČÍSLO ] [ root | parent ČÍSLO_RODIČE
      ] [ estimator INTERVAL KONSTANTA ] [ [ TYP_ŘAZENÍ ] [
      help | VOLBY ]
kde TYP_ŘAZENÍ = { [p|b]fifo | cbq | tbf | red | prio |…}
```

- handle reprezentuje jedinečné označení řazení do front, přiděluje jej uživatel, resp. administrátor. Řazení do front mají druhou složku čísla nulovou (např. 1:0)
- root značí, že řazení do front je kořenem v hierarchii, jinak je nutné uvést číslo rodiče
- estimator se používá k zjištění, zda jsou parametry řazení do front nastaveny správně. Více informací je v [6].

FIFO

Syntax je:

```
# tc … [p|b]fifo [ limit ČÍSLO ]
```

Písmeno p znamená paket, písmeno b znamená bajt. Číslo udává velikost fronty v paketech nebo bajtech.

TBF

Syntax je:

```
# tc … tbf limit LIMIT burst BURST[/BURST2] rate RYCHLOST [
      mtu MTU[/MTU2] ] [ peakrate PEAKRATE ] [ latency ODEZVA
      ]
```

Limit udává velikost fronty, rychlostí se myslí šířka výstupního pásma (neboli rychlost generování nových tokenů). Burst udává počet paketů, které jsou poslány v jednom časovém okamžiku. MTU udává minimální velikost paketu.

V případě zapojení dvou algoritmů TBF do série, peakrate udává výstupní rychlost prvního z nich. Parametry s 2 pak náleží prvnímu průchodu. Je jasné. že peakrate musí být větší než parametr rate. Velikost první fronty odpovídá MTU druhé fronty. Požadovanou odezvu je možné nastavit, jinak je určena vzorcem:

```
ODEZVA := max((LIMIT-BURST)/RYCHLOST,(LIMIT-MTU)/PEAKRATE)
```

RED

Syntax je:

```
# tc … red limit LIMIT min MIN max MAX avpkt PRUM burst
      BURST probability PROB bandwidth RYCHLOST
```

Limit udává maximální velikost fronty, jestliže algoritmus pracuje správně, tohoto limitu nebude nikdy dosaženo. Proto tento parametr neovlivňuje chování algoritmu. Limit musí být větší než max+burst.

Min udává minimální velikost fronty, kdy je již nenulová pravděpodobnost zahazování paketů. Naopak, dosáhne-li velikost fronty max, jsou zahazovány všechny příchozí pakety.

Parametr prob udává Max_Prob, prum pak průměrnou velikost paketů vstupujících do tohoto řazení do front. Ostatní parametry už byly popsány výše.

Parametr burst by měl být větší než podíl min/avpkt. Dokument [11] udává jako experimentálně optimální hodnotu `( 2 * min + max ) / ( 3 * avpkt ).`

SFQ

Syntax je:

```
# tc … sfq [ perturb DOBA ] [ quantum VELIKOST ]
```

Quantum udává jednotkovou velikost dat, která jsou ukládána do tabulky a většinou je hodnota rovná MTU. Jádro obnovuje hašovací tabulku (řeší kolize) s periodou uvedenou v parametru perturb. Čím delší je tato perioda, tím je méně zatížen procesor. Na druhou stranu se vyskytuje více kolizí.

CBQ

CBQ má nejsložitější syntax:

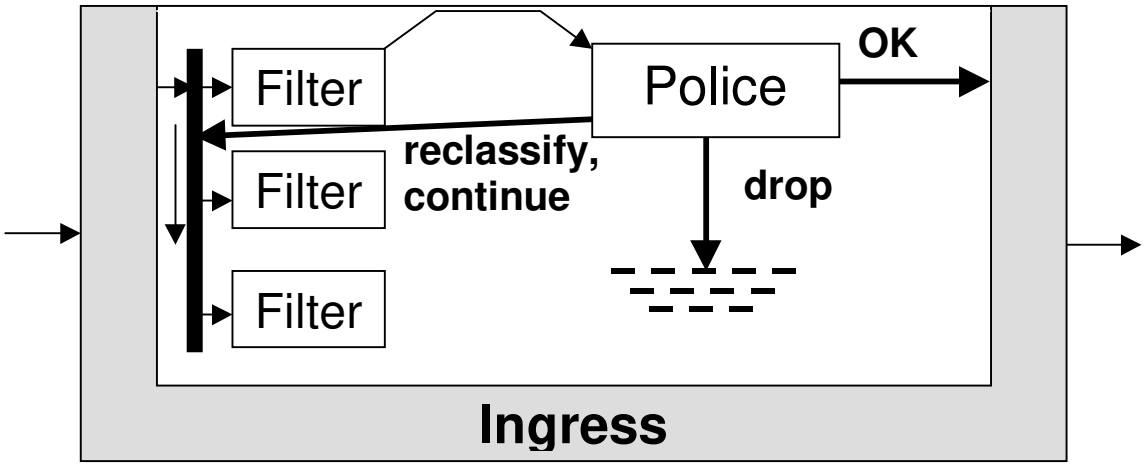
```
# tc … cbq bandwidth BANDWIDTH [ avpkt AVPKT ] [ mpu MPU ] [
      cell CELL ] [ ewma EWMA ]
```

Význam voleb je následující:

- Bandwidth označuje maximální šířku pásma dostupnou na rozhraní, ke kterému je toto řazení do front přiřazeno.Obvykle to bývá maximální rychlost fyzického média, v případě ethernetu 10 Mbps
- avpkt uvádí průměrnou velikost paketů. Ta se používá k výpočtu doby přenosu jako podíl avpkt / bandwidth. Velikost je možné stanovit experimentálně, většinou vychází na 50-60 % MTU (maximální velikost paketu)
- MPU je minimální velikost paketu. Menší pakety jsou uměle prodlužovány
- cell reprezentuje počet bajtů, v kterých bude jádro měřit dobu přenosu paketu. Používá se jako index do tabulky, která udržuje k jednotlivým délkám paketů jejich dobu přenosu.
- ewma je zkratka exponential weighted moving average. Jedná se o konstantu, kterou používá algoritmus WRR ke stanovení průměrné doby čekání na odbavení. Přesná definice je uvedena v [6].

Ingress

Ingress je složitější řazení do front. Neobsahuje žádné třídy, rozhoduje se ale na základě filtrů
a police. Syntax je jednoduchý, protože neobsahuje žádné parametry. Fungování tohoto řazení do front vysvětluje obrázek 6:



obrázek 6 – schéma řazení do front ingress

Řazení do front Ingress se používá tam, kde je potřeba omezit přenosovou rychlost pro nějakou přesně definovanou třídu paketů. Typicky třeba pro pakety s danou zdrojovou IP adresou a nebo pakety protokolu ICMP. Výhodné je toto řazení do front jako hráz proti záplavě SYN paketů. Pomocí filtru fw se označí pak se jim přidělí nějaká malá šířka pásma.

2.2.2.3 Třídy v CBQ

Existuje více řazení do front, které obsahují třídy, já zde ale popíši jen nejpoužívanější CBQ.

Základní syntax je:

```
# tc class … [ classid CLASSID ] [ root | parent PŘEDEK ]
      cbq bandwidth BANDWIDTH rate RATE maxburst MAXBURST [
      avpkt AVPKT ] [ minburst MINBURST ] [ bouded ] [ iso-
      lated ]
      [allot ALLOT ] [ mpu MTU ] [ weight WEIGHT ] [ prio
      PRIO ] [ cell CELL ] [ ewma EWMA ] [ estimator INTERVAL
      KONSTANTA ] [ split TŘÍDA ] [ defmap MASKA/ZMĚNA ]
```

Popis jednotlivých voleb je následující:

- Classid je uživatelem přidělené číslo ve formátu *hlavní:vedlejší*. Hlavní číslo je shodné s předkem, vedlejší pak volí uživatel. Například je-li rodič 1:0, může mít třída číslo 1:2.
- root značí, že třída je kořenem hierarchie
- parent označuje číslo předka v hierarchii
- bandwidth reprezentuje maximální šířku pásma, která je dostupná řazení do front, jež vlastní tuto třídu
- rate je šířka pásma alokovaná pro tuto třídu
- maxburst je maximální počet bajtů, které mohou být vyslány naráz
- avpkt je průměrná velikost paketu, které náleží této třídě
- minburst je minimální počet bajtů, které mohou být vyslány naráz
- bounded znamená, že tato třída nemůže využívat nevyužitou šířku pásma od svých předků. Bez tohoto parametru může třída využívat šířku pásma svého rodiče
- isolated znamená, že třída nemůže sdílet šířku pásma s jinou třídou, jenž není potomkem
- weight je parametr, který by měl být proporcionální k rate. Zatím je jej možné ponechat na jedničce.

- split určuje uzel v hierarchii, od kterého budou na tuto třídu používány filtry. Jinými slovy, na tuto třídu mají vliv jen filtry od split-třídy a níže. Používá se kvůli optimalizaci klasifikování paketů. Implicitně je to kořen hierarchie
- prio udává prioritu této třídy v rozsahu 1 až 10. Čím nižší hodnota, tím je priorita vyšší.
- parametr defmap určuje, do které třídy náleží pakety, které neprojdou žádným filtrem. Jako logická priorita paketu je využíváno pole ToS v hlavičce nebo hodnota SO_PRIORITY v lokálně generovaných paketech. Jestliže paket projde nějakým filtrem, je podle něj zařazen a logická priorita se nepoužívá. V opačném případě je paket zařazen do takové třídy, u níž parametr defmap bitově pokrývá logickou prioritu. Zřejmě defmap ffff značí implicitní třídu pro všechny neklasifikované pakety.
- estimator slouží k optimalizaci parametrů algoritmu CBQ. První parametr určuje, jak často jádro vyhodnocuje průměrnou šířku pásma přidělenou této třídě. Druhý parametr pak značí délku vyhodnocení. První parametr se obvykle pohybuje od 1 do 2 sekund, druhý pak 8-16 sekund.

2.2.2.4 Filtry

Filtry slouží v CBQ ke klasifikaci paketů. Základní syntax je:

```
# tc filter [ add | del | change | get ] dev ROZHRANÍ [ prio
     PRIO ] [ protocol PROTOCOL ] [root | parent CLASSID ]
     [ handle FILTERID ] [ [ TYP_FILTRU ] [ help | OPTIONS ]
```

kde `TYP_FILTRU := { rsvp | u32 | fw | route | tcindex }`

Význam jednotlivých voleb je následující:

- prio určuje prioritu filtru. To znamená, že určuje pořadí, ve kterém se filtry aplikují
- protocol určuje protokol paketů, které tento filtr klasifikuje. Žádné dva filtry nemohou mít stejnou prioritu a zároveň protokol
- handle je identifikační číslo filtru, jeho formát se liší podle různých typů filtrů.

U filtrů se ve zdrojových textech zaměňuje klíčové slovo *flowid* a *classid*. Zatím jsem nenarazil na případ, který by tato slova v případě filtrů rozlišoval.

Routovací filtr

Routovací filtr se rozhoduje na základě informací (většinou značek) z routovacích tabulek. Značkování paketů se provádí příkazem `ip`. Syntax je následující:

```
# tc filter … route [from REALM | formif TAG ] [ to REALM ]
     [ flowid TŘÍDA ] [ police POLICE ]
```

Realm[1] určuje značku přidělenou příkazem `ip` a flowid pak cílovou třídu. Jestliže tyto parametry chybí, pakety náleží do třídy, která je shodná se značkou. Parametry police jsou podobné s algoritmem TBF a budou popsány dále. Vše ozřejmí příklad:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 100
     route
# tc filter add dev eth0 parent 1:0 protocol ip prio 100
     route to 1 flowid 1:3
# tc filter add dev eth0 parent 1:0 protocol ip prio 100
     route to 2 flowid 1:4
# ip route add 10.0.0.100 via 10.0.0.4 dev eth0 realm 1
# ip route add 10.0.0.200 via 10.0.0.4 dev eth0 realm 2
```

V tomto případě provoz s cílovou adresou 10.0.0.100 přes 10.0.0.4 náleží do třídy 1:3 a cílová adrese 10.0.0.200 pak do třídy 1:4. Obecně pak:

```
# ip rule add 10.0.0.100 realms cesta
# ip rule add tos 0x10 realms cesta
# tc filter add dev eth0 parent 1:0 protocol ip prio 100
      route to cesta flowid 1:2
```

Starší konfigurace routovacího filtru uváděná především v [1] se dnes již nepoužívá a novější verze programů `tc` a především `ip` ji ani nepodporuje. Tady byla směrovacímu pravidlu přímo přiřazena třída volbou flowid příkazu `ip`.

Filtr fw

Filtr fw se rozhoduje na základě značek, které do paketů vloží firewallovací pravidla příkazem `ipchains` s parametrem `-m`. Firewall uloží značku do pole `skb->fwmark` ve struktuře paketu. Velikost pole je 32 bitů, prvních 16 náleží cílovému řazení do front, dalších 16 už pak konkrétní třídě. Tak například může být značka ve formátu `-m 0x1000a`. Písmeno *a* značí číslo třídy 1:a.

Filtr fw je podobný routovacímu filtru. Oba dva využívají značení, které do paketů přidají různé externí subsystémy. Filtr fw má tu výhodu, že postihuje i vyšší vrstvy než je IP. Program `ipchains` umí rozlišit i čísla portů v TCP a UDP a nebo typy zpráv protokolu ICMP.

Syntax filtru je jednoduchý:

```
# tc filter ... fw [ classid TŘÍDA ] [ police POLICE ]
```

Příklad konfigurace může být následující:

```
# ipchains -A output -p tcp -b -s 0.0.0.0/0 telnet -m
     0x10001 -j ACCEPT
# ipchains -A output -p tcp -b -s 0.0.0.0/0 22 -m 0x10001 -j
     ACCEPT
# ipchains -A output -p ip -d 192.168.1.0/24 -m 0x10001 -j
     ACCEPT
# ipchains -A output -p tcp -m 0x10002 -j ACCEPT
# tc filter add dev eth0 parent 1:0 protocol ip prio 90 fw
```

Filtr rozdělí data do dvou tříd 1:1 a 1:2.

Filtr u32

Filtr u32 je nejobecnějším klasifikátorem v tomto systému. Cenou za to je oproti předchozím dvěma menší rychlost vyhodnocení. Nicméně vše, co umí filtry fw a route, zvládne i filtr u32. Filtr může klasifikovat pakety na základě zdrojové a cílové IP adresy, zdrojového a cílového portu, protokolu, pole ToS. Obecně může filtr u32 využívat jakoukoliv část všech hlaviček paketu.

Konfigurace filtru je poměrně složitá a skládá se s několika kroků.

Nejprve musíme inicializovat hašovací tabulku klíčů příkazem:

```
# tc filter … u32 divisor VELIKOST
```

Parametr velikost udává počet pozic v hašovací tabulce.

Dalším příkazem určíme, podle kterých položek z hlavičky má filtr klasifikovat:

```
# tc filter … u32 match SAMPLE [ match SAMPLE [ … ] ]
     [ flowid TŘÍDA ]
kde SAMPLE  = { ip | ip6 | udp | tcp | icmp | u{32|16|8} }
     ARGUMETY
```

Filtr má ještě daleko více voleb, které popíši dále. Teď se zaměřím na specifikaci dat, podle kterých se paket klasifikuje.

Položka SAMPLE určuje podtyp filtru u32. Tyto podtypy rozdělím na *specifické* a *obecné* (toto nemá nic společného se specifickým a obecným filtrem). Takže volby ip, ip6, udp, tcp a icmp jsou specifické podtypy (říkejme jim selektory protokolu), které umožňují klasifikaci podle údajů z příslušných hlaviček. Za selektorem hlavičky následuje selektor pole v hlavičce. Tímto je jednoznačně určeno místo v hlavičce paketu. Teď je nutné uvést jen vzor a masku, podle které se budou pakety klasifikovat. V následující tabulce jsou uvedeny možné selektory pole.

selektor protokolu	selektor pole	parametry	popis
match ip	src	prefix/32	zdrojová IP adresa
	dst	prefix/32	cílová IP adresa
	tos		pole TOS
	dsfield	tos u8	pole TOS
	precedence		
	ihl	ihl u8	velikost hlavičky v 32bitových slovech
	protocol	prot u8	protokol vyšší vrstvy
	nofrag		paket není fragmetován
	firstfrag		příznak první fragment
	df		příznak „nefragmetovat“
	mf		příznak „více fragmetů“
	sport	port u16	zdrojový port protokolu vyšší vrstvy
	dport	port u16	cílový port protokolu vyšší vrstvy
	icmp_type	typ u8	typ zprávy ICMP
	icmp_code	kod u8	kód zprávy ICMP
match ip6	src	prefix/128	zdrojová IP adresa
	dst	prefix/128	cílová IP adresa
	flowlabel	flow u32	identifikace toku v IPv6
udp, tcp	src	src u16	zdrojový port
	dst	dst u16	cílový port
icmp	type	typ u8	typ zprávy ICMP
	code	kod u8	kód zprávy ICMP

tabulka 2 – parametry speciálního filtru u32

Jsou-li podmínky napsány v jednom volání příkaz za sebou, musí platit všechny zároveň. Implicitní logická operace je konjunkce. Disjunkce se docílí voláním podmínek ve více příkazech.

Obecný podtyp filtru je obecnější varianta předchozího. Jádro vlastně žádný specifický filtr u32 nezná, už volání příkazu `tc` je převádí na filtry obecné. Tento převod je popsán v souboru `f_u32.c`, který se nachází v zdrojovém balíku `iproute2`.

Obecný filtr u32 má následující syntax:

```
# tc … match { u32 | u16 | u8 } VZOR MASKA [ at OFFSET
      | nexthdr+OFF ]
```

Číslo 32, 16 a nebo 8 udává, jaká je velikost pole v bitech, podle kterého se má klasifikovat. Vzor a maska mají obvyklý význam a jejich velikost je opět 32, 16 nebo 8 bitů. Parametr *at* udává pozici od začátku paketu a *nexthdr* udává pozici od začátku paketu vyšší vrstvy.

Tak například následující příkazy jsou ekvivalentní:

```
# tc … match ip tos 0x10 0xff flowid 1:1
```

a

```
# tc … match u8 0x10 0xff at 1 flowid 1:1
```

Filtr u32 má ještě několik parametrů, jejíž přesný význam se mi nepodařilo přesně zjistit. Uvedu alespoň syntax.

```
# tc filter … u32 [ link HTID ] [ police POLICE ] [ offset
     OFFSET ] [ ht HTID ] [ hashkey HASHKEY ] [ sample
     SAMPLE ]
```

Některé z těchto parametrů budou dále použity v konfiguracích. Jejich použití jsem našel v několika příkladech konfigurace a bez další hlubší znalosti je používám. Výjimku tvoří parametr *police*, který vysvětluje následující kapitola.

2.2.2.5 Police

Klíčové slovo police je možné používat na více místech, syntax parametrů je ale vždy táž.

```
# tc … police rate RATE burst BURST[/BURST2] [ mtu
     MTU[/MTU2] ] [ peakrate PEAKRATE ] [ avrate AVRATE ] [
     AKCE ]
```

kde `AKCE := { reclassify | drop | continue | shot | pass | ok}`

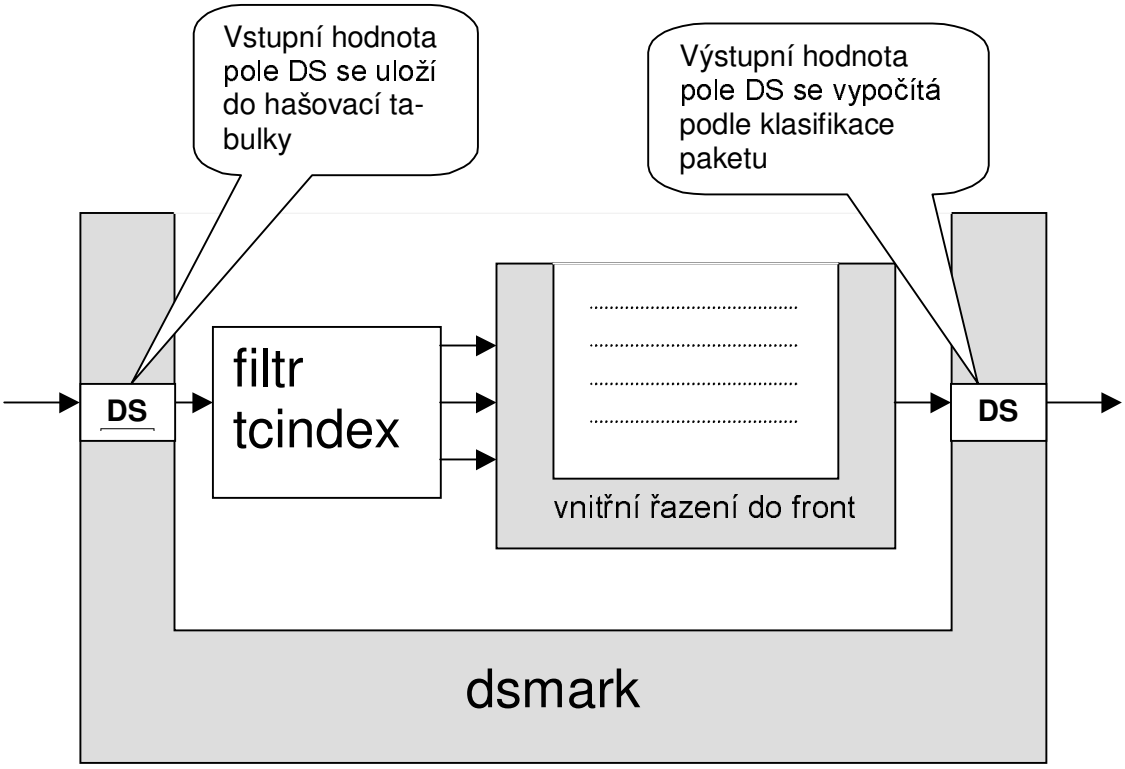
Parametry drop a shot jsou ekvivalentní, stejně tak ok a pass. Ostatní parametry mají stejný význam jako u algoritmu TBF.

2.2.2.6 Podpora Differentiated Services

Ačkoliv architekturu DiffServ lze budovat pomocí CBQ a výše popsaných filtrů, existuje speciální řazení do front dsmark. To přímo odpovídá návrhu podle normy DiffServ. Toto řazení do front může obsahovat další třídy a filtry. V struktuře paketu je položka `skb->tc_index`, do které si řazení dsmark ukládá značku paketu. Tato značka se nachází v hlavičce paketu jako pole DS.

Řazení do front dsmark pak pomocí filtrů klasifikuje pakety a zatřiďuje je do tříd. Při tomto procesu se může položka `skb->tc_index` změnit. Pro řazení dsmark existuje filtr tcindex, který klasifikuje pakety na základě pole `skb->tc_index`.

Schéma fungování řazení dsmark ukazuje obrázek 7.



obrázek 7 – schéma řazení dsmark

Řazení do front dsmark si vytvoří tabulku dvojic [maska, hodnota], podle kterých se bude počítat výstupní hodnota pole `skb->tc_index`. Jakmile je vyvolána funkce dequeue při odesílání paketu, změní se pole DS v hlavičce paketu následovně:

```
DS := ( DS & maska ) | hodnota
```

Teď je stačí určit, která položka s tabulky dvojic se použije. Položka se volí na základě klasifikace filtrem tcindex.

Syntax řazení do front dsmark je následující:

```
# tc … dsmark indices INDICES  [ default IMPLICITNÍ_INDEX ]
      [ set_tc_index ]
```

Položka indices značí maximální velikost tabulky dvojic [maska, hodnota]. Volba set_tc_index značí, že do pole DS se má při vstupu paketu zkopírovat do `skb->tc_index`. Řazení dsmark má za úkol klasifikovat paket a výsledek uložit do `skb->tc_index`, není-li paket klasifikován, použije se implicitní index.

Příkazem

```
# tc class change dev ROZHRANÍ classid TŘÍDA dsmark mask
      MASKA value HODNOTA
```

se vytvoří položka v tabulce. Takže položkám v tabulce odpovídají třídy a ty řazení do front dsmark obsahuje již na začátku.

Filtr tcindex má syntax:

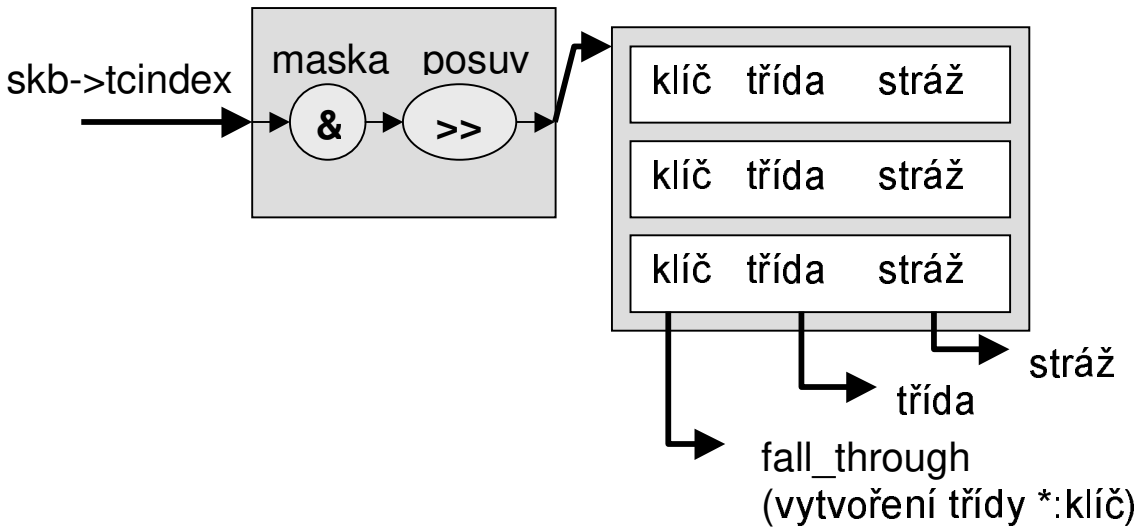
```
# tc filter … [ handle ČÍSLO ] tcindex [ maska MASKA ]
      [shift POSUN ] [ classid TŘÍDA ] [ pass_on ] [
      fall_through ] [ police POLICE ]
```

Jestliže je zadána maska a posun a handle a třída zadány nejsou, pak výsledný klíč je vypočítán následovně:

```
klíč := ( skb->tcindex >> posun ) && maska
```

Naopak jestli je zadáno handle, musí být zadána i třída. Pass_on znamená, že když neexistuje třída s hodnotou handle, pak se pokračuje klasifikováním dalším filtrem.

Volba fall_through se bude snažit vytvořit novou třídu, jestliže neexistuje taková, která by odpovídala klíči. Bez této volby se jinak pokračuje s dalším filtrem. Filtr tcindex ukazuje obrázek 8.



obrázek 8 – filtr tcindex

S architekturou DiffServ souvisí poslední řazení do front, a to GRED (Generalized RED). Obsahuje virtuální fronty, do kterých třídí pakety podle posledních čtyř bitů v `skb->tcindex`.

2.2.2.7 Další operace

Doposud bylo popsáno, jak se programem `tc` přidávají různé struktury. Vždy se používalo klíčové slovo *add*.

Jestliže ale konkrétní struktura již existuje, parametry se mění výměnou *add* za *change*. Všechny parametry (i ty, které se nemění) se musí uvést znova.

Podobně se příkazem *del* zruší konkrétní struktura. Opět se musí uvést všechny povinné parametry, ne jenom identifikační číslo. Spolu se strukturou se zruší i všichni potomci.

Příkaz

```
# tc  [ -s[tatistic] | -d[etails] | -r[aw] ] { qdisc | class
      | filter } ls dev ROZHRANÍ
```

vypíše parametry všech konkrétních struktur na daném síťovém rozhraní. Volba *-s* vypíše statistiku přenesených paketů a různé další výpočty podle typu řazení do front. Tato volba se výborně hodí pro ladění parametrů.

2.3 Programové rozhraní a další vývoj

Struktura zdrojového kódu programu `tc` je poměrně přehledná, stejně tak je tomu i u řazení do front a filtrů v jádře.

2.3.1 Program tc

Zdrojový kód programu `tc` zajišťuje především syntaktickou analýzu. Data jsou převedena do struktur a předána jádru systému. Řazení do front jsou implementována v souborech `q_*.c` (například `q_cbq.c` nebo `q_red.c`), filtry pak v souborech `f_*.c` (`f_u32.c` ...). Analýzou těchto souborů lze zjistit přesný syntax příkazu `tc` (s tím, že zdrojový kód je nedokumentován). Bohužel předávání parametrů jádru je řešeno nepřehledně a jen stěží se nachází vazba na struktury uvnitř systému.

Tento nedostatek řeší několik projektů, jeden z nich je popsán v [12]. V tomto případě se s strukturami řazení do front, tříd a filtrů pracuje pomocí volání `set_qdisc()`, `set_class()` a `set_filter()`. Na dalším API pracují W. Almesberger, J. Hadi Salim a A. Kuznetsov.

Programové rozhraní by pak mohlo sloužit k práci na dalších uživatelských systémech. Program `tc` je jak je vidět výše z uživatelského hlediska značně komplikovaný. Postupem času má být vybudováno i grafické rozhraní, funkčně podobné jako program `tc`.

2.3.2 Jádro systému

Zdrojový kód jednotlivých řazení do front a filtrů je uložen v adresáři `Linux/net/sched/`. Podobně jako v případě programu `tc`, řazení do front jsou uložena v souborech `sch_*.c` a filtry v `cls_*.c`. Jednotlivé soubory je pak možné zkompilovat do jádra nebo jako moduly. Tyto zdrojové kódy jsou celkem obstojně komentovány.

Dokonce tady existuje API pro implementaci dalších filtrů (soubor `cls_api.c`) a generické řazení do front (soubory `sch_api.c` a `sch_generic.c`).

3 Směrovače Cisco

Tato kapitola popisuje základní možnosti kontroly provozu na směrovačích značky Cisco. Přesněji se jedná o operační systém Cisco IOS, který tyto směrovače používají. Všechny prvky se týkají verze 11.1c, případně pak 12.

Vnitřní fungování Cisco IOS není možné zjistit, protože se jedná o proprietární software. Zdrojový kód není veřejně přístupný a dokumentace se zabývá především uživatelskými aspekty.

Průchod paketu síťovým rozhraním lze rozdělit do čtyř sekvencí:

1. Po přijetí paketu následuje jeho tzv. barvení. V podstatě se jedná o police, jak byla vysvětlena v předchozích kapitolách. Jestliže paket splňuje požadavky police, je obarven neboli označkován pro pozdější použití.
2. Druhým krokem je omezování a limitování paketu. Omezování (shaping) se na pakety dívá jako na toky a může třeba omezit jejich rychlost. Mechanizmus omezování ale sleduje dopady na celý tok a podle toho se chová. Naopak limitování je jednoduší a vybraný paket zahodí bez hlubšího zkoumání. Limitování může být aplikováno na příchozí i odchozí pakety, omezování jen na odchozí.
3. V tomto okamžiku následuje řazení do front. Zde se využívá obarvení paketu k jeho klasifikaci.
4. Posledním krokem je poslání paketu na cílovou adresu.

3.1 Řazení do front

Terminologie firmy Cisco rozlišuje podle použití dva typy řazení do front. Většina řazení se používá k zvládání přetížení sítě, druhé se používají pro předcházení přetížení. Nejprve tedy řazení do front pro zvládání přetížení:

- FIFO
- PQ (Priority Queuing)
- CU (Custom Queuing)
- WFQ (Weighted Fair Queuing)
- CBWFQ (Class Based Weighted Fair Queuing)
- IP RTP Priority
- WRR (Weighted Round Robin)
- 2Q2T (2 Queues, 2 TresholdsManaging)

3.1.1 Priority Queuing

Prioritní řazení do front klasifikuje pakety do čtyř tříd. Vyšší třída má vždy absolutní přednost, takže se může stát, že provoz s nižší prioritou je úplně zablokován. Prioritní řazení podporuje i jiné protokoly než IP jako jsou IPX nebo AppleTalk.

Toto řazení se používá pro důležitý provoz, který ale není náročný na přenosovou kapacitu. Typicky to jsou řídící informace, například DNS.

3.1.2 Custom Queuing

Uživatelské řazení klasifikuje provoz až do šestnácti tříd. Každá třída má přidělenu svou frontu typu FIFO. Pomocí police lze třídám přiřadit procentuelní využití linky. Pak jsou podle těchto vah třídy obsluhovány algoritmem round robin. Jestliže je nějaká třída nevyužita, mohou její šířku pásma dostat proporcionálně ostatní.

3.1.3 Weighted Round Robin

Algoritmus prioritní round robin je známý. Pakety jsou rozděleny do čtyř různých tříd podle pole IP precedence. Každý fronta má svou uživatelem danou váhu a odezvu. Jestliže se doba odezvy paketu zvyšuje je váha fronty dočasně zvýšena.

3.1.4 Další algoritmy

Class Based Weighted Fair Queuing algoritmus je kombinací WFQ a CU. Jednotlivé toky mohou být klasifikovány až do 64 tříd. Klasifikování to tříd se děje opět pomocí nějaké police.

IP RTP Priority je jednoduché řazení do front, které dává absolutní přednost hlasovému provozu.

Algoritmus 2Q2T rozděluje provoz do dvou tříd podle pole IP precedence. Ty jsou pak obsluhovány podle svých vah. Třídy mají své fronty FIFO, které mají navíc dva prahy. Prahy jsou různé velikosti fronty, opět se určují podle IP precedence. Takto je provoz rozdělen do čtyř typů.

3.1.5 Předcházení přetížení – WRED

Algoritmy RED a WRED slouží k předcházení přetížení neboli k rychlému nalezení optimální rychlosti toku dat.

WRED je prioritní varianta algoritmu RED. Na prioritě paketu (nebo obecně na jeho obarvení) závisí pravděpodobnost zahození a dolní a horní limit fronty.

Další verzí algoritmu je DWRED (Distributed WRED). V tomto případě je provádění přeneseno na speciální procesor zvaný VIP (Versatile Interface Procesor). Stejně tak existují i distribuovaná verze algoritmu WFQ zvaná DWFQ.

3.2 Signalizační techniky

Signalizační techniky zná Cisco IOS dvě. Jednak podporuje protokol RSVP a navíc lze posílat signalizační data v datových paketech v poli IP precedence. Tedy dva přístupy podle architektur IntServ a DiffServ.

Signalizace v poli IP precedence je pak přijímána algoritmy WRED a WFQ. Protokol RSVP byl již popsán výše a tak jen uvádím, že se zde používá spolu s řazeními WFQ, WRED a CBWFQ.

4 Srovnání popisovaných systémů

Co se týče podpory standardů IntServ a DiffServ, splňují je oba systémy. Cisco IOS má navíc možnost kontroly provozu i na vstupních zařízeních. Dalšími výhodami Cisco IOS oproti implementaci v Linuxu jsou rozšířené verze algoritmů řazení do front. Nutno ale zdůraznit, že většiny těchto rozšíření lze v Linuxu dosáhnout kombinací více typů řazení do front. Nespornou předností směrovačů Cisco je daleko snazší konfigurace. Tuto nevýhodu může linuxová implementace brzy dohnat lepším uživatelským a programovým rozhraním.

Přednosti linuxové implementace se nachází především v její univerzalitě. Ve směrovačích Cisco je předepsáno několik algoritmů pro konkrétní situace, v Linuxu si může uživatel zvolit přesně, co potřebuje.

Dalo by se také hovořit o cenovém srovnání. Pořizovací cena směrovačů Cisco nesnese s linuxovými žádné srovnání, cena linuxového směrovače závisí jen na hardwaru. Jinak to bude s provozními náklady (na administraci), které mohou být v případě Linuxu i větší.

4.1 Další implementace kontroly provozu

Kromě operačního systému Linux a směrovačů firmy Cisco existuje mnoho dalších implementací kontroly provozu, speciálně se zaměříme na CBQ.

Hlavní část teoretické práce na CBQ pochází od Sally Floyd, která působí na universitě v Berkeley. Tam také vznikla první implementace LBNL's CBQ v1.1 a pozdější LBNL/SUN CBQ v2.0.

LBNL's CBQ v1.1 je experimentální implementací v tamnějším unixovém systému BSD. Jedná se ale o prototyp, který nemá praktické využití.

Naproti tomu LBNL/SUN CBQ v2.0 novější implementace pro operační systémy Solaris a SunOS 4.x. Obsahuje kód jádra i uživatelské programy pro ovládání. Další rozšíření podporuje i protokol RSVP.

Pro systém FreeBSD vznikla další implementace ALTQ (Alternate Queuing for FreeBSD). Ta již podporuje i další řazení do front jako třeba RED a SFQ. Dále obsahuje rozhraní spolupracující s protokolem RSVP.

Komerční implementace pochází od firmy Xedia, která nyní patří pod společnost Lucent. CBQ je implementováno v přístupových směrovačích.

5 Experimentální konfigurace

Následující odstavce jsou praktickou částí této práce. Vysvětlím několik konkrétních konfigurací v operačním systému Linux, dále pak srovnání algoritmů řazení do front.

5.1 Podmínky měření

Měření probíhala na vyhrazeném počítači s operačním systémem Linux. Detailněji, jedná se o počítač s procesorem 486/66, 16 MB RAM, pevný disk 1200 MB a síťové karta 3Com 3i509. Počítač je zapojen do fakultní sítě na FI a tím pádem i do celého internetu. Jedná se o ethernetovou síť s maximální rychlostí 10 Mbit/s. Další síťové rozhraní je sériová linka, která vede k dalšímu počítači. Tato linka slouží čistě pro testovací účely, mezi inkriminovanými počítači je dostupné i normální ethernetové spojení na teoretické rychlosti 10 Mbit/s.

Operační systém je tedy RedHat Linux 6.0, verze jádra 2.3.45. Počítač má IP adresu 147.251.48.208 a doménové jméno scylla.fi.muni.cz. Sériové rozhraní ppp0 má IP adresu 10.0.0.2 a ústí na počítači calypso.fi.muni.cz s IP adresami 10.0.0.1, 10.0.0.3, 10.0.0.4 a 10.0.0.5. Na sériové lince je vypnutá Van Jacobsonova komprese hlaviček (volba *novj* programu *pppd*), jinak se jedná o běžná nastavení sériové linky. Přes zařízení ppp0 je směrována jen adresa 10.0.0.1. Sériová i ethernetová linka má nastavenu maximální velikost paketů na 1500 bajtů.

5.2 Měřící nástroje

Pro účely měření se používal především program `ttcp`. Tento program měří šířku pásma mezi dvěma uzly. Pomocí voleb na příkazové řádce lze nastavit většinu parametrů spojení (TCP/UDP, počet a velikost paketů, komunikační porty...). Zdrojový kód musel být upraven tak, aby přenosovou rychlost vypisoval průběžně a ne jen statistické vyhodnocení na konci. Aktuální rychlost se počítá jako průměr rychlostí za poslední jednu sekundu.

Druhým měřícím nástrojem je program `ping`. Ten posílá pakety ICMP Echo. Program slouží pro měření doby odezvy. Tu není možné měřit použitím protokolu TCP, poněvadž třícestné navazování spojení by výsledky značně ovlivnilo. Použití protokolu ICMP zde tedy simuluje protokol UDP. Použitá délka paketů je 64 bajtů. To se ale děje při interaktivním provozu, kdy je samostatně posílán každý stisk klávesy na terminálu. Mezi odesláním jednotlivých paketů se čeká podle situace jednu až tři sekundy.

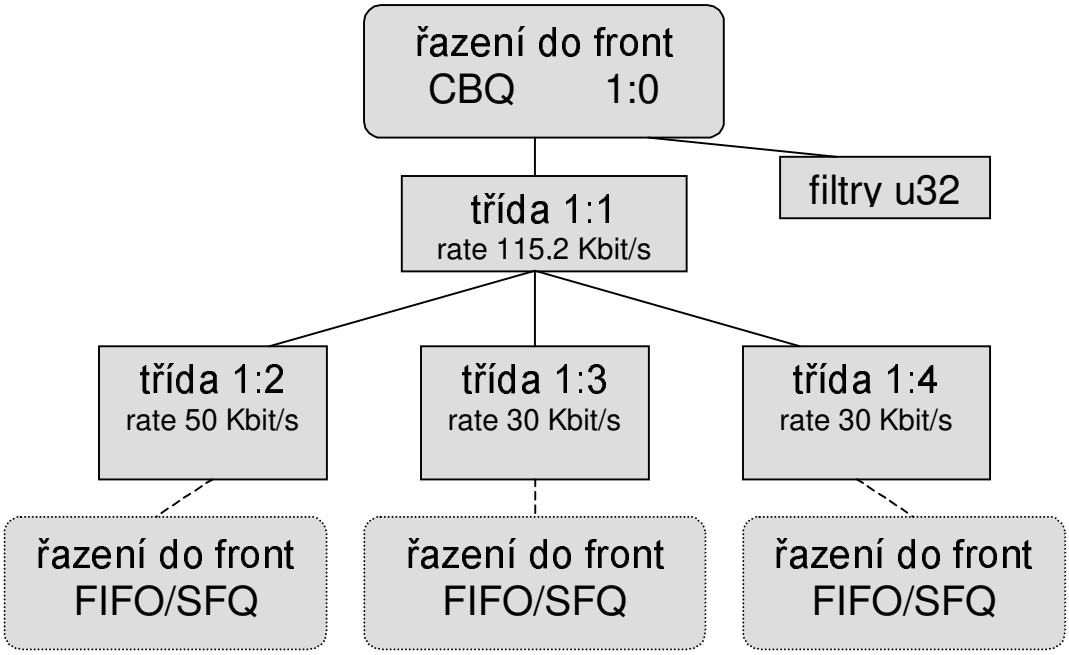
Průměrná rychlost sériové linky naměřená výše popsaným měřícím programem činí 10,64 KB/s v případě jednoho TCP a postupně 3,20 3,37 s 3,37 KB/s v případě tří rovnocenných TCP spojení. Tyto hodnoty byly neměřeny bez použití jakékoliv konfigurace kontroly provozu programem `tc`.

Podobně v případě ethrnetové linky činila šířka pásma 746 KB/s.

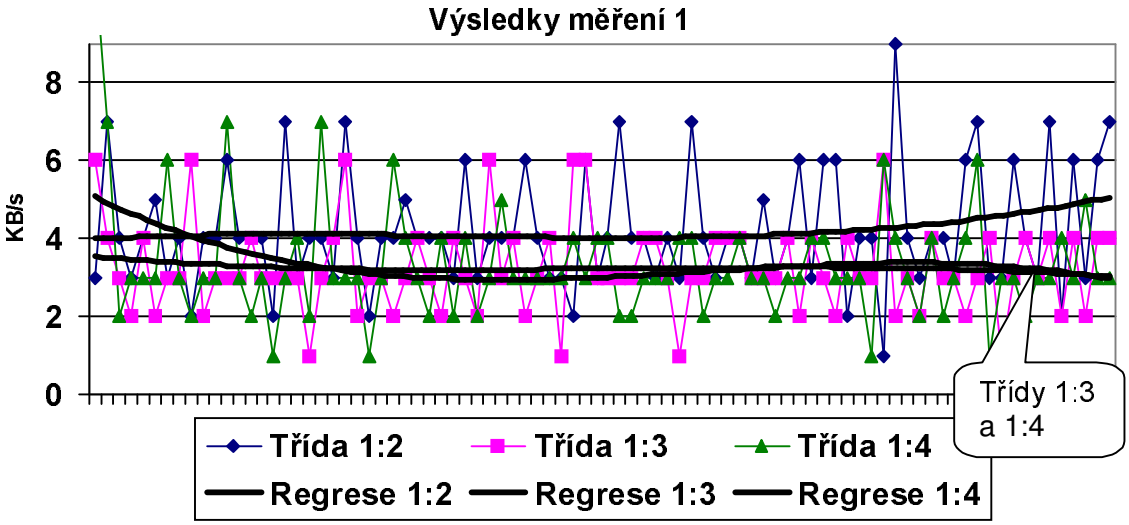
5.3 Konkrétní konfigurace

5.3.1 Meření 1 – rozdělování šířky pásma

První konfigurace má ukázat, jak algoritmus CBQ rozděluje šířku pásma. Tři listové třídy mají přiřazenu rychlost 50, 30 a 30 Kbit/s. Celková šířka pásma je teoreticky 115,2 Kbit/s. Klasifikace se provádí pomocí filtru u32 na základě cílové IP adresy. Stejného efektu lze dosáhnout i pomocí firewallového a routovacího filtru. Listové třídy mají řazení do front FIFO (implicitní). Použitím algoritmu SFQ se naměřené hodnoty prakticky nezměnily, jelikož testování probíhá posíláním paketů z jedné aplikace. Použitý protokol byl TCP a velikost paketů 1500 bajtů, což je maximum na sériových linkách (MTU).



obrázek 9 – konfigurace 1



	Třída 1:2	Třída 1:3	Třída 1:4
Aritmetický průměr [KB/s]	4,20	3,24	3,35
Směrodatná odchylka	1,51	1,14	1,49

tabulka 3 – měření 1

Měření ukázalo poměrně velké odchylky od průměru. Proto jsou v grafu pro větší přehlednost i regresní křivky. Regrese je prováděna polynomem třetího stupně. Horní čára představuje třídu 1:2, dvě dolní pak zbylé třídy.

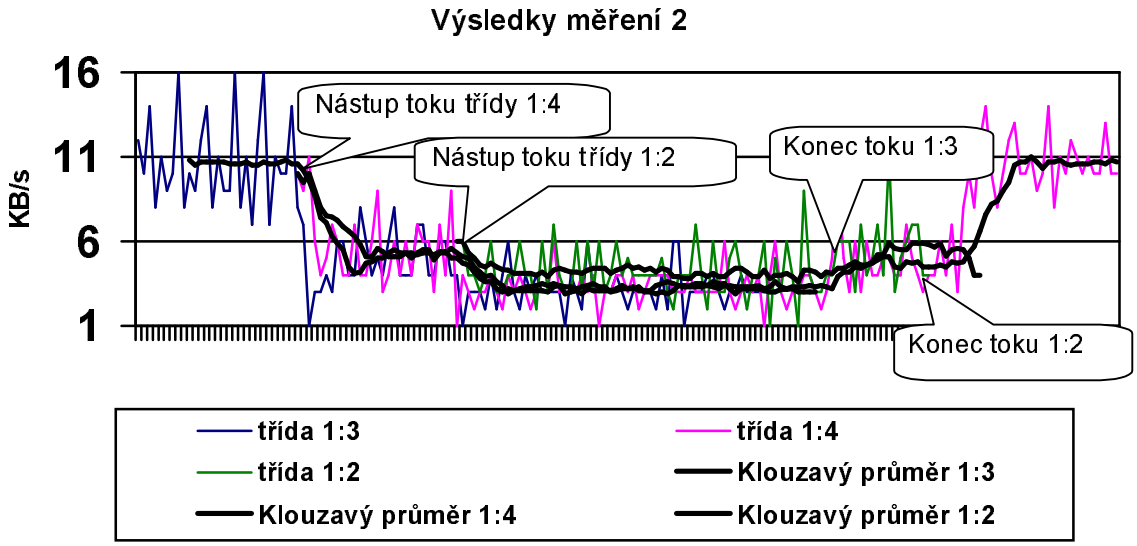
Měření ukázalo, že rychlosti jsou nižší, než je uváděno v konfiguraci. Důležité je, jejich poměr k původně daným je u všech tříd skoro stejný.

5.3.2 Měření 2 – asynchronní toky

Druhé měření využívá shodné konfigurace rozhraní, jako se používala v prvním případě. Jednotlivé toky dat (TCP spojení) byly spouštěny v různých časových okamžicích. Nejdříve začal provoz klasifikovaný do třídy 1:3, pak následovala třída 1:4 a po ní 1:2. Poté byl ukončen provoz třídy 1:3, pak 1:2 a nakonec 1:4.

Místo polynomiální regrese zde bylo využito klouzavého průměru za deset posledních měření.Při nástupu nového toku dat dochází k poměrně velikému pádu toku původního. Naopak při ukončení jednoho toku se ostatní dostávají na vyšší rychlostní úroveň jen pozvolna.

Aritmetický průměr rychlostí v tomto případě nemá smysl, uvedu jen směrodatnou odchylku. Ta je v tomto případě daleko vyšší, což je zapříčiněno různými podmínkami na síti v průběhu měření. Výsledky ukazuje tabulka 4.

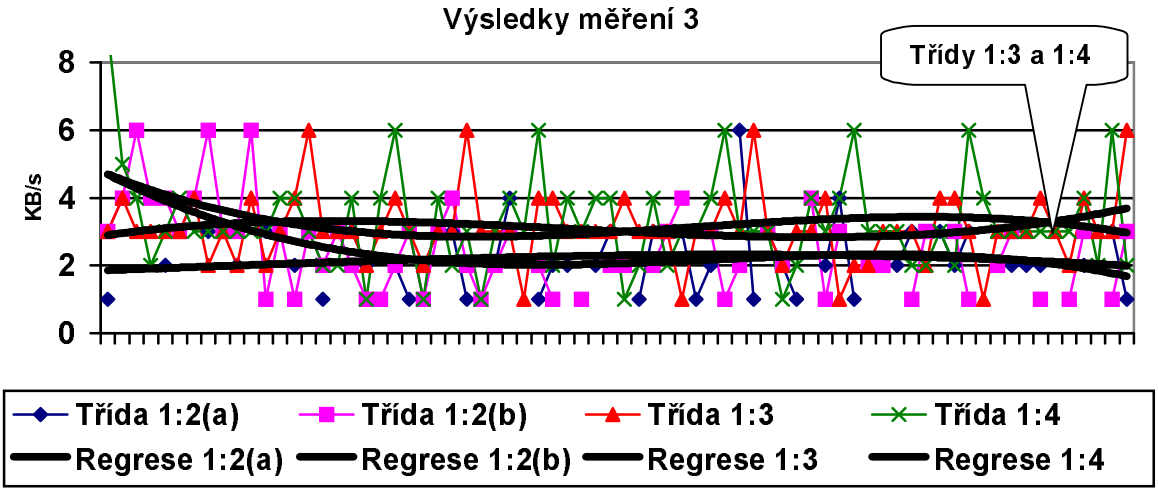


Třída 1:2	Třída 1:3	Třída 1:4
1,68	3,61	3,10

tabulka 4 – měření 2, směrodatná odchylka

5.3.3 Měření 3 – více spojení v jedné třídě

Tomto měření má ukázat, jak se chová více TCP spojení klasifikovaných v jedné listové třídě. Konfigurace je shodná s měřením jedna. Jen to třídy 1:2 přichází dvě TCP spojení.



	Třída 1:2(a)	Třída 1:2(b)	Třída 1:3	Třída 1:4
Aritmetický průměr [KB/s]	2,19	2,52	3,11	3,28
Směrodatná odchylka	0,99	1,29	1,03	1,40

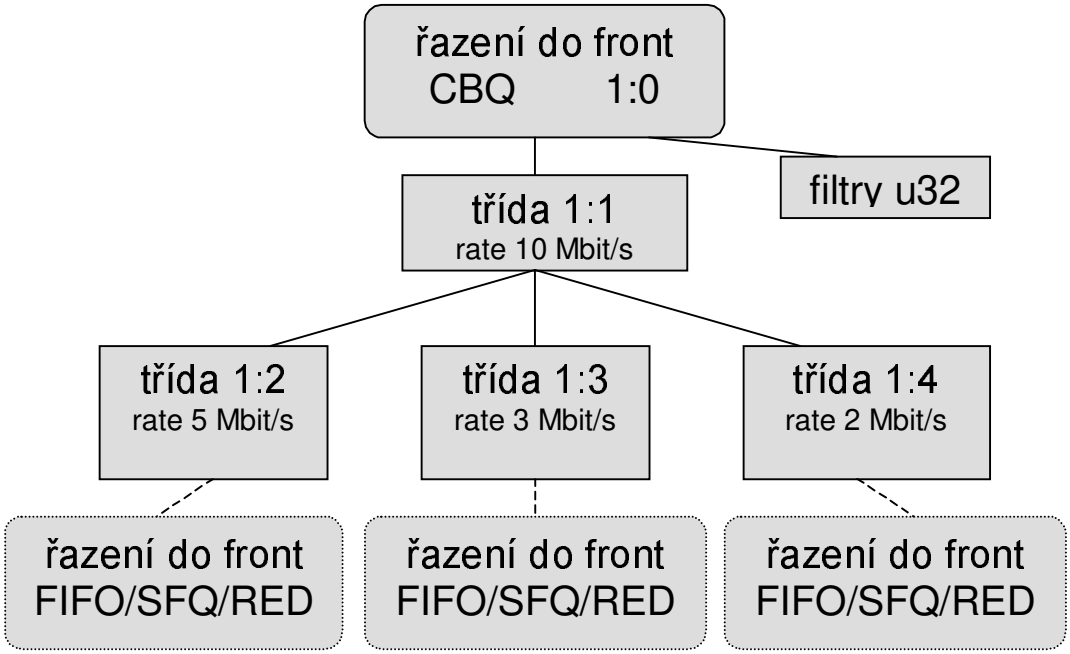
tabulka 5 – měření 3

Nabízí se porovnání s měřením jedna. Jediný rozdíl spočívá v tom, že v třídě 1:2 jsou klasifikovány dvě TCP spojení. Na první pohled by se zdálo, že součet jejich rychlostí bude shodný s výsledkem měření 1. Součet ale je 4,71 KB/s, což je více než v případě jedna – 4,20 KB/s.

Vysvětlení je možné hledat v tom, že při omezování šířky pásma je zasaženo v jednom okamžiku vždy jen jedno TCP spojení a druhé může pokračovat na své rychlosti. Při omezování rychlosti vysílající stanice pozná, že data proudí pomaleji a zpomalí jejich vysílání. V tomto případě se ale zpomalí v jednom okamžiku jen jedno spojení. Kdyby se použilo protokolu UDP namísto TCP, měl by součet rychlostí dvou paralelních odpovídat jednomu solitérnímu.

5.3.4 Měření 4 – omezování provozu na rychlé lince

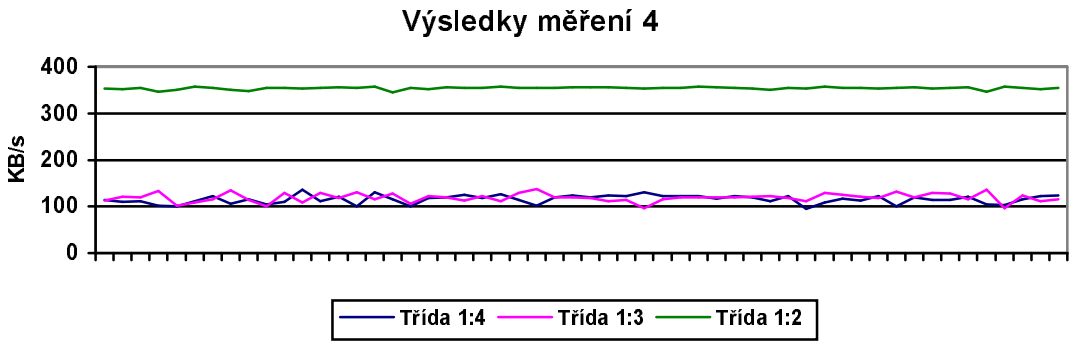
Další měření rychlosti se zaměřuje už na rychlejší ethernetovou linku. V konfiguraci opět budou figurovat tři listové třídy, na které budeme navěšovat různé řazení do front. Třídy jsou nastaveny jako ohraničené a izolované, aby dostávali jen tu šířku pásma, která je jim předepsána.



obrázek 10 – konfigurace 2

	Třída 1:2	**Třída 1:3**	**Třída 1:4**
Aritmetický průměr [KB/s]	353,9	123,8	119,8
Směrodatná odchylka	2,61	9,27	8,92

tabulka 6 – měření 4

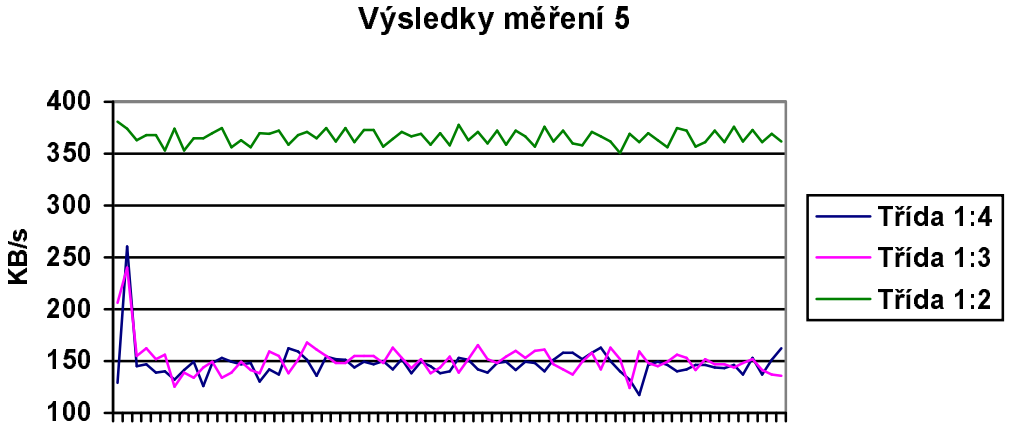


Listové třídy neobsahovaly žádné řazení do front (resp. používaly implicitní FIFO). Abychom dostali výsledek v bitech za sekundu, je nutné výsledky vynásobit přibližně deseti. Ani tak se ale nedostáváme na rychlosti přidělené v konfiguraci tříd. Uváděné rychlosti jsou na ethernetové síti jen teoretickými maximy. Spíše překvapuje malý rozdíl mezi třídami 1:3 a 1:4 i když mají přiděleny rozdílné rychlosti.

Stejné měření jsem provedl i s listovým řazením do front typu SFQ. Jelikož jsem ale měřil vždy s jedním tokem na jednu třídu, výsledky se nelišily. Naměřené rychlosti byly jen nepatrně pomalejší, pravděpodobně kvůli vyšší režii algoritmu SFQ.

5.3.5 Měření 5 – omezování provozu algoritmem RED

Podobné měření s listovým řazením do front typu RED dokazuje, že je možné dosáhnout lepších výsledků. Důležité je citlivě nastavit parametry algoritmu RED, které jsou uvedeny v konfiguraci v příloze.



	Třída 1:2	Třída 1:3	Třída 1:4
Aritmetický průměr [KB/s]	366,1	151,0	147,2
Směrodatná odchylka	6,87	15,55	16,00

tabulka 7 – měření 5

K nastavení lepších parametrů výrazně pomohly statistiky, které se vypisují následujícím příkazem:

```
# tc -s qdisc ls dev ppp0
```

5.3.6 Měření 6 – odezva

Doba odezvy byla měřena programem `ping`. Výše používané konfigurace tady nefungovaly, protože filtry u32 byly nastaveny jen na klasifikaci TCP paketů. Proto jsem musel použít jiný typ filtrů a to směrovací.

Konfigurace opět obsahuje tři listové třídy. Dvě jsou používány pro zatížení linky (dva TCP toky) a jedna pro testování odezvy. Velikost posílaných paketů činila 64 bajtů a perioda posílání pak dvě sekundy. První testování probíhalo bez jakékoliv konfigurace kontroly provozu, druhé pak s listovým řazením do front FIFO a třetí použitím algoritmu TBF. Tady bylo opět nutné citlivě nastavit parametry, aby bylo dosaženo přijatelnějších výsledků.

	Bez konfigurace	Fronta FIFO	Fronta TBF
Aritmetický průměr [ms]	615	421	383
Směrodatná odchylka	172	99	63

tabulka 8 – měření 6

Konfigurací řazení CBQ a přiřazením priorit se dosáhne poměrně výrazného snížení doby odezvy. Výrazně se také sníží odchylka od průměru, doba odezvy je stabilnější. Rozdíl mezi algoritmy FIFO a TBF už tak výrazný není, nicméně je pozorovatelný.

Připomínám, že rychlost zbylých dvou TCP toků se při měření odezvy nezměnila. Počet dat v testované prioritní třídě byl tak malý, že prostředky přidělené dalším dvěma se prakticky nezmenšily.

5.4 Výsledky měření

Nejprve se zaměříme na sériovou linku jakožto zástupce vytáčené linky. Vytáčení v testovacích podmínkách není třeba, protože nemá vliv na provoz na lince.

Měření ukázalo, že výsledné hodnoty šířky pásma odpovídají proporcionálně hodnotám zadaným v konfiguraci tříd CBQ. Graf u měření 2 ukazuje, že požadovaná šířka pásma se alokuje prakticky bezprostředně po ustanovení spojení, což je netriviální požadavek. Přidělená šířka pásma je velice záhy velmi stabilní v průměru. Oproti tomu, absolutní rychlost vykazuje velmi vysokou směrodatnou odchylku. Patrně je to důsledek algoritmu WRR, který sleduje počet přenesených dat v jednotlivých třídách. Měření 3 ukazuje, jak se jedna třída chová k více souběžným spojením. Vyjma efektu popsaného výše, tady žádné anomálie sledovány nebyly.

Měření 6 se zaměřilo na dobu odezvy na zatížené sériové lince. Použité konfigurace latenci poměrně výrazně snižují. Dalším testováním a nastavováním parametrů listových řazení do front by patrně šlo dosáhnout ještě lepších výsledků.

Měření 4 a 5 byla prováděna na ethernetové lince. Tady byla testována tři řazení do front – FIFO, SFQ a RED. Algoritmus RED přináší jisté zvýšení rychlosti. Naproti tomu u SFQ žádný přínos pozorován nebyl kvůli způsobu měření s malým počtem spojení.

6 Závěr

Tato práce by měla sloužit (zatím jen česky rozumějícím) administrátorům sítí v lepši orientaci při konfiguraci kontroly provozu. Popis programu `tc` je poměrně detailní a v současné době nemá obdoby ani v anglicky psané dokumentaci.

Poměrně rozsáhlá část je věnována filtru u32. Princip jeho fungování a konfigurování se dal zjistit z konkrétních konfigurací a studiem zdrojových textů. Uváděný popis dává čtenáři podrobný návod, jak dosáhnout požadovaných cílů. Stále ale zůstaly některé možnosti filtru neodkryty, jejich význam je zdá se jen optimalizační.

Řazení do front dsmark a ingress jsou pevně spjata s architekturou DiffServ. V práci je podán princip fungování a nástin použití.

Praktická část přiblíží čtenáři použití konkrétních konfiguracích v daných situacích. Nastavení konečných parametrů je pak experimentální záležitost. V případech, kde je možný alespoň odhad parametru, jsem uvedl přibližný vzorec. Ten samozřejmě nelze brát doslova.

Měření prokázala výrazné zlepšení požadovaných aspektů kvality služby. Možnosti kontroly provozu v operačním systému Linux sahají dále než mohly ukázat tyto konfigurace. Ale i tak bylo možné ukázat cestu, jak používat především řazení CBQ.

Další směry práce uvádím v bodech:

- podrobněji popsat řazení do front ingress a dsmark. Já jsem uvedl jen princip fungování a bylo by přínosné zdokumentovat jejich praktické použití. V balíku `iproute2` se nacházení komentované příklady a tak by to nemělo představovat větší problém.
- pochopit rozhraní mezi uživatelskými programy (dnes jen `tc`) a jádrem. Na základě těchto poznatků pak vytvořit srozumitelné API. Jestli autor v tomto směru neposkytne žádnou dokumentaci, bude poměrně těžké tyti vazba zmapovat. Nicméně se projekty na toto téma začínají objevovat.
- přepracovat program `tc` do lepší uživatelské podoby. Jak je zřejmé z textu výše, ovládání kontroly provozu je v Linuxu složité. Program `tc` by mohl býti napsán tak, aby přecházel chybám uživatele a nezatěžoval jej redundantními informacemi.
- detailněji popsat stávající implementace CBQ, především v systémech Solaris a FreeBSD
- vysvětlit spolupráci CBQ a protokolu RSVP.

7 Literatura

[1] Radhakrishnan, Saravanan: Quality od Service Support in Linux, The University of Kansas, 25.12.1998

[2] Almesberger, Werner: Linux Traffic Control – Implementation Overview, 30.11.1998

[3] Floyd, Sally; Van Jacobson: Random Early Detection Gateways for Congestion Avoidance, 1993, IEEE/ACM Transactions on Networking

[4] Krawczyk, Pawel: Sterowanie przeplywem danych w Linuxie 2.2, 1999

[5] Floyd, Sally; Van Jacobson: Link-sharing and Resource Management Models for Packet Networks, IEEE/ACM Transactions on Networking, Vol. 3 No., 4. srpen 1995; `http://www-nrg.ee.lbl.gov/papers/link.pdf`

[6] Floyd, Sally: Notes on Class-Base Queueing: Setting Parameters, únor 1996, `ftp://ftp.ee.lbl.gov/papers/params.ps.Z`

[7] Floyd, Sally: Notes on CBQ and Guaranted Service, 1995

[8] The Linux Traffic Shaper; `http://www.lwn.net/1998/1119/shaper.html`

[9] Kuznetsov, Alexey N.: IP Command Reference, duben 1999 (součást iproute2)

[10] IP QoS FAQ, Stardust Forum, `http://www.qosforum.com/`

[11] Hubert, Bert; Maxwell, Gregory; Oosterhout, Martijn: Linux 2.4 Advance Routing HOWTO, verze 0.0.2

[12] Gopi, Vaddi; Pramodh, Malipatna: An API for Linux QoS Support, The University of Kansas, 1999

[13] Introduction: Quality of Service Overview, Dokumentace Cisco Systems, 1999, `http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/qos_c/qcintro.htm`

[14] Plannig Quality of Service, Dokumentace Cisco Systems, 1999, `http://www.cisco.com/univercd/cc/td/doc/product/rtrmgmt/ciscoasu/class/qpm1_1/using_qo/c1plan.htm`

[15] Warhol, Andy: Od A k B a zase zpět

7.1 Normy IETF

[16] Differenticated Services (diffserv), `http://www.ietf.org/html.charters/diffserv-charter.html`

[17] Integrated Services (intserv), `http://www.ietf.org/html.charters/intserv-charter.html`

[18] Integrated Services over Specific Link Layers (issll), `http://www.ietf.org/html.charters/issll-charter.html`

[19] Resource Reservation (rsvp), `http://www.ietf.org/html.charters/rsvp-charter.html`

[20] RSVP Admission Policy (rap), `http://www.ietf.org/html.charters/rap-charter.html`

[21] IP Next Generation (ipngwg), `http://www.ietf.org/html.charters/ipngwg-charter.html`

7.2 Normy DiffServ

[22] RFC 2698, A Two Rate Three Color Marker. J. Heinanen, R. Guerin. Září 1999.

[23] RFC 2697, A Single Rate Three Color Marker. J. Heinanen, R. Guerin. Září 1999.

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

[25] RFC 2598, An Expedited Forwarding PHB. V. Jacobson, K. Nichols, K. Poduri. Červen 1999.

[26] RFC 2597, Assured Forwarding PHB Group. J. Heinanen, F. Baker, W. Weiss, J. Wroclawski. Červen 1999.

[27] RFC 2475, An Architecture for Differentiated Service. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss. Prosinec 1998.
[28] RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. K. Nichols, S. Blake, F. Baker, D. Black. Prosinec 1998.
[29] RFC 2430, A Provider Architecture for Differentiated Services and Traffic Engineering (PASTE). T. Li, Y. Rekhter. Říjen 1998.

7.3 Normy IntServ

[30] RFC 2689, Integrated Services Mappings for Low Speed Networks. S. Jackowski, D. Putzolu, E. Crawley, B. Davie. Září 1999.
[31] RFC 2688, Integrated Services Mappings for Low Speed Networks. S. Jackowski, D. Putzolu, E. Crawley, B. Davie. Září 1999.
[32] RFC 2212, Specification of Guaranteed Quality of Service.
[33] RFC 2213, Integrated Services Management Information Base using SMIv2.
[34] RFC 2214, Integrated Services Management Information Base Guaranteed Service Extensions using SMIv2.
[35] RFC 2215, General Characterization Parameters for Integrated Service Network Elements.
[36] RFC 2216, Network Element Service Specification Template.

7.4 Specifikace RSVP

[37] RFC 2210, The Use of RSVP with IETF Integrated Services.
[38] RFC 2209, Resource ReSerVation Protocol (RSVP) -- Version 1 Message Processing Rules.
[39] RFC 2208, Resource ReSerVation Protocol (RSVP) -- Version 1 Applicability Statement Some Guidelines on Deployment.
[40] RFC 2207, RSVP Extensions for IPSEC Data Flows
[41] RFC 2205, Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification.

7.5 Diskuzní skupiny

[42] cz.comp.linux
[43] fa.linux.net

8 Slovník pojmů

class ..... třída
classifier ..... třídidlo, sada filtrů, klasifikátor
delay..... zpoždění
filter ..... filtr
flow..... tok
jitter..... změna latence
latency..... zpoždění, latence
metering ..... měření provozu
policing ..... sledování provozu, stráž
quality of service ..... kvalita služby
queuing discipline..... řazení do front
queuing..... řazení do front
router..... směrovač
traffic control..... kontrola provozu
traffic shaping..... omezování provozu
type of service..... typ služby

9 Seznam obrázků

OBRÁZEK 1 – PRŮCHOD PAKETU SÍŤOVÝM UZLEM 4
OBRÁZEK 2 – ALGORITMUS RED 6
OBRÁZEK 3 – NORMY INTSERV A DIFFSERV V LINUXU 8
OBRÁZEK 4 – OBECNÉ SCHÉMA ŘAZENÍ DO FRONT 10
OBRÁZEK 5 – PŘÍKLAD HIERARCHIE CBQ 12
OBRÁZEK 6 – SCHÉMA ŘAZENÍ DO FRONT INGRESS 16
OBRÁZEK 7 – SCHÉMA ŘAZENÍ DSMARK 21
OBRÁZEK 8 – FILTR TCINDEX 23
OBRÁZEK 9 – KONFIGURACE 1 29
OBRÁZEK 10 – KONFIGURACE 2 32

10 Seznam tabulek

TABULKA 1 – PŘEHLED JEDNOTEK RYCHLOSTI 13
TABULKA 2 – PARAMETRY SPECIÁLNÍHO FILTRU U32 20
TABULKA 3 – MĚŘENÍ 1 30
TABULKA 4 – MĚŘENÍ 2, SMĚRODATNÁ ODCHYLKA 31
TABULKA 5 – MĚŘENÍ 3 31
TABULKA 6 – MĚŘENÍ 4 32
TABULKA 7 – MĚŘENÍ 5 33
TABULKA 8 – MĚŘENÍ 6 34

11 Přílohy

11.1 Konfigurace 1

```
# kořenové řazení do front
tc qdisc add dev ppp0 root handle 1: cbq bandwidth 115Kbit allot
          1514 cell 8 avpkt 1500 mpu 64

# kořenová třída
tc class add dev ppp0 parent 1:0 classid 1:1 cbq bandwidth
          115Kbit rate 115Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 20 weight 11Kbit prio 2

# další třídy
tc class add dev ppp0 parent 1:1 classid 1:2 cbq bandwidth
          115Kbit rate 50Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 20 weight 4Kbit prio 3

tc class add dev ppp0 parent 1:1 classid 1:4 cbq bandwidth
          115Kbit rate 30Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 20 weight 3Kbit prio 3


tc class add dev ppp0 parent 1:1 classid 1:3 cbq bandwidth
          115Kbit rate 30Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 20 weight 3Kbit prio 3

#tc qdisc add dev ppp0 parent 1:2 handle 2: sfq perturb 8 quan-
          tum 1500
#tc qdisc add dev ppp0 parent 1:3 handle 3: sfq perturb 8 quan-
          tum 1500
#tc qdisc add dev ppp0 parent 1:4 handle 4: sfq perturb 8 quan-
          tum 1500

# filtry
tc filter add dev ppp0 parent 1:0 prio 8 protocol ip u32

tc filter add dev ppp0 parent 1:0 prio 8 handle 1: u32 divisor
          256

tc filter add dev ppp0 parent 1:0 prio 8 u32 ht 1:6: match ip
          dst 10.0.0.1 flowid 1:2

tc filter add dev ppp0 parent 1:0 prio 8 u32 ht 1:6: match ip
          dst 10.0.0.3 flowid 1:3

tc filter add dev ppp0 parent 1:0 prio 8 u32 ht 1:6: match ip
          dst 10.0.0.4 flowid 1:4

tc filter add dev ppp0 parent 1:0 prio 8 handle ::1 u32 ht 800::
          match ip nofrag offset mask 0x0F00 shift 6 hashkey mask
          0x00FF0000 at 8 link 1:
```

```
# nastavení směrování
ip route add 10.0.0.3 dev ppp0
ip route add 10.0.0.4 dev ppp0
ip route add 10.0.0.5 dev ppp0
```

11.2 Konfigurace 2

```
# kořenové řazení do front
tc qdisc add dev eth0 root handle 1: cbq bandwidth 10Mbit allot
         1514 cell 8 avpkt 1000 mpu 64

# kořenová třída
tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth
         10Mbit rate 10Mbit allot 1514 cell 8 avpkt 1000 max-
         burst 20 weight 1Mbit prio 2

# další třídy
tc class add dev eth0 parent 1:1 classid 1:2 cbq bandwidth
         10Mbit rate 5Mbit allot 1514 cell 8 avpkt 1000 maxburst
         20 weight 500Kbit prio 3 isolated bounded

tc class add dev eth0 parent 1:1 classid 1:3 cbq bandwidth
         10Mbit rate 3Mbit allot 1514 cell 8 avpkt 1000 maxburst
         20 weight 300Kbit prio 3 isolated bounded

tc class add dev eth0 parent 1:1 classid 1:4 cbq bandwidth
         10Mbit rate 2Mbit allot 1514 cell 8 avpkt 1000 maxburst
         20 weight 200Kbit prio 3 isolated bounded

# filtry u32
tc filter add dev eth0 parent 1:0 prio 8 protocol ip u32

tc filter add dev eth0 parent 1:0 prio 8 handle 1: u32 divisor
         256

# porty 6010, 6011 a 6012
tc filter add dev eth0 parent 1:0 prio 8 u32 ht 1:6: match tcp
         dst 0x177a 0xffff flowid 1:2

tc filter add dev eth0 parent 1:0 prio 8 u32 ht 1:6: match tcp
         dst 0x177b 0xffff flowid 1:3

tc filter add dev eth0 parent 1:0 prio 8 u32 ht 1:6: match tcp
         dst 0x177c 0xffff flowid 1:4

tc filter add dev eth0 parent 1:0 prio 8 handle ::1 u32 ht 800::
         match ip nofrag offset mask 0x0F00 shift 6 hashkey mask
         0x00FF0000 at 8 link 1:
```

11.3 Konfigurace 3

```
# dodatek k předchozímu
# listové řazení do front RED
tc qdisc add dev eth0 parent 1:2 handle 2: red limit 8000 min
         4000 max 7000 avpkt 1500 burst 10 probability 0.02
         bandwidth 5Mbit
```

```
tc qdisc add dev eth0 parent 1:3 handle 3: red limit 10000 min
          4000 max 9000 avpkt 1500 burst 10 probability 0.02
          bandwidth 3Mbit
tc qdisc add dev eth0 parent 1:4 handle 4: red limit 20000 min
          10000 max 18000 avpkt 1500 burst 10 probability 0.02
          bandwidth 2Mbit
```

11.4 Konfigurace 4

```
# kořenové řazení do front
tc qdisc add dev ppp0 root handle 1: cbq bandwidth 115Kbit allot
          1514 cell 8 avpkt 1500 mpu 64

# kořenová třída
tc class add dev ppp0 parent 1:0 classid 1:1 cbq bandwidth
          115Kbit rate 115Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 20 weight 11Kbit prio 1

# další třídy
tc class add dev ppp0 parent 1:1 classid 1:2 cbq bandwidth
          115Kbit rate 80Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 2 weight 10Kbit prio 1

tc class add dev ppp0 parent 1:1 classid 1:4 cbq bandwidth
          115Kbit rate 30Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 20 weight 3Kbit prio 8

tc class add dev ppp0 parent 1:1 classid 1:3 cbq bandwidth
          115Kbit rate 30Kbit allot 1514 cell 8 avpkt 1500 max-
          burst 20 weight 3Kbit prio 8

# testované řazení do front
tc qdisc add dev ppp0 parent 1:2 handle 2: tbf burst 1600b rate
          80Kbit latency 30ms


# směrovací filtry
tc filter add dev ppp0 parent 1:0 protocol ip prio 100 route
tc filter add dev ppp0 parent 1:0 protocol ip prio 100 route to
          2 classid 1:2
tc filter add dev ppp0 parent 1:0 protocol ip prio 100 route to
          3 classid 1:3
tc filter add dev ppp0 parent 1:0 protocol ip prio 100 route to
          4 classid 1:4

# směrovací pravidla
ip route add 10.0.0.1 dev ppp0 realm 2
ip route add 10.0.0.3 dev ppp0 realm 3
ip route add 10.0.0.4 dev ppp0 realm 4
```