Princip a užití HTB QoS disciplíny

Martin Devera

CDI computers s.r.o.
Email: devik@cdi.cz

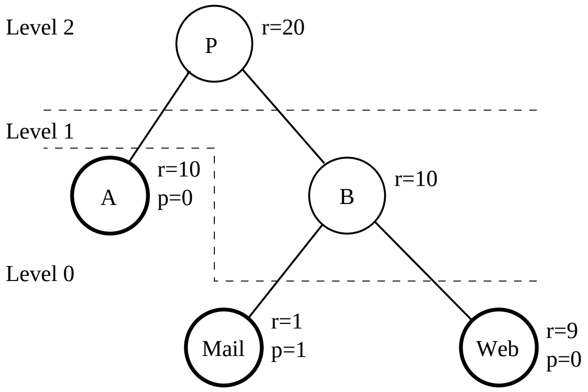
Abstrakt: Stále širší nasazení Linuxu jako routeru či firewallu zvyšuje zájem administrátorů o jeho QoS subsystém. Bohužel, uživatelé jsou často odrazeni složitostí nastavení a ne zcela zřejmými omezeními jak subsystému, tak zejména CBQ disciplíny. Tato přednáška si klade za cíl objasnit tato omezení a vysvětlit princip poměrně nové disciplíny HTB, která většinu uvedených problémů řeší.

Klíčová slova: SLT 2002, Linux, kernel, QoS, HTB, CBQ.

1 Úvodem

Tento článek popisuje impementaci nové QoS disciplíny HTB (Hierarchical Token Bucket) v Linuxu. Je předpokládáno, že čtenář je seznámen s administrací sítě v Linuxu a dokáže používat program `tc` pro nastavení Qdisců, tříd a filtrů. Tuto problematiku lze nastudovat v LARTC HOWTO na webové adrese http://www.lartc.org.

2 Hierarchické sdílení



Obrázek 1. Sdílení kapacity linky

Jan Kasprzak, Petr Sojka (editoři): SLT 2002 – sborník semináře o Linuxu a TEXu, str. 1–10, 2002.

Běžný problém, který administrátor řeší, je znázorněn na obrázku 1. Provider P potřebuje rozdělit internetové pásmo mezi firmy označené A a B. Každá firma má specifický požadavek na garantovaný datový tok r a maximální odezvy. Velikost odezvy přitom záleží na prioritě p třídy. Firma B si navíc přeje oddělit datové toky pro mailové a webové služby. Zavedeme si základní terminologii:

třída je uzlem sdílecího stromu a má definovaný požadovaný tok r,
uzel je pouze jiný název pro třídu,
list označuje uzel, který nemé žádné další následovníky, pouze listy mohou "obsahovat" pakety ve své interní frontě,
vnitřní uzel je uzel, který není listem,
plný list je list, který obsahuje nějaké pakety,
plný vnitřní uzel je uzel s potomkem typu plný list,
podlimitním uzlem rozumíme uzel, jehož aktuální tok je menší než jeho r, o podlimitnosti rozhoduje *estimátor*,
nespokojeným uzlem nazveme uzel, který je současně plný i podlimitní – jinými slovy tento uzel má "nárok", aby byl obsloužen,
úroveň uzlu (level) je "hloubka" uzlu, která je pro list definována jako 0. Způsob číslování je zřejmý s obrázku 1.

Cílem QoS disciplíny je pro každé vyvolání určit který další paket má být odeslán. Obecně vzato můžeme odeslat paket ze všech *nespokojených* listů nejépe v pořadí priorit. Pokud takový list není, pokračujeme *plnými* listy, které si mohou "půjčit" od svých předků.

V následujícím textu budeme sledovat, jak se která disciplína staví k problému "kdy si uzel může půjčit tok" a jakým způsobem se určí, zda je třída pod nebo nad svým limitem.

2.1 CBQ a půjčování

Autor CBQ ukázal, že jedním z pravidel které je nutno respektovat, je:

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

Pokud toto pravidlo nebude algoritmus respektovat, nastane situace, kdy listy s vyšší prioritou (A a Web) zahltí celou kapacitu linky a $Mail$ nedostane ani svůj garantovaný tok. Při respektování pravidla 1 toto nenastane, neboť $Mail$ je *nespokojeným* uzlem a ostatní si nemohou půjčovat z úrovně jiné než 0 (neboli nemohou si půjčovat vůbec).

Algoritmus CBQ implementuje toto pravidlo zavedením globální proměnné $toplevel$, která odráží stav X z pravidla 1. Proměnná je přepočítávána vždy při zafrontování nového paketu a při odebrání paketu. Proč, to je zřejmé, obě tyto akce ovlivní, zda je list *plný* či ne a definice *nespokojeného* uzlu (použitá v pravidle 1) závisí na *plnosti* listu.

Nespokojenost uzlu záleží také na aktuálním toku uzlu (zda je *podlimitní*). Uzel, který je nad limitem, se ovšem může stát *podlimitním* nejen během zafrontování či odebrání paketu, ale i po uplynutí dostatečné doby, kdy uzel nebyl využíván. Tato *asynchronní* událost není v “toplevel” algoritmu zohledněna a proto CBQ implementace které “toplevel” využívají[1], jsou zatíženy jistou chybou.

2.2 CBQ estimátor

Originální CBQ práce ([1]) nedefinuje jaký estimátor se má použít. Avšak stejně jako téměř každá implementace používá “toplevel” (viz odstavec 2.1), tak i ve volbě estimátoru neprokázali implementátoři příliš invence a použili stejný algoritmus navržený autory CBQ.

Tento algoritmus detekuje limitní stav třídy měřením doby, která uplyne mezi odesláním dvou za sebou následujících paketů (patřících měřené třídě). Čas se porovná s očekávanou dobou (vypočítanou z požadované rychlosti) a znaménko získané odchylky[2] určuje, zda je třída nad či pod limitem.

Pomineme-li složitost měření mezipaketové mezery s dostatečnou přesností (řádově mikrosekundy), zjistíme, že tento algoritmus potřebuje ke své funkci znát přesnou rychlost fyzického rozhraní. Není problém určit rychlost Ethernetového rozhraní, ale je to nemožné u zařízení jako je Wireless LAN nebo softwarová zařízení (tunely). Podobná zařízení totiž nemají pevnou rychlost odesílání dat ale jejich aktuální rychlost kolísá a závisí na mnoha podmínkách (kvalita signálu, zatížení routeru ...). Navíc nastavení takového estimátoru není vůbec jednoduchá záležitost a autoři CBQ sepsali dokument [2], který pojednává výhradně o tom jak estimátor nastavit.

3 HTB

Výše uvedené nesnáze byly prvotním impulzem pro vývoj nové disciplíny, která nevykazuje žádný podobný problém. Vývoj prošel během dvou let třemi fázemi, během každé byl algoritmus kompletně přepsán. Následující řádky se týkají verze 3.

subsectionHTB estimátor HTB dělí třídy nejen na *podlimitní* a *nadlimitní*. Místo toho budeme stav z hlediska aktuální velikosti toku označovat barvami semaforu. Každá třída má krom garantovaného toku r i maximální tok c[3].

Nyní nadefinujeme barvu třídy pro daný časový okamžik t a aktuální tok $a(t)$ jako

zelenou pokud platí $a(t) < r$, tedy třída je pod limitem,

žlutou pokud $c \geq a(t) \geq r$, taková třída je sama nad limitem, ale nedosáhla stropu, tudíž si může zkusit půjčit od rodiče, nebo

červenou pro $a(t) > c$. Červená třída přesáhla strop, a proto již nemůže odeslat žádný paket.

[1] Veškeré implementace které mi jsou známé využívají “toplevel”.

[2] Ve skutečnosti se odchylka ještě exponenciálně průměruje ale to není pro nás podstatné

[3] Angl. **c**eil = strop.

Implementace estimátoru využívá tzv. Leaky Bucket (Dále jen LB). Výhodou je tolerance vůči rozlišení systémového časovače a zejména nezávislost na fyzické rychlosti rozhraní. Jak vidíme, použití LB řeší oba problémy z odstavce 2.2. LB je definován dvěma parametry, datovým tokem r (v bajtech za sekundu) a *burstem* $burst$ (v bajtech). Běžně má takový datový tok rychlost r. Pokud je ovšem $a(t) < r$, pak si LB "zapamatuje", kolik bajtů bylo nevyužito. Maximální velikost této "paměti" je právě $burst$. Pokud je požadavek na přenesení dat nad limit r, LB v každou chvíli umožní přenést až tolik bytes bez omezení rychlosti, kolik jich je právě "zapamatováno".

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

$$burst \geq \Delta t \times r \tag{1}$$

Utilita `tc` pro HTB s tím počítá, a pokud $burst$ neuvedete, nastaví automaticky nejmenší možnou hodnotu pro daný systém podle (1).

Pro implementaci stropu (c) musí mít každá třída přiřazeny dva LB, jeden pro garantovaný tok r a druhý pro c. Abychom se v textu lépe vyznali, budeme *burst* příslušný ke "stropovému" LB označovat $cburst$.

3.1 HTB hierarchie a půjčování

Definujme algoritmus pro výběr dalšího paketu k odeslání

Algoritmus 2 *Ze všech plných listů volme takový, který by si při odesílání paketu mohl půjčit tok od rodiče na nejnižší úrovni. Pokud je takových listů více, vybereme ten s nejvyšší prioritou. Pokud máme stále více listů, střídáme je pravidelně podle poměrů jejich* r.

Uvědomte si prosím, že list si může korektně půjčit sám od sebe ! Algoritmus 2 splňuje pravidlo 1 ze strany 2. Důkaz je triviální. Krom toho algoritmus 2 splňuje očekávání, která na podobný systém klademe

- Toky tříd se stejnou prioritou si půjčují tok od společného rodiče v poměru jejich r.
- Třída s vyšší prioritou získá tok od společného rodiče přednostně.
- Třída s vyšší prioritou má kratší odezvy.
- Garantované toky jsou vždy splněny.

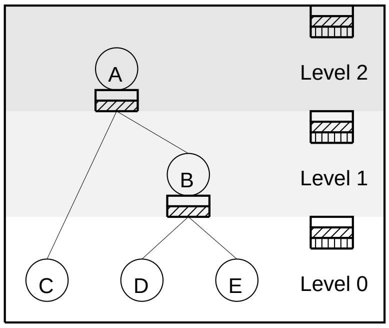
3.2 Implementace výběru paketu

Nyní zůstává pouze otázka, zda a jak lze algoritmus 2 efektivně implementovat. Během výběru paketu musíme najít *zelenou* třídu s nejnižší úrovní, která má jako potomka plný list a k potomkovi "vede" cesta se *žlutými* třídami (včetně listu).

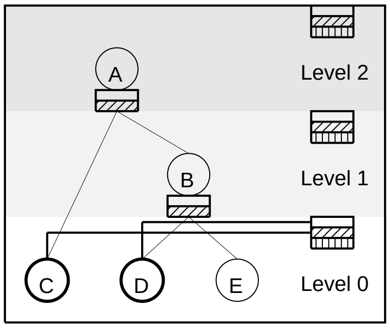
Není možné toto hledání provést průchodem přes všechny třídy během výběru paketu, neboť složitost by byla $O(N)$, kde N je počet tříd, a to je neúnosné.

Budeme tedy muset udržovat informace o aktuálním stavu a inkrementálně je upravovat. Podívejme se nyní na obrázek 2. Pro účel vysvětlení algoritmu jsme pozměnili označení tříd. List je znázorněn tučnou kružnicí pokud je *plný*, a vnitřek kruhu je prázdný pokud je třída *zelená*, jednoduše šrafovaný pokud je *žlutá*. Dvojitě šrafovaná je *červená* třída.

Každý vnitřní uzel si udržuje seznam potomků, kteří by si od něj rádi půjčili nějaký tok. Podle definice to mohou být pouze plné *žluté* třídy. Tento seznam (tzv. *interní půjčovací seznam*) musí být veden zvlášť pro jednotlivé priority, neboť uzel může být předkem listu s libovolnou prioritou. Naše příklady mají pouze dvě priority, vysoká s interním seznamem znázorněným prázdným obdélníčkem pod uzlem a nízká (šrafovaný obdélníček o něco níže).



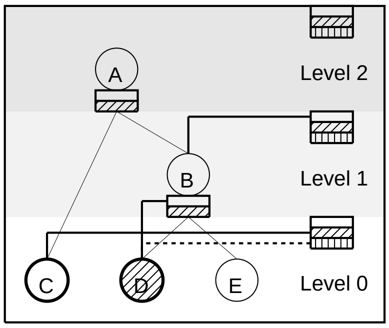
Obrázek 2. Žádné pakety, třídy jsou *zelené*, pouze D ma vyšší prioritu



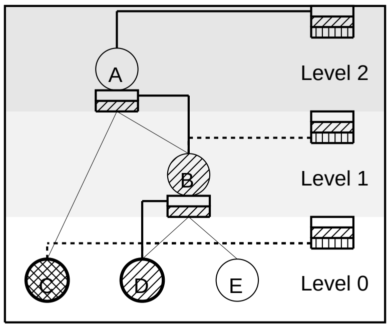
Obrázek 3. Přišly pakety pro C a D

V pravé části obrázku vidíme podobné obdélníčky. Ignorujme prozatím nejspodnější (svisle šrafovaný). Dva horní tvoří tzv. *globální půjčovací seznam*. Seznam obsahuje třídy určité úrovně a priority, které jsou *plné* a zároveň *zelené* (tedy *nespokojené*). Jinými slovy, třídy, které jsou na jakémkoli globálním seznamu, jsou připraveny odeslat paket. Nyní tedy v souladu s algoritmem 2 stačí vybrat nejspodnější úroveň s neprázdným globálním seznamem, zde seznam s nejvyšší prioritou, a sledovat interní seznamy, které nás dovedou až k listu, který má odeslat další paket.

Obrázek 2 ukazuje stav, kdy v HTB není žádný paket a všechny třídy jsou *zelené*. Pokud přijdou pakety určené listům C a D, listy se změní na *plné* a protože jsou zatím i *zelené* (jsou pod limitem), můžeme je připojit do globálních seznamů pod patřičné priority (obrázek 3). Pokud by si síťová karta nyní řekla o paket, velmi rychle zjistíme, že máme poslat z listu D a případný další z C (pokud D již bude *prázdný*). Podívejme se, co by se stalo pokud tedy pošleme paket z D a estimátor rozhodne, že D je *žlutá* (došlo k překročení r). Tento stav vidíme na obrázku 4. List D byl odpojen od globálního listu neboť již nemůže odesílat bez půjčky a zároveň byl zařazen na vysokoprioritní interní seznam třídy B. Třída B sama je zařazena na globální list úrovně 1, vysoké priority. To proto, aby třída D mohla nadále posílat pakety výpůjčkami od B.



Obrázek 4. D je nyní *žlutá*

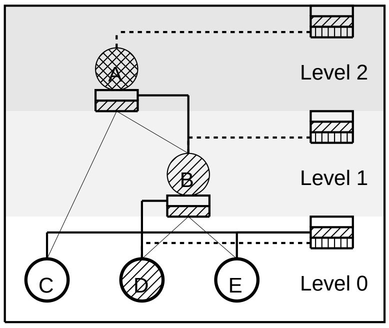


Obrázek 5. C je *červená* a zároveň B je teď *žlutá*

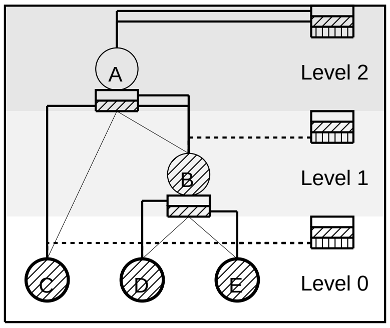
Protože, pokud *D* po nějakou dobu neodešle paket, tak se její barva v určitém čase změní opět na *zelenou*, zařadíme *D* na speciální *čekací seznam*. Každá úroveň má svůj čekací seznam a zařazení je naznačeno čárkovanou čarou. Po uplynutí předem spočítaného času se změní barva třídy. Čekací seznam řeší problém CBQ, popsaný v odstavci 2.1.

Zde si ale povšimněme důležitého faktu – při žádosti o další paket bude vybrána třída *C* nikoli *D* a to i přesto, že *D* má vyšší prioritu. Je to proto, že v globálním seznamu začíná cesta k *C* na nižší úrovni. Tento krok je vlastně implementací pravidla 1 ze strany 2. Proč by měla dostat *D* přednost, když *C* má nárok na svůj garantovaný tok.

Obrázek 5 zobrazuje situaci po odeslání paketu z *C*. Řekněme, že tato třída překročí *c* a je tudíž *červená*. Navíc *B* je nyní *žlutá*. Vidíme, jak se *B* připojí na interní seznam *A*, a místo *B* je v globálním seznamu nyní *A*. Toto již známe. Nové je pro nás chování *červené* třídy. Jak vidno, *C* je pouze na čekacím seznamu (nemůže sama poslat paket ani si půjčit). Jediná třída, která může posílat je *D*, jak je zřejmé z globálního seznamu. Na obrázku 6 vidíme situaci, kdy *A* je



Obrázek 6. A je *červená*, a C i E jsou *plné* a *zelené*



Obrázek 7. Všechny listy si půjčují od A

červená, a *C* i *E* jsou *plné* a *zelené*. Zde lze pozorovat, že cesta po interních seznamech existuje, i když nejvyšší třída nemá žádný spoj na globální seznam. Navíc *C* i *E* jsou připojeny ke společnému globálnímu seznamu. V tomto případě je nutno pravidelně střídat odebírání paketu mezi *C* a *E* v poměru jejich r. To je v implementaci zajištěno implementací seznamů pomocí *RB-tree*, kde klíčem je $classid$ třídy. Pro seznam si HTB pamatuje $classid$, které bylo naposledy použito a díky perzistenci $classid$ je střídání férové i při změnách barev a tedy i obsahu seznamů. Váhování podle r je řešeno pomocí *DRR* [3].

Obrázek 7 poukazuje na možnost napojení více tříd na společné interní seznamy. Nebýt vyšší priority *D*, byli bychom nuceni střídat odebírání paketu mezi všemi listy. Neboť *D* ale vyšší prioritu má, bude z globálního seznamu vybrána pouze *D*.

4 Praktické použití HTB

HTB je již maturovaný systém a v současnosti je vložen ve verzích Linuxového kernelu 2.4.20pre a posledních kernelů řady 2.5. Upravený nástroj `tc` by měl být k dispozici koncem roku 2002 v oficiálním balíku `iproute2`, do té doby je nutno stáhnout `tc` z http://luxik.cdi.cz/˜devik/qos/htb/. Je součástí TAR s poslední verzí HTB.

Pro detailní vysvětlení syntaxe a nuancí nastavení viz. manuál na výše uvedené adrese.

4.1 Jednoduchá konfigurace

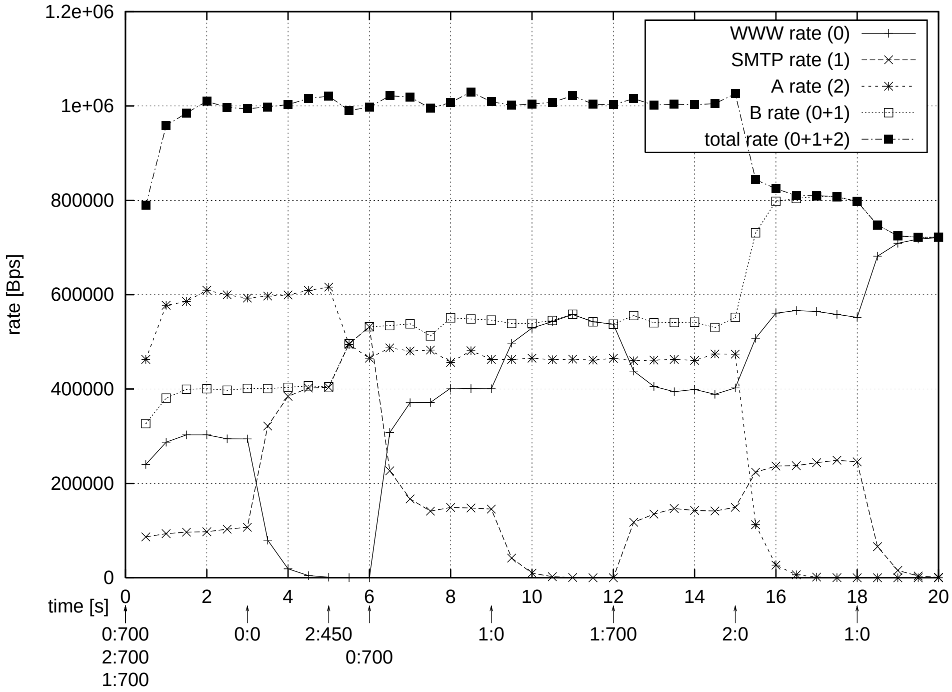
Ukážeme si triviální konfiguraci a její výsledné toky. Zadáme tyto příkazy

```
tc qdisc add dev eth0 root handle 1: htb default 12

tc class add dev eth0 parent 1: classid 1:1 htb \
   rate 1000kbps ceil 1000kbps
tc class add dev eth0 parent 1: classid 1:2 htb \
   rate 1000kbps ceil 800kbps
tc class add dev eth0 parent 1:2 classid 1:10 htb \
   rate 300kbps ceil 1000kbps prio 1
tc class add dev eth0 parent 1:2 classid 1:11 htb \
   rate 100kbps ceil 1000kbps prio 1
tc class add dev eth0 parent 1:1 classid 1:12 htb \
   rate 600kbps ceil 1000kbps prio 1

tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
   match ip src 1.2.3.4 match tcp dport 80 0xffff flowid 1:10
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
   match ip src 1.2.3.4 flowid 1:11
```

První řádek vytvoří instanci disciplíny HTB, další čtyři řádky sestaví hierarchii podobnou té na obrázku 1. Webové pakety z adresy 1.2.3.4 půjdou do listu 1:10, ostatní pakety z adresy 1.2.3.4 do 1:11 a ostatní do 1:12 (to zajistí *default 12* z první řádky). Výsledný tok je na obrázku 8. Kódy zobrazené pod grafem indikují



Obrázek 8. Výsledný datový tok

místa, kde se změnil vstupní datový tok. Např. 0:700 znamená, že třída 0 (1:10) má od té chvíle vstupní tok 700 kbytes.

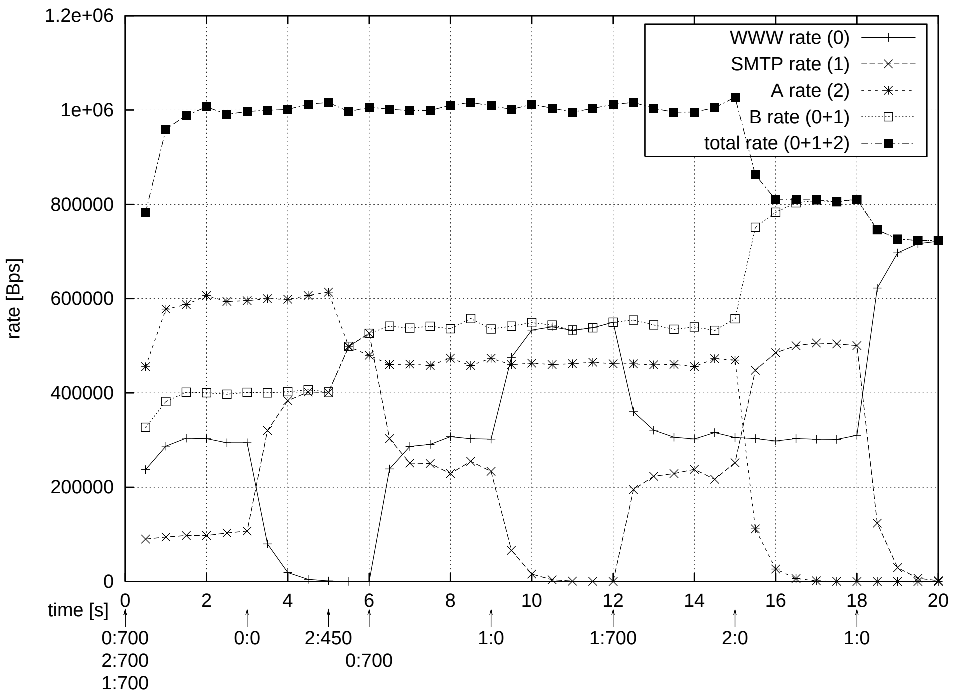
Můžeme sledovat jak postupné vypínání jednotlivých paketových zdrojů způsobí přerozdělení toků. V čase 3s jsme vypnuli tok 0 a z grafu je zřejmé, že jeho pásmo si zabral tok 1. Proč kousek nezískal i tok 2 ? Protože uzel B (rodič tříd pro toky 0 a 1) má garantováno 400kbps.

V čase 8s využívá tok 2 pouze 450kbps, zbývajících 150kbps si rozdělili toky 0 a 1 v poměru jejich r. Pokles celkováho toku na 16s je způsoben hodnotou $ceil$=800kbps u třídy 1:2 (uzel B).

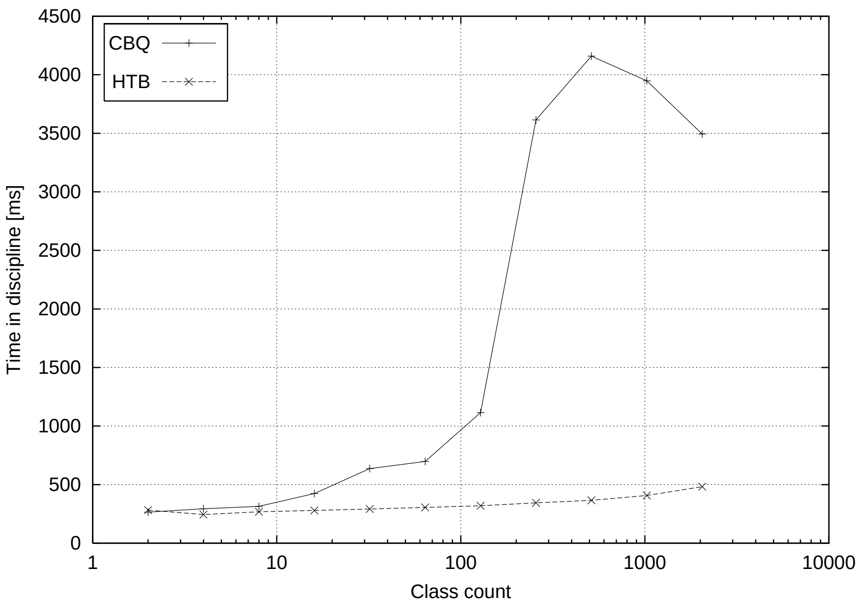
Ukážeme si ještě co se stane když zvýšíme prioritu třídy 1:11 pomocí

```
tc class change dev eth0 parent 1:2 classid 1:11 htb \
   rate 100kbps ceil 1000kbps prio 0
```

(nižší číslo u $prio$ znamená vyšší prioritu). Na obrázku 9 je znatelný pokles toku 0 v časech 8s a 14s – prostor získaný poklesem toku 2 si nyní vzala kompletně SMTP třída 1 a pro 0 zůstává pouze její garantovaný tok.



Obrázek 9. Datový tok po zvýšení priority u SMTP



Obrázek 10. Závislost spotřebovaného CPU času pro 5 sekundový testovací běh na 30MBit

5 Výsledky

HTB bylo podrobeno mnoha testům výkonu i stability. Z měření výkonu vyplývá, že HTB škáluje dobře i nad 5000 plných tříd v jediné konfiguraci a jeho rychlost je přitom stále vyšší než u CBQ. Na grafu 10 je jasně vidět srovnání škálovatelnosti HTB a CBQ. Zploštění křivky u CBQ u hranice 200 tříd je způsobeno nedostatkem CPU cyklů.

Vezmeme-li v úvahu i čerstvé zařazení HTB do Linuxového kernelu a jednoduchost nastavení z hlediska uživatele, je pravděpodobné že HTB plně nahradí CBQ disciplínu.

Reference

1. Sally Floyd, Van Jacobson, *Link-sharing and Resource Management Models for Packet Networks*, IEEE/ACM Transactions on Networking, Vol. 3 No. 4, August 1995
2. Sally Floyd, *Notes on Class-Based Queuing: Setting Parameters*, LBNL, February 1996
3. M. Shreedhar, George Varghese, *Efficient Fair Queuing using Deficit Round Robin*, Washington University, October 1995