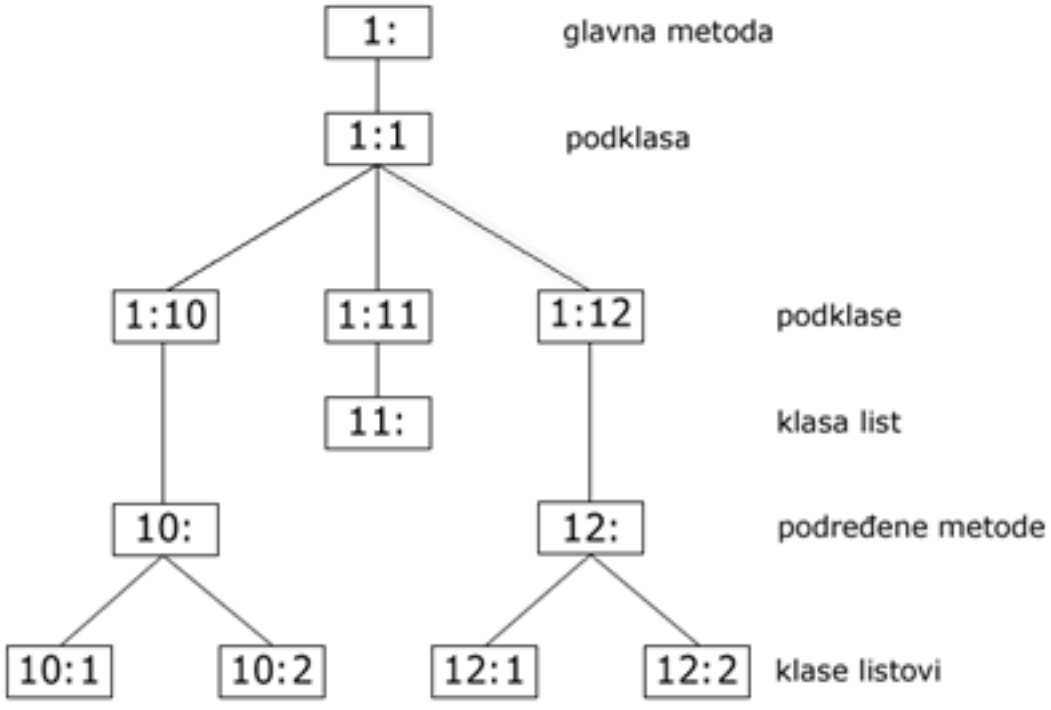
# Filtriranje mrežnog prometa na aplikacijskoj razini pomoću Linux rješenja - 3.dio

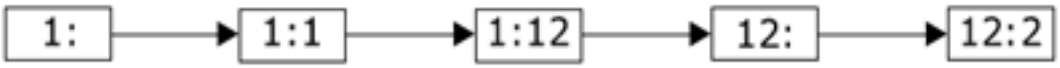
Kod Linux operativnih sustava kontrola mrežnog prometa (ulazak paketa u redove čekanja) je bazirana na klasama, tj. metoda upravljanja može sadržavati klase (eng. *classful qdisc*). Taj je princip vrlo koristan u slučaju da je promet na mrežnom sučelju različite vrste i da je svaku vrstu potrebno oblikovati na željeni način. Upotreba metoda koje koriste klase je vrlo fleksibilna i to na način da klase mogu sadržavati jednu ili više podređenih klasa ili jednu metodu za upravljanje. Dalje, podređene klase također mogu imati svoje podređene klase ili jednu metodu za upravljanje, što dovodi do moguće vrlo kompleksnog sustava za upravljanje mrežnim prometom. Klase koje nemaju svojih podređenih klasa zovu se klase listovi (eng. *leaf classes*) i one imaju dodijeljenu samo metodu za upravljanje. Također, klasi se može dodijeliti jedan ili više filtara kojima se omogućava usmjeravanje mrežnog prometa na određenu pod klasu ili se obavlja njegova selekcija (njegova ponovljena klasifikacija ili brisanje). Slučaj kad se mrežni promet usmjerava na neku od podređenih klasa se zove klasificiranje mrežnog prometa (eng. *classify*). Većina metoda koje koriste klase također omogućavaju i oblikovanje mrežnog prometa, što je vrlo korisno ako se želi kontrolirati brzina i raspoređivati redoslijed toka (npr. pomoću metode SFQ). Oblikovanje se najčešće koristi u slučaju da je izlazno mrežno sučelje velike brzine (npr. mrežna kartica) spojeno na mrežno sučelje male brzine (npr. kabelski modem). Tada se brzina izlaznog sučelja smanjuje na brzinu ulaznog sučelja i ne dolazi do zagušenja mrežnog prometa ili prioritiziranja nekog toka.

Svako mrežno sučelje ima jednu metodu za upravljanjem redom (eng. *root qdisc*) i unaprijed definirano, to je već opisana metoda pfifo_fast. Svakoj metodi i klasi se dodjeljuje jedinstvena oznaka (eng. *handle*) preko koje se u kasnijoj konfiguraciji pristupa toj metodi tj. klasi. Jedinstvene oznake se sastoje od dva dijela tj. broja, *major* i *minor* koji tvore jedinstvenu oznaku na sljedeći način: *<major>:<minor>*. Običaj je da se glavna metoda označi s oznakom '1'. Kako sve metode imaju *minor* broj '0', onda puna oznaka glavne metode glasi '1:0'. Klase imaju isti *major* broj kao i njihovi nadređeni objekt (metoda ili klasa). Tako *major* broj mora biti jedinstven za ulazni tj. izlazni red, a *minor* brojevi jedinstveni unutar metode i njenih klasa. Tipična hijerarhija može izgledati kao na slici 1.



Važno je naglasiti da paketi ulaze u red i izlaze iz njega samo u glavnoj metodi, to je jedini dio ovog stablastog prikaza s kojim komunicira jezgra operativnog sustava. Paketi ulaze u stablo na vrhu i onda se klasificiraju prema dnu da bi zatim izlazili prema vrhu redoslijedom određenim u samim metodama i njima pridruženim filtrima.

Npr. paket se može klasificirati u lancu na način prikazan na slici 2.

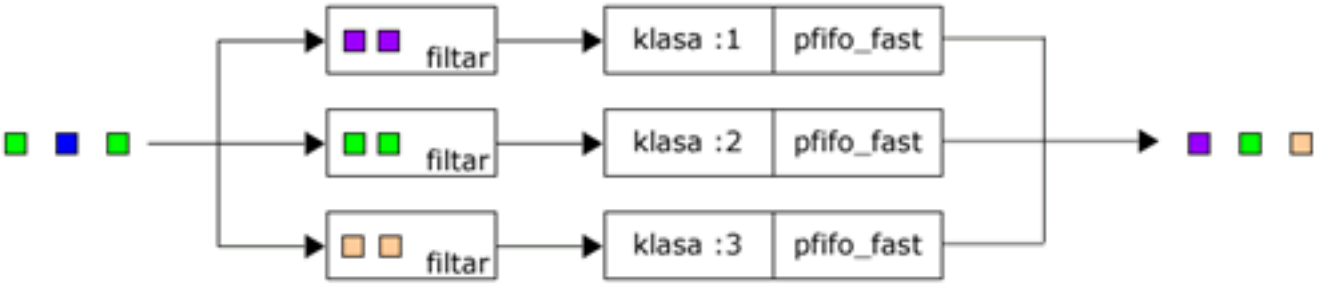



Ovdje je paket dodijeljen metodi u klasi 12:2. U ovom primjeru svaka grana stablastog prikaza ima svoj filtar i paket je postupno klasificiran. Također je moguće da paket bude klasificiran na drugi način, prikazan na slici 3.


U ovom se slučaju paket iz glavne metode usmjerava direktno u klasu 12:2, ali ovaj način je manje fleksibilan zbog nepostojanja postupnog klasificiranja.

**Prio metoda za upravljanje**

Ova metoda ne oblikuje mrežni promet nego ga samo klasificira na osnovu parametara definiranim u filtrima. Slična je besklasnoj metodi pfifo_fast s razlikom što umjesto tri podređena reda, unaprijed definirano ima tri klase koje sadrže FIFO metodu bez unutrašnje strukture, ali moguće ih je zamijeniti bilo kojom metodom. Ova je metoda korisna u slučaju da se određenom dijelu mrežnog prometa želi dodijeliti veći prioritet ne koristeći samo TOS polja u zaglavlju paketa, nego upotrebom svih prednosti klasa. Upravo zbog razloga što ne oblikuje mrežni promet, kao i metoda SFQ, ova je klasna metoda najbolja u slučaju kad je mrežno sučelje potpuno opterećeno ili se koristi unutar metode koja oblikuje mrežni promet. Shema metode prio je prikazana na slici 4.

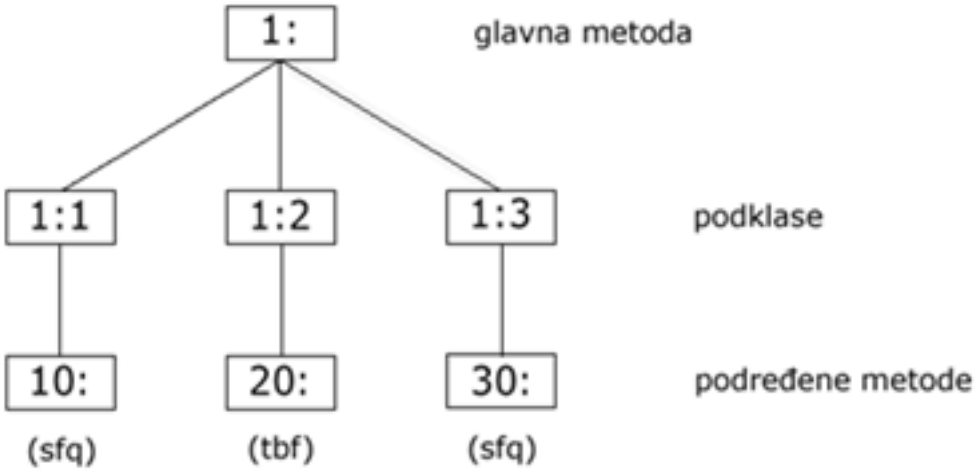



Prilikom napuštanja reda prvo se prazni podređena klasa 1. Kad se ona isprazni, podređena klasa 2 dolazi na red i na kraju se prazni podređena klasa 3.

Parametri dostupni uz prio metodu su:

- *bands* – broj klasa tj. filtara koji se kreiraju
- *priomap* – parametar usko povezan s *band* parametrom, ista funkcija kao i kod metode pfifo_fast

Za primjer konfiguracije kreirat ćemo stablo prikazano na slici 5.




Neklasificirani mrežni promet ćemo usmjeriti na klasu 30, a interaktivni mrežni promet na klase 10 i 20.

```
# tc qdisc add dev eth0 root handle 1: prio
## sljedece tri linije kreiraju klase 1:1, 1:2 i 1:3
# tc qdisc add dev eth0 parent 1:1 handle 10: sfq
# tc qdisc add dev eth0 parent 1:2 handle 20: tbf rate 20kbit buffer 1600 limit 3000
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
```

**Class Based Queuing (CBQ)**
Ovo je najkompleksnija metoda koja postoji i najteža za ispravno konfiguriranje. Razlog tome je nepreciznost algoritma koji ne prati najbolje rad Linux operativnog sustava. Ova metoda, osim što koristi klase, oblikuje mrežni promet, ali to radi dobro samo u određenim slučajevima. Metoda CBQ radi na način da osigurava dovoljnu neopterećenost mrežne veze i time se brzina mrežnog prometa svodi na konfiguriranu vrijednost. To se radi na način da metoda izračunava vrijeme koje bi trebalo proći između prosječnih paketa. Tijekom rada mjeri se efektivno vrijeme praznog hoda (vrijeme kad nema paketa koji pristižu na mrežno sučelje) i to pomoću EWMA (*Exponential Weighted Moving Average*) metode koja pretpostavlja da je zadnji paket koji je stigao na mrežno sučelje eksponencijalno važniji od prethodnog paketa. Izračunato vrijeme praznog hoda se oduzima od izmjerenog i dobiveni broj se zove *avgidle*. Kod savršeno izbalansirane mrežne veze *avgidle* iznosi nula, što znači da paket stigne točno u izračunatom intervalu. Kako je vrlo teško precizno izmjeriti vrijeme praznog hoda, CBQ ponekad potpuno promaši zadane vrijednost. Razlog više mogu biti različiti problemi poput loše implementiranog upravljačkog programa (*drivera*) mrežne kartice, nemogućnost sabirnice da propusti određenu količinu mrežnog prometa i slično.
Neopterećena mrežna veza uzrokuje vrlo velik pozitivan *avgidle*, što bi nakon dužeg vremena praznog hoda omogućilo neograničenu mrežnu propusnost. Da bi se to spriječilo, koristi se *maxidle* parametar.
S druge strane, mrežna veza koja je preopterećena ima negativan *avgidle* i ako on postane previše negativan, CBQ prestaje s radom na određeni period i tada se nalazi u *overlimit* stanju. Teoretski, u tom bi se slučaju CBQ trebala zagušiti točno onoliko vremena koliko je izračunato vrijeme između dva paketa, zatim propustiti jedan paket, ponovo ući u *overlimit* stanje i tako sve dok zagušenje ne prođe. U tu se svrhu koristi *minburst* parametar.
Parametri dostupni uz metodu CBQ su:

- *avpkt* – prosječna veličina paketa u oktetima, potrebna za izračunavanje *maxidle* parametra
- *bandwidth* – fizička pojasna širina mrežne veze, potreban za izračunavanje vremena praznog hoda
- *cell* – vrijeme potrebno da se paket pošalje može rasti u koracima, ovisno o veličini paketa. Npr. paketima veličine 800 i 806 okteta treba isto vremena da se pošalju. Ovaj parametar

određuje granularnost. Obično se postavlja na vrijednost 8, a mora biti potencija broja 2
- *maxburst* – u slučaju da *avgidle* broj ima vrijednost *maxidle*, ovoliki broj paketa će biti propušten prije nego se *avgidle* vrati na nulu. Vrijednost parametra *maxidle* je jedino moguće postaviti pomoću ovog parametra
- *minburst* – u slučaju *overlimit* stanja idealno je propustiti jedan paket unutar točno izračunatog vremena praznog hoda. Međutim, kako je kod Unix jezgri vrlo teško raspoređivati događaje unutar vremena od 10ms, bolja je situacija biti u zagušenju duže vremena, zatim propustiti *minburst* broj paketa pa onda ući u stanje praznog hoda isto toliko vremena
- *minidle* – vrijednost na koju se postavlja *avgidle* ukoliko postane premalen (da bi se spriječilo ispadanje mrežne veze)
- *mpu* – na ethernetu paketi veličine nula okteta se postavljaju na veličinu od 64 okteta i kao takvi uzimaju vrijeme potrebno za slanje. Ovaj parametar služi za precizno izračunavanje vremena praznog hoda
- *rate* – parametar brzine

Osim samog oblikovanja mrežnog prometa, metoda CBQ radi istim načinom kao i klasna metoda prio, što znači da klase unutar metoda imaju prioritete. Svaki put kad se zatraži paket, započinje težinski Round Robin proces (*Weighted Round Robin - WRR*) i to s klasom najvećeg prioriteta. Klase se grupiraju i ispituju da li u njima postoje paketi koji čekaju. Ako postoje, onda se te klase prve obrađuju. Parametri koji su dostupni uz WRR proces su:

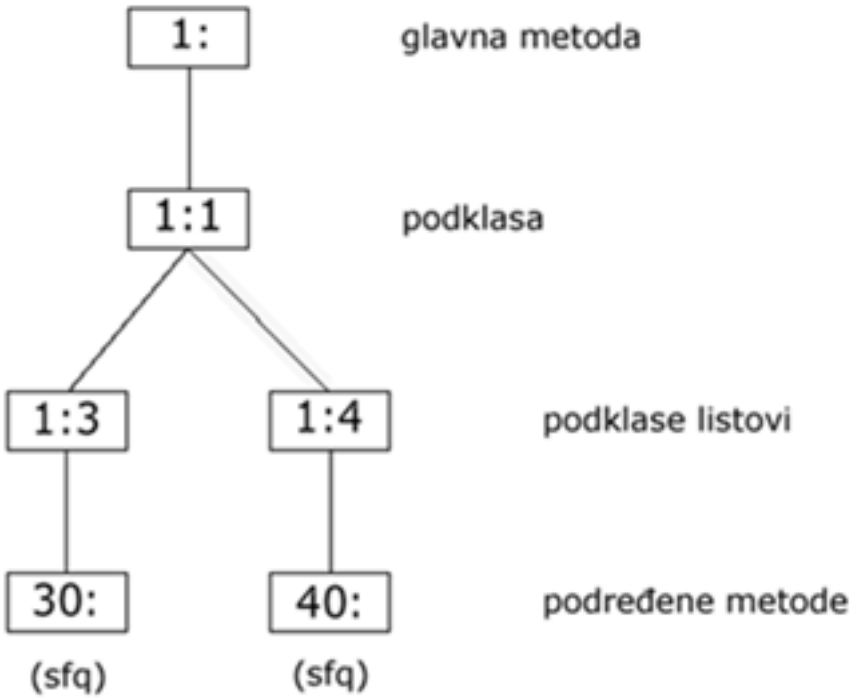
- *allot* – prilikom obrade klasa po prioritetima svaka klasa može poslati određenu količinu podataka. Ovaj parametar je osnovna jedinica te veličine, a koristi se zajedno s *weight* parametrom
- *prio* – parametar koji određuje prioritet
- *weight* - parametar koji se množi s parametrom *allot* da bi se dobila količina podataka koju svaka klasa može poslati kad dođe na red

U svakoj metodi CBQ postoje mehanizmi koji omogućavaju fino podešavanje rada. Tako se npr. ne ispituju klase u kojima nema paketa koji čekaju, a klasama koje su u *overlimit* stanju se smanjuje prioritet.

Jedna od naprednijih mogućnosti metode CBQ jest mehanizam posuđivanja pojasne širine (eng. *bandwidth*). Moguće je specificirati neku klasu koja će posuditi pojasnu širinu od neke druge klase, kao i da jedna klasa posudi dio svoje pojasne širine drugoj klasi. Parametri dostupni uz ove mogućnosti su:

- *isolated/sharing* – klasa koja je konfigurirana kao *isolated* neće posuditi svoju rezerviranu pojasnu širinu nekoj drugoj klasi. Obratno, klasa kojoj je postavljen parametar *sharing* će posuditi svoju pojasnu širinu
- *bounded/borrow* – klasa kojoj je postavljen parametar *bounded* neće posuđivati pojasnu širinu od neke druge klase. Obratno, klasa koja je konfigurirana kao *borrow* će pokušati posuditi pojasnu širinu od druge klase

Tipična situacija bi bila da na jednoj mrežnoj vezi postoje dvije klase od kojih je svaka i *isolated* i *bounded*. Na taj se način osigurava limitiranost svake klase na postavljene vrijednosti. Također je moguće da unutar neke klase koja nema mogućnost dijeljenja pojasne širine postoje druge klase koje mogu dijeliti svoju pojasnu širinu.
Slijedi primjer konfiguracije koja limitira mrežni promet web poslužitelja na 5Mb, SMTP mrežni promet na 3Mb, a zajedno ukupno ne mogu preći 6Mb. Mrežna kartica je pojasne širine 100Mb i klase mogu posuđivati pojasnu širinu jedna od druge. Shema primjera prikazana je na slici 6




Sljedeće dvije linije kreiraju glavnu metodu unutar reda (eng. *root qdisc*) i podređenu klasu 1:1. ta je podređena klasa postavljena na *bounded*, što znači da ukupni mrežni promet ne može preći zadanu vrijednost, u ovom slučaju 6Mb.

```
# tc qdisc add dev eth0 root handle 1:0 cbq bandwidth 100Mbit avpkt 1000 cell 8
# tc class add dev eth0 parent 1:0 classid 1:1 cbq bandwidth 100Mbit rate 6Mbit weigh
t 0.6Mbit
   prio 8 allot 1514 cell 8 maxburst 20 avpkt 1000 bounded
```

Sad se kreiraju dvije klase listovi koje su pomoću *weight* parametra ograničene na određene vrijednosti pojasne širine koju mogu koristiti, ali, kako su vezane za klasu 1:1, njihov zbroj nikad neće biti veći od maksimalne vrijednosti dostupne pojasne širine koja je postavljena za klasu 1:1.

```
# tc class add dev eth0 parent 1:1 classid 1:3 cbq bandwidth 100Mbit rate 5Mbit
   weight 0.5Mbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000
# tc class add dev eth0 parent 1:1 classid 1:4 cbq bandwidth 100Mbit rate 3Mbit
   weight 0.3Mbit prio 5 allot 1514 cell 8 maxburst 20 avpkt 1000
```

Obje podređene klase imaju FIFO kao unaprijed definiranu metodu i ovdje se postavlja metoda SFQ da bi se osigurala jednakost obje vrste mrežnog prometa.

```
# tc qdisc add dev eth0 parent 1:3 handle 30: sfq
# tc qdisc add dev eth0 parent 1:4 handle 40: sfq
```

Sljedeće dvije linije se odnose na glavnu metodu za upravljanje i šalju mrežni promet s priključnih točaka 25 i 80 na prethodno kreirane podređene klase.

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 80 0xffff f
lowid 1:3
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip sport 25 0xffff f
lowid 1:4
```

Ako obje vrste mrežnog prometa pokušaju preći ukupnih 6Mb, pojasna širina će se podijeliti prema *weight* parametru postavljenom prilikom kreiranja klasa, što znači da će 5/8 pojasne širine biti rezervirano za web poslužitelj, a 3/8 za SMTP mrežni promet.
U ovom slučaju sav ostali promet nije klasificiran, što znači da će moći iskoristiti svu preostalu pojasnu širinu.

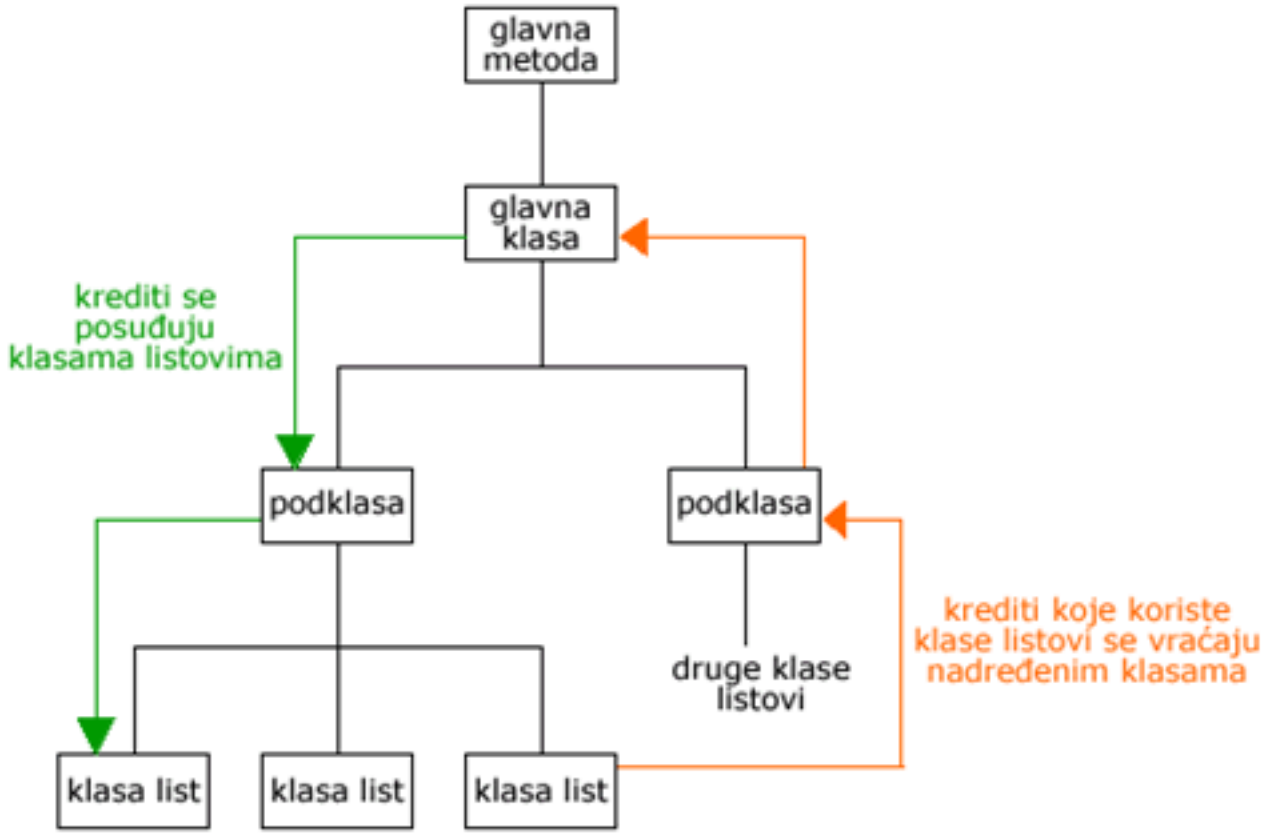
**Hierarchical Token Bucket (HTB)**
Nakon spoznaje kompleksnosti metode CBQ i obzirom da ona nije optimizirana na mnoge tipične situacije, stvorena je metoda HTB. Ona koristi princip metode TBF (privremena memorija koja se puni imaginarnim podacima) vezan za klase, i filtara uz korištenje mehanizama posuđivanja (iz metode CBQ) i kao takva omogućava vrlo precizno kontroliranje i oblikovanje mrežnog prometa. Mehanizam posuđivanja se odnosi na imaginarne podatke – kredite za slanje (eng. *tokens*) i to na način da podređene klase posuđuju kredite od svojih nadređenih objekata u slučaju da su prešli svoju zadani parametar brzine slanja paketa. Posuđivanje kredita i slanje paketa se nastavlja dok se ne dosegne limit postavljen od strane korisnika. U tom će trenutnu podređena klasa početi stavljati pakete u stanje čekanja sve dok još kredita ne postane dostupno.
Može se reći da metoda HTB osigurava da je pojasna širina koja se daje nekoj klasi na korištenje najmanje onolika koliko je ta klasa zatražila i koliko je za nju rezervirano. Ova se metoda pokazala najbolja u slučajevima kada postoji određena pojasna širina koja se želi podijeliti na način da svaki njen dio ima zagarantiranu vrijednost uz istovremeno zadržavanje mogućnosti posuđivanja pojasne širine za slučaj da se koristi manje pojasne širine od rezervirane.
Parametara dostupnih uz ovu metodu ima samo nekoliko:

- *rate* – parametar brzine, određuje pojasnu širinu koju klasa može koristiti
- *ceil* – parametar koji određuje maksimalnu pojasnu širinu za pojedinu klasu, što limitira pojasnu širinu koju klasa može posuditi od druge klase. Unaprijed definirana vrijednost je ista kao i postavljeni *rate* parametar, a njegova vrijednost se može nalaziti između veličine parametra *rate* te klase te veličine *ceil* parametra svojih podređenih klasa
- *burst/cburst* – parametar koji određuje količinu podataka koja može biti poslana maksimalnom brzinom koju dozvoljava sklopovlje bez posluživanja neke druge klase. Također treba biti najmanje velik koliko i vrijednost *burst/cburst* parametra svake podređene klase
- *prio* – parametar koji određuje prioritet na način da klase s višim prioritetom prve dobivaju šansu za posuđivanje pojasne širine
- *quantum* – ključni parametar za kontrolu mehanizma posuđivanja. Njega "po defaultu" određuje sama metoda HTB, a služi za podjelu mrežnog prometa između podređenih klasa (količinom većom od *rate*, a manjom od *ceil*) i slanje paketa iz tih istih klasa

Kako postoje samo dvije primarne klase koje se mogu kreirati pomoću metode HTB, tablica 1 opisuje moguća stanja mehanizma za posuđivanje, a slika 7 ponašanje mehanizma za posuđivanje.

| Tip klase | Stanje klase | Poduzeta akcija |
|---|---|---|
| list | <rate | klasa će poslati onoliko paketa koliko ima dostupnih kredita |
| list | >rate, <ceil | klasa će pokušati posuditi kredite od nadređenog objekta. ako postoje slobodni krediti, oni će biti posuđeni u quantum količini i podređena će klasa poslati onoliko podataka koliko je određeno *burst* parametrom |
| list | >ceil | nijedan paket neće biti poslan, što dovodi do kašnjenja |
| podređena klasa, glavna klasa | <rate | nadređena klasa će posuditi kredite svojim podređenim objektima |
| podređena klasa, glavna klasa | >rate, <ceil | klasa će pokušati posuditi kredite odvoog nadređenog objekta i onda ih proslijediti svojim podređenim objektima ukoliko oni to zatraže |
| podređena klasa,glavna klasa | >ceil | klasa neće pokušati posuditi kredite od svog nadređenog objekta i, prema tome, neće ih prosljeđivati svojim podređenim objektima |

Slika 7.




Primjer konfiguracije je gotovo isti kao i kod metode CBQ.

```
# tc qdisc add dev eth0 root handle 1: htb default 30
# tc class add dev eth0 parent 1: classid 1:1 htb rate 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 5mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 3mbit ceil 6mbit burst 15k
# tc class add dev eth0 parent 1:1 classid 1:30 htb rate 1kbit ceil 6mbit burst 15k
```

Preporuka je da se za podređene klase koristi metoda SFQ:

```
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc qdisc add dev eth0 parent 1:30 handle 30: sfq perturb 10
```

Na kraju se dodaju filtri koji usmjeravaju mrežni promet na podređene klase 1:10 i 1:20.

```
# U32="tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32"
# $U32 match ip dport 80 0xffff flowid 1:10
# $U32 match ip sport 25 0xffff flowid 1:20
```

Iz priloženog se vidi da je ova konfiguracija jednostavnija od one korištene s metodom CBQ, uz isti rezultat. Klase 10 i 20 imaju svoju rezerviranu pojasnu širinu i ako im je potrebno više, posuđivanje se obavlja u odnosu 5:3. Neklasificirani mrežni promet se usmjerava na klasu 30 koja može posuditi svu preostalu pojasnu širinu, a korištenjem metode SFQ dobivena je ravnomjerna raspodjela mrežnog prometa.

## Zaključak

Metode za upravljanje redovima koje koriste klase omogućavaju da se na jednom fizičkom linku simulira više sporijih linkova te se preko njih na različite načine šalju različite vrste mrežnog prometa. Njihove napredne mogućnosti dovele su do njihove integracije u Linux kernel (metoda CBQ od kernel verzije 2.1, a metoda HTB od kernel verzije 2.4). Metodu CBQ će preferirati korisnici koji još uvijek koriste starije verzije kernela. Međutim, iz razloga što ta metoda i oblikuje mrežni promet, a zbog kompleksnosti algoritma to ne radi na adekvatan način, preporuka je da se pređe na metodu HTB koja ne ovisi o karakteristikama mrežnog sučelja. Njena manje kompleksna konfiguracija, brzina te odlična dokumentacija čine je odličnim izborom za klasificiranje i oblikovanje mrežnog prometa.

- Logirajte [1] se za dodavanje komentara

čet, 2008-07-24 14:30 - Mirko Lovričević**Kategorije:** Mreža [2]
**Vote:** 0

No votes yet

**Source URL:** https://sysportal.carnet.hr/node/411

**Links**
[1] https://sysportal.carnet.hr/sysportallogin
[2] https://sysportal.carnet.hr/taxonomy/term/29