Sergej Rožman

Upravljanje čakalnih vrst

Queue Management

Povzetek

Ena od osnovnih nalog paketnih omrežij je učinkovita delitev omrežnih virov med komunicirajoče subjekte. V paketnih omrežjih komunikacijski toki med sabo namreč niso izolirani, ampak se potegujejo za skupne mrežne vire, kar zagotavlja njihovo boljšo izkoriščenost in s tem nižjo ceno komunikacijskih storitev.

Bistveni lastnosti vsakega komunikacijska toka sta njegov pretok in zakasnitve na prenosni poti. Pretok komunikacijskega toka je določen z velikostjo paketov in s pogostnostjo (frekvenco) pošiljanja teh skozi omrežje. Zakasnitve pa upravljamo z izbiro vrstnega reda paketov različnih komunikacijskih tokov. Vprašanje je torej, kateri paket poslati in kdaj.

Za diferencirano obravnavo komunikacijskih tokov glede na zahtevano kakovost storitev uporabljamo več različnih pristopov: nadzor pristopa v omrežje, oblikovanje ali omejevanje presežnega prometa, prekategorizacijo v nižje prioritetni promet ali v promet, ki se prenaša brez zagotovil, ter označevanje presežnega prometa, ki se po potrebi zavrže na prenosni poti v nadaljevanju. Vse naštete akcije izvajamo z upravljanjem čakalnih vrst. Prihajajoče pakete razvrstimo v ustrezne čakalne vrste in jih obdelujemo po tem vrstam pripadajočih disciplinah.

Najenostavnejše in največkrat uporabljeno razvrščanje v prihajajočem vrstnem redu nam ponavadi ne da zadovoljivih rezultatov, saj pri tem nimamo nobenega vpliva na delitev omrežnih virov posameznim komunikacijskim tokom. Vsi toki so obravnavani enako in sprememba kateregakoli ima velik vpliv na vse ostale. To najlažje rešimo tako, da različnim komunikacijskim tokom podelimo različne prioritete in jih odpošiljamo v skladu s temi. Slabost razvrščanja po prioritetah je potencialna možnost, da je preveč paketov z visoko prioriteto in tisti z nizko prioriteto nikoli ne pridejo na vrsto.

Opisana težava pripelje do potrebe po delitvi mrežnih virov na pravičen način. Pravična razvrščanja predstavljajo celo skupino postopkov, ki lahko posameznim komunikacijskim tokom zagotavljajo pretok in omejujejo zakasnitve. Med kompleksnejša pravična razvrščanja prištevamo uteženo pravično razvrščanje (WFQ), ki poizkuša časovno posnemati le teoretično mogoč vzporedni pretok več tokov. Za določitev vrstnega reda odpošiljanja paketov v realnem paketnem omrežju uporabi končne čase odpošiljanja teh v pripadajočem tokovnem modelu. Zaradi zaporednega odpošiljanja so v paketnem omrežju nekateri paketi sicer odposlani prej, prav noben pa kasneje kot pri pripadajočem tokovnem

modelu.

Med manj kompleksna pravična razvrščanja spadajo krožna razvrščanja. Pri krožnih razvrščanjih se vsak komunikacijski tok usmerja v svojo čakalno vrsto, iz katerih se pripadajoči paketi odpošiljajo v krožnem procesu. V primeru paketov različnih velikosti obstajajo različice, primer: krožno razvrščanje s primanjkljajem (DRR), ki beležijo količino odposlanih podatkov in prilagajajo število odposlanih paketov posameznega toka v naslednjem ciklu ter s tem izravnavajo pretok.

Vsi našteti postopki spadajo med neintenzivna razvrščanja, saj delujejo le, ko imamo v vrstah čakajoče pakete. Nasprotno od teh delujejo intenzivna razvrščanja, ki začasno zadržujejo tudi posamezne pakete in jih odpošiljajo naprej šele ob predvidenem času. S tem zmanjšujejo rafalnost komunikacijskih tokov. Intenzivna razvrščanja izvajamo s postopki, katerih delovanje temelji na principu vedra in/ali žetonov.

Osnovna razvrščanja delujejo na posamezne komunikacijske toke. Včasih želimo podobno obravnavo celih skupin podobnih komunikacijskih tokov, ki pa mora biti drugačna od drugih podobno združenih skupin. Zato komunikacijske toke grupiramo in jih razvrščamo po hierarhičnih postopkih z razredi. Mrežne vire v tem primeru najprej razdelimo znotraj svojega razreda in šele če kaj ostane, ponudimo v okvirih hierarhije še sosednjim razredom.

Meritve v prav ta namen zasnovanem preprostem testnem omrežju večinoma potrjujejo teoretično zasnovane lastnosti različnih razvrščanj paketov. Vseeno pa nas ne prav redko presenetijo posebnosti, na primer: lastnosti komunikacijskih protokolov in njihova izvedba na posameznih sistemih, dodatne skrite čakalne vrste v nekaterih mrežnih gradnikih, težavno kvantitativno upravljanje z zakasnitvami, v nekaterih primerih ne dosežemo želenih rezultatov, če se upravljanja čakalnih vrst lotimo na nepravem mestu.

Z učinkovitim upravljanjem čakalnih vrst je kljub vsemu mogoče združevati izrazito različne komunikacijske storitve in tako graditi več storitvena poenotena (konvergenčna) omrežja, oziroma kot jih danes popularno imenujemo, omrežja naslednje generacije.

Ključne besede: razvrščanje paketov, oblikovanje prometa, omejevanje prometa, nadzor pristopa, pravična razvrščanja, pravičnost, kakovost storitev

Abstract

One of the principal goals of a packet switched network is to optimize sharing of common network resources between communication entities. In a packet switched network flows are not isolated, they compete for common resources ensuring their better utilization which reduces cost of communication.

Bandwidth and end-to-end delay are the most important properties of each flow. Bandwidth is determined by the size and frequency of packet transmission through the network while delay depends on the order of packets belonging to different flows. The question therefore is: which packet to send, and when.

In order to offer differential treatment of flows according to required Quality of Service (QoS) several different approaches are in use: admission control, policing and shaping, downgrading of violating traffic to lower priority or even to best-effort service, marking nonconformant packets which should be dropped first when the network experiences lack of resources. All the actions above are performed by the queue management. Incoming packets are classified and placed into the appropriate queues served according to a specific service discipline.

The use of a single queue and FCFS (first-come, first-served) scheduling to serve packets has major limitations. It does not provide any form of resource assurance to traffic flows. No packet receives special treatment. Greedy sources can cause an extremely unfair distribution of the network resources among all traffic flows. One simple way to provide differential treatment to flows is to use multiple queues with associated priorities. The scheduler always serves packets from higher priority queue before it attends to the lower priority queue. The problem with this method is that lower priority queues may not get serviced at all if high priority traffic is excessive.

To override the problem we should distribute resources in a more fair way. Fair queuing represents a whole group of scheduling algorithms that support fair bandwidth allocation and delay bounds. One of most complex algorithms is weighted fair queuing (WFQ). WFQ algorithm is often explained with the fluid model in which we assume that traffic is infinitely divisible and multiple flows can be served simultaneously. In a real network packets are processed one at a time. An order of the packets is determined by calculating the departure (finish) time from a corresponding fluid model. Because of the

serialization some packets finish earlier, but all packets finish no later than they did in the corresponding fluid system.

A simpler implementation of fair queuing algorithms is round robin scheduling method. The round robin scheduler maintains one queue for each flow. The queues are served in a round robin fashion. If packet sizes are variable a minor modification to round robin exists – deficit round robin (DRR). Deficit round robin records the amount of already transmitted data of each flow by using deficit counter and then adjusts number of served packets in next turn.

Most of the well-known queuing disciplines are work conserving. Work conserving scheduler is idle only when there are no packets waiting to be transmitted. In contrast, in a non-work-conserving scheduler a node can transmit a packet only when the packet is eligible. If no packets are eligible for transmission, the node will become idle even when there is packet in the queue. The reason for this idle time at the non-work-conserving scheduler is to reduce the burstiness of traffic. Examples of non-work-conserving queuing disciplines include leaky bucket and token bucket.

Simple classless queuing disciplines schedule individual flows. Often the same approach is needed to schedule a whole class of flows which should differ from scheduling of other classes. To achieve hierarchical link sharing, classful queuing disciplines are used. In a hierarchical system the resources will first be redistributed within its own class and, if there are still excessive resources, will be redistributed among sibling classes.

Our measurement results in a simple test network environment essentially confirmed the theoretical expectations of various scheduling algorithms. However, we should consider: the characteristics of communication protocols and their implementation on individual system types, the additional concealed queues in some network elements mostly served by FCFS discipline, the problem in providing quantitative delay bounds, the queue management has no effect if there is no queue of waiting packets.

Effective queue management makes it possible to combine highly diverse communication services into a single all purpose network (network convergence), nowadays known as next generation networks (NGN).

Keywords: scheduling, shaping, policing, admission control, fair queuing, fairness, quality of service

Kazalo vsebine

Uvod

Osnovna naloga paketnih omrežij je učinkovita delitev omrežnih virov med komunicirajoče subjekte. V današnjem času so paketna omrežja praktično popolnoma nadomestila nekdanja vodovno komutirana omrežja. V vodovnih omrežjih so bili vsi komunikacijski toki med sabo izolirani. Vsak tok je imel povsem svoje omrežne vire. Če jih ni izkoristil v celoti, so ostali neizkoriščeni. V paketnih omrežjih je stanje drugačno. Omrežni viri so skupni. Posamezni sočasni komunikacijski toki si jih med sabo delijo. Vsak nima več zagotovljenega svojega popolnoma neodvisnega kanala. S tem sicer dosežemo boljšo izkoriščenost omrežnih virov. Po drugi strani to pomeni, da komunikacijski toki za omrežne vire med sabo tekmujejo in s tem drug drugemu motijo optimalen pretok. Z različnimi algoritmi razvrščanja poizkušamo do neke mere posnemati neodvisnosti, ki jih nudijo vodovna omrežja, in hkrati ohraniti možnost dodeljevanja neizkoriščenih virov drugim, ki v istem trenutku te lahko učinkovito uporabijo.

Paketna omrežja, kot je Internet, so bila ob svojem nastanku namenjena izključno prenosu računalniških podatkov, na primer datotek in elektronske pošte. Prometa je bilo malo in ni bil občutljiv na spreminjajoče komunikacijske pogoje: spremembe hitrosti prenosa, večje ali manjše zakasnitve, izgube in ponovno pošiljanje paketov. Ker paketni značaj prenosa podatkov omogoča učinkovito multipleksiranje komunikacijskih tokov, so omrežni viri uporabljeni bolj učinkovito. To vodi do nižjih cen in porodila se je zamisel, da bi tudi druge bolj zahtevne storitve prenašali prek paketnih omrežij. To je pomenilo večjo zasedenost omrežja z občutljivejšimi storitvami, ki zahtevajo bolj nadzorovane in urejene prenosne pogoje. Običajna enakopravna obravnava in neodvisnost pretoka posameznih paketov ne zadostuje več. Promet je treba uravnavati aktivno in povezano glede na pripadnost paketov posamezni storitvi, kar dejansko na najnižjem nivoju pomeni, da moramo aktivno prepoznavati in razvrščati pakete v čakalnih vrstah. Celoten proces običajno imenujemo upravljanje kakovosti storitev.

Omrežni viri so fizikalna količina. To pomeni, da je njihova **razpoložljivost omejena**. Podobno kot za mnoge fizikalne količine veljajo ohranitveni zakoni, na primer: zakon o ohranitvi mase, energije, gibalne količine, veljajo podobni zakoni tudi za omrežne vire. Intuitivno ugotovimo, da lahko pri polno zasedenem prenosnem kanalu posamezen komunikacijski tok povečuje svoj pretok le na račun zmanjšanja pretoka drugih tokov. Fizični prenosni kanali imajo ponavadi konstantno prepustnost. Vsota vseh pretokov komunikacijskih

tokov r_i je lahko največ enaka prepustnosti kanala R:

$$\sum_{i=1}^{N} r_i \leq R = konstanta$$

Nekoliko manj trivialen je pogled na zakasnitve, ki so tudi en od omrežnih virov. Če zanemarimo čas procesiranja paketov in celotno zakasnitev pripišemo čakanju v čakalnih vrstah, potem pravi ohranitveni zakon pri zakasnitvah, da je vsota povprečnih zakasnitev v čakalnih vrstah komunikacijskih tokov ponderiranih z lastnim pretokom konstantna in je torej neodvisna od načina razvrščanja paketov. To podobno kot pri pretoku pomeni, da lahko nekemu toku zmanjšamo zakasnitve edino na račun drugih komunikacijskih tokov. Zakon velja spet le v primeru, da je komunikacijski kanal polno zaseden in brez izgub, torej da nimamo dodatnih prostih virov in da uporabljamo neintenzivno razvrščanje, kar pomeni, da učinkovito izkoriščamo vse razpoložljive vire. Ohranitveni zakon za zakasnitve podaja sledeča formula:

$$\sum_{i=1}^{N} r_i t_i = konstanta$$

Kjer so:

- r_i srednja vrednost pretoka komunikacijskega toka i,
- t_i povprečna zakasnitev paketov komunikacijskega toka i in
- N število komunikacijskih tokov.

Formula dejansko predstavlja dobro znan, le nekoliko drugače zapisan Littlov teorem, predstavljen tudi v nadaljevanju pri razvrščanju v prihajajočem vrstnem redu na strani 25. Celotna vsota, ki je pri stalno polno zasedenem kanalu (stacionarno stanje) konstanta, v tem primeru predstavlja količino vseh podatkov v sistemu in je vsota čakalnih vrst vseh komunikacijskih tokov.

Poglejmo si na primeru, kjer imamo dva komunikacijska toka i in j, ki tečeta skozi usmerjevalnik. Privzemimo, da je pretok komunikacijskega toka i 10 Mbit/s in toka j 40 Mbit/s. Uporabimo razvrščanje v prihajajočem vrstnem redu, ki povzroča enake povprečne zakasnitve 3 ms vsakemu od teh dveh komunikacijskih tokov. Nek drug algoritem razvrščanja bi zmanjšal zakasnitve toka i na 1 ms. Po zgornji formuli lahko potem izračunamo povprečne zakasnitve toka j, ki se povečajo na 3,5 ms. Komunikacijski tok i doseže zmanjšanje zakasnitev na račun povečanja zakasnitev toka j.

Aktivno razvrščanje paketov

Z aktivnim razvrščanjem paketov nadzorujemo porabo omrežnih virov posameznih komunikacijskih tokov. Če ni dovolj prostih omrežnih virov, paketi ne morejo biti odposlani in v omrežnih napravah se začnejo tvoriti čakalne vrste paketov. S pravili razvrščanja določamo, katerim paketom in v kakšnem vrstnem redu bodo dodeljeni omrežni viri. Pri razvrščanju pomnimo, koliko paketov je vseboval posamezen komunikacijski tok in primerjamo s količino temu toku rezerviranih omrežnih virov. Naslednji paketi so poslani samo, če niso vsi viri že porabljeni.

Način razvrščanja glede na tip storitve

Način razvrščanja mora biti odvisen tudi od tipa in lastnosti storitev. Večpredstavnostne storitve, kot so prenos zvoka in slike v realnem času, delujejo zadovoljivo tudi v primeru izgub posameznih paketov, zelo pa so občutljive na zakasnitve. Drugače je pri storitvah prenosa podatkov, kjer je veliko bolje, da uporabimo povratno informacijo za krmiljenje pretoka in podatki ob zasedenih omrežnih virih začasno počakajo na izvoru, kot da so zavrženi nekje na poti skozi omrežje.

Namen različnih razvrščanj

V kratkem časovnem intervalu znotraj prenosa posameznega omrežnega paketa poteka prenos z največjo hitrostjo, ki je fizikalno oziroma tehnološko pogojena. Pretok komunikacijskega toka je tako določen z velikostjo paketov in s pogostnostjo (frekvenco) pošiljanja teh skozi omrežje. Zakasnitve pa upravljamo z izbiro vrstnega reda paketov, ki pripadajo različnim komunikacijskim tokom. Osnovna parametra razvrščanja sta tako:

- določanje časovnih intervalov pošiljanja paketov skozi omrežje,
- izbira vrstnega reda poslanih izmed množice čakajočih paketov.

Vprašanje je torej, kateri paket poslati in kdaj. Na posameznih vozliščih omrežja lahko pakete pošiljamo naprej takoj ali z zakasnitvijo. Izmed prispelih paketov pa po nekem pravilu izberemo vrstni red, kako se bodo odposlani paketi zvrstili.

Problem ni trivialno rešljiv, saj sta parametra med sabo povezana. Tudi vpliv na oba parametra je omejen. Vsako vozlišče lahko samo povečuje kumulativne zakasnitve, zmanjševati jih ne more. Prav tako lahko iz vozlišča v vozlišče le vedno bolj striktno omejujemo pretok posameznega toka.

Z razvrščanjem paketov v paketnih omrežjih tako nekatere lastnosti prenosa skozi omrežje določamo neposredno in druge posredno: pretok, zakasnitve, časovno kolebanje zakasnitev, delež izgub.

Pričujoče delo je sestavljeno iz treh delov:

- v prvem delu so predstavljene načrtovalske zahteve in izvedbene možnosti za dosego zadanih ciljev;
- drugi del predstavlja nekaj najbolj uporabljanih postopkov razvrščanj paketov v omrežjih;
- v tretjem delu prikazujemo razvrščanja v praksi in meritve v dejanskem testnem omrežju ter primerjave med posameznimi postopki.

1. DEL
Načrtovalske zahteve in možnosti

1.1 Aktivno razvrščanje paketov

Z aktivnim razvrščanjem mrežnih paketov poizkušamo zagotavljati čimbolj optimalno delitev omrežnih virov in s tem učinkovito delovanje omrežja.

V grobem lahko postopke razvrstimo na:

- postopke za zagotavljanje prepustnosti,
- postopke za omejevanje zakasnitev.

1.1.1 Postopki za zagotavljanje prepustnosti

Prepustnost omrežja je osnovni omrežni vir, ki ga najpogosteje želimo deliti med sočasne komunikacijske toke na nadzorovan način.

Prepustnost delimo na dva načina:

- absolutno;
  Posamezen komunikacijski tok prejme določeno vrednost prepustnosti omrežja, na primer 5 Mbit/s, ne glede na skupno prepustnost in zasedenost z drugimi tokovi, ki bi morda občasno ob prostem kanalu dopuščala tudi večje prepustnosti.
- relativno.
  Posamezen komunikacijski tok prejme administrativno določen delež skupne prepustnosti. To si sočasni toki praviloma v celoti razdelijo. Na primer: v primeru dveh sočasnih komunikacijskih tokov prvi prejme ¼ in drugi ¾ skupne prepustnosti. Ko se število sočasnih tokov spreminja, komunikacijski toki prilagodijo svoje vrednosti pretokov, ki pa ostanejo v nastavljenem razmerju.

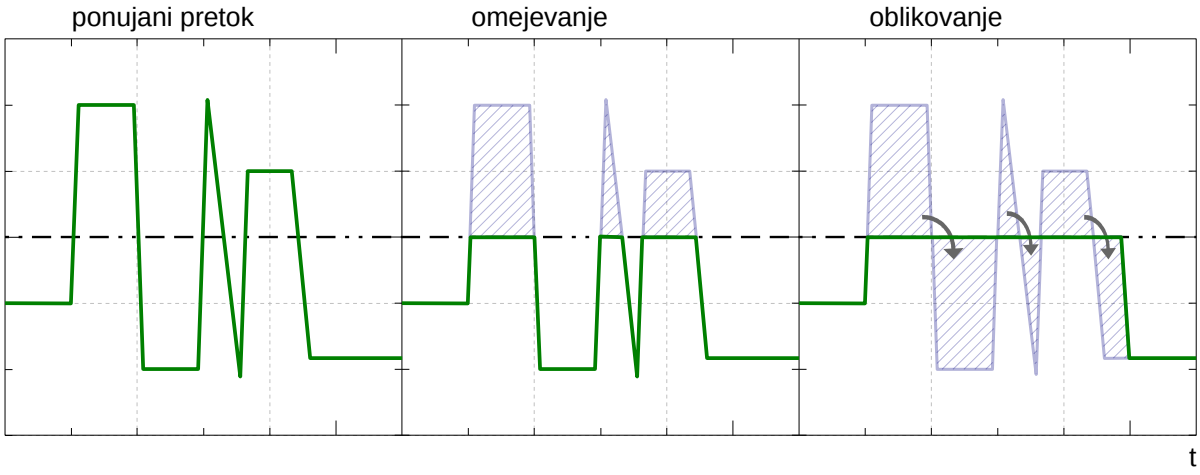
Prvo, absolutno delitev zagotavljajo intenzivna razvrščanja in jih običajno izvajamo s pomočjo omejevanja z vedri in/ali žetoni. Nevarnost absolutnih delitev je v tem, da lahko vsota dodeljenih prepustnosti presega skupno kapaciteto kanala, kar vodi do nestabilnega delovanja. Komunikacijski toki v takem primeru izmenično poizkušajo zasesti dodeljeno prepustnost na račun drugih, kar vodi v pulzirajoče vrednosti pretokov in zakasnitev. Ko je celotna prepustnost razdeljena, praviloma ne smemo več dopustiti novih komunikacijskih tokov. V nasprotnem primeru, pri premajhnem številu komunikacijskih tokov, prihaja do

neizkoriščenosti omrežja, saj del prepustnosti ostane nerazdeljen.

Absolutne delitve so primerne za storitve, ki rabijo zagotovljene vrednosti pretokov, ker v nasprotnem sploh ne delujejo. To so predvsem večpredstavnostne storitve in storitve, ki delujejo v realnem času.

Drugo, relativno delitev zagotavljajo neintenzivna razvrščanja in jih običajno izvajamo z uteženimi pravičnimi razvrščanji. Značilnost relativnih delitev je, da praviloma vedno razdelimo celotno prepustnost. Minimalnih posameznih deležev ne zagotavljamo. Pri velikem številu sočasnih komunikacijskih tokov bodo posamezni pretoki zelo majhni. Relativne delitve so primerne za razne prenose podatkov, kjer želimo prav vsakomur zagotoviti vsaj del prepustnosti.

Če ponujani pretok presega dovoljeno ali rezervirano prepustnost, imamo na voljo več možnosti:



Slika 1: omejevanje in oblikovanje prometa

- tak komunikacijski tok v celotni zavrnemo, komunikacije sploh ni mogoče vzpostaviti ali nadaljevati;
- zavrnemo ali zavržemo del prometa, ki presega dovoljene okvire – omejevanje prometa (na sliki 1 v sredini);
- presežni del prometa shranimo – zadržimo v čakalni vrsti – in ga odpošljemo naprej v manj obremenjenih časovnih terminih, ko se ponujani pretok zmanjša in se del omrežnih virov sprosti – oblikovanje prometa (na sliki 1 desno);

- presežni del prometa prekategoriziramo v nižje prioritetni promet ali v promet, ki se prenaša brez zagotovil v najboljši možni meri;
- presežni del prometa označimo in ga odpošljemo naprej na račun proste prepustnosti drugih komunikacijskih tokov, ki te trenutno ne izkoriščajo v polni meri. Ta neskladni del prometa postane prvi kandidat za zavrženje na prenosni poti v nadaljevanju ob morebitnem pomanjkanju omrežnih virov.

S pretokom so posredno povezane tudi izgube paketov. Preveliki pretoki ali premočni rafali pripeljejo do zamašitev v omrežju. Zaradi pomanjkanja prostora v medpomnilnikih v takem primeru nekateri paketi ne morejo biti sprejeti ali zaradi izteka časovnikov ne morejo biti pravočasno odposlani, kar privede do njihovih izgub na prenosni poti. Na same izgube paketov torej direktno težko vplivamo. Vpliv nanje je samo posreden z dobro regulacijo oziroma omejevanjem pretoka.

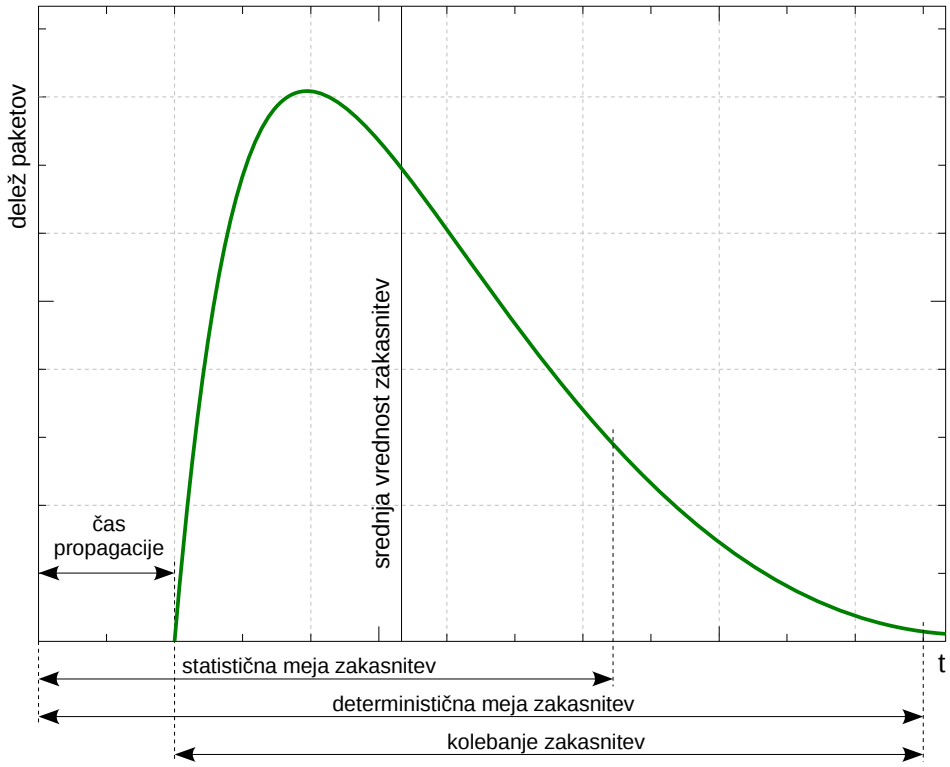
1.1.2 Omejevanje zakasnitev

Postopki razvrščanja morajo zagotavljati nadzor nad zakasnitvami. Omejevanje zakasnitev je lahko deterministično ali statistično. Storitve zagotovljene kakovosti, ki delujejo v realnem času, zahtevajo strogo deterministično omejevanje zakasnitev tudi v najbolj neugodnem primeru. Za manj zahtevne storitve z nadzorovano obremenitvijo je dovolj, da ne prihaja do dolgotrajnejših zastojev. Občasni rafali so dovoljeni.

Pri determinističnem omejevanju zakasnitev se bolj približamo načinu delovanja vodovnih omrežij. To ima svojo ceno, saj moramo biti pazljivi pri multipleksiranju prometa. Rafale je treba strogo omejevati. Vedno moramo imeti zagotovljeno obvezno rezervo virov. Pomeni, da potrebujemo predimenzionirano omrežje. Pri tem moramo računati z večjimi zakasnitvami kot pri statističnem omejevanju, saj načrtovane zakasnitve ne sme prekoračiti praktično noben paket. Točnost je kradljivec časa[1].

Statistično omejevanje zagotavlja zelo učinkovito izrabo omrežja. Računamo lahko z nižjimi zakasnitvami (slika 2), saj dopuščamo, da vsake toliko dejanska zakasnitev za kratek čas tudi preseže pričakovano.

1 *Punctuality is the thief of time. (Oscar Wilde)*



Slika 2: shematični prikaz statistične porazdelitve zakasnitev

Z zakasnitvami je posredno povezana še ena lastnost, to je časovno kolebanje zakasnitev. Nekatere storitve niso občutljive na same vrednosti zakasnitev paketov. Bolj jih moti, če se zakasnitve paketov s časom spreminjajo. V omrežjih, kjer je pretok paketov asinhron in če so poleg tega paketi še različnih velikosti, je kolebanje zakasnitev zelo težko omejevati v ozkih mejah. Zato se temu ne posveča velike pozornosti in se problem raje rešuje na strani klientov z izravnalniki, ki začasno shranjujejo neenakomeren pretok paketov in jih odpošiljajo aplikacijam v urejenih časovnih tokih. Slaba lastnost izravnalnikov je dodatno povečanje zakasnitev. Rešitev je povsem sprejemljiva za prenos pretočnih večpredstavnostnih vsebin, medtem ko je vnos dodatnih zakasnitev lahko problematičen za interaktivne večpredstavnostne vsebine.

Postopki za omejevanje zakasnitev

Razvrščanje z uporabo postopkov za omejevanje zakasnitev odpošilja najprej pakete, katerim bo prvim potekel dovoljen čas zakasnitve. Vsakemu paketu se ob prihodu priredi skrajni čas odpošiljanja, potem se pakete razvrsti po tem času in v tem vrstnem redu odpošilja. Prednost postopkov za omejevanje zakasnitev je, da neodvisno obravnavajo zakasnitve in prepustnost omrežja. Ponavadi namreč velja, da toki, ki jim omogočimo večjo

prepustnost omrežja, dobijo tudi manjše zakasnitve in obratno. S postopki za omejevanje zakasnitev lahko tudi toku, ki ima rezervirano majhno prepustnost, kljub temu zagotavljamo nizke zakasnitve. Proces nadzora je dokaj težaven. Rabimo kar dvojni nadzor:

- da vsota pretokov vseh tokov ne preseže celotne prepustnosti kanala in
- da ne zamudimo dopustne zakasnitve pri nobenem paketu.

Naslednji problem, ki se pojavi, je lahko nepravičnost. Za dva toka, ki rabita enak pretok in zahtevata enake zakasnitve, ni nobenega zagotovila, da dobita enake deleže.

V nadaljevanju si oglejmo nekaj lastnosti, ki so pomembne pri aktivnih razvrščanjih.

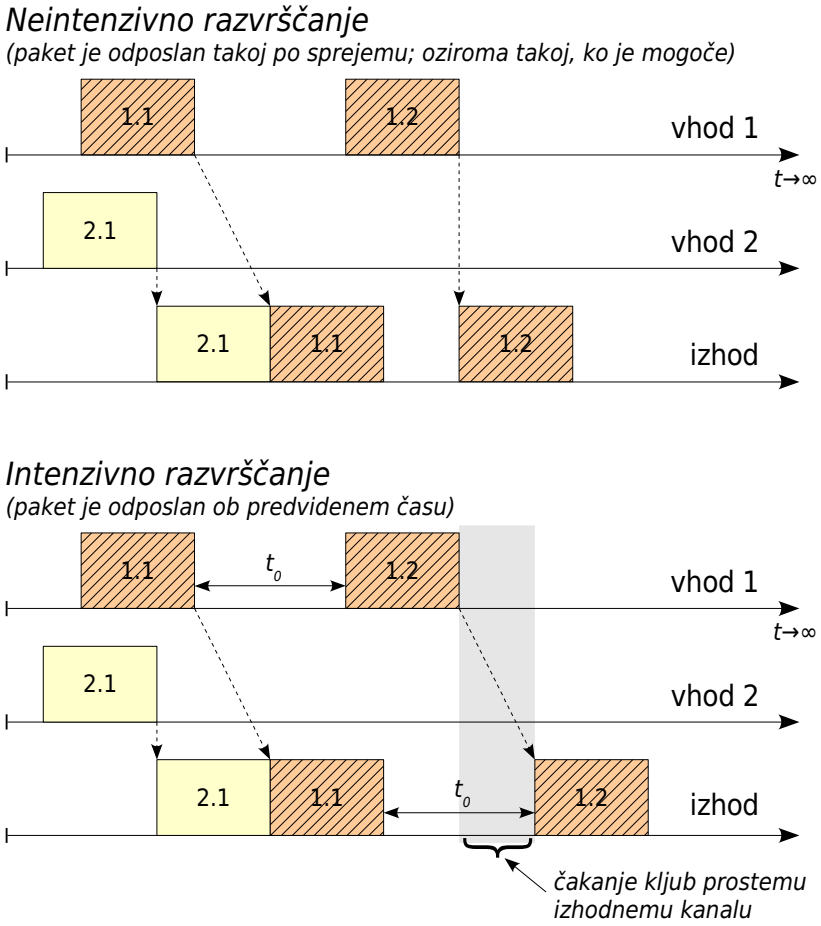
1.2 Lastnosti aktivnih razvrščanj

1.2.1 Intenzivno in neintenzivno razvrščanje

Kadar v čakalni vrsti ni čakajočih paketov, se nad prispelimi paketi običajno ne izvaja nikakršna obdelava in so ti odposlani nemudoma. Tak način delovanja imenujemo neintenzivno razvrščanje. Večina postopkov razvrščanj deluje na tak način in tako spadajo med neintenzivna razvrščanja.

Intenzivno razvrščanje deluje drugače. Pri intenzivnem razvrščanju se tudi posamezni paketi začasno zadržijo v čakalni vrsti in se odpošljejo naprej šele ob vnaprej določenih časih ne glede na dejstvo, da sta tako procesna enota kot tudi izhodni kanal prosta. Na prvi pogled je čudno, zakaj bi mrežni paket moral čakati, kadar se zdi, da mu ni treba. Razlog za namerno neaktivnost pri intenzivnem razvrščanju je zmanjševanje turbulentnosti prometa. Med intenzivna razvrščanja prištevamo postopke, katerih delovanje temelji na principu vedra in/ali žetonov. Najbolj običajna postopka sta razvrščanje z vedrom in žetoni (TBF) ter razvrščanje s puščajočim vedrom.

Intenzivno razvrščanje uporabljamo tudi, kadar želimo omejiti časovno kolebanje zakasnitev paketov ali zmanjšati velikosti potrebnih medpomnilnikov v naslednjih mrežnih elementih. Slaba stran intenzivnega razvrščanja je potencialna neizkoriščenost prenosnega kanala in s tem tratenje sistemskih virov, ki so takrat na razpolago v zadostnem obsegu, ter povečanje povprečnih zakasnitev komunikacijskega toka.



Slika 3: intenzivno in neintenzivno razvrščanje

Lastnosti neintenzivnega razvrščanja:

- izhodni kanal ni nikoli prost, če imamo v vrsti čakajoče pakete,
- dobro izkorišča sistemske vire,
- rafalnost tokov lahko narašča, s tem se v nadaljevanju na prenosni poti skozi omrežje veča verjetnost izgub,
- zakasnitve paketov so spremenljive.

Lastnosti intenzivnega razvrščanja:

- omogoča nadzor prenosne hitrosti komunikacijskega toka,
- vsakemu paketu se določi primeren čas odpošiljanja (odvisen od časovne reže med paketi – t_0 na sliki 3),
- izhodni kanal je lahko prost ne glede na prisotnost čakajočih paketov (če za noben paket še ni napočil primeren čas za odpošiljanje),

- rafali in varianca zakasnitev so nadzorovani,
- tratenje sistemskih virov.

1.2.2 Pravičnost

Kadar skupni viri niso zadostni za zadovoljitev vseh potreb, jih moramo med vse sočasne komunikacijske toke nekako razdeliti. Če ni posebnih zahtev za privilegirano obravnavo posameznih komunikacijskih tokov, dostikrat želimo, da so sistemski viri razdeljeni na pravičen način. Težava je, da pravičnost ni tehničen pojem in nima enolične definicije. Niti ni jasno, za koga naj bi bila delitev pravična. Morda za uporabnike, lahko pa za omrežje? Na prvi pogled se zdi najbolj pravično, da vire delimo med vse komunikacijske toke v enakih deležih – komunikacijske promete enake prioritete naj bi obravnavali enako. Takoj se pojavi dilema, ali je pravično, da nekdo z enako prenosno hitrostjo obremenjuje več vej omrežja, saj komunikacija poteka na večjo razdaljo, kot nekdo, ki komunicira s sosednjim vozliščem.

Poznamo več tipov pravičnosti. Različni tipi pravičnosti dajejo prednost ali zapostavljajo razrede prometov po različnih kriterijih. Cilj je lahko čim bolj optimalno izrabiti vire omrežja ne upoštevaje posameznih izvorov (maksimiranje prepustnosti) ali nasprotno, cilj je lahko čimbolj enaka delitev virov med uporabnike (max-min pravičnost).

Matematičen pristop k pravičnosti si lahko zamislimo kot optimizacijski problem. Cilj je najti razporeditev virov, ki je ekstrem uporabnostne funkcije specifične za posamezen kriterij pravičnosti. Pri tem se lahko odločamo tudi na podlagi cene, ki v funkciji tudi lahko nastopa kot parameter.

Za kvantitativno ugotavljanje stopnje pravičnosti poznamo empirični Jainov pravičnostni indeks [1]:

$$pravičnost = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2}$$ Jainov indeks pravičnosti

Pri tem je x_i količina poljubnega splošnega vira, ki ga med n subjekti prejme subjekt i.

Indeks je uporaben pri katerikoli delitvi virov. Zgornja vrednost indeksa je 1 (popolna pravičnost), spodnja vrednost indeksa je $\frac{1}{n}$ (skrajna nepravičnost) in se pri velikem številu

subjektov bliža vrednosti 0. Vrednost indeksa je neodvisna od dejanskih vrednosti virov. Iz vrednosti indeksa je mogoče ugotoviti stopnjo nepravičnosti. Rezultat 0,1 namreč lahko pomeni, da je delitev skrajno nepravična za 90% udeležencev – 10% jih dobi vse, 90% ne dobi nič.

Možno je tudi, da dodelimo nekaterim subjektom že v osnovi pravice do večjega deleža virov, tako da uvedemo uteži, ki predstavljajo faktor, po katerih delimo deleže. S tem pridemo do utežene pravične delitve.

Proporcionalna pravičnost

Proporcionalna pravičnost je postopek pravične delitve virov, ki daje prednost učinkoviti **izrabi omrežja** in s tem pravično dodeljevanje posameznega vira uporabniku potiska nekoliko v ozadje. Proporcionalna pravičnost obravnava vire bolj celovito in ne vsakega posebej. Običajno vire cenovno združimo in potem vsakemu uporabniku dodelimo tolikšno količino skupnih virov, da je ta cenovno enaka. V praksi lahko to pomeni, če nek komunikacijski tok v primerjavi s konkurenčnim tokom teče skozi dvakratno število vozlišč, mu zato pripada le polovična prepustnost omrežja.

Proporcionalno delitev virov v praksi zelo enostavno dosežemo tako, da pri uteženem pravičnem razvrščanju podelimo uteži, ki so obratno sorazmerne ceni oziroma porabi sistemskih virov.

$$\phi_i = \frac{1}{c_i}$$

kjer je c_i poraba virov oziroma cena pretoka za enoto komunikacijskega toka i in ϕ_i dodeljena utež temu toku.

Max-min pravičnost

Max-min pravičnost se pri razvrščanju paketov uporablja zelo pogosto, zato si jo oglejmo nekoliko podrobneje. Velja za **s stališča uporabnikov** enega najbolj pravičnih delitev virov. Max-min pravičnost poizkuša čim bolj pravično razdeliti sistemske vire med vse sočasne subjekte, kljub temu da taka razdelitev pogosto ni najoptimalnejša za izkoriščenost omrežja.

Max-min pravična delitev določa, da skromnejši komunikacijski toki dobijo kolikor rabijo, ostali si potem enakomerno razdelijo kar ostane. Torej:

- noben subjekt ne prejme več virov kot jih potrebuje in
- vsem subjektom, katerim ne moremo pokriti potreb, dodelimo tolikšno največjo količino virov, **da bi bilo te mogoče nadalje povečati edino na škodo drugega, ki ima ali bo s tem imel manj virov** (glej formalno definicijo v nadaljevanju).

Max-min pravična delitev torej privilegira tiste, ki rabijo manj virov; maksimira minimalne deleže. Od tod naziv.

Možno je tudi, da dodelimo nekaterim subjektom že v osnovi pravice do večjega deleža, tako da vsakemu subjektu določimo utež $\phi_1, \phi_2, \dots, \phi_k$ in glede na te sorazmerno delimo deleže. S tem pridemo do utežene pravične max-min delitve virov.

Formalna definicija max-min pravičnosti

Prejšnje ugotovitve lahko strnemo v preciznejšo matematično obliko, pri čemer imamo z delitvijo virov najpogosteje v mislih delitev prepustnosti omrežja.

Vzemimo poljubno omrežje z n izvori komunikacijskih tokov, ki tečejo proti n ciljem. Iz razporeditev prepustnosti, ki jih zasedajo toki na vseh vejah omrežja, tvorimo vektor $\boldsymbol{r} = [r_1, r_2, \dots, r_n]$ razporeditev prepustnosti.

Definicija: Vektor $\boldsymbol{r}$ razporeditev prepustnosti v omrežju je max-min pravičen:

- če obstaja in
- če za vsak komunikacijski tok i in pri katerikoli drugi možni razporeditvi prepustnosti $\boldsymbol{r}^*$, kjer bi veljalo $r_i^* > r_i$, lahko najdemo nek drug tok j, kjer je $r_j \le r_i$ in $r_j^* < r_j$.

Iz definicije sledi nekaj izrekov:

Izrek 1: Vektor razporeditev prepustnosti $\boldsymbol{r}$ je max-min pravičen tedaj in le tedaj, ko teče vsak komunikacijski tok i skozi vsaj eno zasičeno vejo omrežja.

Izrek 2: Če teče vsak komunikacijski tok i skozi vsaj eno zasičeno linijo, potem je vektor razporeditev prepustnosti $\boldsymbol{r}$ v omrežju max-min pravična.

Izrek 3: Vektor max-min pravičnih razporeditev prepustnosti $\boldsymbol{r}$ je v vsakem omrežju enolično določen.

Dokaze lahko najdemo v [5].

Določitev max-min pravičnega deleža – polnilni postopek

Max-min pravične deleže najlažje poiščemo z iteracijskim polnilnim postopkom na

sledeč način:

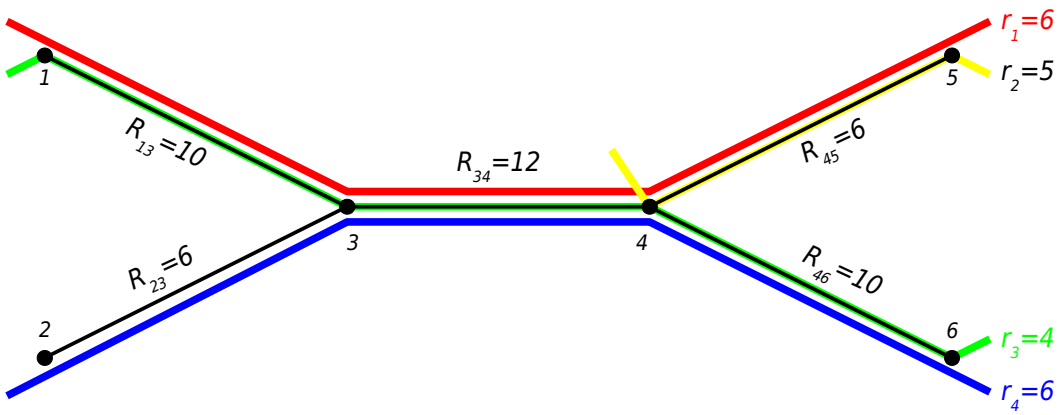
- vsi komunikacijski toki začnejo s pretokom nič,
- vsem komunikacijskim tokom enako povečujemo pretok:
  - dokler nekateri ne dosežejo zahtevanega pretoka ali
  - dokler ne zasičimo katere od vej omrežja;
- komunikacijske toke na zasičeni veji fiksiramo pri tej vrednosti in nadaljujemo postopek povečevanja pretoka (prejšnja točka) preostalim tokom.

Značilnosti postopka:

- očitno zahteva centralno upravljanje omrežja;
- po nekaj iteracijah algoritem konvergira proti max-min pravičnim deležem;
- obstajajo tudi decentralizirane verzije; omrežni elementi morajo uporabljati bitno krožno strežbo (posplošeno razvrščanje) in samoregulacijo pretoka (z drsečim oknom).

Primer max-min pravičnih deležev v omrežju

Poglejmo si primer določitve max-min deležev prepustnosti na omrežju s slike 4. Omrežje sestavljajo štiri robna in dve notranji vozlišči, med katerimi potekajo veje omrežja označene z R_{ij}, ki imajo različne prepustnosti. Te so razvidne s slike.



Slika 4: določitev max-min deležev prepustnosti v omrežju

Skozi omrežje tečejo štirje komunikacijski toki označeni z različnimi barvami. Vsak tok ima določen tudi največji možni pretok r_j prikazan na desni, ki bi ga tok dosegel, če ne bi bilo drugih omejitev. Omejuje ga lastni izvor oziroma pripadajoča storitev.

Vrednosti polnilnega postopka po iteracijah prikazuje tabela 1. Ob začetku polnilnega

algoritma so vsi deleži nič. V vsaki iteraciji vsem tokom enako povečujemo pretok. Do tretje iteracije gre gladko, vsi toki dosežejo pretok po 3 enote. Po tretji iteraciji tokoma r_1 in r_2 ne moremo več povečevati pretoka, saj dosežemo omejitev 6 enot na veji R_{45}. V naslednji iteraciji povečujemo pretok samo še tokoma r_3 in r_4. Po četrti iteraciji tok r_3 doseže omejitev lastnega izvora, to je 4 enote. Povečuje se lahko samo še tok r_4, ki s pretokom 5 enot naleti na omejitev veje R_{34} po peti iteraciji. S tem smo prišli do zaključka postopka, kjer vsak od tokov doseže svoj max-min pravični delež pretoka.

iteracija	r_1	r_2	r_3	r_4	***opomba***
–	*6*	*5*	*4*	*6*	*največje vrednosti tokov (omejitev izvorov)*
0	*0*	*0*	*0*	*0*	*začetek*
1	*1*	*1*	*1*	*1*	
2	*2*	*2*	*2*	*2*	
3	*3*	*3*	*3*	*3*	*dosežena omejitev veje R_{45} – r_1 in r_2 se ne moreta več povečevati*
4			*4*	*4*	*r_3 doseže omejitev izvora*
5				*5*	*dosežena omejitev veje R_{34} – r_4 se ne more več povečevati*
	3	***3***	***4***	***5***	***max-min pravični deleži***

Tabela 1: max-min pravična delitev prepustnosti – polnilni postopek

Kljub temu, da večina vej omrežja še ni zasičena, vrednosti komunikacijskih tokov ni mogoče več povečevati, saj je vsak od njih že naletel na omejitev. Po izreku 1 je značilnost max-min pravične delitve ravno ta, da vsakega od tokov omejuje vsaj ena veja – lahko tudi lastni izvor.

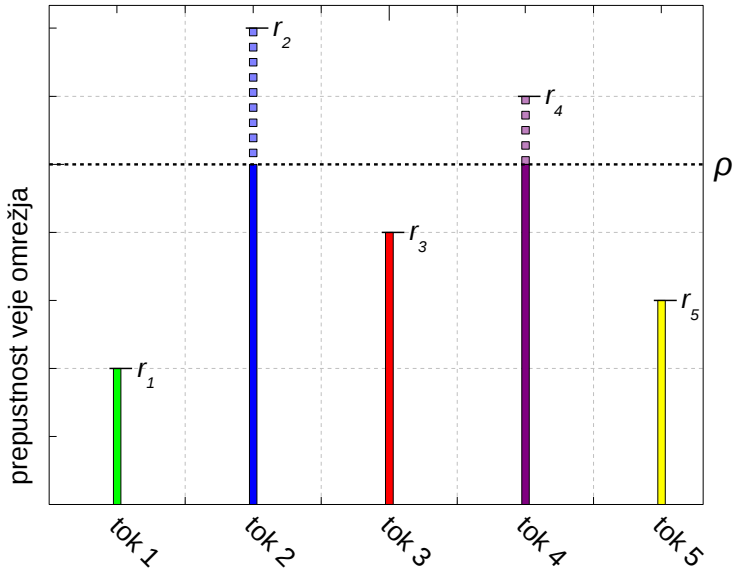
Na prvi pogled se zdi, da je dosežen rezultat pretokov v danem primeru najboljši možen. To je le deloma res, saj je odvisno od zornega kota opazovanja. S stališča komunicirajočih subjektov je dosežen rezultat verjetno res najbolj ugoden kompromis. Nobenemu ni mogoče povečati pretoka, ne da bi ga nekomu zmanjšali. S stališča omrežja pa se zdi, da je omrežje slabo izkoriščeno. Zasičeni sta samo dve veji, vsota vseh štirih tokov znaša 15 enot. Z razdelitvijo $r_1=2$, $r_2=4$, $r_3=4$ in $r_4=6$ bi lahko na račun zmanjšanja pretoka

toku r_1 za eno enoto za prav toliko povečali pretok kar dvema tokoma (r_2 in r_4) ter tako dosegli večji skupni pretok skozi omrežje – 16 enot – in zasičenje štirih vej omrežja. Kot kaže primer, je max-min pravičnost optimalna za komunicirajoče subjekte, medtem ko za izkoriščenost omrežja ni najugodnejša.

Zaradi poenostavitve in nazornosti v primeru povečujemo pretok stopenjsko po 1 enoto. Pri splošnem postopku bi morali pretok povečevati zvezno. Pri polnilnem postopku je enostavno ugotoviti, kdaj dosežemo katero mejo, saj sta možnosti samo dve: lastna omejitev toka ali zasičenje omrežne veje. Če bi max-min pravične deleže iskali z računanjem, bi bil postopek mnogo težji.

Max-min pravična delitev prepustnosti na eni veji omrežja

Podobno kot v celotnem omrežju lahko določimo max-min pravične deleže tudi na posamezni veji omrežja.



Slika 5: določitev max-min deležev prepustnosti na veji omrežja

Zaradi skrčenega števila pogojev je možno dokaj enostavno deleže določiti tudi računsko. Pogoja sta:

- da posameznemu komunikacijskemu toku podelimo prepustnost, ki jo potrebuje;
- da vsota prepustnosti za vse toke ne preseže kapacitete omrežne veje.

Drugi pogoj je veliko blažji kot v prejšnjem primeru, saj je omejen samo na eno vejo omrežja. Če s prvim pogojem kršimo drugega, pomeni da toku ne moremo zagotoviti

potrebne prepustnosti. Dodelimo mu torej največ, kolikor je mogoče – manjšo vrednost med želenim pretokom r_i in razpoložljivim za vse komunikacijske toke enakim deležem prepustnosti ρ.

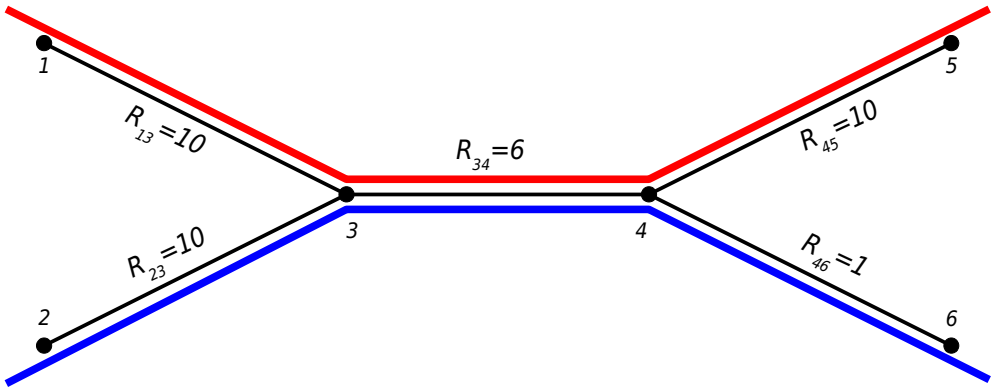
$$\sum_{i \in S_l} min(r_i, \rho) \leqslant R_l$$

Kjer so:

- S_l množica komunikacijskih tokov, ki uporabljajo vejo l,
- r_i ponujan pretok izvora i,
- ρ max-min pravični delež prepustnosti linije, ki pripada komunikacijskemu toku, če mu ne moremo zagotoviti celotne želene prepustnosti,
- R_l celotna prepustnost veje l.

Primer delitve virov v omrežju brez regulacije pretoka

Poglejmo si primer, kako se razporedi pretok komunikacijskih tokov v omrežju brez samoregulacije pretoka.



Slika 6: delitev virov v omrežju brez regulacije pretoka

Privzemimo, da vsako vozlišče zase deli prepustnost izhodnega kanala in s tem hitrost prenosa proporcionalno med vse komunikacijske toke v enakih deležih. V omrežju imamo:

- en komunikacijski tok brez omejitve označen rdeče od vozlišča 1 proti vozlišču 5 in
- pet komunikacijskih tokov brez omejitve označeni modro od vozlišča 2 proti vozlišču 6.

Ponujani pretoki v omrežju so naslednji:

- v veji od vozlišča 1 do 3 imamo en tok z 10 enotami pretoka,
- v veji od vozlišča 2 do 3 imamo pet tokov, vsak po 2 enoti pretoka,
- v vozlišču 3 se toki združijo in v skupni veji od vozlišča 3 do 4 imamo šest tokov, vsak po 1 enoto pretoka,
- v vozlišču 4 se toki spet razdružijo in v veji od vozlišča 4 do 5 imamo en tok z 1 enoto pretoka,
- v veji od vozlišča 4 do 6 imamo pet tokov, vsak po 0,2 enoti pretoka.

Dejansko opravljeni pretok je torej [1; 0,2; 0,2; 0,2; 0,2; 0,2] – eno enoto toka od vozlišča 1 do vozlišča 5 in 0,2 enoti vsakega od petih tokov od vozlišča 2 do vozlišča 6.

Pri upoštevanju opravljenega pretoka, so viri omrežja od vozlišča 1 do vozlišča 5 neizkoriščeni, saj z opravljenim pretokom nikjer ne izkoristimo celotne prepustnosti kanala. Delitev virov ni v skladu z definicijo in izreki max-min pravične delitve virov, razpored komunikacijskih tokov torej ni max-min pravična.

Če bi bil uporabljen samoregulativen komunikacijski protokol, bi se ponujan pretok izenačil z opravljenim. Tokovi od vozlišča 2 do vozlišča 6 bi imeli pretok po 0,2 enote na vsej prenosni poti. To bi razbremenilo omrežje nepotrebnega neopravljenega pretoka. Prvi tok od vozlišča 1 do 5 bi se lahko povečal na 5 enot, s čimer bi veja med vozliščema 3 in 4 prišla v nasičenje. Kot sledi iz izrekov, je to zadosten pogoj, da je taka razporeditev pretokov [5; 0,2; 0,2; 0,2; 0,2; 0,2] max-min pravična. S tem smo pokazali, da je samoregulacija komunikacijskih tokov lahko en od pogojev za doseganje max-min pravičnosti na decentraliziran način.

Uporabnostna pravičnost

Zanimiv pristop z namenom poenotenja različnih tipov pravičnosti v enotno teorijo zasledimo v virih [2] in [3].

Ideja uporabnostne pravičnosti je poiskati splošno uporabnostno funkcijo $u_i(r_i)$ za delitev virov (največkrat prepustnosti omrežja). Iskanje pravičnosti je potem matematično optimizacijski problem, iskanje maksimuma ali minimuma uporabnostne funkcije za posamezen kriterij pravičnosti. Če funkcija $u_i(r_i)$ predstavlja pretok, ki ga v omrežje pošilja izvor i, je skupna obremenitev omrežja vsota pretokov, ki pripadajo množici vseh izvorov S:

$$U = \sum_{i \in S} u_i(r_i)$$

Za čim boljšo izkoriščenost poizkušamo omrežje čimbolj obremeniti. Poizkušamo torej poiskati maksimum *U*.

Kot uporabna funkcija za delitev prepustnosti in posledično pretoka se izkaže:

$$u(r) = \begin{cases} \log r & \alpha = 1 \\ \dfrac{r^{1-\alpha}}{1-\alpha} & \alpha \neq 1 \end{cases}$$

kjer je *α* parameter, s katerim nastavljamo tip pravičnosti. Izbira nekaterih vrednosti parametra *α* vodi v naslednje posebne primere pravičnosti:

α	***poseben primer***	$\max_r U$
0	*maksimiranje celotne prepustnosti omrežja*	$\max_r \sum r_i$
1	*proporcionalna pravičnost*	$\max_r \sum \log r_i$
2	*minimiranje potencialnih zakasnitev*	$\min_r \sum \frac{1}{r_i}$
∞	*max-min pravičnost*	$\max_r \min_{i \in S} r_i$

Tabela 2: posebni primeri uporabnostne pravičnosti

Ob tem naletimo na dve novi pravični delitvi. Poleg omenjene proporcionalne in max-min pravičnosti se srečamo še s primerom maksimiranja celotne prepustnosti omrežja in minimiranjem potencialnih zakasnitev.

Maksimiranje celotne prepustnosti omrežja

V primeru maksimiranja celotne prepustnosti omrežja pustimo potrebe posameznih subjektov nekoliko ob strani in se osredotočamo na izkoriščenost omrežja kot celote. Iščemo tako razporeditev, ki da maksimum vsote pretokov vseh komunikacijskih tokov.

$$U = \max_{r} \sum_{i \in S} r_i$$

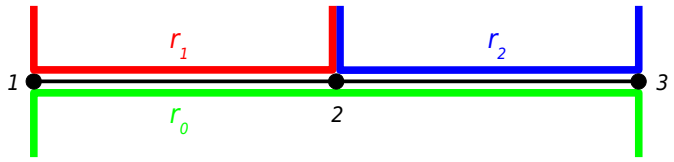
Kot bomo videli na primeru v naslednjem poglavju, lahko maksimiranje celotne prepustnosti omrežja vodi v nepravičnost, saj nekaterim subjektom lahko tudi popolnoma odreče delež prepustnosti. V splošnem ta razporeditev daje prednost lokalnim komunikacijskim tokom in zavira toke na daljših prenosnih poteh.

Minimiranje potencialnih zakasnitev

Pri minimiranju potencialnih zakasnitev ne gre za zakasnitve posameznih paketov. Cilj te razporeditve je, v čim krajšem času skozi omrežje prenesti neko količino podatkov. Čas prenosa podatkov je obratno sorazmeren s pretokom. Iščemo take vrednosti pretokov, da bo čas prenosa čim krajši skupaj za vse izvore.

$$U = \min_{r} \sum_{i \in S} \frac{1}{r_i}$$

Ponazoritev uporabnostne pravičnosti na enostavnem omrežju linearne topologije



Slika 7: uporabnostna pravičnost v omrežju linearne topologije

Komunikacijski tok na dolgi povezavi ima indeks 0. Indeksi ostalim komunikacijskim tokom gredo po vrsti od 1 dalje. Vse povezave imajo prepustnost 1.

α	*poseben primer*	r_0	$r_{1,2}$	*skupna obremenjenost omrežja* $\sum r_i$
0	*maksimiranje celotne prepustnosti omrežja*	*0*	*1*	*2*
1	*proporcionalna pravičnost*	*0,33*	*0,67*	*1,67*
2	*minimiranje potencialnih zakasnitev*	*0,41*	*0,59*	*1,59*
∞	*max-min pravičnost*	*0,5*	*0,5*	*1,5*

Tabela 3: posebni primeri uporabnostne pravičnosti v omrežju linearne topologije

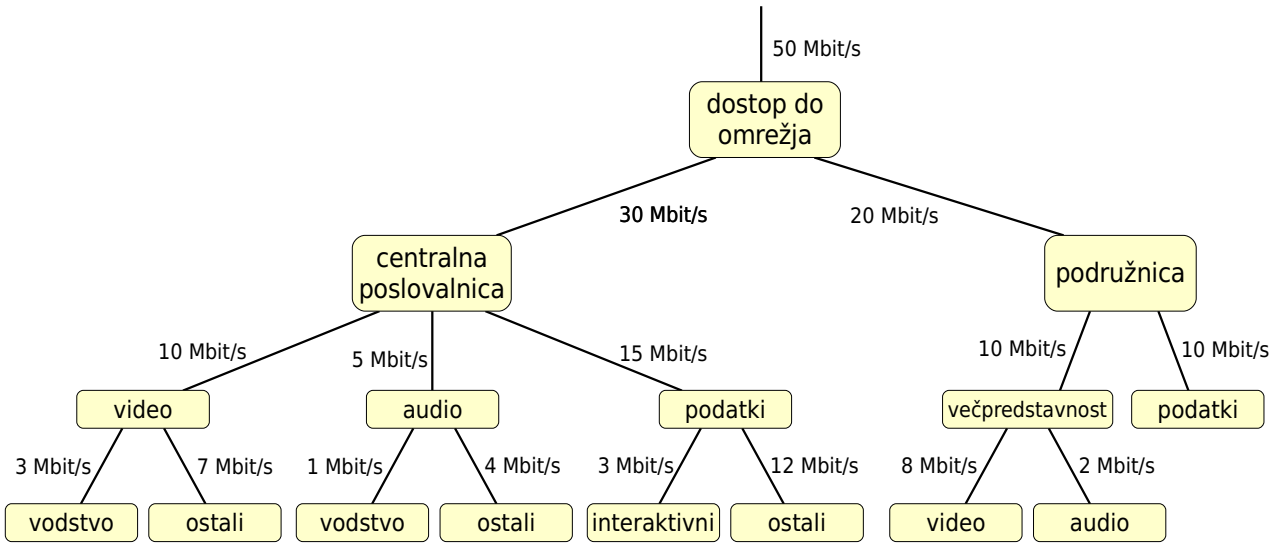
Opazimo lahko, da majhne vrednosti parametra *α* dajejo prednost izrabi omrežja na račun posameznega subjekta, medtem ko velike vrednosti parametra *α* povečujejo pravičnost s stališča posameznega subjekta, še posebej skromnejšim subjektom.

1.2.3 Hierarhično razvrščanje z razredi

Osnovna entiteta eno nivojskih razvrščanj je komunikacijski tok. Skupine paketov, ki pripadajo posameznemu toku, so obravnavane na isti način. Različni komunikacijski toki med sabo so obravnavani popolnoma neodvisno in enakovredno. V praksi se izkaže potreba po bolj kompleksnih delitvah omrežnih virov. Nekateri komunikacijski toki morajo biti obravnavani povezano v medsebojni odvisnosti. V ta namen poznamo strukturno obliko razvrščanj, kjer promet najprej delimo v hierarhično povezane razrede.

Razred predstavlja komunikacijski tok ali množico tokov, ki so združeni v skupine glede na administrativno določene lastnosti. Primeri razredov: področna organizacijska delitev, delitev po aplikacijah, delitev glede na transportni protokol, delitev glede na izvorni ali ciljni naslov in podobno. Slika 8 prikazuje primer hierarhične delitve 50 Mbit/s komunikacijskega kanala med dve poslovalnici organizacije. Pod organizacijsko delitvijo imamo še delitev glede na vrsto aplikacije in nivo nižje po prioritetah glede na tip uporabnikov.

Potrebno je poudariti, da je v posameznem razredu lahko več komunikacijskih tokov. Viri so torej dodeljeni razredom in ne direktno posameznim komunikacijskim tokom. Več audio tokov si bo delilo vire, ki so dodeljeni audio razredu.



Slika 8: hierarhično razvrščanje

V hierarhično urejene razrede delimo promet s filtri. Filter odloči v katerega od razredov bo posamezen paket razvrščen. Vsak podrazred lahko vsebuje še svoje filtre za finejše razvrščanje. Šele na koncu razvrstimo pakete v ustrezne čakalne vrste, katerim so prirejeni ustrezni algoritmi razvrščanja posameznih komunikacijskih tokov.

Vsakemu razredu je administrativno dodeljena določena količina omrežnih virov, v prikazanem primeru je to prepustnost omrežja, ki jo ima zagotovljeno v primeru, da jo potrebuje. V primeru, kadar nek razred ne potrebuje celega obsega rezerviranih omrežnih virov, pride do veljave hierarhično delovanje. Višek omrežnih virov se najprej ponudi prvemu nadrejenemu razredu znotraj svoje hierarhične skupine. Šele če ta še vedno ostane neizkoriščen, se razporedi navzgor po hierarhiji drugim višjim razredom, ki ga spet razdelijo po svoji hierarhični veji navzdol v deležih, ki so sorazmerni rezervacijam. V ta namen lahko razredom določimo nekaj dodatnih lastnosti:

Izoliranje/deljenje

Razredi, ki so določeni kot izolirani, ne bodo dovolili posojanja neizkoriščenih omrežnih virov drugim razredom. Omrežni viri ostanejo neizkoriščeni. Nasprotje od izoliranosti je deljenje, ko si razredi med sabo v okviru hierarhije dovolijo izposojati neizkoriščene vire.

Omejevanje/izposojanje

Izoliranje/deljenje iz druge perspektive je omejevanje/izposojanje. Razred označen kot

omejen si ne bo poizkušal izposoditi omrežnih virov od drugih. Če je označen za izposojanje, lahko poizkuša dobiti dodatne omrežne vire od razreda, ki jih trenutno ne potrebuje.

Če imamo dva razreda, ki sta med sabo izolirana in omejena, pomeni, da sta res zaklenjena v nastavljene okvire, saj ne bosta dovolila delitve svojih rezerviranih virov in tudi ne bosta poizkušala dobiti dodatnih virov od drugih.

Primeri hierarhičnih razvrščanj

Za hierarhično razvrščanje obstajajo hierarhične različice postopkov, ki si jih bomo ogledali v nadaljevanju. Od eno nivojskih se razlikujejo le po tem, da imajo prigrajene zmožnosti združevanja in hierarhičnega povezovanja. Zato ima večina postopkov razvrščanj tudi hierarhično različico. Nekaj primerov:

- hierarhično razvrščanje z vedrom in žetoni,
- hierarhično uteženo pravično razvrščanje (HWFQ – Hierarchical Wighted Fair Queuing),
- hierarhične pravične prenosne krivulje (HFSC Hierarchical Fair Service Curve),
- hierarhično krožno razvrščanje (HRR – Hierarchical Round Robin).

2. DEL
Postopki razvrščanj

2.1 Razvrščanje v prihajajočem vrstnem redu (FIFO – First In First Out)

V literaturi zasledimo tudi poimenovanje: »prvi prispel – prvi postrežen« (FCFS – First Come First Served).

To je najstarejši in najenostavnejši postopek razvrščanja in še danes največ v uporabi kot privzeti način. V cenenih usmerjevalnikih je razvrščanje v prihajajočem vrstnem redu edino vgrajeno razvrščanje.

Slika 9: razvrščanje v prihajajočem vrstnem redu

Princip delovanja je preprost, paketi prihajajo v eno samo čakalno vrsto in v istem vrstnem redu so tudi obdelani ter poslani naprej. Ves promet je obravnavan povsem enako. Do različnih komunikacijskih tokov je nepravičen, saj ne zagotavlja nikakršne enakomerne, oziroma pravične razdelitve prepustnosti ali drugih virov. Medsebojni vplivi med različnimi komunikacijskimi toki so veliki. Pohlepni izvori zlahka zavzamejo večino čakalne vrste in povzročijo velike zakasnitve ostalim komunikacijskim tokom. Oškodovani so zlasti protokoli, ki povratno krmilijo pretok glede na zasedenost omrežja (v IP okolju UDP protokol izrine TCP). Medsebojnega izoliranja komunikacijskih tokov ni.

Včasih je medsebojni vpliv tudi zaželen, saj deluje kot neke vrste blažilnik. Na račun drugih komunikacijskih tokov rafal posameznega toka sebi povzroči manjše povečanje zakasnitev, kot če bi komunikacija potekala v popolnoma izoliranih kanalih. Nepravičnost se bolj pokaže pri kratkih paketih, ki sledijo dolgim. Vendar pri tem razvrščanju ni nevarnosti, da bi prednostni komunikacijski toki izpodrinili toke nižjih prioritet kot pri prioritetnem razvrščanju. Prej ali slej so obdelani vsi paketi, če le niso preseženi dovoljeni časovni okviri ali če se čakalna vrsta ne prenapolni. Prihajajoči paketi, ki bi zatekli polno čakalno vrsto, se

namreč enostavno zavržejo.

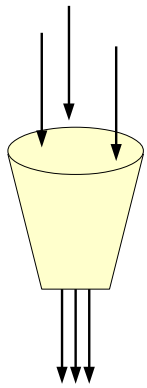
Delovanje razvrščanja v prihajajočem vrstnem redu je stabilno, če je število paketov N v čakalni vrsti končno in manjše od razpoložljivega pomnilnika. Littlov teorem pravi, da to velja v primeru, ko sta povprečna hitrost dotekanja paketov r in čas zadrževanja v sistemu T omejena.

$N = r\,T$ Littlov teorem

Razvrščanje v prihajajočem vrstnem redu ne omogoča nobenega aktivnega nadzora prometa.

2.2 Razvrščanje s puščajočim vedrom Leaky Bucket

Razvrščanje s puščajočim vedrom ureja kaotično in neenakomerno bruhanje paketov v urejen komunikacijski tok. Gre za različico razvrščanja v prihajajočem vrstnem redu z dodatno funkcionalnostjo.



Slika 10: razvrščanje s puščajočim vedrom

Vhodni promet v vedro je lahko zelo neenakomeren, v vedru se prispeli paketi začasno shranjujejo in v rednih nadzorovanih intervalih spuščajo naprej v omrežje s točno določeno hitrostjo r oktetov na sekundo. Vedro je omejene velikosti in lahko sprejme samo b oktetov. Dodatni prispeli paketi se prelijejo čez rob vedra – se zavržejo.

Pretok skozi puščajoče jedro se pogosto uporablja za uravnavanje komunikacijskega toka in za nadzor skladnosti največjega pretoka s prometno pogodbo.

Poglejmo si primer nadzora največjega pretoka z enostavnim puščajočim vedrom. Želimo nadzor največjega pretoka 1000 paketov na sekundo, torej moramo imeti pretok paketa na 1 ms. Velikost vedra naj bo 2 paketa. Tabela 4 prikazuje čase prispetja pet paketov, število paketov v vedru po prispetju in odločitev nadzora za vsak paket. Paketi se odpošiljajo naprej tik pred vsako celo milisekundo.

časi prispetja paketa (ms)	*število paketov v vedru*	*odločitev nadzora*
0,0	*1*	*skladen*
1,0	*1*	*skladen*
1,5	*2*	*skladen*
1,7	*2 (3)*	*ni skladen – se zavrže*
2,0	*2*	*skladen*

Tabela 4: odločitev nadzora za pet paketov pri nadzorovanju največjega pretoka

V primeru vidimo, da paketi prihajajo nekoliko prehitro. V intervalu od prve in pred drugo milisekundo prispejo namesto enega kar trije paketi. V vedru je prostor samo za dva, zato se tretji, ki je prispel ob 1,7 ms, zavrže. Tik pred 2 ms je en že odposlan naprej, zato lahko ob 2 ms že sprejmemo nov paket. Pri tem nam je ostalo polno vedro, kar pomeni, da bo v naslednjih intervalih še manj tolerance do predčasno prispelih paketov. Vedro se bo izpraznilo šele, če v kakšnem od intervalov sploh ne bo prispelega paketa ali če se bo v povprečju frekvenca prihodov vsaj nekoliko zmanjšala pod 1000 paketov na sekundo in se bodo s tem intervali prihodov nekoliko podaljšali.

Poznamo tudi posebne različice puščajočih veder, ki neskladnih paketov ne zavržejo. Namesto tega jih označijo in tako postanejo prvi kandidati za zavrženje drugod v omrežju, če pride do zamašitev. Možno je tudi, da se take pakete zadrži v posebni čakalni vrsti, od koder se odpošljejo naprej z nižjo prioriteto, torej šele, ko bo vedro prazno.

V zgornjem primeru smo pri merjenju pretoka šteli pakete. To je v redu za omrežja s enakimi dolžinami paketov. V omrežjih z različnimi dolžinami paketov namesto paketov štejemo dejansko količino podatkov v oktetih. Da postopek sploh deluje, mora vedro

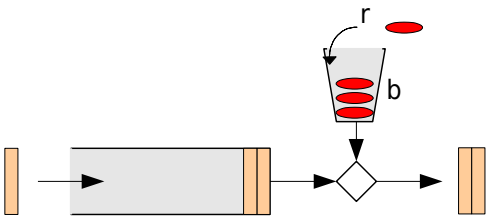
zadoščati vsaj za hranjenje enega paketa.

Razvrščanje s puščajočim vedrom ne izrablja optimalno omrežnih virov. Tudi če je izhodni kanal prost, se paketi v vedru vedno zadržijo in odpošljejo šele ob določenem trenutku, ki ustreza nastavljenemu pretoku. Tako delovanje uvršča razvrščanje s puščajočim vedrom med intenzivna razvrščanja.

2.3 Razvrščanje z vedrom in žetoni (TBF – Token Bucket Filter)

Razvrščanje z vedrom in žetoni je enostaven postopek za regulacijo toka. Dopušča odpošiljanje paketov z vnaprej administrativno določeno hitrostjo in občasno dovoljenimi rafali, ki to hitrost presegajo. Po načinu delovanja je tudi vedro z žetoni različica razvrščanja v prihajajočem vrstnem redu.

Razvrščanje z vedrom in žetoni je zelo točno in pretirano ne obremenjuje omrežnih in procesorskih virov, zato je najboljša izbira v primeru, če želimo preprosto umiriti in zgladiti tok paketov.



Slika 11: razvrščanje z vedrom in žetoni

Delovanje je preprosto in ga shematično prikazuje slika 11. Glavni del tvori števec – vedro, ki se v rednih intervalih z vnaprej določeno hitrostjo polni z virtualnimi koščki informacij imenovanimi žetoni. Žetoni prihajajo v vedro s konstantno hitrostjo *r*. To je hitrost, ki ustreza povprečnemu prometnemu pretoku, ki ga želimo nadzirati. Vedro lahko vsebuje samo administrativno določeno količino žetonov *b*. Vsak paket, ki gre skozi sistem, porabi en žeton. To pomeni, da gredo lahko paketi skozi sistem samo, če je dovolj žetonov oziroma z enako ali manjšo hitrostjo kot pritekajo žetoni. Če paketi dotekajo z manjšo hitrostjo, se višek žetonov nabira v vedru. Ko je vedro polno, se dotekajoči žetoni prelijejo – zavržejo. V primeru akumuliranih žetonov v vedru lahko pride do kratkotrajnega rafala podatkovnih

paketov, ki zapustijo sistem z največjo administrativno določeno hitrostjo ali z največjo hitrostjo, ki jo sistem zmore. Velikost vedra določimo, da ustreza dovoljenim velikostim takih rafalov.

Za razliko od puščajočega vedra z vedrom in žetoni nadzorujemo promet z spremenljivo bitno hitrostjo, ki ima določeni dve lastnosti: povprečen pretok in velikost občasnih rafalov. Velikost vedra *b* določa velikost rafalov, hitrost generiranja žetonov *r* pa omejuje povprečen pretok čez daljši čas. Največje število paketov *N*, ki gredo lahko skozi v času *t* je:

$$N = r\,t + b$$

Če ob prispelem paketu v vedru ni žetona, se šteje, da ni skladen s prometno pogodbo in se ustrezno ukrepa. Lahko se zavrže, označi ali naprej odpošlje z nižjo prioriteto.

Na primeru si oglejmo nadzor povprečnega pretoka 1000 paketov na sekundo, kar pomeni prihod žetona vsako milisekundo. Vedro naj bo veliko za 10 žetonov. Sprejem rafala paketov ne pomeni, da paketi prispejo ob istem času, saj komunikacijski kanal prenaša pakete zaporedno. Pri največji hitrosti si torej paketi sledijo zaporedno brez presledkov, s tako imenovano zaporedno zakasnitvijo. V tem primeru privzamemo, da je zaporedna zakasnitev 0,1 ms, kar pomeni, da pošiljanje enega paketa traja 0,1 ms. Pred začetkom je kanal dovolj časa prost, da se v vedru nabere začetna količina 10 žetonov.

časi prispetja paketov (ms)	*število žetonov v vedru*	*odločitev nadzora*
0,0; 0,1; 0,2; 0,3; 0,4; 0,5	*4*	*skladni*
1,0	*4*	*skladen*
2,0; 2,1; 2,2; 2,3; 2,4; 2,5	*0 (-1)*	*5 skladnih, 1 neskladen*

Tabela 5: odločitev nadzora z vedrom in žetoni

Tabela 5 prikazuje čase prispetja paketov, število žetonov po prispetju in odločitev nadzorne funkcije za vsak paket. Prihajajoči tok je sestavljen iz dveh rafalov po šest paketov in enega paketa med njima ob času 1,0 ms. Prvi rafal je sprejet v celoti, kar zmanjša število žetonov z 10 na 4. V času od 0 do 2 ms prispeta še 2 žetona. Enega porabi samostojen paket

ob 1,0 ms, tako za zadnji rafal šestih paketov ostane le 5 žetonov. Torej je 5 paketov skladnih s prometno pogodbo, medtem ko zadnji – šesti ni.

Dejanske izvedbe običajno ne obravnavajo števila paketov ampak količino podatkov – število podatkovnih oktetov. Na koncu tega dela, na strani 83, si bomo ogledali tudi primer uporabe več veder z žetoni kot način pravičnega razvrščanja.

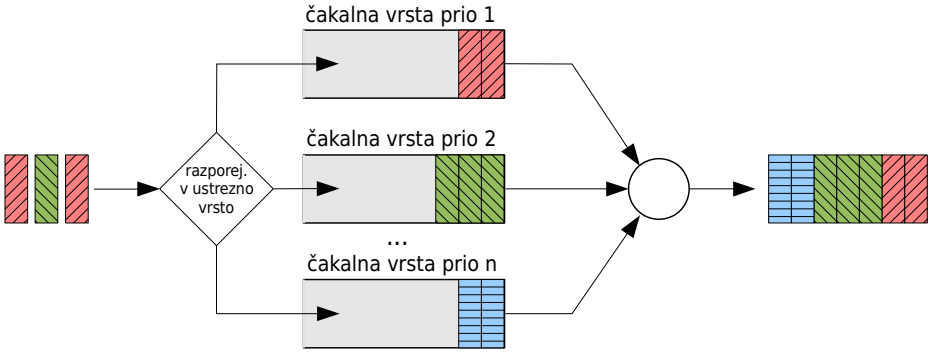
2.3.1 Kombiniran sistem – razvrščanje z dvojnim vedrom

Kadar želimo komunikacijskemu toku nadzorovati tri parametre: največji pretok, povprečni pretok in velikosti rafalov, uporabimo oba načina v paru: puščajoče vedro in vedro z žetoni.

S puščajočim vedrom najprej nadzorujemo največji pretok in skladnemu prometu potem z vedrom z žetoni še povprečni pretok ter velikosti rafalov. Skladen promet pri prvem nadzoru maksimalnega pretoka lahko drugi nadzor še vedno uvrsti med neskladen, če presega vrednosti nastavljenega povprečnega pretoka ali velikosti rafalov.

2.4 Razvrščanje po prioritetah (PQ – Priority Queuing)

Razvrščanje po prioritetah je veliko v uporabi med gradniki znotraj računalniških sistemov. Pri razvrščanju po prioritetah imamo fiksno število čakalnih vrst z različnimi prioritetami, v katere se po vnaprej znanih pravilih razvrščajo komunikacijski toki. Višja prioriteta vedno pomeni prednost pred nižjo. Ko je prenosni kanal prost, se vedno najprej izbirajo paketi iz čakalne vrste z najvišjo prioriteto. Paketi z nižjimi prioritetami pridejo na vrsto šele, ko ni nobenega paketa z višjo prioriteto več. Strežba posamezne prioritetne čakalne vrste se izvaja v prihajajočem vrstnem redu.



Slika 12: razvrščanje po prioritetah

To je zelo enostavno razvrščanje. Slabost razvrščanja po prioritetah je potencialna možnost, da je preveč paketov z visoko prioriteto in tisti z nizko prioriteto nikoli ne pridejo na vrsto. Zato se najvišja prioriteta praviloma uporablja samo za kritični promet, kot je nadzor in upravljanje omrežja. Delež visoko prioritetnega prometa moramo strogo nadzorovati in omejevati.

Višja prioriteta pomeni hkrati manjše zakasnitve, večjo prepustnost in manjše izgube.

Razvrščanje s prioritetami ne oblikuje prometa, ampak ga samo deli in prioritizira glede na pogoje nastavljene s filtri. Razvrščanje s prioritetami spada med neintenzivna razvrščanja in opravlja svojo nalogo le, kadar so paketi v čakalni vrsti. Če je prometa manj in paketom ni potrebno čakati, gredo brez razvrščanja takoj skozi sistem.

2.5 Pravična razvrščanja

Pravična razvrščanja predstavljajo celo skupino postopkov, ki lahko posameznim komunikacijskim tokom zagotavljajo pretok in omejujejo zakasnitve. Vsaj nekatera od pravičnih razvrščanj so vgrajena v večino zmogljivejših usmerjevalnikov, ki omogočajo upravljanje kakovosti storitev.

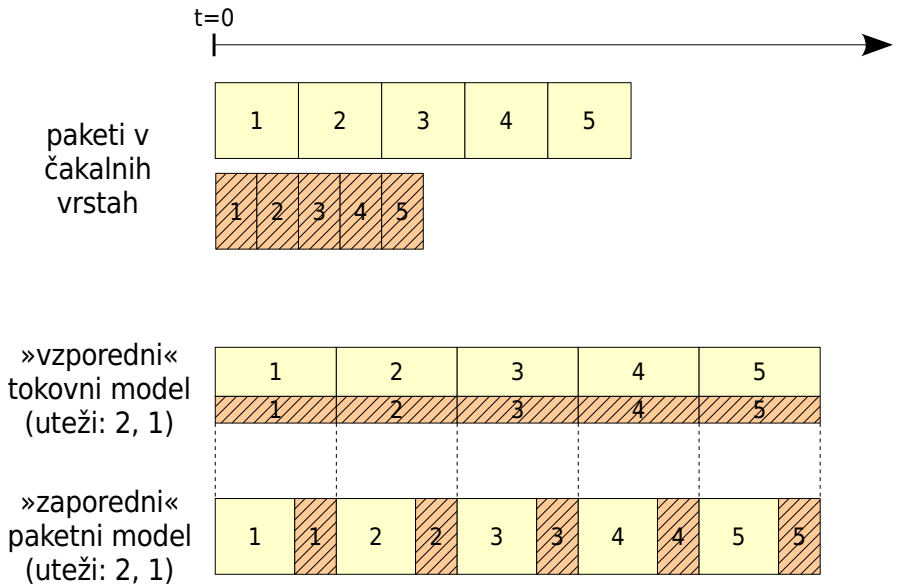
2.5.1 Tokovni model

Tokovni model je temelj večine pravičnih razvrščanj. Uteženo pravično razvrščanje (WFQ) običajno razlagamo s tokovnim modelom. Pri tokovnem modelu privzamemo, da je omrežni promet neskončno deljiv in da sistem lahko ob istem času obdeluje več komunikacijskih tokov vzporedno. V resničnem svetu so seveda paketi odpošiljani zaporedno,

drug za drugim. Različne dolžine paketov dodatno otežijo in vplivajo na točnost postopka. Enostavnost tokovnega modela nam omogoča lažjo predstavo in v večini primerov samoumevno izpeljavo rezultatov za paketni sistem.

Oglejmo si na primeru. Privzemimo, da imamo dva komunikacijska toka, ki si uteženo pravično delita izhodni kanal (slika 13). Prvi tok naj ima dodeljeno utež 2 in drugi 1. V enakem razmerju naj bodo tudi velikosti paketov. Pri tokovnem modelu sta dva paketa teh dveh komunikacijskih tokov obdelana hkrati, vzporedno. V prikazanem primeru se ujemajo začetni časi paketov obeh tokov kakor tudi končni.

V realnem paketnem sistemu si paketi sledijo drug za drugim. Paket prvega komunikacijskega toka bo odposlan pred paketom drugega toka. Še vedno pa velja dejstvo, da sta ob času, ko sta paketa obeh tokov že v celoti odposlana naprej v omrežje pri tokovnem modelu, prav tako tudi v celoti en za drugim odposlana oba paketa pri paketnem modelu.



Slika 13: tokovni model

2.5.2 Posplošeno razvrščanje (GPS – Generalized Processor Sharing)

Posplošeno razvrščanje je idealno pravično razvrščanje. Posplošeno razvrščanje je teoretični model razvrščanja, saj temelji na tokovnem modelu in predstavlja točno max-min uteženo pravično delitev.

Posplošeno razvrščanje spada med neintenzivne postopke. To pomeni, da deluje le, če

med komuniciranjem paketi čakajo v čakalni vrsti. V kolikor izvori generirajo manj prometa kot je njihov pravični delež, ne prihaja do razvrščanja in so paketi odposlani brez čakanja, takoj ko prispejo. Preostala prepustnost, ki je taki toki ne izkoristijo, se porazdeli med ostale v sorazmerju z utežmi, kar je lastnost utežene max-min pravičnosti.

Poenostavljeno lahko gledamo na posplošeno razvrščanje, kot bi vsak komunikacijski tok imel svojo ločeno čakalno vrsto. Procesiranje potem krožno v kratkem časovnem intervalu obdela infinitezimalni delček podatkov iz vsake neprazne čakalne vrste. Vsaka čakalna vrsta ima lahko prirejeno utež in količina obdelanih podatkov je sorazmerna z utežmi. V terminologiji posplošenega razvrščanja imenujemo komunikacijski tok, povezavo ali čakalno vrsto zasedeno, kadar ima čakajoče pakete v čakalni vrsti.

Primer: če imamo *N* zasedenih čakalnih vrst z enakimi utežmi, potem vsak komunikacijski tok prejme točno 1/*N* delež virov. Če je čakalna vrsta nezasedena, ker komunikacijski tok rabi manj, kot je njegov max-min pravični delež, se preostanek njegovih virov porazdeli med druge komunikacijske toke.

Vzemimo, da imamo *N* komunikacijskih tokov, ki jih obdeluje strežnik s hitrostjo strežbe *R*. Komunikacijskemu toku *i* dodelimo utež ϕ_i. *S(i, τ, t)* naj bo količina podatkov toka *i* v intervalu *(τ, t)*. Pri posplošenem razvrščanju za katerikoli zaseden tok *i* in katerikoli drug tok *j* v intervalu *(τ, t)* velja:

$$\frac{S(i,\tau,t)}{S(j,\tau,t)} \geq \frac{\phi_i}{\phi_j}$$

To pomeni, da na intervalu *(τ, t)* dobi tok *i* **vsaj pravičen delež** prepustnosti sorazmeren njegovi uteži:

$$\rho_i = \frac{\phi_i}{\sum_{j=1}^{N} \phi_j} R$$

kjer *N* predstavlja število vseh zasedenih komunikacijskih tokov v danem časovnem intervalu *(τ, t)*.

Omejevanje zakasnitev

Omejevanje zakasnitev je ena od pomembnih lastnosti posplošenega razvrščanja. Pogoj pri tem je, da je izvor omejen z vedrom in žetoni.

Pri omejevanju z vedrom z žetoni privzemimo, da je ob začetku vedro polno žetonov.

Prihaja rafal paketov b_i komunikacijskega toka i. Ti paketi porabijo vse žetone in so uvrščeni v čakalno vrsto, ki jo streže posplošeno razvrščanje. Ker je hitrost strežbe najmanj z zagotovljeno hitrostjo ρ_i po zgornji formuli, bo zadnji bit zadnjega paketa odposlan z največjo zakasnitvijo d_{max}:

$$d_{max} = \frac{b_i}{\rho_i} = \frac{b_i}{\frac{R\,\phi_i}{\sum \phi_j}}$$

Če je izvor omejen z vedrom in žetoni, ki ima parametre: dovoljen rafal (velikost vedra) b in pretok $r = \rho_i$ (enak pripadajočemu deležu posplošenega razvrščanja), potem ta postopek zagotavlja zakasnitve v mejah $\frac{b}{r}$. Zakasnitve so torej direktno odvisne od parametrov vedra z žetoni. Komunikacijskemu toku so s tem postopkom zagotovljeni vsaj enaki pogoji, kot bi jih nudil nadomestni namenski vod s prepustnostjo ρ_i.

Posplošeno razvrščanje je idealna shema, saj strežba infinitezimalnih delčkov podatkov dejansko ni izvedljiva. Zato v nadaljevanju predstavljamo nekaj različic posplošenega razvrščanja, ki jih lahko izvedemo v realnih sistemih.

2.5.3 Uteženo pravično razvrščanje (WFQ – Weighted Fair Queuing)

Uteženo pravično razvrščanje je aproksimirana realizacija posplošenega razvrščanja. Razlikuje se v tem, da gre za zaporedno posplošeno razvrščanje paketov in ne za vzporedno posplošeno razvrščanje komunikacijskih tokov. Uteženo pravično razvrščanje poizkuša posnemati posplošeno razvrščanje tako, da računa končne čase odpošiljanja paketov pri pripadajočem modelu posplošenega razvrščanja in to uporabi za razvrščanje paketov. V resnici se izračunani čas uporabi samo kot indeks za ureditev vrstnega reda odpošiljanja paketov in ne kot dejanski čas odpošiljanj. Na primer, če paket A konča odpošiljanje pred paketom B v sistemu posplošenega razvrščanja, bo v sistemu uteženega pravičnega razvrščanja paket A razvrščen pred paketom B.

Navidezni sistemski čas

Dejanski čas nas v resnici ne zanima, zato čase raje računamo kot celoštevilčni indeks

in z enoto v številu bitov. To naj bo naš navidezni sistemski čas. Če se paket začne ob času 0 in je dolg 1000 bitov, pravimo, da se konča ob času 1000. Tudi če se dejanska hitrost prenosa vmes spreminja zaradi novih komunikacijskih tokov, se spremeni samo dejanski čas prenosa. Trajanje prenosa v navideznem času je v vsakem primeru 1000. Pomembno je le, da ostaja vrstni red dogodkov nespremenjen med dejanskim in navideznim časom. S tem si poenostavimo računanje, saj novi toki ne spreminjajo že obstoječih izračunov končnih navideznih časov.

Tako definiran navidezni čas lahko enačimo tudi s števcem bitnih iteracij pri posplošenem razvrščanju. Spomnimo se, da je posplošeno razvrščanje krožno procesiranje infinitezimalnega delčka podatkov iz vsake neprazne čakalne vrste, kar je v praksi lahko najmanj en bit. Iteracija je en krog procesiranja vseh čakalnih vrst, ne glede koliko dejanskega časa traja. Čas trajanja iteracije je odvisen od števila aktivnih čakalnih vrst, ki jih posplošeno razvrščanje streže. Več čakalnih vrst kot streže, dlje časa bo trajala ena iteracija.

Izkaže se, da računanje končnega časa odpošiljanja v sistemu posplošenega razvrščanja kljub temu ni enostavno. Če si za začetek pogledamo enostaven primer, kjer vsi izvori pošiljajo več, kot je njihov pravični uteženi delež in so zato vsi regulirani. V tem primeru je končni čas odpošiljanja paketa *k* toka *i* enak končnemu času odpošiljanja prejšnjega paketa istega toka povečanemu za čas odpošiljanja trenutnega paketa. Če F_i^k predstavlja končni čas odpošiljanja paketa *k* toka *i*, dobimo:

$$F_i^k = F_i^{k-1} + \frac{L_i^k}{\Phi_i}$$

kjer sta L_i^k dolžina paketa *k* in ϕ_i utež toka *i*. V bolj splošnem primeru seveda ni nujno, da izvori ves čas oddajajo s polno hitrostjo in zato se lahko zgodi, da imamo med paketi proste časovne intervale:

$$F_i^k = max[F_i^{k-1}; V(a_i^k)] + \frac{L_i^k}{\phi_i}$$

a_i^k je linearno naraščajoča funkcija – čas prispetja paketa *k* toka *i* v dejanskem času in $V(a_i^k)$ odsekoma linearna naraščajoča funkcija – prispetje paketa v navideznem času v sistemu pripadajočega posplošenega razvrščanja. Hitrost naraščanja funkcije $V(t)$, navideznega sistemskega časa, narašča obratno sorazmerno vsoti uteži vseh **trenutno aktivnih** reguliranih tokov:

$$\frac{dV}{dt} = \frac{R}{\sum \phi_i}$$

kjer je *R* skupna prepustnost kanala za vse regulirane toke.

Primer izračuna končnih časov

Primer izračuna končnih navideznih časov si oglejmo na naslednjem primeru. Imamo dve enako uteženi povezavi *i in j* z vrednostjo uteži 1. Prepustnost kanala naj bo 1 enoto/s. Pakete poimenujemo kar s številčnim vrstnim redom: ena, dve, tri,… Dospetje paketov naj bo naslednje:

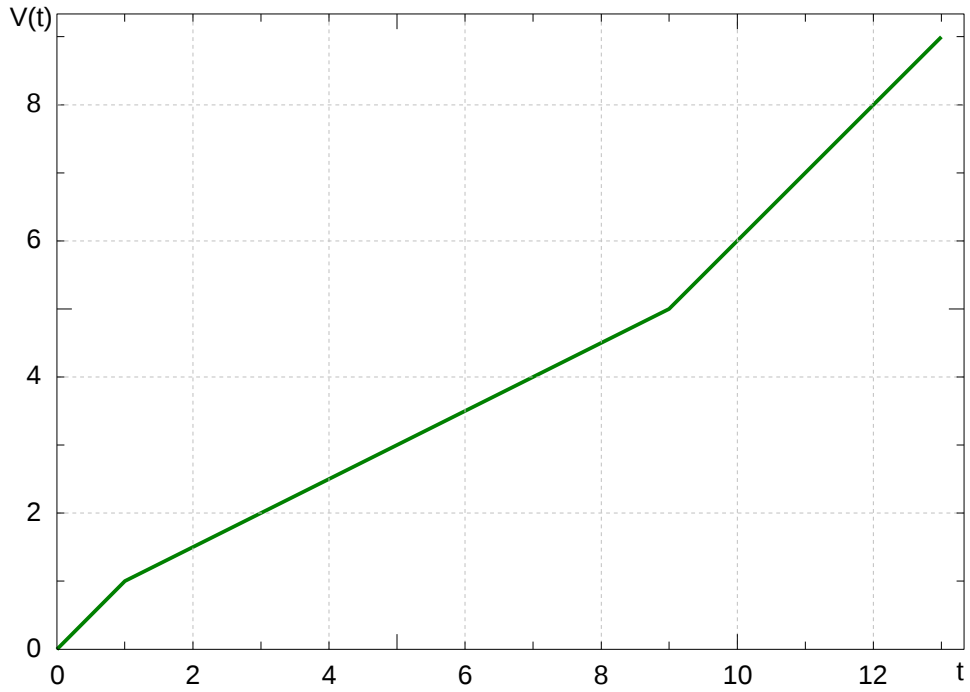
- ena: prispe ob času 0 po povezavi j in je dolžine 3 enote;
- dve: prispe ob času 1 po povezavi i in je dolžine 1 enoto;
- tri: prispe ob času 2 po povezavi i in je dolžine 1 enoto;
- štiri: prispe ob času 3 po povezavi i in je dolžine 2 enoti;
- pet: prispe ob času 5 po povezavi j in je dolžine 2 enoti;
- šest: prispe ob času 9 po povezavi j in je dolžine 2 enoti;
- sedem: prispe ob času 11 po povezavi i in je dolžine 2 enoti.

Primer bomo razložili tudi s pomočjo prenosnih diagramov v naslednjem poglavju in ga prikazuje slika 15, kjer si lahko dospetje paketov pogledamo tudi bolj nazorno. Vse začetne časovne vrednosti naj bodo 0. Izračuni končnih časov paketov so prikazani v nadaljevanju:

- ena: $F_j^1 = max(0;0)+3=3$, kjer je *V(0) = 0*, prvi paket na povezavi *j*;
- dve: $F_i^1 = max(0;1)+1=2$, kjer je *V(1) = 1*, prvi paket na povezavi *i*;
- tri: $F_i^2 = max(2;1{,}5)+1=3$, kjer je *V(2) = 1,5* in $F_i^1 = 2$;
- štiri: $F_i^3 = max(3;2)+2=5$, kjer je *V(3) = 2* in $F_i^2 = 3$;
- pet: $F_j^2 = max(3;3)+2=5$, kjer je *V(5) = 3* in $F_j^1 = 3$;
- šest: $F_j^3 = max(5;5)+2=7$, kjer je *V(9) = 5* in $F_j^2 = 5$,
- sedem: $F_i^4 = max(5;7)+2=9$, kjer je *V(11) = 7* in $F_i^3 = 5$.

Hitrost naraščanja navideznega časa je odvisna od števila aktivnih povezav in jo prikazuje slika 14.

Na intervalih *[0, 1]* in *[9, 12]* dejanskega časa je aktiven samo en komunikacijski tok, na intervalu *[1, 9]* ima navidezni čas polovično strmino, saj sta aktivna oba toka in posamezna iteracija traja dvakrat dlje časa.



Slika 14: potek navideznega časa

Razvrščanje uporablja končne čase pri izbiri paketa, ki ga je treba poslati. Dejanski vrstni red odpošiljanj za naš primer se zdi (z navedbo končnih časov):

dve(2), tri(3), ena(3), štiri(5), pet(5), šest(7), sedem(9).

Vendar ta vrstni red ni pravilen. Ne smemo namreč pozabiti na kronološki vrstni red dogodkov. Ob pričetku, ko prispe paket ena, v sistemu še ni nobenega drugega paketa niti še ne moremo vedeti, da bo ob času 1 prispel naslednji paket, ki bo imel nižjo vrednost končnega časa. Pri sestavljanju vrstnega reda moramo torej slediti času. Ob času 0, ko prispe paket ena, je to edini paket in **ob tistem trenutku ima ta paket najmanjši končni čas** z vrednostjo 3. Pravi vrstni red odpošiljanja paketov je tako naslednji:

- ena, odpošiljanje začne ob dejanskem času 0 in konča ob 3;
- dve, odpošiljanje začne ob 3 in konča ob 4;

- tri, odpošiljanje začne ob 4 in konča ob 5;
- štiri, odpošiljanje začne ob 5 in konča ob 7;
- pet, odpošiljanje začne ob 7 in konča ob 9;
- šest, odpošiljanje začne ob 9 in konča ob 11;
- sedem odpošiljanje začne ob 11 in konča ob 12.

Vrstni red paketov štiri in pet bi bil lahko tudi zamenjan, saj imata enak končni navidezni čas (5) in ob času odpošiljanja kateregakoli od njiju sta tudi že oba prisotna v sistemu.

Opazimo lahko, da se resnični končni časi ne ujemajo vedno z izračunanimi, tudi če primerjamo dejanske čase. Paket dve bi po izračunu začel ob navideznem času 1 in končal ob 2, kar ustreza dejanskima časoma 1 in 3. V resnici je čas odpošiljanja paketa dve od 3 do 4 po dejanskem času. Razlika nastopi, ker je izračun narejen za tokovni model, kjer so paketi lahko odposlani vzporedno. To tudi pojasnjuje, kako je sploh mogoče, da se eno enoto dolg paket odpošilja dve enoti časa. Ne smemo pozabiti, da izračunani končni časi služijo izključno za določitev vrstnega reda odpošiljanj.

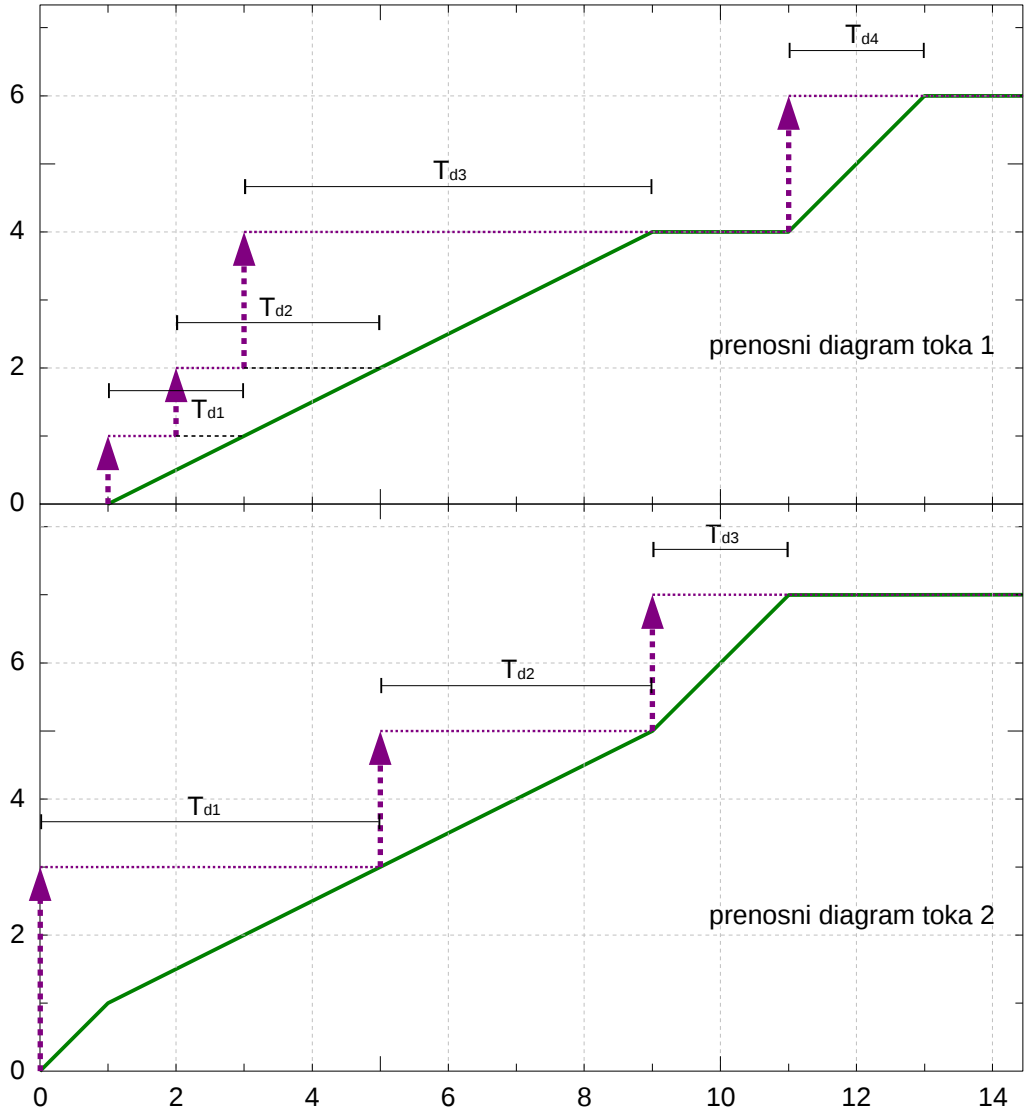
Za poenostavitev primera smo določili povezavam enake uteži. Če bi določili različne uteži, bi bile povezave obravnavane sorazmerno vrednosti utežem. Uteži povezavam lahko določimo administrativno ali avtomatično z uporabo signalizacijskih protokolov. Zapleteno izračunavanje navideznega časa se izvrši ob vsakem prispetju in odpošiljanju paketa, kar povzroča, da je izvedba tega postopka problematična pri zelo hitrih povezavah.

Prenosni diagrami

Računanje končnega časa odpošiljanja lahko bolj jasno razložimo s prenosnimi diagrami. Vzemimo isti primer iz prejšnjega poglavja, ko si dva komunikacijska toka enakomerno delita kanal in v prvem toku paketi prihajajo ob časih $t = 1, 2, 3$ in 11 z dolžinami $1, 1, 2$ in 2 ter v drugem ob $t = 0, 5$ in 9 z dolžinami $3, 2$ in 2. Slika 15 prikazuje prenosna diagrama.

Prekinjena črta kaže sprejemni proces – količino sprejetih podatkov v času. Vsaka puščica predstavlja sprejem paketa. Višina puščice predstavlja dolžino paketa. Ob sprejemu vsakega paketa se prekinjena črta dvigne za dolžino paketa. Polna črta predstavlja količino podatkov, ki jih obdelamo. V intervalu $[0, 1]$ je tok 1 še v mirovanju, ko že prispe paket komunikacijskega toka 2 ob času $t = 0$. Prvi paket toka 2 je torej obdelan s hitrostjo 1 v

časovnem intervalu *[0, 1]*. Ob času *t = 1* prispe prvi paket toka 1, zato sta oba toka obdelana s hitrostjo ½ do časa *t = 9*. Do tega časa zapustijo sistem trije paketi toka 1, naslednji paket pa še ne prispe. V intervalu *[9, 11]* je tok 2 spet obdelan s hitrostjo 1. Ob času *t = 11* prispe nov paket toka 1, ampak do tega časa so oddani že vsi paketi toka 2, zato se ta paket odpošlje s polno hitrostjo 1 in konča ob času *t = 13*.



Slika 15: prenosni diagrami

Z uporabo prenosnih diagramov lahko dobimo končne čase odpošiljanj v sistemu posplošenega razvrščanja. Če pogledamo tok 1, tretji paket prispe ob času *t = 3*, skupaj do takrat prispejo 4 enote podatkov. Polna črta kaže, da količina odposlanih podatkov doseže 4 ob času *t = 9*. Tretji paket toka 1 torej prispe ob času *t = 3* in se odda ob času *t = 9*. Celotni čas zakasnitve tega paketa je 6.

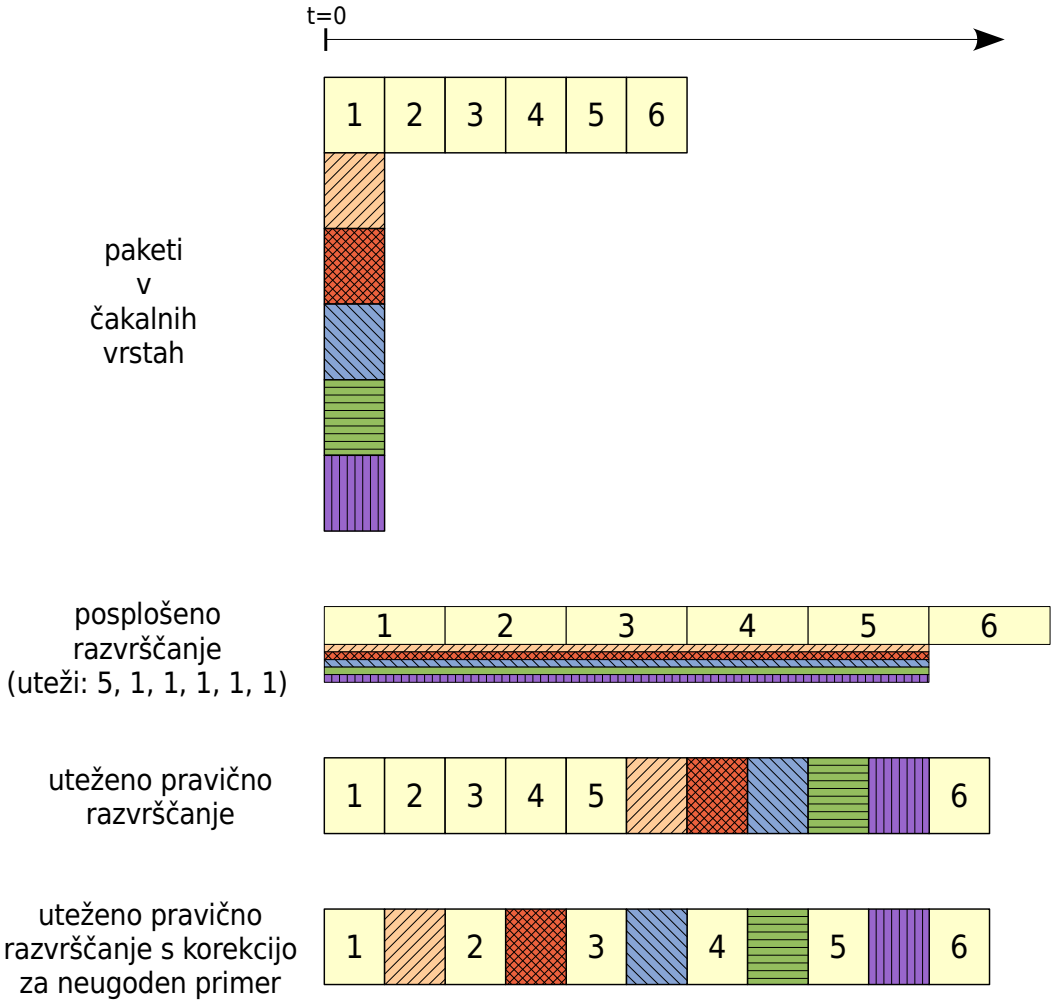
Razdalja med prekinjeno puščico in polno črto v smeri osi x predstavlja zakasnitev paketa. Diagram tudi kaže, da je končni čas odpošiljanja v sistemu posplošenega razvrščanja po kosih linearna funkcija časa, katere strmina se spreminja glede na število trenutno aktivnih tokov. **Težavnost računanja končnega časa odpošiljanja v sistemu posplošenega razvrščanja je v tem, da na novo aktivni toki lahko naknadno spremenijo končni čas odpošiljanja prejšnjih paketov ostalih komunikacijskih tokov.** Pomeni, da prispetje novega paketa v sistem povzroči, da bi morali na novo preračunati končne čase vsem paketom v sistemu.

Uteženo pravično razvrščanje s korekcijo za neugoden primer (WF^{2}Q – Worst–Case Fair Weighted Fair Queuing)

Obstaja veliko različic uteženih pravičnih razvrščanj, od katerih nekatera ponujajo poenostavljene algoritme izračunavanja časov in s tem določanja vrstnega reda odpošiljanj paketov, medtem ko druga ponujajo boljšo pravičnost v posebnih robnih primerih.

»Navadno« uteženo pravično razvrščanje ne nudi popolne pravičnosti v zelo kratkih časovnih intervalih. Uteženo pravično razvrščanje s korekcijo za neugoden primer rešuje ta problem z vpeljavo novega pravila pri izbiri paketa, katerega odposlati naslednjega. Pri uteženem pravičnem razvrščanju se vrstni red določa izključno glede na končni čas odpošiljanja v pripadajočem sistemu posplošenega razvrščanja. Uteženo pravično razvrščanje s korekcijo za neugoden primer je izboljšava uteženega pravičnega razvrščanja in upošteva tako začetni kot končni čas odpošiljanja paketov. **Naslednji obdelan paket bo tisti, ki ima najmanjši končni čas med paketi, ki bi jih posplošeno razvrščanje že začelo obdelovati.**

Najlažje to ponazorimo s pomočjo primera na sliki 16. Imamo šest komunikacijskih tokov sestavljenih iz enako dolgih paketov. Čas prispetja prvih paketov vseh tokov v čakalne vrste je ob času *t=0*. Prvi komunikacijski tok vsebuje 6 paketov, dodelimo mu utež 5. Ostali toki imajo vsak po en paket in utež 1. Pri posplošenem razvrščanju se komunikacijski toki porazdelijo vzporedno sorazmerno glede na vrednosti uteži. Konci paketov komunikacijskega toka 1 so ob časih *t={2, 4, 6, 8, 10, 12}* in zavzamejo pol prepustnosti kanala. Toki 2 do 6 vsi končajo ob času *t=10* in rabijo po desetino prepustnosti. Uteženo pravično razvrščanje zato najprej odpošlje večino paketov toka 1 – pakete od 1 do 5, potem pakete ostalih tokov in na koncu spet zadnji, šesti paket toka 1. Torej imamo najprej rafal prvega komunikacijskega toka in potem precejšnjo zakasnitev do naslednjega paketa tega toka.



Slika 16: uteženo pravično razvrščanje s korekcijo za neugoden primer

Uteženo pravično razvrščanje s korekcijo lepše razporedi pakete in sicer po že omenjenem pravilu: najprej tiste, ki imajo manjši končni čas, ampak le med paketi, ki bi jih posplošeno razvrščanje že začelo obdelovati. Najprej se enako kot pri uteženem pravičnem razvrščanju zvrsti prvi paket prvega komunikacijskega toka. Za njim ima drugi paket prvega toka končni čas v sistemu posplošenega razvrščanja (*t=4*) pred končnimi časi paketov ostalih tokov (*t=10*), vendar ob času *t=1* še ni v obdelavi posplošenega razvrščanja, saj začne šele ob času *t=2* in zato ne pride v poštev. Po prvem paketu toka 1 je tako ob času *t=1* na vrsti paket enega od preostalih tokov. Šele kot naslednji se zvrsti drugi paket toka 1, ker ob času *t=2* že izpolnjuje pogoj in ima najmanjši končni čas. Po istem pravilu se do konca izmenoma med paketi komunikacijskega toka 1 zvrstijo paketi vseh preostalih tokov.

V primerjavi s pravičnim uteženim razvrščanjem nam sistem s korekcijo nudi vsaj enake zakasnitve ob boljši pravičnosti. Nudi tudi bolj gladko prepletanje paketov. Tako je

uteženo pravično razvrščanje s korekcijo za neugoden primer še nekoliko bolj zapleteno od »navadnega« uteženega pravičnega razvrščanja, vendar zagotavlja boljšo pravičnost v kratkih časovnih intervalih.

2.5.4 Krožna razvrščanja (RR – Round Robin)

Krožno razvrščanje je enostavna izvedba posplošenega razvrščanja, kjer je neizvedljiva infinitezimalna količina podatkov zamenjana z enim paketom. Zaradi problema pravičnosti pri razvrščanju v prihajajočem vrstnem redu ima krožno razvrščanje za vsak komunikacijski tok ali skupino tokov svojo čakalno vrsto. Vsak prispeli paket je razvrščen v pripadajočo čakalno vrsto. Čakalne vrste so obdelane v krožnem procesu, tako da se iz vsake čakalne vrste obdela natanko po en paket. Prazne čakalne vrste se pri tem preskočijo. Ta način je pravičen v smislu, da vsak aktiven komunikacijski tok dobi priložnost odpošiljanja natančno enega paketa v vsakem krogu.

Pohlepnost posameznih komunikacijskih tokov sebi ne prinaša nobene prednosti. Pohlepnim komunikacijskim tokom se čakalne vrste napolnijo, kar jim povzroči povečanje zakasnitev, medtem ko ostali komunikacijski toki zaradi tega niso prizadeti.

Pri enako velikih paketih, kot na primer v ATM omrežjih, krožno razvrščanje zagotavlja pravično razporeditev prepustnosti. Pri paketih različnih velikosti, kot je primer v IP omrežjih, je pravičnost problematična. Vzemimo primer čakalne vrste z zelo velikimi paketi in več drugih čakalnih vrst z majhnimi paketi. Krožno razvrščanje bo hitro obdelalo vse majhne pakete in se dolgo zadržalo pri dolgem paketu. V povprečju bo komunikacijski tok velikih paketov prejel levji delež prepustnosti kanala. Naslednja težava navadnega krožnega razvrščanja je enaka obravnava vseh čakalnih vrst in tako ne moremo zagotoviti posebne obravnave ali rezervacij prepustnosti posameznim komunikacijskim tokom.

Uteženo krožno razvrščanje (WRR – Weighted Round Robin)

Uteženo krožno razvrščanje je nekoliko spremenjen način »navadnega« krožnega razvrščanja. Pri uteženem krožnem razvrščanju seznam tokov obdelujemo krožno in odpošiljamo število paketov sorazmerno glede na dodeljene uteži. Kot primer vzemimo dva

komunikacijska toka: *A* z utežjo 1 in *B* z utežjo 2, ki si delita skupni komunikacijski kanal. V takem sistemu odpošljemo po en paket toka *A* in za tem dva paketa toka *B*. Naprej postopek krožno ponavljamo. Uteženo krožno razvrščanje deluje dobro pri paketih enake dolžine, oziroma kadar so znane povprečne dolžine paketov komunikacijskih tokov. V nasprotnem primeru uteženo krožno razvrščanje prepustnosti ne deli pravično.

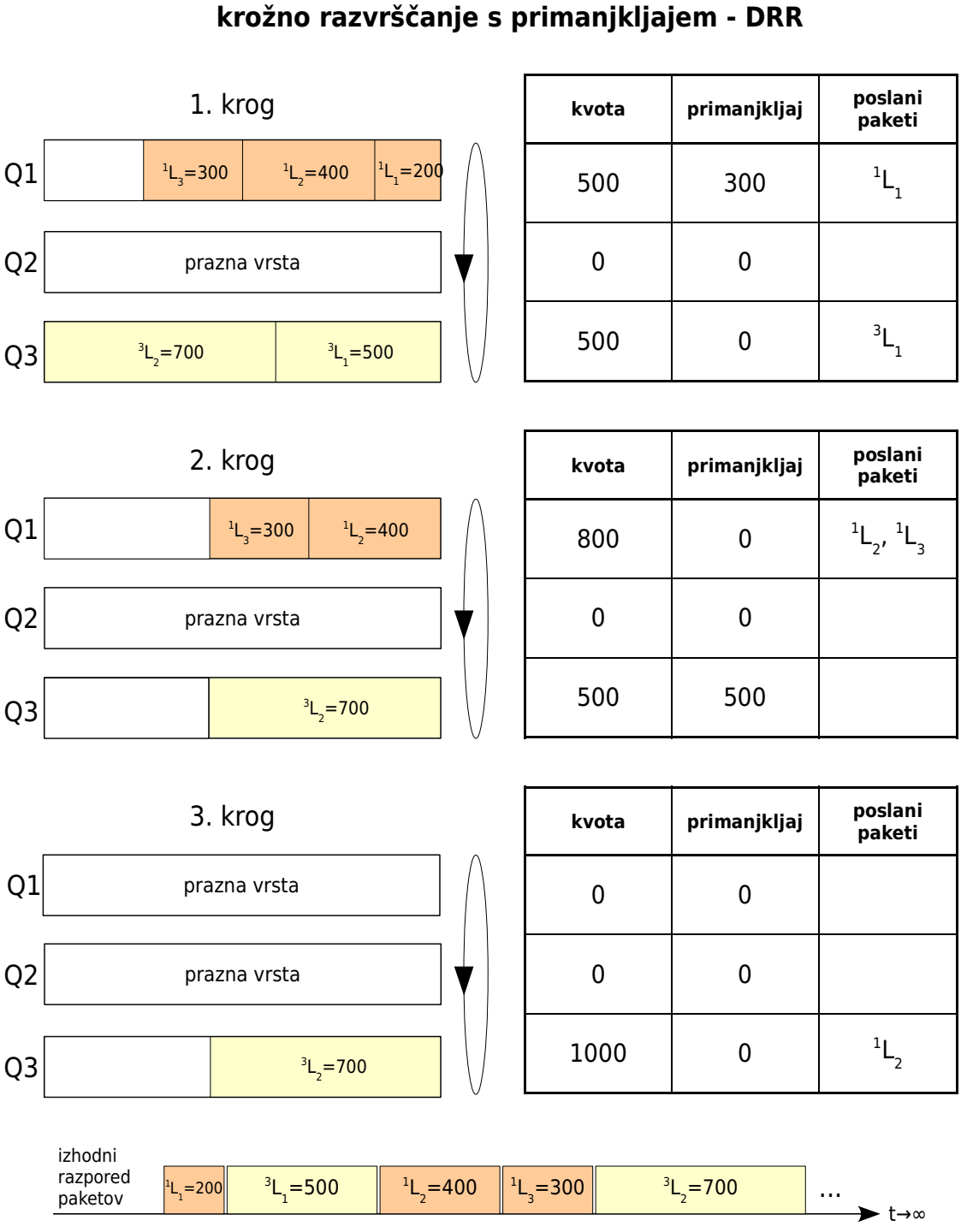
Vzemimo za primer tri ATM izvore – imajo enake velikosti celic – z utežmi *0,75; 1,0* in *1,5*. Če te uteži normaliziramo v cela števila, potem bo pri teh izvorih obdelano 3, 4 in 6 ATM celic v vsakem krogu.

Uteženo krožno razvrščanje je v nekaterih primerih nepravično v krajših časovnih intervalih. Če podajamo celoštevične uteži in te pomenijo kar število obdelanih paketov posameznega komunikacijskega toka, moramo izbirati čim manjše vrednosti, ki še zagotavljajo želeno medsebojno razmerje. Če namesto uteži 1 in 2 izberemo 1000 in 2000, bomo namesto enega in potem dveh paketov vsakega toka prenašali rafale 1000 in za tem 2000 paketov skupaj. Povprečno čez daljši čas bo rezultat enak, medtem ko bodo v krajših intervalih toki veliko bolj turbulentni in nepravični en do drugega.

Krožno razvrščanje s primanjkljajem (DRR – Deficit Round Robin)

Krožno razvrščanje s primanjkljajem je podobno uteženemu krožnemu razvrščanju, s tem da upoštevamo spremenljivo dolžino paketov. Pri krožnem razvrščanju s primanjkljajem vsakemu toku priredimo števec za primanjkljaj, ki ima na začetku vrednost 0. Imamo tudi vnaprej nastavljeno kvoto dovoljene količine prenesenih podatkov. V enem ciklu iz vsakega toka obdelamo toliko paketov, kolikor dovoljuje kvota prenesenih podatkov. Ko velikost naslednjega paketa presega preostalo kvoto, se neizrabljena kvota prepiše v števec za primanjkljaj. Toliko sistem še dolguje toku. V naslednjem ciklu se ta količina prišteje novi začetni kvoti. Kvoto nastavimo administrativno za posamezen tok in je v bistvu utež obdelave. Če imamo prekinitve v toku, se števec za primanjkljaj briše in tok ne izkoristi celotne dovoljene kvote.

Delovanje krožnega razvrščanja s primanjkljajem si poglejmo na primeru. Slika 17 prikazuje tri čakalne vrste: Q1 do Q3.



Slika 17: krožno razvrščanje s primanjkljajem

Začetne vrednosti vseh števcev za primanjkljaj naj bodo 0 in stanje čakalnih vrst naslednje:

- Q1 vsebuje tri pakete velikosti 200, 400 in 300 oktetov;
- Q2 je prazna;

- Q3 vsebuje dva paketa velikosti 500 in 700 oktetov.

Vsem čakalnim vrstam dodelimo enako kvoto dovoljene količine oddanih podatkov v enem ciklu. Ta znaša 500 oktetov. Kvote med čakalnimi vrstami bi bile lahko tudi različne.

1. krog:

Čakalni vrsti Q1 in Q3 dobita začetno kvoto 500, kar zadostuje za odpošiljanje prvega paketa iz vsake vrste. Q1 odpošlje paket dolžine 200 oktetov in Q3 500 oktetov. Čakalno vrsto Q2 proces popolnoma preskoči, saj nima čakajočih paketov.

Čakalni vrsti Q1 po poslanem prvem paketu od kvote ostane še 300, kar ne zadostuje za pošiljanje drugega paketa. Vrednost se shrani in prišteje začetni vrednosti naslednjega cikla. Preostala kvota se imenuje primanjkljaj, saj je bila čakalna vrsta Q1 v tem ciklu za to vrednost prikrajšana. Čakalna vrsta Q3 v tem ciklu svojo kvoto izkoristi v celoti, primanjkljaj je 0.

2. krog:

Vse čakalne vrste razen Q2, ki je prazna in ne sodeluje v procesu, prejmejo novo kvoto v vrednosti 500. Kvoti čakalne vrste Q1 se poleg tega prišteje primanjkljaj iz 1. cikla in dobi vrednost 800. To zadostuje za odpošiljanje obeh čakajočih paketov dolžine 400 in 300. Od kvote preostane še 100, kar se ne shrani, saj v čakalni vrsti ni več čakajočih paketov. To onemogoča kopičenje kredita prek daljšega obdobja in zagotavlja bolj gladek pretok. Čakalna vrsta Q2 ima kvoto 500. To ne zadostuje za odpošiljanje čakajočega paketa dolžine 700. Celotna kvota postane primanjkljaj te vrste za naslednji krog.

V tem ciklu naletimo na zanimivost. Obe čakalni vrsti Q1 in Q3 imata enako začetno količino čakajočih podatkov. Kljub temu lahko zaradi primanjkljaja od prej vrsta Q1 odpošlje celotno količino, medtem ko Q3 čisto nič.

3. krog:

Čakalna vrsta Q3 spet pridobi h kvoti novih 500. Skupaj s primanjkljajem ima že 1000 in končno odpošlje paket velik 700 oktetov. Končna vrednost števca je 0, ker je vrsta prazna. Drugi dve prazni vrsti ne sodelujeta v procesu.

Za gladek pretok in dobro pravičnost pri krožnem razvrščanju s primanjkljajem določimo kvoto, ki omogoča odpošiljanje največ enega paketa iz vsake čakalne vrste. Postopek namreč ni pravičen v časovnih intervalih krajših od časa odpošiljanja podatkov, kolikor dovoljuje kvota. Prednost krožnega razvrščanja s primanjkljajem je enostavna izvedba.

Poznamo tudi manj razširjeno različico krožnega razvrščanja s primanjkljajem in sicer kreditno krožno razvrščanje, CRR – Credit Round Robin. V tem primeru obdelamo en paket več, kot dovoljuje kvota in si količino izposodimo od naslednje začetne kvote.

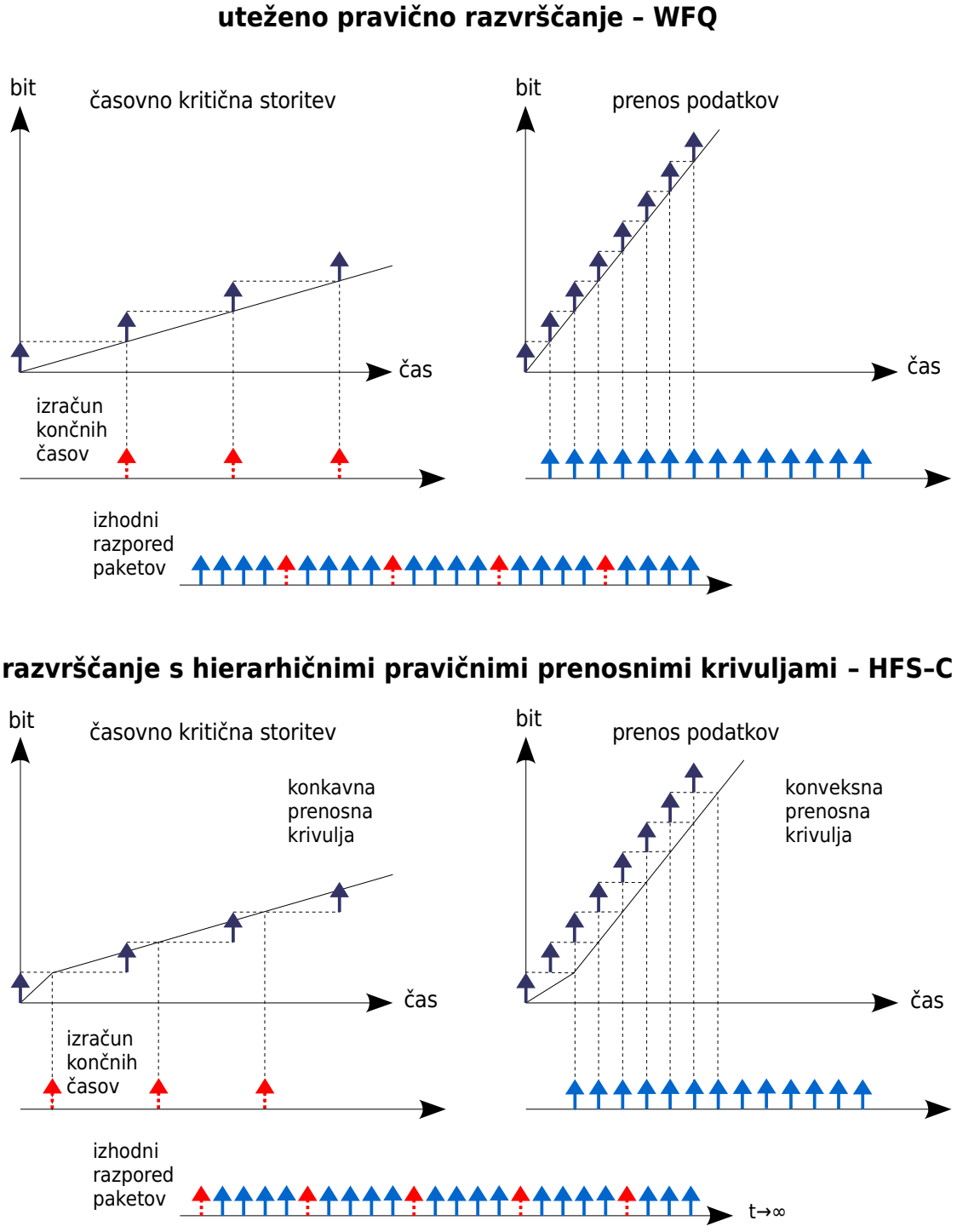
2.5.5 Hierarhične pravične prenosne krivulje (HFSC – Hierarchical Fair Service Curve)

Tradicionalna pravična razvrščanja paketov skušajo zagotoviti pravično delitev omrežnih virov med komunicirajoče subjekte. Pod omrežne vire največkrat štejemo prepustnost omrežja in sprejemljive zakasnitve. Razvrščanja obravnavana do zdaj imajo linearno prenosno funkcijo, ki hkrati določa tako prepustnost kot tudi zakasnitve. To pomeni, da ti dve lastnosti obravnavamo povezano. Večja prepustnost avtomatično pomeni krajše časovne zakasnitve in obratno.

Pri pravičnih razvrščanjih z linearno prenosno funkcijo z manjšo zvijačo lahko zmanjšamo zakasnitve tako, da komunikacijskemu toku povečamo administrativno določeno utež. Hkrati seveda povečamo potencialno prepustnost, česar morda ne želimo. Prepustnost moramo zato omejiti na drug način ali zaupati, da izvor tega ne bo zlorabil in ne bo oddajal preveč podatkov. Ločitev omejevanja zakasnitev in zagotavljanja prepustnosti lahko dosežemo tudi na drugačen način – z razvrščanjem s hierarhičnimi pravičnimi prenosnimi krivuljami.

Glavna lastnost razvrščanja s hierarhičnimi pravičnimi prenosnimi krivuljami je, da ločuje povezavo med zagotavljanjem prepustnosti in omejevanjem zakasnitev. Prenosna karakteristika tega postopka ni več enostavna linearna funkcija, ampak je konveksna ali konkavna krivulja. Aproksimiramo jo s prelomljeno linearno funkcijo.

Ker gre za novejši postopek, razvrščanje s hierarhičnimi pravičnimi krivuljami že od začetka predvideva uporabo v hierarhični postavitvi. Delovanje si najlažje ponazorimo s pomočjo prenosnega diagrama in primerjavo z uteženim pravičnim razvrščanjem.



Slika 18: razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami

Kot je bilo prikazano v poglavju o prenosnih diagramih na strani 37, pravična razvrščanja računajo skrajni čas odpošiljanja s pomočjo prenosnega diagrama v pripadajočem sistemu posplošenega razvrščanja. Pri tem bolj strma prenosna karakteristika pomeni večjo prepustnost in manjše časovne zakasnitve ter bolj položna manjšo prepustnost in hkrati večje

zakasnitve.

Če pogledamo na primeru s slike 18, želimo preko izhodnega 10 Mbit/s kanala prenašati časovno kritično storitev – rdeči paketi, ki ima pretok 2 Mbit/s. Hkrati želimo prenašati tudi podatke z 8 Mbit/s – modri paketi. Prenos pri časovno kritični storitvi zahteva manjše časovne zakasnitve. Običajno uteženo pravično razvrščanje nam v tem primeru tega ne nudi, ker ima prenos podatkov večjo utež in zato prednost pri prenosu. Če namesto uteženega pravičnega razvrščanja uporabimo razvrščanje s pravičnimi prenosnimi krivuljami in za kritično storitev nastavimo konkavno prenosno karakteristiko, nam to kljub manjši prepustnosti zagotovi časovno prednost paketov in s tem manjše zakasnitve. Za prenos podatkov nastavimo konveksno obliko prenosne krivulje, ki zagotavlja večjo prepustnost, vendar ob večjih časovnih zakasnitvah.

Delovanje temelji na značilnostih paketnih omrežij, kjer velja:

- pretok znotraj časovnega intervala pošiljanja posameznega paketa je vedno največja možna, ki jo omejuje tehnologija in izvedba omrežja;
- ob kateremkoli trenutku se vedno prenaša največ le en sam paket.

Manjši pretok torej ne pomeni, da se paketi skozi omrežje pretakajo počasneje. Pretok določa izključno velikost paketov in njihova frekvenca pošiljanja skozi omrežje. Iz tega sledi, če prvi paket neke seje odpošljemo nekoliko prej in hkrati obdržimo frekvenco odpošiljanja naslednjih paketov nespremenjeno pri isti vrednosti, se pri pretoku nismo dosti pregrešili, le zakasnitve tudi vseh naslednjih paketov bodo manjše. To v praksi pomeni, da moramo spremeniti zakasnitev – ga pospešiti ali zavreti – samo prvemu paketu ali morda prvim nekaj paketom. V začetnem delu spremenjen naklon prenosne karakteristike naredi ravno to, kot nazorno prikazuje slika 18.

2.6 Naključno zgodnje odmetavanje (RED – Random Early Drop)

Za zaokrožitev tematike si na koncu oglejmo še nekoliko drugačne pristope pri upravljanju čakalnih vrst. Do zdaj smo si ogledali akcije pri začetku čakalne vrste, kjer podatki čakalno vrsto zapuščajo in gredo v obdelavo. Pri začetku vrste lahko obdelujemo podatke deterministično z izbranimi postopki. Na drugi strani podatki prihajajo v čakalno vrsto in to je naključen proces. Na prihajajoče podatke nimamo neposrednega vpliva. Vendar

se tudi tam odvijajo določeni procesi. Najbolj običajen proces je zapolnitev pomnilnika čakalne vrste. Pomnilnik je dejanski fizični vir, ki ga ni v neomejeni količini. Tudi če bi lahko imeli neomejeno čakalno vrsto, bi to pomenilo izredno dolgo čakanje na obdelavo in posledično ogromne zakasnitve pri komunikaciji, oziroma iztek časovnikov in s tem ponovno pošiljanje paketov, ki bi kakor pri verižni reakciji še dodatno daljšali čakalne vrste.

Postopek, ki ga običajno uporabljajo usmerjevalniki in za katerega sploh ni potreben poseben algoritem, saj nastane tako rekoč sam od sebe, lahko poimenujemo »zavrzi rep« (tail drop). Algoritem zavrzi rep vsebuje pomnilnik, ki je končne dolžine. Ko čakalna vrsta zapolni ves razpoložljiv pomnilnik, se zavrže ves dodatni promet, ki presega razpoložljivo dolžino. To je zelo nepravično delovanje do posameznih komunikacijskih tokov. Ker se paketi izgubljajo istočasno pri več komunikacijskih tokih, sinhrono in z enakimi postopki za okrevanje reagirajo tudi njihovi izvori. Dejanje vodi v obsežno in hkratno ponovno pošiljanje paketov iz množice izvorov, ki lahko v ponavljajočih rafalih vedno znova zapolnijo čakalno vrsto.

Da se izognemo temu stopničastemu prehodu med normalnim delovanjem in množičnim izgubam paketov, lahko namesto algoritma zavrzi rep uporabimo naključno zgodnje odmetavanje.

Naključno zgodnje odmetavanje statistično naključno zavrže nekaj paketov še preden količina prometa doseže zgornjo mejo. Postopek sproti določa verjetnost, da se paket zavrže na podlagi trenutne dolžine čakalne vrste. Če so čakalne vrste prazne, je verjetnost, da se paket zavrže približno enaka 0. Ko se dolžina čakalne vrste približuje določenemu maksimumu, se verjetnost povečuje proti 1. Zaradi naključnosti postopka paketov ne izgubljajo vsi komunikacijski toki istočasno[2].

Posledično samoregulativni izvori veliko bolj umirjeno zmanjšujejo količino ponujanega prometa in zato ne prihaja do močnih rafalov ponavljajočega prometa od več izvorov hkrati. Poleg enakomernejše prepustnosti nam naključno zgodnje odmetavanje prinaša tudi krajše čakalne vrste in s tem bolj enakomerne in predvidljive zakasnitve.

Naključno zgodnje odmetavanje je bolj pravično od postopka zavrzi rep. Več kot izvor oddaja, večja je verjetnost, da se zavržejo ravno njegovi paketi. To na drugi strani povzroča težave pri uteženih razvrščanjih, ki dajejo določenim izvorom pravice do večjega pretoka. Potrebujemo torej uteženi algoritem tudi za odmetavanje paketov – uteženo naključno zgodnje odmetavanje (WRED – Weighted Random Early Drop). Obstaja tudi podobna

2 *Nesinhroniziranost postopka je pomembna.*

različica, ki omrežni promet najprej deli na posamezne toke in izvaja algoritem na vsakem komunikacijskem toku posebej: FRED – Flow Based (W)RED. Omenjene različice vsebujejo več čakalnih vrst z različno nastavljenimi parametri: prag zasedenosti in verjetnost, da se paket zavrže. Pakete ob prispetju razvrščamo v čakalne vrste na podlagi njihovih lastnosti. Na ta način lahko bolj učinkovito upravljamo promet, saj lahko dopustimo, da se z večjo verjetnostjo zavržejo samo paketi manj pomembnega prometa.

Naključno zgodnje odmetavanje je enostaven algoritem primeren za hitra hrbtenična omrežja, kjer si ne moremo privoščiti sledenja in pravičnega razvrščanja posameznih komunikacijskih tokov.

2.7 Razvrščanja namenjena testiranju

2.7.1 Omrežni emulator (NETEM – Network emulator)

Omrežni emulator je testno razvrščanje vgrajeno v nekatere operacijske sisteme, ki oponaša lastnosti delovanja realnega prostranega omrežja. Omrežni emulator omogoča naslednje lastnosti pri razvrščanju paketov:

- emuliranje zakasnitev prostranih omrežij,
- časovno kolebanje zakasnitev,
- izgubljanje paketov,
- podvojevanje paketov,
- kvarjenje paketov,
- zamešanje vrstnega reda paketov.

Vse naštete lastnosti so pri običajnih omrežjih nezaželene, zato je razumljivo, da omrežni emulator ni primeren za običajno uporabo, ampak samo za razvoj in testiranje protokolov in omrežij. Pri naštetih lastnostih je običajno mogoče nastavljati pogostost ali statistično porazdelitev verjetnosti pojavov. Več o lastnostih in uporabi v [15].

2.8 Izvedba

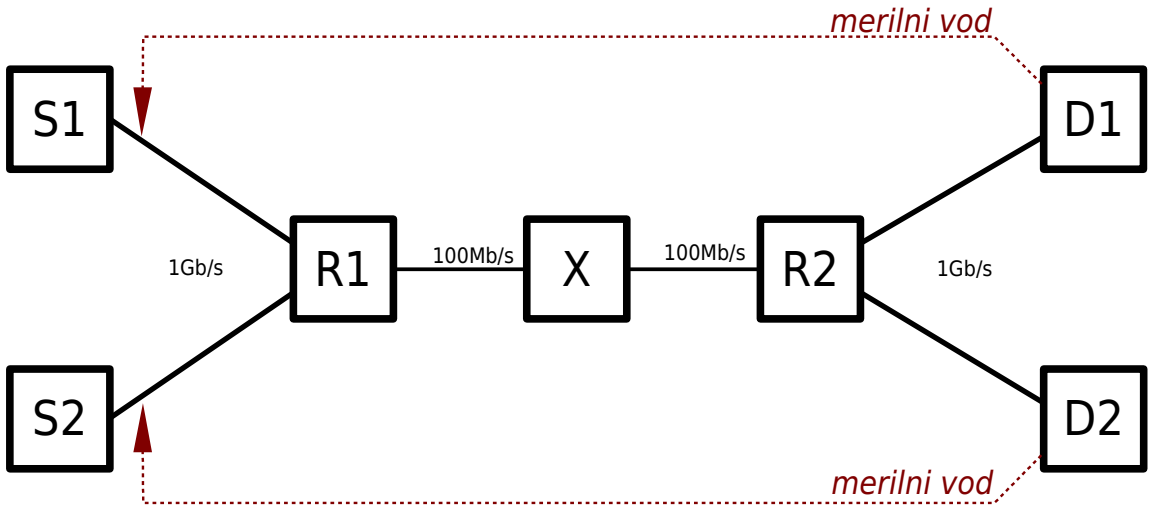
Večina od naštetih naprednih tehnik razvrščanj je razvita na raznih unix operacijskih sistemih za IP okolje. Linux vsebuje orodje iproute2 za nastavljanje raznih razvrščanj, ki so vgrajena že v samo jedro operacijskega sistema. Alternate queuing (ALTQ) je naslednji nabor pripomočkov, ki je na voljo v BSD unix okoljih. V naslednjem delu si oglejmo uporabo in rezultate tc (traffic control) orodja iz paketa iproute2 za urejanje mrežnega prometa na linux sistemih.

3. DEL
Praktični prikaz razvrščanj in meritve v dejanskem omrežju

Do zdaj obravnavane tehnike razvrščanj prikažimo na praktičnem primeru. Meritve nazorno prikazujejo različne lastnosti posameznih postopkov.

3.1 Testno omrežje

Za testni primer izberemo TCP/IP omrežje, kot ga prikazuje slika 19. Vsa vozlišča so izvedena z računalniškimi sistemi z operacijskim sistemom linux. V času meritev je bila aktualna in pri meritvah uporabljena različica jedra 2.6.38.7[3]. Linux že privzeto vsebuje zelo veliko opcij pri upravljanju omrežij in komunikacijskih tokov. Posledično se danes že v osnovi uporablja tudi v vedno več komercialnih omrežnih elementih.



Slika 19: shema omrežja

3.1.1 Topologija omrežja

Na skrajni levi strani imamo dva izvora S1 in S2, ki pošiljata promet proti ciljema skrajno desno, D1 in D2, in sicer tako, da komunikacija poteka od izvora S1 proti cilju D1 in

3 *originalna, imenovana tudi vanilla različica – http://www.kernel.org*

od izvora S2 proti cilju D2.

Komunikacijska toka obeh izvorov se združita v usmerjevalniku R1, ki ju usmerja naprej proti vozišču X. Pri izhodu iz omrežja usmerjevalnik R2 komunikacijska toka spet razdruži in vsakega posebej posreduje proti svojemu končnemu cilju.

Na sredini omrežja imamo nameščeno vozlišče X, ki emulira delovanje realnega prostranega omrežja z razvrščanjem NETEM opisanim na strani 49. V omrežje lahko vnaša zakasnitve, občasno lahko izgublja, podvaja in celo kvari pakete ali včasih nekoliko zameša njihov vrstni red. V našem primeru smo emulacijo nastavili dokaj enostavno. V omrežje samo vnaša dodatno zakasnitev v trajanju 5 ms. To naredimo za vsako smer posebej z naslednjimi nastavitvami:

```
tc qdisc add dev eth1 root netem delay 5ms
tc qdisc add dev eth2 root netem delay 5ms
```

S tem se delovanje omrežja in komunikacijskih protokolov približa realnemu delovanju v znatno večjem omrežju. Ostalih parazitnih lastnosti nismo vključevali, da so bila opazovanja in meritve bolj predvidljive.

Pravilno delovanje preizkusimo enostavno s programom ping od izvora do cilja, ki mora pokazati približno 10 ms zakasnitve, 5 ms v vsaki smeri.

```
ping 6.6.6.4
PING 6.6.6.4 (6.6.6.4) 56(84) bytes of data.
64 bytes from 6.6.6.4: icmp_req=1 ttl=61 time=10.4 ms
64 bytes from 6.6.6.4: icmp_req=2 ttl=61 time=10.1 ms
```

3.1.2 Potek komunikacije

Dostopovni deli omrežja, to so vodi, ki povezujejo izvorna in ciljna vozlišča z omrežjem, so izvedeni z 1 Gb/s ethernet povezavami. Vodi znotraj omrežja, od usmerjevalnika R1 do usmerjevalnika R2, pa so 100 Mb/s ethernet povezave. To ima za posledico, da pri prehodu iz hitrih vodov od izvorov v počasnejši vod omrežja v

usmerjevalniku R1 prihaja do zgostitve prometa. Pri izhodni povezavi usmerjevalnika R1 v omrežje nastaja čakalna vrsta paketov. Teoretično je tudi možno, da od obeh izvorov prispeta paketa natančno ob istem času.

V usmerjevalniku torej poteka odločitev, kako obravnavati pakete, ki jih ta pošilja naprej v omrežje, katerim dati prednost in katere zadržati. Največje zakasnitve prometa so torej ravno v tem usmerjevalniku R1 in tu želimo z različnimi pristopi in obravnavo paketov opazovati učinke, kot so zakasnitve in hitrost pretoka.

Zanima nas primer, ko imamo dva komunikacijska toka:

- **Od izvora S2 do cilja D2 poteka tok brez omejitev** značilen za prenos podatkov, na primer kopiranje datotek. Pri tej komunikaciji nas zakasnitve paketov ne zanimajo, pomembna je le čim večja prepustnost komunikacijskega kanala.
- Od izvora S1 do cilja D1 smo želeli skozi zelo obremenjeno omrežje, ki ga je zasedal prej opisan tok, speljati časovno kritično storitev, na primer interaktivno računalniško storitev, kot je terminalska seja, ali celo prenos zvoka. Značilnost tega prometa je, da gre za prenos manjšega števila majhnih paketov. Pretok je majhen, zato prepustnost komunikacijskega kanala ni kritična. Pomembne pri tem so zakasnitve posameznih paketov. Za dobro uporabniško izkušnjo, oziroma pri nekaterih storitvah celo pogoj za samo delovanje storitve, morajo biti zakasnitve čim manjše.
  Promet te vrste smo simulirali s TCP tokom, ki ima hitrost 100 kbit/s in velikosti IP paketov 100 oktetov. V paketih je tako približno polovica uporabniške vsebine in polovica protokolnega pribitka.

Opisane parametre prenosa nastavimo na izvoru S1 z uporabo vedra z žetoni in nastavitvijo velikosti paketov (mtu) na 100 oktetov:

```
tc qdisc replace dev eth0 root tbf rate 100kbit burst 116 \
        latency 1s peakrate 100008bit minburst 256
ip link set eth0 mtu 100
```

Parametri vedra z žetoni imajo naslednji pomen:

- rate 100kbit pretok 100 kbit/s.
- burst 116 velikost vedra za žetone 116 oktetov.

Eksperimentalno določimo čim manjšo vrednost, pri kateri tok še teče – nekoliko več od velikosti okvirja v ethernet sloju.

- latency 1s — določa velikost čakalne vrste za podatke pred vstopom v razvrščanje.
  Ker gre za lokalno generirane podatke, določimo »veliko« vrednost in s tem zmanjšamo procesiranje – ponavljanje izgubljenih segmentov po protokolnem skladu od transportnega TCP sloja navzdol. Čakalna vrsta je dovolj velika za 1 sekundno čakanje.
- peakrate 100008bit — vrhnji pretok 100,008 kbit/s, če je v vedru zaloga žetonov.
  Eksperimentalno izberemo čim bližjo vrednost (kolikor sistem dopušča) osnovnemu pretoku, saj želimo vedno imeti 100 kbit/s.
- minburst 256 — velikost vedra za žetone vrhnjega pretoka (v resnici gre pri tej izvedbi za razvrščanje z dvema vedroma).
  Eksperimentalno izberemo čim manjšo vrednost (kolikor sistem dopušča), da omejimo morebitne rafale.

3.1.3 Postopek generiranja prometa

Na obeh ciljih odpremo poljubno izbrana TCP vrata za prihajajoči promet s priročnim programom netcat[4] (nc) in sprejete podatke sproti odmetavamo:

```
nc -l -p 9000 > /dev/null
```

Na izvorni strani pošiljamo podatke v omrežje proti ustreznemu ciljnemu IP naslovu in izbranim TCP vratom s preusmeritvijo podatkovnega toka iz datoteke, ki smo jo pred tem napolnili z naključnimi podatki:

```
cat datoteka2.dat > /dev/tcp/6.6.6.4/9000
```

Z velikostjo vnaprej pripravljene datoteke nastavljamo časovno trajanje komunikacije.

4 *http://nc110.sourceforge.net/*

Pri pretoku 100 kbit/s za 30 sekundni interval TCP komunikacije rabimo 175 kB veliko datoteko, saj uporabniški podatki predstavljajo samo približno pol prenesene vsebine.

Pomembno! Da pri merjenju dobimo pravilne rezultate, mora biti branje datoteke z diska hitrejše od mrežnih povezav.

3.1.4 Posebnosti protokola

V testnem primeru smo uporabili transportni komunikacijski protokol TCP. To vnaša v proces dodatne spremenljivke, saj gre za samoregulativen protokol, ki sam aktivno prilagaja pretok glede na razpoložljive vire. Pri uporabi protokola, ki te značilnosti nima, na primer UDP, bi bil rezultat lažje predvidljiv, saj bi bil ponujan pretok izključno v domeni aplikacij na izvoru. S tem pa bi se oddaljili od želene simulacije realnega stanja. Večina prometa v IP omrežjih namreč še vedno poteka po protokolu TCP.

Z uporabo protokola UDP s konstantnim pretokom brez regulacije je pri polnem pretoku tudi nemogoče doseči stabilno stanje. To je stanje, ko se zakasnitve paketov s časom le malo spreminjajo in ko ni izgub paketov ali so te zelo majhne. Če izvor oddaja z manjšo hitrostjo, kot jo zmore omrežje, se čakalne vrste ne tvorijo nikjer v omrežju. Zakasnitev praktično ni. Takoj ko to hitrost presežemo, se začnejo tvoriti čakalne vrste, v našem primeru v usmerjevalniku R1, in te se ves čas povečujejo, dokler ne zmanjka razpoložljivega pomnilnika. Nakar se začne stalno odmetavanje presežne količine paketov.

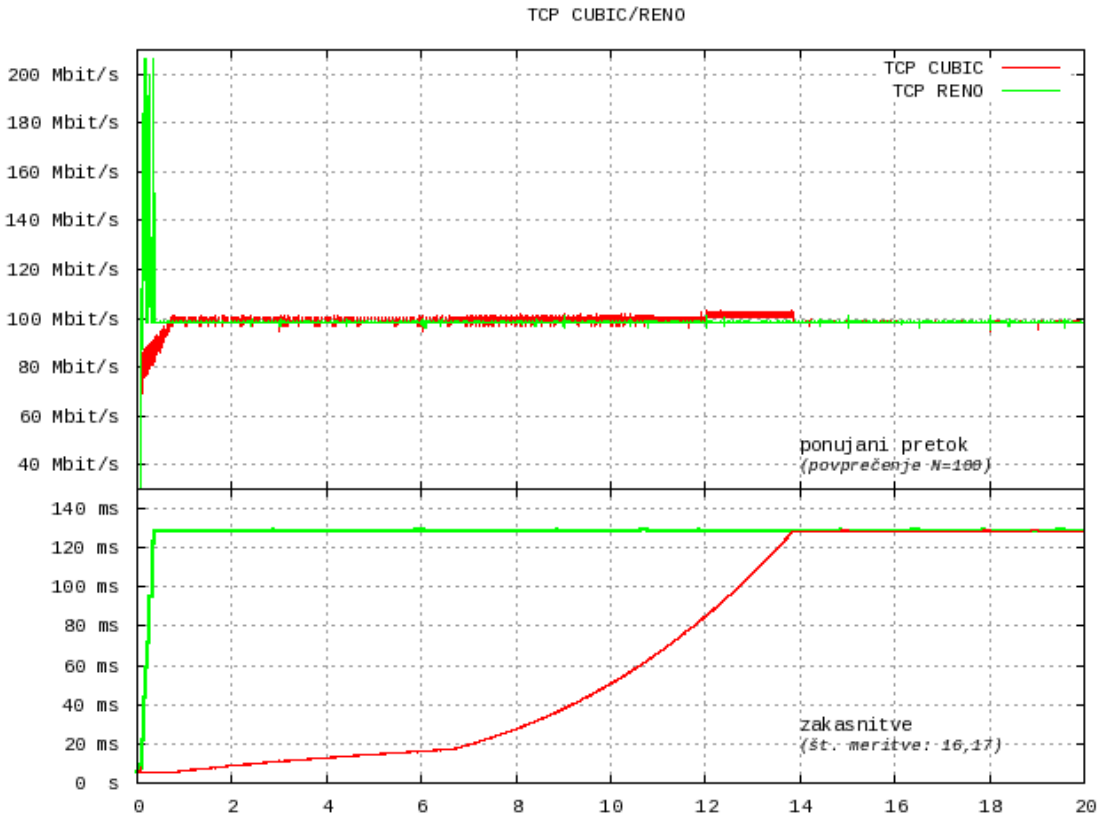
Okolje je še manj predvidljivo zaradi dejstva, da TCP protokol nima enolične strogo predpisane izvedbe. V resnici obstaja več kot deset podtipov TCP protokola. Med sabo se razlikujejo ravno pri regulaciji pretoka ter obravnavi zgostitev in zamašitev prometa. Zakasnitve in izgube paketov so zato odvisne od izvedbe in od uporabljenega tipa TCP protokola.

Slika 20 prikazuje primerjavo komunikacijskih tokov od S2 do D2 v primeru uporabe dveh tipov TCP protokola, ki sta v običajni konfiguraciji linux operacijskega sistema v tem času na izbiro: TCP CUBIC in TCP RENO.

Protokol CUBIC TCP je bil zasnovan za prostrana zmogljiva omrežja[5] z dolgimi obhodnimi časi in velikimi bitnimi pretoki. Regulacija pretoka protokola CUBIC TCP ni odvisna od obhodnega časa, kar je razvidno tudi iz literature [10]. Veliko tipov TCP protokola spreminja oddajno okno in s tem pretok ponujanega prometa v postopkih počasnega zagona

5 *LFN – long fat networks, RFC 1072*

ali izogibanja zamašitvam glede na hitrost sprejemanja potrditvenih paketov. Če je obhodni čas kratek, se potrditveni paketi vračajo z manjšimi zakasnitvami, ponujani pretok se povečuje hitreje. Pri komunikaciji na večje razdalje, kjer je obhodni čas daljši, tak tip TCP protokola pri kratkoživih sejah morda sploh ne bi dosegel zadovoljivega pretoka. Če po isti liniji teče komunikacijski tok na daljši in krajši razdalji lahko pride tudi do nepravičnosti, saj bi slednji zaradi hitrejšega povečevanja pretoka izrival prvega. CUBIC TCP deluje drugače. Povečevanje pretoka sledi kubični funkciji časa, katere oblika je odvisna izključno od pogostnosti zamašitvenih dogodkov. Zamašitveni dogodki so: izgube paketov, zamešanje vrstnega reda paketov in ponavljanje potrditvenih paketov.



Slika 20: primerjava protokolov TCP cubic in TCP reno

Protokol TCP RENO v prikazanem primeru zelo agresivno povečuje pretok, dokler ne doseže ustaljenih vrednostih zakasnitev 128 ms pri največjem možnem opravljenem pretoku 100 Mbit/s. TCP CUBIC pospešuje bolj umirjeno. Ko doseže ustaljene vrednosti zakasnitev pri isti vrednosti kot TCP RENO, vendar šele pri 14. sekundi, nekoliko zmanjša ponujani pretok.

V prikazanem primeru noben od TCP komunikacijskih tokov nima izgub paketov, kar

se na prvi pogled zdi presenetljivo. **Če izgub ni, kaj potem ustali pretok in zakasnitve?** Dokaj neobičajen je tudi izgled diagramov, saj so krivulje kar preveč ravne in gladke. Meritev pretoka je zaradi preglednosti sicer zglajena s tekočim povprečenjem prek 100 paktov (N=100), kar pa ne velja za zakasnitve, katerih meritve so prikazane za vsak paket posebej. Tudi če bi bilo ravnovesje posledica samoregulativnosti TCP protokola, bi pričakovali, da zaradi vgrajenih korekcijskih postopkov trenutne vrednosti ves čas lovijo stabilno stanje, kar bi se odražalo v valovitosti krivulje. Prikazano stanje daje slutiti, da je ustalitev vrednosti zakasnitev posledica lastnosti linux sistemov in izvedbe TCP protokolnega sklada.

Izvedba TCP protokolnega sklada na linux sistemih vsebuje še eno optimizacijsko zanimivost. Nove mrežne povezave med istima dvema sistemoma niso povsem neodvisne. Tudi po končanju ali prekinitvi povezave linux še nekaj časa pomni parametre, ki so bili v uporabi pri tej povezavi. Nove povezave potem ne pričnejo z začetnim počasnim zagonom, ampak se začetno stanje postavi kar na shranjene vrednosti prejšnjih povezav. Diagram na sliki 20 je rezultat povezav te vrste. Če naredimo meritev takoj po zagonu sistema ali če opisano pomnjenje izključimo, je potek nekoliko drugačen. V takem primeru tudi TCP CUBIC začne z zelo agresivnim začetkom in ima podoben potek kot TCP RENO. Potek protokola TCP RENO pa se bistveno ne spremeni. Ob tem se postavi domneva, da bi to lahko bil vzrok, za tako urejen pretok in zakasnitve, saj bi lahko parametri povezave (oddajno okno) ostali optimalno nastavljeni od prejšnjih povezav. Tudi ta domneva se pri naslednjih meritvah izkaže kot napačna in smo jo ovrgli pri referenčni meritvi v neobremenjenem omrežju opisani na strani 60.

Pri meritvah v nadaljevanju je v vseh primerih uporabljen TCP CUBIC tip protokola. Ta je privzet na linux operacijskih sistemih od verzije jedra 2.6.19, to je od konca leta 2006.

3.1.5 Postopek merjenja

Zanima nas hitrost pretoka in zakasnitve paketov komunikacijskih tokov. To izračunamo iz meritev časov in velikosti paketov pri izhodu iz izvorov S1 in S2 ter pri prispetju do ciljev D1 in D2.

Problem sinhronizacije časovnikov

Prva ideja je bila, da čas odpošiljanja paketa odčitamo kar pri izvoru in čas prispetja na cilju. Pri tem se pojavi potreba po čim natančnejši sinhronizaciji sistemskih časovnikov izvornega in ciljnega sistema. To je še nekako izvedljivo z uporabo namenskih

sinhronizacijskih paketov pred začetkom vsake komunikacije, ko je komunikacijski kanal še prost in so razmere dovolj deterministične. Potem zelo hitro naletimo na novo težavo. Izkaže se, da kljub začetni sinhronizaciji sistemskih časovnikov za dovolj natančno meritev, ti ne tečejo dovolj sinhrono. V praksi se, ob začetni sinhronizaciji in po 30 sekundni meritvi, časovnika izvora in cilja pogosto razlikujeta že več kot 1 ms. Čedalje večje odstopanje se direktno prišteva ali odšteva od izmerjene vrednosti in pomeni pri merjenju zakasnitev v velikosti okoli 5 ms nesprejemljivo veliko napako. V izogib tolikšni napaki moramo oba časa, tako odpošiljanja kot dospetja paketa, izmeriti z istim časovnikom. V ta namen od vsakega cilja D1 in D2 do pripadajočega izvora potegnemo dodatni merilni vod, po katerem odčitavamo čase odpošiljanja paketov (črtkani vodi na sliki 19). Priključitev izvedemo s pomočjo mrežnih stikal. V konfiguraciji stikal nastavimo zrcaljenje vhodnega prometa od izvorov proti izhodoma, kamor priključimo merilna voda. Vse čase potem merimo s ciljnima sistemoma.

Zajemanje in izračun vrednosti

Za zajemanje časov in velikosti paketov uporabimo sistemsko orodje tcpdump[6]:

```
tcpdump -n -e -S -v -s 100 -i eth0
17:56:52.826341 00:24:8c:55:bb:31 > 00:24:8c:59:38:b4,
 ethertype IPv4 (0x0800), length 1514: (tos 0x0, ttl 61, id 40720,
 offset 0, flags [DF], proto TCP (6), length 1500)
 5.5.5.3.40389 > 6.6.6.4.9000: . 905300923:905302371(1448)
 ack 760425548 win 46 <nop,nop,timestamp 717311991 717292894>
```

Sistemsko orodje tcpdump nam vrne čas paketa v mikrosekundni ločljivosti in vse pomembne podatke iz glav drugega (ethernet), tretjega (IP) in četrtega (TCP) sloja protokolnega sklada. Za našo meritev so pomembni podatki:

- čas paketa …….. `17:56:52.826341,`
- velikost paketa v drugem sloju …… `1514,`
- sekvenca paketa …… `905300923:905302371.`

  S primerjavo sekvence identificiramo isti paket v različnih delih omrežja.

6 *http://www.tcpdump.org/*

Zakasnitev vsakega od paketov dobimo kot razliko časov med ciljem in izvorom pri istem paketu.

$$d_i = t_i^{Dx} - t_i^{Sx}$$

Trenutni opravljeni pretok merimo samo pri vsakem od ciljev in je enak količniku velikosti paketa in časovnega intervala – razlike časov trenutnega in predhodnega paketa, ki pripada istemu komunikacijskemu toku.

$$r_i^{Dx} = \frac{L_i^{Dx}}{t_i^{Dx} - t_{i-1}^{Dx}}$$

Trenutni pretok je zelo spremenljiva količina. Pri pretoku 100 Mbit/s in 1500 oktetov velikih paketih ti prihajajo vsakih 120 µs. V praksi časovni intervali niso zelo stabilni in tudi meritve z našo metodo v takem velikostnem razredu niso prav točne. Zato pretok povprečimo prek več paketov:

$$\bar{r}_i^{Dx} = \frac{\sum_{i-(N-1)}^{i} L_i^{Dx}}{t_i^{Dx} - t_{i-N}^{Dx}}$$

Primer: pri tekočem povprečenju pretoka $\bar{r}_i$ pri cilju D_2 prek 2 paketov (*N=2*) upoštevamo velikosti obeh paketov $L_{i-1} + L_i$, ki prispeta vsak v svojem časovnem intervalu. Paket *i–1* v intervalu od t_{i-2} do t_{i-1} in paket *i* od t_{i-1} do t_i. Skupni interval je torej $t_i - t_{i-2}$. *N* je v našem primeru število paketov, prek katerih povprečimo trenutni pretok. Vrednost *N* določimo s poizkusom, da postane diagram dovolj čitljiv in nazoren, kar pomeni z gladko krivuljo pretoka, in je na vsakem diagramu tudi naveden.

Pri velikosti paketa upoštevamo velikost v drugem (ethernet) sloju protokolnega sklada. Celoten ethernet okvir na liniji je dejansko še za nekaj oktetov večji. Ta dodatek upravlja direktno ethernet krmilnik in spada v prvi (fizični) sloj. Ta pribitek programsko sploh ni viden.

Postopki razvrščanja paketov so programsko pripeti direktno na omrežni krmilnik in upoštevajo celotno dolžino, ki je programsko vidna. Podane hitrosti in velikosti vsebujejo tudi protokolni pribitek drugega sloja.

Posamezna meritev traja približno 30 sekund in v tem času je zajetih nekaj več kot pol milijona paketov. Za shranjevanje in analizo te količine podatkov se zato kot najprimernejša

izkaže uporaba podatkovne zbirke in SQL poizvedb. Za grafično predstavitev uporabimo program gnuplot[7].

Pomembno! Za pravilno meritev rabimo dovolj hiter sistem, da orodje tcpdump pri zajemu ne izgublja paketov. Po koncu meritve (pritisnemo ctrl-c) tcpdump izpiše nekaj kratkih statističnih podatkov, med katerimi je tudi število izgubljenih paketov. Da lažje dosežemo ta cilj, pri klicu uporabimo določilo -s 100. To določa, da namesto celih paketov zajemamo samo prvih 100 oktetov, kar je več kot dovolj za zajem glav vseh protokolnih slojev.

3.2 Referenčna meritev v neobremenjenem omrežju

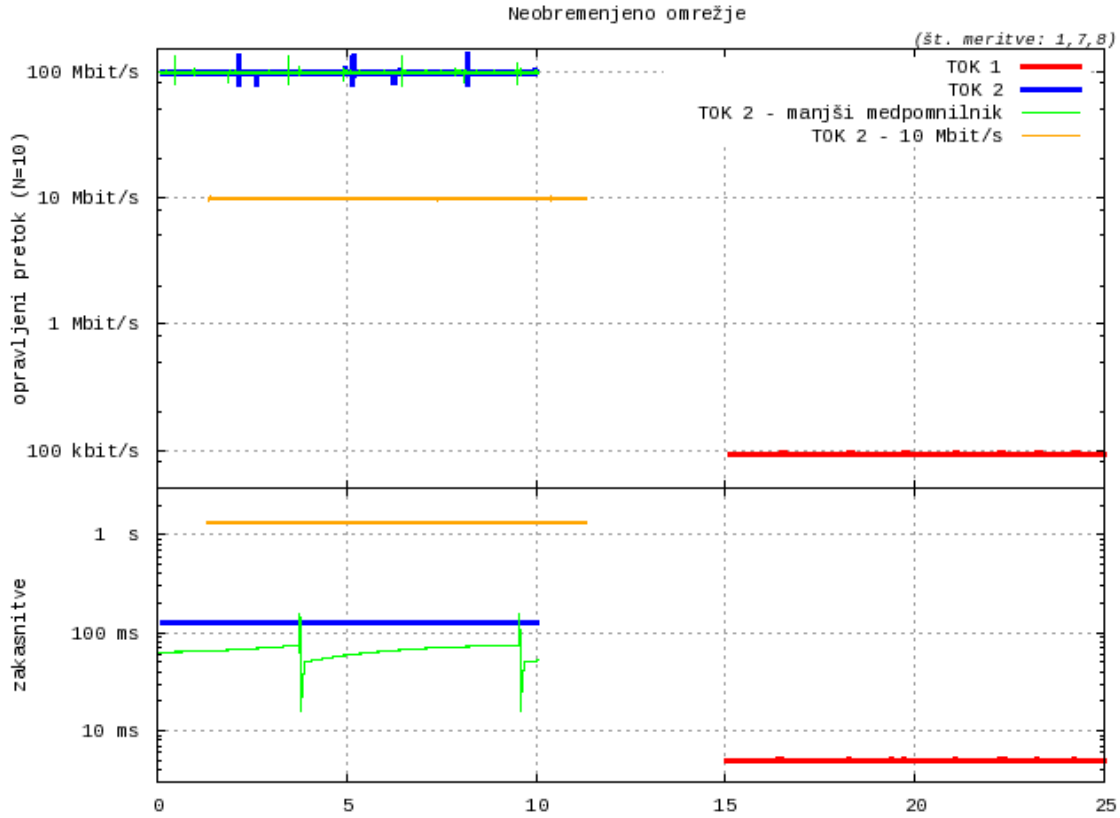
Meritev v neobremenjenem omrežju je prva, ki jo izvedemo. Tako pridemo do referenčnih rezultatov, s katerimi lahko primerjamo vse ostale meritve. Skozi omrežje spustimo posebej samo po en komunikacijski tok naenkrat ter po nekem času, ko se tok ustali, izmerimo pretok in zakasnitve.

Rezultati meritev v neobremenjenem omrežju namreč predstavljajo mejo, ki je zaradi fizikalnih in tehnoloških omejitev ne moremo preseči z nobenimi metodami razvrščanja, ko omrežje hkrati uporablja več subjektov, ki si morajo vire med sabo deliti. Pri neobremenjenem omrežju dosežemo največji pretok in najmanjše možne zakasnitve.

Slika 21 na levi strani prikazuje meritve komunikacijskega toka 2. Kljub temu, da so rezultati predstavljeni skupaj v istem diagramu in v istem časovnem intervalu, gre za ločene meritve. Meritve so ponazorjene z različnimi barvami v skupnem diagramu z namenom medsebojne primerjave.

Komunikacijski tok 1, ki ga prikazuje slika 21 z rdečo črto na desni strani, ima omejen pretok približno 100 kbit/s in nima zgostitev nikjer v omrežju. Ker se čakalne vrste ne tvorijo, zakasnitve povzroča samo čas procesiranja v elementih omrežja in razsežnost omrežja. V našem omrežju imamo na sredini vozlišče X, ki emulira razsežno omrežje. Od tod tudi pričakovana zakasnitev približno 5 ms, ki jo izmerimo pri komunikacijskem toku 1.

7 *http://www.gnuplot.info/*



Slika 21: referenčna meritev v neobremenjenem omrežju

Komunikacijski tok 2 nima omejenega pretoka. Zgostitev toka se pojavi v obliki čakalne vrste v usmerjevalniku R1, kjer pride do največje zakasnitve. Ob privzetih nastavitvah se zakasnitve paketov stabilizirajo pri srednji vrednosti 128 ms in pretok pri 100 Mbit/s, kot prikazuje modra črta na sliki 21. O tem, zakaj ravno pri teh vrednostih, smo nekoliko ugibali že v poglavju o posebnostih protokola na strani 55. Pretok omejuje prepustnost izhodnega kanala, ki je 100 Mbit/s. Vrednost zakasnitev je odvisna predvsem od dolžine čakalne vrste. Ta pri izmerjenih vrednostih zakasnitev in izražena v številu paketov N velikosti $L=1514$ oktetov znaša:

$$N = \frac{d\,R}{L} \times \frac{1}{8(bit/oktet)} = 1015\ paketov$$

d je zakasnitev paketov samo v usmerjevalniku R1 (123 ms, ko od celotne odštejemo 5 ms zakasnitev v vozlišču X), R je hitrost praznjenja čakalne vrste – izhodni pretok (100 Mbit/s).

Največja možna dolžina čakalne vrste je fizično omejena z razpoložljivim medpomnilnikom, ki je zanjo predviden. Dejanska dolžina čakalne vrste je odvisna od

algoritma samoregulacije TCP komunikacijskega protokola in od omejitev/izvedbe TCP protokolnega sklada na našem sistemu, ki z regulacijo pretoka določata zasedenost tega medpomnilnika. Linux ima privzeto velikost medpomnilnika pri ethernet krmilnikih nastavljeno na 1000 paketov največje dolžine (običajno 1514 oktetov).

Velikost medpomnilnika in izmerjena dolžina čakalne vrste se neverjetno dobro ujemata. Še posebej, če pri izmerjeni vrednosti upoštevamo, da sta dejanska zakasnitev v čakalni vrsti in s tem tudi čakalna vrsta v resnici malenkostno krajši. Pri določitvi zakasnitve v čakalni vrsti namreč nismo upoštevali, da je vzrok majhnega dela zakasnitve tudi v času procesiranja v mrežnih elementih in smo to kar pripisali čakanju v čakalni vrsti. Na hitro bi postavili hipotezo, da samoregulacijski algoritem TCP protokola prilagodi dolžino čakalne vrste na razpoložljivo velikost medpomnilnika. Tako bi lahko nastavljali zakasnitve s spreminjanjem velikosti medpomnilnika, kar je parameter pri omrežnih nastavitvah na operacijskem sistemu.

Hipoteza se hitro izkaže kot napačna. Če velikost medpomnilnika poljubno povečamo, se zakasnitve kljub temu ne spremenijo, kar pomeni da dolžina čakalne vrste ostane pri približno 1000 paketih. Ugotovljeno ujemanje je torej zgolj slučaj oziroma vgrajena značilnost izvedbe TCP protokolnega sklada. Zanimivo bi bilo izvedeti, od katerih zunanjih pogojev so odvisne zakasnitve, oziroma ali dano omrežje nanje lahko kakorkoli vpliva. Če povečanje medpomnilnika ne vpliva na zakasnitve, kaj pa zmanjšanje? Medpomnilnik zmanjšamo na velikost 300 paketov, kar je približno tretjina običajne vrednosti. Če zgornjo formulo obrnemo, ta vrednost zadošča za največjo možno stabilno zakasnitev v usmerjevalniku R1:

$$d_{max} = \frac{N\,L}{R} \times 8\,bit/oktet = 36\,ms$$

Meritev prikazuje na sliki 21 zelena krivulja. Pri pretoku ni vidnih sprememb, medtem ko zakasnitve se zmanjšajo. Zmanjšanje ni v pričakovanem obsegu in tudi vrednosti zakasnitev postanejo nestabilne. Medpomnilnik očitno ni dovolj velik za stabilno delovanje, kar vodi k večjim in pogostejšim neželenim izgubam paketov. Pri 3,7 sekunde se izgubi 5 paketov in pri 9,5 sekunde še eden. Izkoriščenost omrežnih virov je zato manj učinkovita.

Poizkusi pokažejo, da je uporabljen CUBIC tip TCP protokola zelo neodvisen od zunanjih dejavnikov. Zakasnitve se ne spremenijo niti, če povečujemo ali zmanjšujemo vgrajeno zakasnitev v samem omrežju – na vozlišču X, saj kot smo že omenili, krmiljenje

pretoka ni odvisno od obhodnega časa. Šele ko pretiravamo in v vozlišču X zelo povečamo zakasnitev, prek 100 ms, se povečajo tudi zakasnitve paketov komunikacijskega toka. Hkrati vrednosti zakasnitev postanejo nestabilne, upade tudi pretok. Meja 100 Mbit/s ni več dosegljiva.

Zadnje daje slutiti, da so vrednosti zakasnitev morda odvisne od pretoka oziroma od razmerja upočasnitve med vhodnim in izhodnim pretokom v usmerjevalniku R1. Dejansko se izkaže, če prepustnost omrežja zmanjšamo na desetino – 10 Mbit/s, se zakasnitve paketov malo več kot podeseterijo in dosežejo 1,33 s – oranžna krivulja na sliki 21. Pri tem je presenetljivo, da komunikacijski tok spet ujame stabilno delovanje pri približno 1000 paketov dolgi čakalni vrsti.

Poizkusimo ugotoviti, od kod pride ta omejitev na približno 1000 odposlanih nepotrjenih paketov. Podrobnejša analiza prometa razkrije, da cilj D2 v potrditvenih paketih oglašuje izvoru S2 velikost sprejemnega okna, ki v našem primeru po nekem času doseže in se ustali pri vrednosti 12.308. Pri tem ne smemo pozabiti, da imamo opravka s CUBIC TCP protokolom, ki je namenjen za prostrana zmogljiva omrežja. Velikost sprejemnega okna zato ni oglaševana v oktetih, saj protokol privzeto vsebuje možnost skaliranja okna[8]. Pri tej opciji se izvor in cilj pri vzpostavitvi TCP povezave v SYN paketih dogovorita za faktor, s katerim se množi velikost oglaševanega sprejemnega okna. V našem primeru dobimo: wscale 7, kar pomeni, da moramo oglaševano velikost pomakniti za 7 bitov v levo, oziroma jo množiti s faktorjem 2^7. Oglaševano okno je torej:

$$W = 12.308 \cdot 2^7 = 1.575.424\, oktetov \quad ,$$

kar ustreza 1040 paketom po 1514 oktetov. Velikost oddajnega okna je manjša vrednost med velikostjo zgostitvenega okna[9] in oglaševanega sprejemnega okna. Velikost zgostitvenega okna se z različnimi postopki dinamično prilagaja obremenjenosti omrežja, medtem ko velikost sprejemnega okna določa sprejemnik glede na svoje razpoložljive pomnilniške vire. Oglaševano sprejemno okno predstavlja torej zgornjo mejo velikosti oddajnega okna. Kot smo lahko ugotovili, je sprejemno okno neodvisno od stanja in obremenjenosti omrežja. Ugotovimo pa, da je velikost oglaševanega sprejemnega okna odvisna od več sistemskih parametrov, s katerimi lahko upravljamo pomnilniške vire TCP povezav. Podmnožica važnejših sistemskih parametrov z navedenimi privzetimi vrednostmi, ki dokazano vplivajo na TCP pretok, je naslednja:

8 *TCP window scale option, RFC 1323*
9 *cwnd – congestion window, RFC 2001*

```
# Linux autotuning TCP buffer limits min, default and max
net.ipv4.tcp_rmem = 4096 87380 4194304
net.ipv4.tcp_wmem = 4096 16384 4194304
```

Iz samih vrednosti sistemskih parametrov ni mogoče na prvi pogled ugotoviti velikosti oddajnih in sprejemnih oken. Toda če s spremembo sistemskih parametrov povečamo pomnilniške kvote, se lastnosti komunikacijskega toka 2 spremenijo. Komunikacijski tok potem lahko ustvari tudi daljše čakalne vrste od 1000 paketov. Če nekje na prenosni poti zanje ni dovolj razpoložljivega pomnilnika, bi pogosteje prihajalo do izgub paketov, kar bi sprožilo ustrezne protokolne regulacijske postopke. V tem primeru bi se pretok in zakasnitve ustalile zaradi samoregulacijskih lastnosti TCP protokola in ne več zaradi velikosti sprejemnega okna.

Običajno s povečanjem parametrov ne pridobimo nič, saj prekomerno izgubljanje paketov ali povečevanje zakasnitev prav gotovo ni cilj, ki bi ga želeli doseči. Povečanje navedenih sistemskih parametrov bi imelo smisel edino v primeru izredno prepustnih prenosnih poti, kjer bi hkrati imeli tudi velike zakasnitve. V takem primeru privzete kvote ne bi bile zadostne za zagotavljanje velike množice paketov na prenosni poti, zaradi česar ne bi dobili optimalnega pretoka. Pri manj prepustnih omrežjih z majhnimi zakasnitvami bi prej lahko razmišljali o zmanjšanju parametrov. Privzete vrednosti pokrivajo širok razpon današnjih omrežij. Privzete vrednosti se pri linux sistemih hkrati tudi ujemajo: na eni strani z največ 1000 oddanimi paketi in na drugi strani s privzeto velikostjo medpomnilnikov za 1000 paketov pri ethernet krmilnikih.

Meritev v neobremenjenem omrežju lahko zaključimo z ugotovitvijo, da je zakasnitev paketov šibkega komunikacijskega toka 1 odvisna samo od omrežja, ki povzroča 5 ms zakasnitve, saj tok v omrežju nima zgostitev. Zakasnitve paketov komunikacijskega toka 2 pa so zaradi neomejene narave in posledične zgostitve v omrežju ter ob zadostnem medpomnilniku, da se pretok stabilizira, lahko odvisne od samoregulacijskih značilnosti uporabljenega TCP protokola ali sistemskih omejitev izvedbe oziroma nastavitev TCP protokolnega sklada in se v našem primeru pri privzetih sistemskih nastavitvah ustalijo pri 128 ms, ko čakalna vrsta doseže dolžino približno 1000 paketov.

3.3 Razvrščanje v prihajajočem vrstnem redu (FIFO)

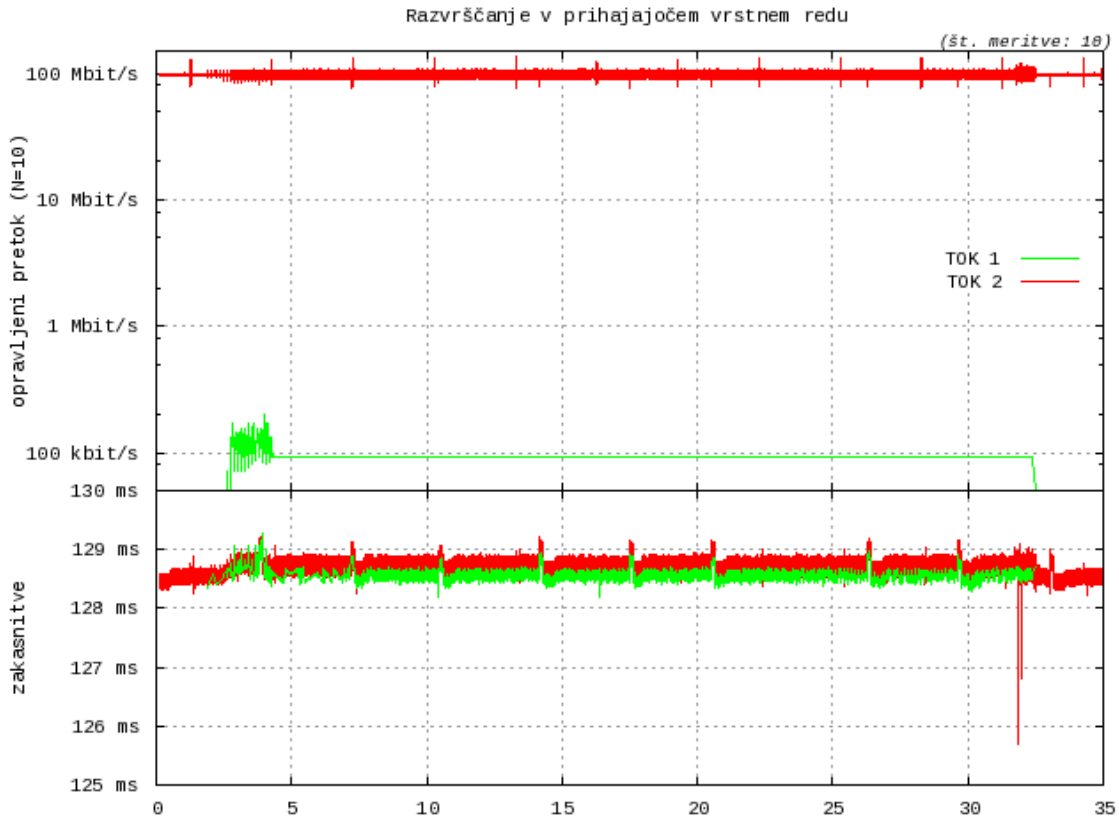
Razvrščanje v prihajajočem vrstnem redu na izhodnem, v našem primeru *eth0*, ethernet krmilniku nastavimo z ukazom:

```
tc qdisc add dev eth0 root bfifo
```

Pri razvrščanju v prihajajočem vrstnem redu usmerjevalnik z obeh vhodnih linij sprejema komunikacijska toka in ju združuje v skupno izhodno čakalno vrsto v vrstnem redu, kot sta prispela. Pomeni, da vsi paketi enako čakajo na odpošiljanje, kar lepo pokaže tudi meritev prikazana na sliki 22. Vsi paketi, ne glede kateremu toku pripadajo, imajo praktično enake zakasnitve. Da ima komunikacijski tok 1 le za malenkost manjše zakasnitve, je posledica znatno manjših paketov, ki nekoliko hitreje pridejo skozi omrežje. Zakasnitev komunikacijskega toka 1 se v primerjavi z neobremenjenim omrežjem v prejšnjem primeru poveča kar za 25-krat in ne izpolnjuje več pogojev za kakovostno delovanje nekaterih storitev, ki delujejo v realnem času.

Pri hitrosti pretokov posameznih tokov večjega medsebojnega vpliva ne zaznamo. Šibek tok 1 ima le neznaten vpliv na močan tok 2, kar se na krivulji pretoka na sliki 22 zgoraj pokaže v obliki povečanega šuma. Zanimivo, da velja tudi obratno, močan komunikacijski tok 2 ne vpliva bistveno na tok 1, saj ta nekako že »najde luknjo« v prepustnosti omrežja, ki je tok 2 ne zapolni popolnoma.

Če bi imeli dva enakovredna komunikacijska toka, bi bila slika pri razvrščanju v prihajajočem vrstnem redu drugačna. Pri pretoku bi se med sabo izrivala in na koncu našla ravnotežje približno na sredini, vsak pri 50 Mbit/s. Kot smo videli v prejšnjem poglavju pri analizi zakasnitev uporabljenega TCP protokola v neobremenjenem omrežju, bi zmanjšanje opravljenega pretoka vplivalo tudi na zakasnitve. Pol manjši pretok kot pri neobremenjenem omrežju bi povzročil dvakrat večje zakasnitve paketov, okoli 250 ms. To bi veljalo edino v primeru, če bi zagotovili dovolj velik medpomnilnik, da bi pretoka dosegla stabilno stanje. Privzeta vrednost za 1000 paketov ne zadostuje. V takem primeru velja za vsak komunikacijski tok posebej, da se stabilizira, ko doseže približno 1000 paketov v čakalni vrsti.



Slika 22: razvrščanje v prihajajočem vrstnem redu

3.4 Razvrščanje po prioritetah

Pri razvrščanju v prihajajočem vrstnem redu se pokaže težava, ker vsi paketi čakajo enako – v skupni čakalni vrsti, ne glede na njihovo pomembnost – prioriteto. Za rešitev tega problema je najbolj logično uvesti prioritete za različne vrste paketov, ki pripadajo različnim komunikacijskim tokom oziroma storitvam. V ta namen naredimo več ločenih čakalnih vrst z različnimi prioritetami odpošiljanja in filtre, ki vanje ustrezno razvrščajo promet.

```
tc qdisc add dev eth0 root handle 1: prio bands 3
   # dobimo tri razrede/vrste 1:1, 1:2, 1:3

# filtri prometa
tc filter add dev eth0 protocol ip parent 1: prio 1 u32 \
  match ip dst 4.4.4.4/32 flowid 1:1          # komunikacijski tok 1

tc filter add dev eth0 protocol ip parent 1: prio 1 u32 \
  match ip dst 6.6.6.4/32 flowid 1:2          # komunikacijski tok 2

tc filter add dev eth0 protocol ip parent 1: prio 2 u32 \
  match ip dst 0.0.0.0/0  flowid 1:3          # ostalo
```

Prvi ukaz ustvari na izhodnem ethernet krmilniku eth0 prio disciplino s tremi čakalnimi vrstami. Čakalne vrste imajo različne prioritete odpošiljanja čakajočih paketov. Vrste z manjšo identifikacijsko številko imajo višjo prioriteto nad vrstami z večjimi identifikacijskimi številkami. Vrste se praznijo izključno po prioritetah. Vedno se odpošiljajo samo paketi iz vrste z najvišjo prioriteto. Šele ko je ta prazna, pridejo na vrsto tiste z nižjo prioriteto.

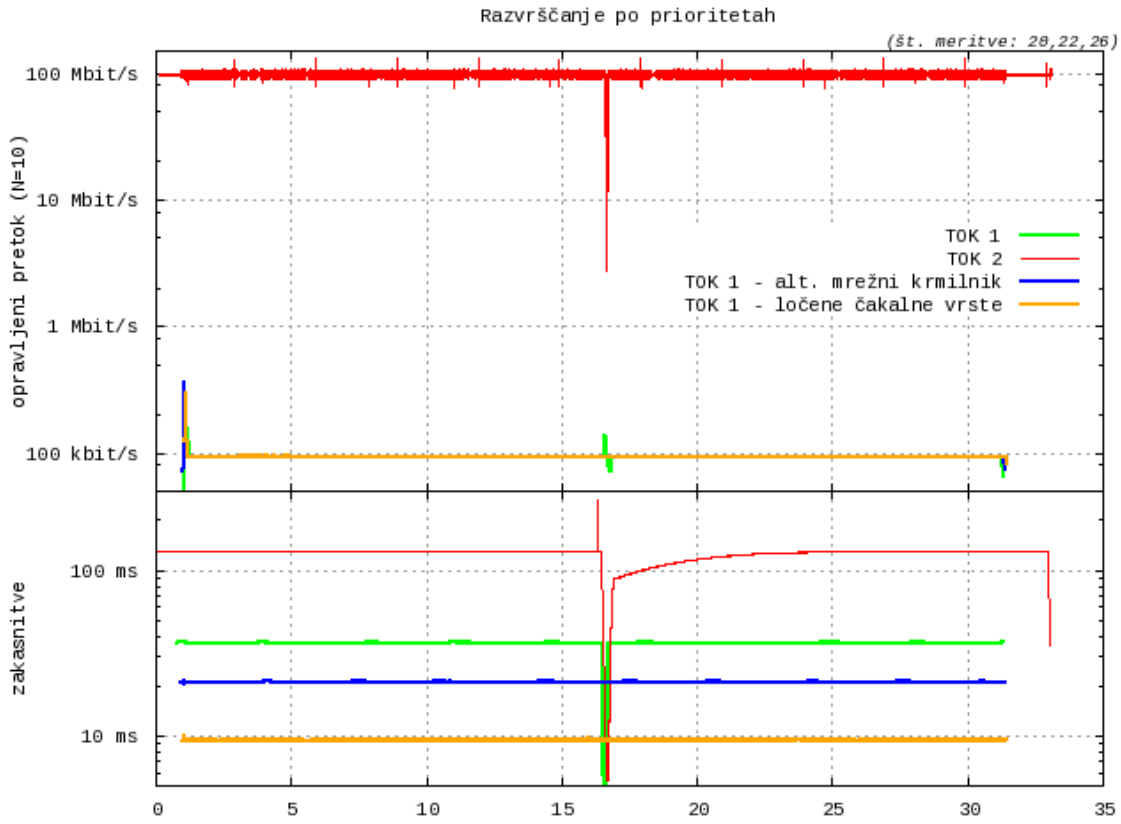
S filtri razvrstimo komunikacijski tok 1 (IP naslov cilja 4.4.4.4) v vrsto 1:1, ki ima najvišjo prioriteto in je tako odposlan brez čakanja – prednostno mimo čakalne vrste komunikacijskega toka 2. Komunikacijski tok 2 (IP naslov cilja 6.6.6.4) razvrstimo v vrsto z nižjo prioriteto 1:2, kjer pride do zgostitve prometa in posledično enakih zakasnitev kot v prejšnjih primerih. Tretja vrsta z najnižjo prioriteto obstaja samo zato, ker je tri najmanjše možno število vrst pri prioritetnem razvrščanju. To je izvedbena omejitev na linux sistemih. Vanjo preusmerimo ves ostali promet, ki ga v našem primeru sploh ni.

Po pričakovanjih bi morali dobiti podobne rezultate, kot smo jih izmerili pri neobremenjenem omrežju. Pri pretoku res ni razlik. Komunikacijska toka dosegata pričakovani vrednosti malo pod 100 Mbit/s in 100 kbit/s. Zanimivost pri tem testu je izguba enega paketa v toku 2 ob času 16,5 sekunde pri prvi meritvi, kot kaže rdeča krivulja na sliki 23. Ta ni posledica razvrščanja, ampak normalno delovanje TCP protokola, ki ves čas tipa, ali je mogoče pretok še povečati. Zato TCP protokol redno izgublja posamezne pakete, v danem omrežju približno en do dva paketa na minuto. Ob tem dogodku izvor bliskovito zmanjša pretok in ga potem postopoma znova povečuje. To za trenutek razbremeni omrežje, kar se vidi tudi na kratkem upadu zakasnitev vseh tokov, ki takrat tečejo skozi. Na podobne izgube paketov bomo naleteli tudi še pri naslednjih primerih in jih pri podobni pogostnosti in vzrokih ni potrebno podrobneje obravnavati.

Pri zakasnitvah meritev ne izpolni pričakovanj. Paketi komunikacijskega toka 1 imajo kljub najvišji prioriteti še vedno zakasnitve večje od 35 ms namesto pričakovanih 5 ms. To pomeni, da poleg 5 ms čakanja v vozlišču X, paketi še vedno čakajo nekaj več kot 30 ms v čakalni vrsti usmerjevalnika R1. Rezultat daje slutiti, da se med protokolnim skladom in samo linijo nekje skriva še dodatni medpomnilnik – v programskem gonilniku ali sami strojni opremi mrežnega krmilnika – in ta odpošilja pakete v prihajajočem vrstnem redu skupaj s komunikacijskim tokom 2. Pričakovana velikost tega medpomnilnika mora biti:

$$N = \frac{d\,R}{L} \times \frac{1}{8(bit/oktet)} = 256\,paketov$$

Zakasnitev *d* je 31 ms, hitrost praznjenja *R* je 100 Mbit/s, velikost paketov *L* je 1514 oktetov.



Slika 23: razvrščanje po prioritetah

Domnevo potrjuje dejstvo, če originalni nVidia MCP61 izhodni mrežni krmilnik usmerjevalnika R1 zamenjamo z drugim modelom – Intel PRO/100+, se zakasnitve paketov komunikacijskega toka 1 prepolovijo na 15 ms v usmerjevalniku R1, oziroma skupaj z zakasnitvami omrežja na 20 ms kot prikazuje modra krivulja na sliki 23. Na komunikacijski tok 2 to ne vpliva, zato ga nismo znova prikazovali.

Za upravljanje in pregled fizičnih parametrov mrežnih krmilnikov linux vsebuje orodje ethtool. Pri večini mrežnih krmilnikov lahko upravljamo tudi interni medpomnilnik z ukazom:

```
ethtool -g eth0

Ring parameters for eth0:
Pre-set maximums:
RX:             4096
RX Mini:        0
RX Jumbo:       0
TX:             4096
Current hardware settings:
RX:             256
RX Mini:        0
RX Jumbo:       0
TX:             256
```

Pri krmilniku za prvo meritev je privzeta vrednost res 256 in pri drugem 128 paketov, kar se popolnoma ujema z izračuni. Z istim orodjem je mogoče vrednosti internega medpomnilnika v omejenih mejah tudi nastavljati. Najmanjša velikost je pogojena z uporabljeno strojno opremo – tipom uporabljenega mrežnega krmilnika – in je običajno med 64 in 128 paketov.

Mrežni krmilniki z ločenimi čakalnimi vrstami

Zdi se, da krajše zakasnitve pri razvrščanju s prioritetami lahko dosežemo le do neke meje z zmanjševanjem internega medpomnilnika pri mrežnih krmilnikih. Vendar ni tako. Obstajajo posebni strežniški mrežni krmilniki, ki vsebujejo več, običajno od 4 do 8, strojno izvedenih medpomnilnikov za ločevanje čakalnih vrst – na primer Intel Gigabit ET. Taki krmilniki prvenstveno niso namenjeni za posebna razvrščanja komunikacijskih tokov temveč za paralelno procesiranje in virtualizacijo. Vsakemu virtualnemu sistemu se namreč priredi ločen medpomnilnik, da so mrežne zakasnitve med njimi neodvisne. Slabost te izvedbe je, da vsaka čakalna vrsta proži ločene prekinitve, vsaka svojemu procesorju. Krmilnik za opisano delovanje torej nujno rabi večprocesorski sistem. To pri današnjih več jedrnih procesorjih ne predstavlja večjega problema.

Linuxov ukaz tc vsebuje manj znano razširitev pri razvrščanju mrežnih paketov. Komunikacijske toke poleg tega, da jih razvrstimo v različne razrede, vsakega usmerimo tudi v ločen medpomnilnik mrežnega krmilnika. Nastavitve so torej identične kot v prejšnjem primeru z manjšim dodatkom:

```
tc qdisc add dev eth0 root handle 1: prio bands 3
   # dobimo tri razrede/vrste 1:1, 1:2, 1:3

# filtri prometa
# komunikacijski tok 1
tc filter add dev eth0 protocol ip parent 1: prio 1 u32 \
  match ip dst 4.4.4.4/32 flowid 1:1 action skbedit queue_mapping 0

# komunikacijski tok 2
tc filter add dev eth0 protocol ip parent 1: prio 1 u32 \
  match ip dst 6.6.6.4/32 flowid 1:2 action skbedit queue_mapping 1

# ostalo
tc filter add dev eth0 protocol ip parent 1: prio 2 u32 \
  match ip dst 0.0.0.0/0  flowid 1:3 action skbedit queue_mapping 2
```

Tak dodatek k nastavitvam lahko s pridom uporabimo tudi v vseh drugih primerih razvrščanja v nadaljevanju, če uporabljajo več čakalnih vrst in imamo ustrezno strojno podporo v mrežnem krmilniku.

V osnovi še vedno ostane enak prioritetni algoritem med razredi kot prej. Ko pa paketi komunikacijskega toka 1 zapustijo sistem razvrščanja, se umestijo v svoj interni medpomnilnik 0 v mrežnem krmilniku, ki je namenjen samo temu toku in je zaradi majhnega pretoka praktično vedno prazen. Velika množica paketov komunikacijskega toka 2 se namreč nabira v sosednjem internem medpomnilniku 1. Paketi iz prvega medpomnilnika so zato odposlani skoraj takoj z zelo majhnimi zakasnitvami, kot prikazuje oranžna črta na sliki 23. Rezultat smo izboljšali še za nekaj ms in zdaj znaša nekaj manj kot 10 ms. Od tega 5 ms odpade na zakasnitev v samem omrežju in le preostalih 5 ms na čakanje v internem medpomnilniku mrežnega krmilnika.

Ko paketi enkrat prispejo v interne medpomnilnike, na čakalne vrste nimamo več nobenega vpliva. Ne moremo vplivati niti na algoritem praznjenja internih medpomnilnikov, ki se izvaja s krožnim procesom, niti na število odposlanih paketov v enem ciklu. Zato se z zakasnitvami ne moremo čisto izenačiti z razmerami pri neobremenjenem omrežju. **Z dodatnimi 5 ms zakasnitvami pri komunikacijskem toku 1 moramo tako pri uporabi tega mrežnega krmilnika računati tudi pri drugih razvrščanjih v nadaljevanju.**

Slabosti razvrščanja po prioritetah

Slabost razvrščanja po prioritetah, ki precej zmanjša njegovo praktično uporabnost, je njegova brezpogojnost. Vedno odpošilja samo pakete iz čakalne vrste z najvišjo prioriteto.

Lahko se zgodi, da je teh preveč in čakalne vrste z nižjimi prioritetami nikoli ne pridejo na vrsto. To lastnost je zelo enostavno tudi namerno zlorabiti.

Primeren kandidat za višjo prioriteto se zdi interaktivno terminalsko delo. To sta storitvi telnet oziroma v novejšem času ssh. Gre za prenos majhne količine podatkov, ki zahtevajo nizke zakasnitve. Zelo enostavno je skozi vrata, ki običajno pripadajo telnet storitvi preusmeriti druge storitve, na primer izmenjavo datotek. Pri ssh storitvi je to še lažje, saj celo storitev sama omogoča tudi druge, bolj obremenjujoče izmenjave podatkov, kot je prenos datotek s storitvijo scp in sftp po istem protokolu in skozi ista vrata. Filtriranje prometa in razvrščanje v čakalne vrste z različnimi prioritetami lahko naredimo zelo kompleksno z namenom, da bolj točno preverimo, če komunikacijski tok res ustreza pogojem za višje prioritete. Kljub temu morebitnih zlorab ni mogoče povsem preprečiti.

Razvrščanje po prioritetah je tako uporabno samo v primerih, ko je količina visoko prioritetnega prometa zagotovljeno zelo omejena. V praksi ga uporabljamo predvsem za krmilne in nadzorne protokole samega omrežja.

Če prioritetno razvrščanje paketov po prehodu skozi omrežje ne da pričakovano nizkih zakasnitev, se najverjetneje nekje v omrežju skriva še kakšna čakalna vrsta z drugačno disciplino, ki kvari pričakovane rezultate.

Zamenjana razvrstitev komunikacijskih tokov

Zelo enostaven poizkus pokaže, kako resna je omenjena pomanjkljivost razvrščanja po prioritetah. Samo obrnemo pogoja, da gre šibek komunikacijski tok 1 v čakalno vrsto z nižjo prioriteto in močan komunikacijski tok 2 v čakalno vrsto z višjo prioriteto.

Pokaže se, da izvor S1 večinoma ne more niti vzpostaviti povezave do cilja D1. Skozi z visoko prioritetnim prometom zasedeno omrežje ne pride niti en sam paket nižje prioritete. Po nekaj deset sekundah poteče čas za vzpostavitev povezave in sistem javi napako, da komunikacija ni mogoča.

3.5 Pravična razvrščanja

Pri pravičnih razvrščanjih se ne more zgoditi, da en od komunikacijskih tokov zaseže vse razpoložljive vire. Značilnost pravičnih razvrščanj je ravno ta, da vsakdo dobi delež, ki pa je lahko poljubno administrativno odmerjen.

Uteženo pravično razvrščanje – WFQ obravnavano na strani 33, je primer pravičnega razvrščanja o katerem je mogoče najti največ opravljenih študij. Pričakovano bi bilo, da si najprej pogledamo primer tega razvrščanja. Verjetno zaradi svoje kompleksnosti algoritma žal na linux sistemih ni splošno uporabne izvedbe. Tudi na nekaterih drugih sistemih so izvedbe zelo statične. Imajo kar vnaprej določene uteži za preddefinirane tipe omrežnega prometa.

3.5.1 Krožno razvrščanje s primanjkljajem (Deficit Round Robin – DRR)

Za začetek zato preizkusimo delovanje enega od krožnih razvrščanj in sicer krožno razvrščanje s primanjkljajem – DRR, ki smo ga obravnavali na strani 42. Gre za zelo eksakten in dobro definiran algoritem, ki daje pričakovane in izredno točne rezultate. To je posledica enostavnosti algoritma, saj ta temelji samo na štetju količine poslanih podatkov in ne vsebuje nikakršnega merjenja časa in preračunavanj pretoka. Štetje nima pogreškov in približkov.

V našem primeru imamo dva komunikacijska toka, ki imata pretok v razmerju približno 1 : 1000. V začetni meritvi zato v tem razmerju nastavimo tudi uteži krožnega razvrščanja:

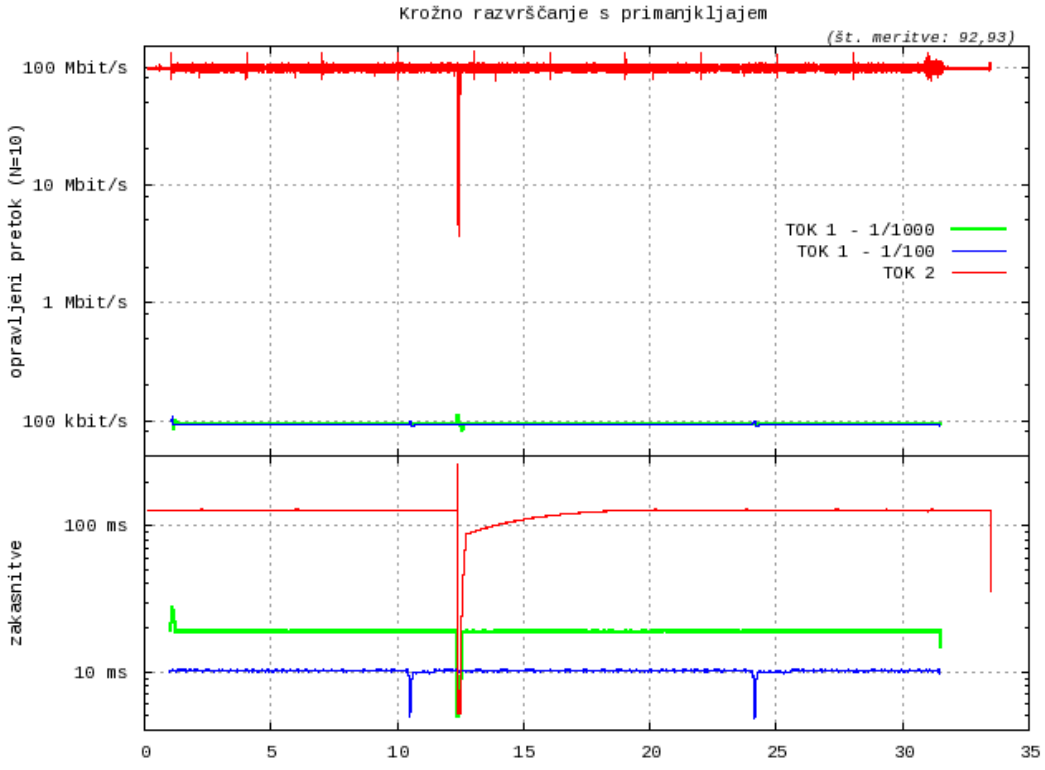
```
tc qdisc add dev eth0 root handle 1: drr

# definiramo tri razrede
# komunikacijski tok 1
tc class add dev eth0 parent 1:0 classid 1:100 drr quantum 1
# komunikacijski tok 2
tc class add dev eth0 parent 1:0 classid 1:200 drr quantum 1000
# ostalo
tc class add dev eth0 parent 1:0 classid 1:300 drr

# in filtre prometa
# komunikacijski tok 1
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
  match ip dst 4.4.4.4/32 flowid 1:100 action skbedit queue_mapping 0

# komunikacijski tok 2
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
  match ip dst 6.6.6.4/32 flowid 1:200 action skbedit queue_mapping 1

# ostalo
tc filter add dev eth0 protocol ip parent 1:0 prio 2 u32 \
  match ip dst 0.0.0.0/0  flowid 1:300 action skbedit queue_mapping 2
```

Nastavitve so zelo podobne kot v primeru razvrščanja po prioritetah. Prvi ukaz ustvari

drr disciplino na izhodnem ethernet krmilniku eth0. Drr disciplina razredov oziroma čakalnih vrst ne naredi samodejno kot smo videli pri prio, ampak moramo te narediti sami. Naredimo torej tri razrede. Prvemu razredu, skozi katerega bomo s pomočjo filtrov usmerili komunikacijski tok 1, dodelimo utež 1. Drugemu dodelimo utež 1000 in skozi usmerimo komunikacijski tok 2. Tretjemu razredu uteži ne določimo posebej in je namenjen morebitnemu ostalemu prometu.

Odpošiljanje paketov se izvaja ciklično samo iz vrst, ki vsebujejo čakajoče pakete. Prazne vrste proces preskoči. Iz vsake vrste se odpošlje toliko oktetov podatkov, kolikor dovoljuje kvota. Količina ne more biti poljubna, ampak vedno navzdol zaokroženo celoštevilčno število paketov. Če je preostala kvota premajhna za odpošiljanje naslednjega paketa, se h kvoti prišteje vrednost uteži in se shrani kot začetna kvota za naslednji krog.

Pretok komunikacijskih tokov je tako zelo točno krmiljen v razmerju nastavljenih uteži, kot za opisan primer prikazujeta zelena in rdeča krivulja na sliki 24.



Slika 24: krožno razvrščanje s primanjkljajem

Opisan princip delovanja zelo vpliva na zakasnitve. Paketi komunikacijskih tokov z večjimi utežmi imajo prednost pred paketi tokov z majhnimi utežmi, saj hitreje dosežejo potrebno kvoto za odpošiljanje paketa. Poglejmo si na primeru:

Komunikacijski tok 1 ima pakete velikosti približno 100 oktetov in pripadajoča čakalna vrsta utež 1. Komunikacijski tok 2 ima pakete velikosti približno 1500 oktetov in čakalna vrsta utež 1000. Ob prispetju prvega paketa toka 1 v usmerjevalnik ima pripadajoča čakalna vrsta kvoto 0. Prvi paket lahko odpošlje šele po 100 ciklih. Tok 2 oddaja približno 2 paketa (2×1500) na vsake 3 cikle (3×1000). Oddajanje 1500 oktetov velikega paketa v komunikacijski kanal s prepustnostjo 100 Mbit/s traja 0,12 ms. Predpostavimo še, da prazni cikli, ko do oddaje paketa ne pride, trajajo zanemarljivo malo časa. V 100 ciklih torej komunikacijski tok 2 odda 66 paketov in to traja 8 ms. Toliko časa mora paket toka 1 čakati, da sploh pride na vrsto. Temu moramo prišteti še 5 ms čakanja v internem medpomnilniku mrežnega krmilnika, kot smo ugotovili pri razvrščanju po prioritetah, in 5 ms zakasnitev v omrežju. Skupna zakasnitev paketov komunikacijskega toka 1 v omrežju je tako približno 18 ms, kar se popolnoma ujema z izmerjeno vrednostjo, ki jo prikazuje zelena krivulja na sliki 24.

Opisano velja tudi v primeru, če uteži sorazmerno povečamo. Čakalni vrsti komunikacijskega toka 1 dodelimo utež 100 in čakalni vrsti toka 2 utež 100.000 ter s tem ohranimo medsebojno razmerje. Potem tok 1 odda prvi paket že po prvem ciklu, ampak pred tem bi tok 2 oddal 100.000 oktetov, kar je spet 66 paketov po 1500 oktetov in traja enako 8 ms kot v prejšnjem primeru.

Pomembno je le razmerje uteži, medtem ko absolutna vrednost ni pomembna. V resnici to velja samo statistično. Večje vrednosti uteži pri ohranjenem razmerju le v povprečju dajo enake rezultate, medtem ko v kratkih intervalih povzročijo večje rafale in manj gladek pretok, saj v enem krogu lahko spustijo skozi omrežje več paketov. V času znotraj posameznega kroga pretok ni reguliran in poteka z največjo možno hitrostjo. Za gladek pretok zato izbiramo vrednosti uteži, ki so vsaj nekajkrat manjše od velikosti paketov.

Iz navedenega sledi, da je mogoče zakasnitve komunikacijskim tokom zmanjšati tako, da jim povečamo utež. S tem se jim hkrati lahko tudi poveča pretok, kar ni vedno zaželeno. Modra krivulja na sliki 24 prikazuje zmanjšanje zakasnitve komunikacijskega toka 1, če spremenimo uteži s prvotnega razmerja 1 : 1000 na 1 : 100. Po prej opisanem izračunu se zakasnitve paketom v krožnem postopku razvrščanja zmanjšajo z 8 ms na 0,8 ms. Skupaj z zakasnitvami v internem medpomnilniku mrežnega krmilnika in v omrežju znaša celotna zakasnitev skozi omrežje potem približno 11 ms. Pretok se pri tem ni spremenil, ker je omejen pri samem izvoru S1.

Slabosti krožnega razvrščanja s primanjkljajem

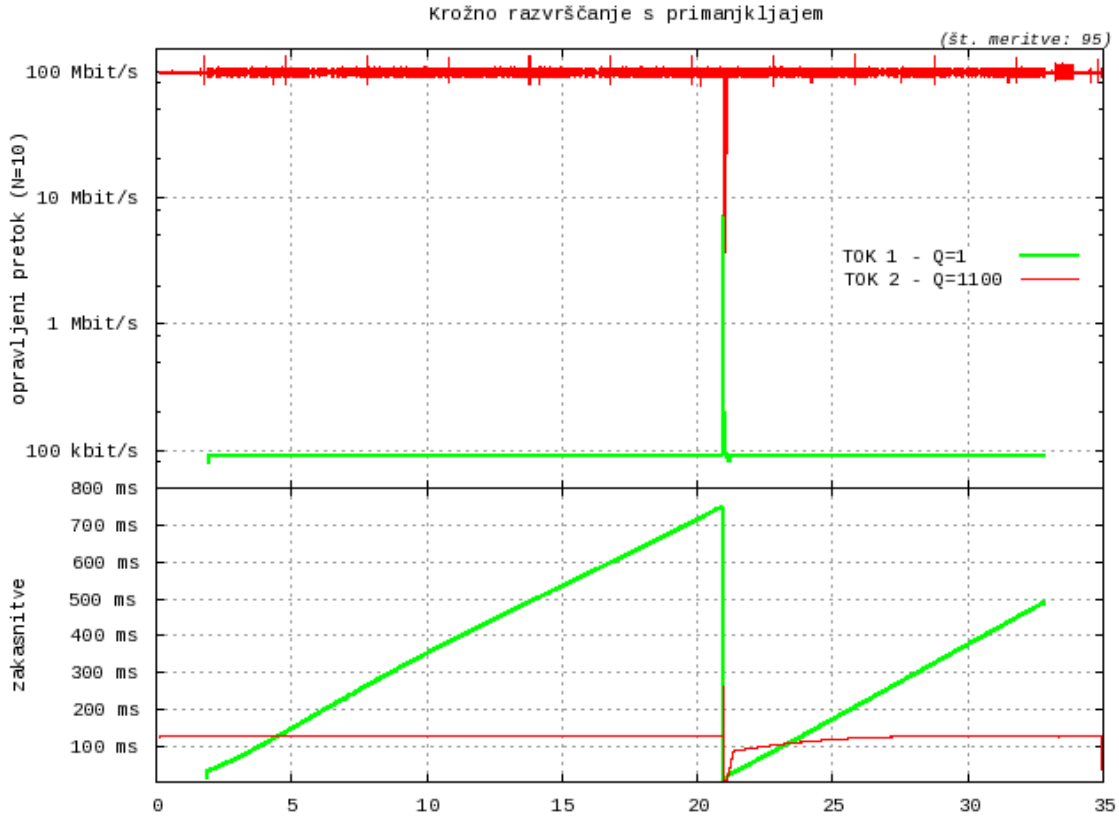
Krožno razvrščanje s primanjkljajem deluje zelo dobro in natančno v primeru, da posameznim tokom zagotovimo dovolj omrežnih virov. Do izvorov, ki razpoložljive vire presežejo tudi samo za malo, je krožno razvrščanje s primanjkljajem zelo neizprosno, saj pri zasedenih linijah ne tolerira prav nobenih rafalov.

Komunikacijski tok 1 pri pretoku 100 kbit/s odpošilja 100 oktetov velike pakete vsakih 8 ms. Če pri krožnem razvrščanju s primanjkljajem nastavimo uteži natančno na vrednosti, ki ustrezajo tem parametrom, in se naključno pripeti, da v nekem intervalu namesto enega prispeta dva paketa, bo prvi odposlan, medtem ko bo drugi na odpošiljanje moral čakati spet celih 8 ms. To je cel običajni interval med dvema paketoma komunikacijskega toka 1. Situacijo dodatno poslabšuje dejstvo, da bo v tem času prispel nov redni paket, ki tudi ne bo mogel biti odposlan v svojem predvidenem času. Zaradi enega samega vrinjenega paketa se od tega dalje zamakne odpošiljanje vsem paketom tega komunikacijskega toka. Zakasnitev razvrščanja se tako podvoji na 16 ms.

V praksi je težko nastaviti parametre tako natančno, da bi se zgodil opisan scenarij, kjer bi en sam paket toliko zmotil delovanje procesa. Kljub temu lahko dejansko naletimo na izredno občutljivost zakasnitev na malo povečan pretok. Slika 25 prikazuje pretok in zakasnitve v omrežju, če spremenimo razmerje uteži samo za 10%. Od prvotnega razmerja 1 : 1000 spremenimo razmerje na 1 : 1100. Tako nastavljena prepustnost kanala ne zadostuje več za nemoten pretok komunikacijskega toka 1.

En paket komunikacijskega toka 1 ne pride več na vrsto na vsakih 66 paketov toka 2 ampak na 73 paketov, kar pomeni vsakih 8,8 ms. Pretok toka 1 se tako tudi zmanjša za 10%. To pomeni, da ne morejo več biti vsi paketi odposlani pravočasno v predvidenem času in se počasi kopičijo v izhodni čakalni vrsti usmerjevalnika R1. Čakalna vrsta se podaljšuje in vsak na novo prispeli paket mora na odpošiljanje čakati linearno vedno več ciklov. Zakasnitve skozi omrežje od prvotnih 18 ms linearno naraščajo skoraj na sekundo. Teoretično bi zakasnitve naraščale brez meja. V praksi bi samoregulacijski mehanizem TCP protokola sčasoma poskrbel za zmanjšanje pretoka pri izvoru.

V prikazanem primeru na sliki 25 se zgodi drugačen scenarij. Komunikacijski tok 2 približno v 21. sekundi izgubi en paket, na kar izvor reagira z bliskovitim zmanjšanjem pretoka. To razbremenitev izkoristi komunikacijski tok 1 in v kratkem času odpošlje vse zaostale pakete.



Slika 25: zakasnitve pri krožnem razvrščanju s primanjkljajem

3.5.2 Hierarhične pravične prenosne krivulje (Hierarchical Fair Service Curve HFSC)

Slabost pravičnih razvrščanj je soodvisnost prepustnosti in zakasnitev omrežja, kar je posledica njihove linearne prenosne karakteristike. Z večanjem uteži posameznim komunikacijskim tokom dodelimo večjo prepustnost. Hkrati s tem ta pomeni tudi manjše časovne zakasnitve. Velja tudi obratno. Manjše dodeljene uteži istočasno določajo manjšo prepustnost in večje zakasnitve.

Ta lastnost je pri uteženem pravičnem razvrščanju posledica principa delovanja tokovnega modela. Komunikacijskim tokom z manjšimi utežmi se končni čas prenosa paketa raztegne dlje v prihodnost, zato so ti paketi odposlani kasneje. Podobno tudi pri krožnih razvrščanjih komunikacijski toki z manjšimi utežmi kasneje dosežejo potrebno kvoto za odpošiljanje.

To slabost odpravlja razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami, kjer nastavimo prelomljeno prenosno karakteristiko in s tem komunikacijski tok glede na potrebe na začetku pospešimo ali zavremo. Gre za nov in posledično še dokaj slabo s

praktičnimi primeri podprt način razvrščanja.

Omeniti velja še, da pri novejši izvedbi razvrščanja s hierarhičnimi pravičnimi prenosnimi krivuljami na linux sistemih lahko določimo ne le eno, ampak kar tri prenosne krivulje, od katerih ima lahko vsaka začetno *m1* in končno *m2* strmino ter točko preloma pri času *d*:

- ls – »link sharing« krivulja; osnovna krivulja, ki določa souporabo povezave. Čeprav je podana v bit/s, to ni trdna določitev prepustnosti, ampak deluje kot utež – pomembno je le medsebojno razmerje. Če vsota vrednosti pri vseh razredih ne doseže prepustnosti celotnega kanala, se preostanek razdeli v istem razmerju in vsak razred prejme sorazmerni dodatek. Obratno je, če vsota vrednosti presega skupno prepustnost, se sorazmerno z vrednostim vsakemu razredu delež prepustnosti odvzame.
- rt – »real time« krivulja; opcijska krivulja realnega časa. Ta trdno določa spodnjo mejo dodeljene prepustnosti. Vsakemu razredu zagotovo pripada vsaj toliko. Manj edino v primeru, ko komunikacijski toki zaradi lastnih omejitev ne izkoristijo cele rt kvote. V tem primeru se višek spet razdeli ostalim v razmerju ls krivulje.
  Vsota prepustnosti določenih z rt krivuljami ne sme presegati celotne kapacitete kanala. To vodi v nestabilno delovanje, saj pride do spora med komunikacijskimi toki. Izmenično poizkuša vsak zasesti dodeljeno prepustnost na račun ostalih, kar vodi v pulzirajoče vrednosti pretokov in zakasnitev. Izkustveno priporočilo je, da vsota rt krivulj doseže največ 80% kapacitete celotnega komunikacijskega kanala.
- ul – »upper limit« krivulja; opcijska krivulja zgornje meje. Ta trdno določa zgornjo mejo dodeljene prepustnosti. Vsakemu razredu pripada največ toliko.

Razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami nastavimo podobno kot v prejšnjih primerih. Najprej nastavimo hfsc disciplino na izhodnem ethernet krmilniku eth0. Potem naredimo tri razrede za posamezne tipe komunikacijskega prometa in s filtri vanje usmerimo pripadajoče toke.

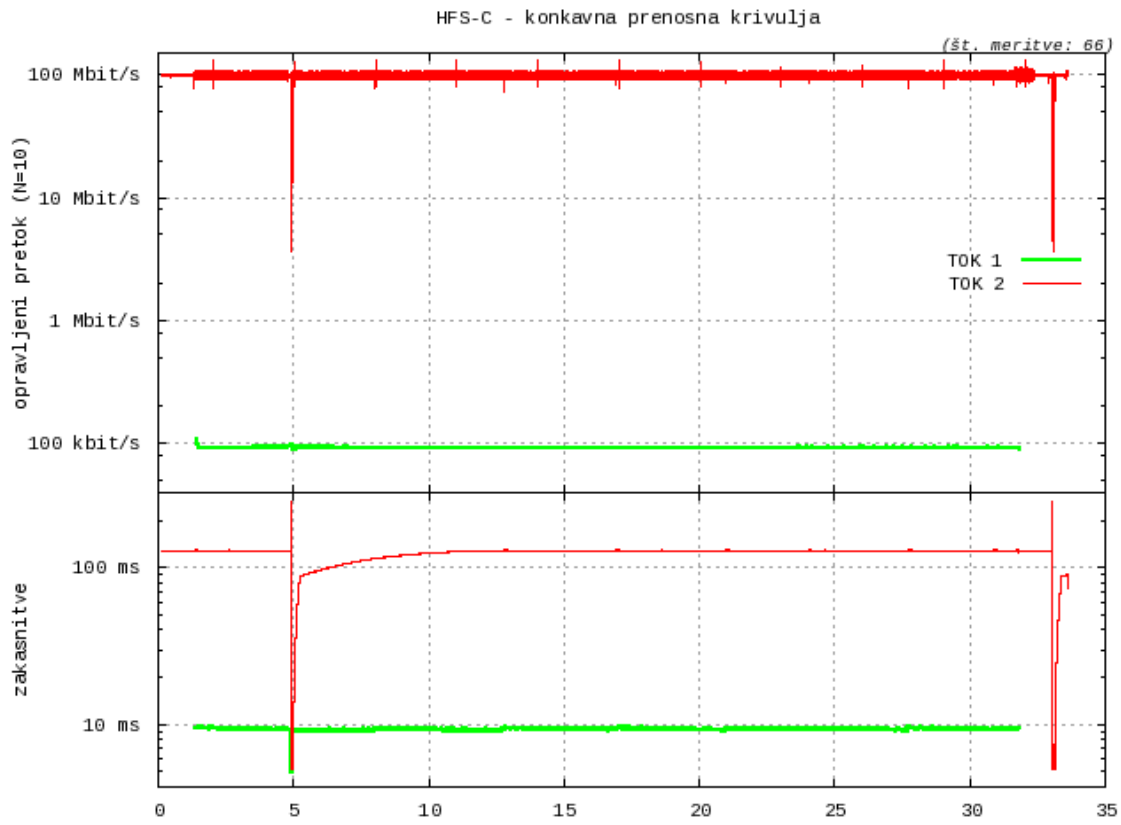
```
tc qdisc add dev eth0 root handle 1: hfsc

# definiramo tri razrede
# komunikacijski tok 1
tc class add dev eth0 parent 1:0 classid 1:100 hfsc ls \
   m1 100Mbit d 10ms m2 100kbit
# komunikacijski tok 2
tc class add dev eth0 parent 1:0 classid 1:200 hfsc ls m2 99.9Mbit
# ostalo
tc class add dev eth0 parent 1:0 classid 1:300 hfsc ls m2 8bit

# in filtre prometa
# komunikacijski tok 1
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
  match ip dst 4.4.4.4/32 flowid 1:100 action skbedit queue_mapping 0

# komunikacijski tok 2
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
  match ip dst 6.6.6.4/32 flowid 1:200 action skbedit queue_mapping 1

# ostalo
tc filter add dev eth0 protocol ip parent 1:0 prio 2 u32 \
  match ip dst 0.0.0.0/0  flowid 1:300 action skbedit queue_mapping 2
```

Prvi razred 1:100 je namenjen šibkemu komunikacijskemu toku 1, ki ga želimo časovno pospešiti. Zato mu v začetnem delu namenimo kar celotno prepustnost kanala – *m1=100 Mbit/s*, vendar samo v trajanju *d=10 ms*, kar je malo več kot interval enega paketa tega toka. V nadaljevanju mu dopustimo običajen pretok – *m2=100 kbit/s*. Drugi razred 1:200 je namenjen komunikacijskemu toku 2. Dodelimo mu linearno karakteristiko s prepustnostjo *m2=99,9 Mbit/s*, saj nas razmere ob začetku tega toka ne zanimajo in jih ne spremljamo. Meritve opravljamo, ko je pretok tega toka že ustaljen. Tretji razred 1:300 za preostali promet nas posebej ne zanima, zato mu dodelimo minimalno prepustnost.

Da paket komunikacijskega toka 1 dobi prednost pred paketom toka 2, mora v pripadajočem tokovnem modelu zaključiti odpošiljanje pred njim. Za osvežitev si lahko podoben primer ogledamo na strani 40, slika 16. Ker so paketi toka 1 petnajstkrat manjši od paketov toka 2, ni potrebno da bi toku 1 dodelili nominalno večjo utež kot toku 2. Dovolj bi bilo že, da dodelimo uteži v razmerju manjšem od 15 : 1. Pri razmerju 15 : 1 se namreč končna časa hkrati prispelega 1500 oktetov velikega paketa toka 1 in 100 oktetov velikega paketa toka 2 ravno izravnata. Dobimo situacijo, ki jo sicer v nekoliko drugačnem razmerju uteži in velikosti paketov na strani 31 prikazuje slika 13. Ker ima tok 2 utež 99,9 Mbit/s, moramo utež – *m1* strmino toka 1 nastaviti na 6,66 Mbit/s ali več (nastavili smo jo na 100 Mbit/s).

Tako nastavljeno razvrščanje nam da optimalne rezultate, ne da bi tok 1 privilegirali,

kot smo ga pri razvrščanju s prioritetami, ali ga prekomerno utežili, kot je bilo potrebno pri krožnem razvrščanju s primanjkljajem. Rezultate prikazuje diagram na sliki 26. Dobimo pričakovan pretok komunikacijskega toka 1 približno 100 kbit/s z zakasnitvami okoli 10 ms in pretok približno 100 Mbit/s z zakasnitvami 128 ms pri toku 2.



Slika 26: razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami

Povečanje zakasnitev

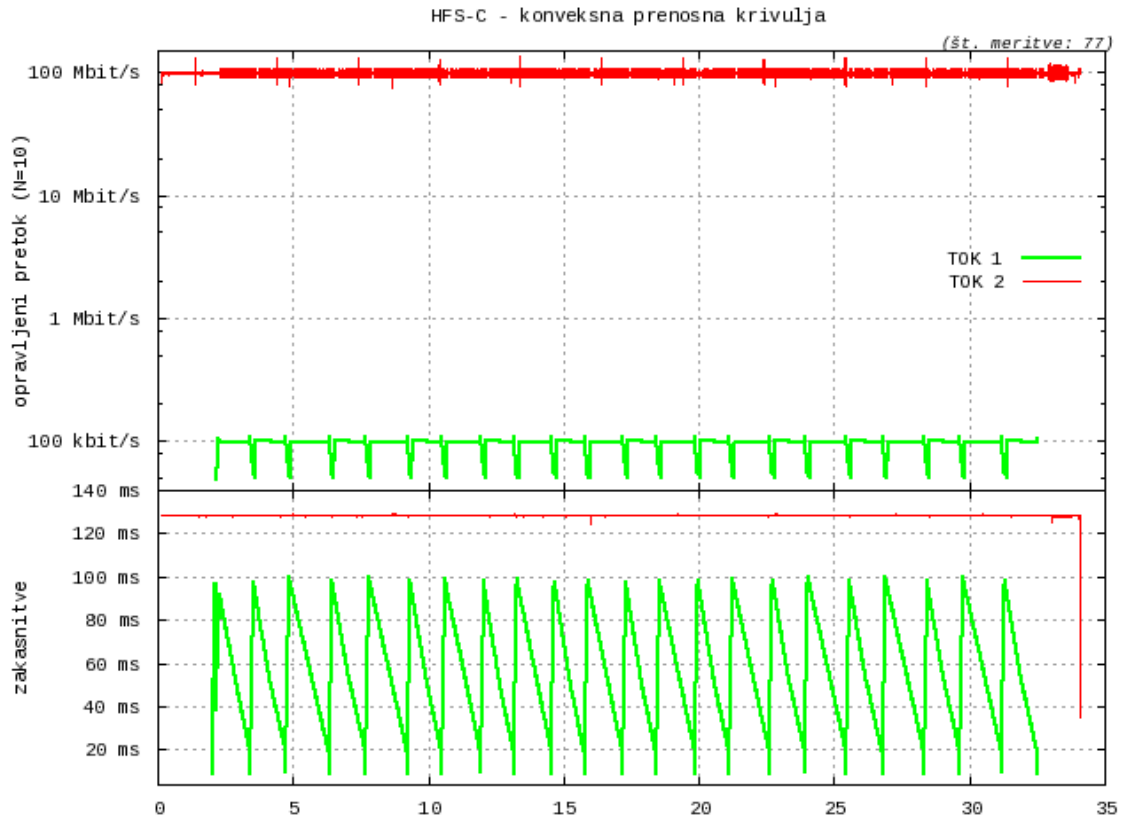
Za test si poglejmo še, ali je mogoče z razvrščanjem s hierarhičnimi pravičnimi prenosnimi krivuljami res povečati zakasnitve. V ta namen komunikacijskemu toku 1 nastavimo obratno kot prej, konveksno prenosno karakteristiko. Ta ima v začetnem intervalu *m1* položnejšo strmino, torej zmanjšano prepustnost, kar zavre prvih nekaj paketov toka 1. Začetno *m1* strmino nastavimo na primer na 10 kbit/s v trajanju 0,1 s. V nadaljevanju ima karakteristika spet običajno strmino 100 kbit/s, kar zagotavlja nemoten pretok toka. Konfiguracija se razlikuje od prejšnjega primera samo v delu, kjer ustvarimo razred 1:100 in je namenjen komunikacijskemu toku 1.

```
  ...

# komunikacijski tok 1
tc class add dev eth0 parent 1:0 classid 1:100 hfsc \
  ls m1 10kbit d 0.1s m2 100kbit ul m1 10kbit d 0.1s m2 100kbit

  ...
```

Test pokaže, da samo z nastavitvijo ls krivulje ni mogoče omejiti toka na tako majhno vrednost. 10 kbit/s v primerjavi s celotno prepustnostjo kanala predstavlja samo 0,01%. Ko TCP protokol s samoregulacijo komunikacijskega toka 2 skuša zasesti kanal, mu to ne uspe popolnoma. Nezasedeni preostanek je vedno večji od 10 kbit/s in to vrzel zasede tok 1 ne glede na postavljeno omejitev. Ls krivulja namreč postavlja samo priporočilo in ne omejitve. V primeru prostih virov je priporočilo lahko preseženo. Meritev zato ne pokaže pričakovanih rezultatov. Pri bolj primerljivih vrednostih tokov na to pomanjkljivost ne naletimo. Da pretok komunikacijskega toka 1 kljub opisani posebnosti omejimo, poleg ls krivulje nastavimo tudi ul krivuljo. Ta postavlja zgornjo mejo, ki strogo omejuje pretok toka, da ta ne preseže nastavljenih vrednosti. Ul krivulji nastavimo kar enake parametre kot ls krivulji.

Rezultate meritev tako nastavljenega razvrščanja prikazuje slika 27. Kljub temu, da na prvi pogled izgledajo nekoliko presenetljivi, lahko ugotovimo, da se je povprečna zakasnitev komunikacijskega toka 1 res povečala in znaša približno 50 ms.

Pri razumevanju rezultatov meritve moramo biti pozorni, kaj sploh je začetek komunikacijskega toka, kjer pride do veljave začetna *m1* strmina prenosne krivulje. Razvrščanje paketov poteka v drugem kanalnem in tretjem mrežnem sloju in ne pozna niti sej, kot tudi ne vzpostavljenih povezav. **Razvrščanje šteje za začetni interval vsako skupino paketov, ki začnejo polniti pred tem prazno čakalno vrsto.** Če se zaradi občasne prekinitve ali manjšega pretoka čakalna vrsta sprazni, bodo naslednji prispeli paketi spet obravnavani kot začetek komunikacijskega toka in mu bo v danem časovnem intervalu *d* ponovno dodeljena večja ali manjša prepustnost, kot jo nastavimo s parametrom *m1*. Tako se lahko zgodi tudi, da močno rafalni tok dobi večjo prepustnost kot je bilo predvideno. Če je to moteče moramo skrbno paziti, da je začetni interval *d* dovolj kratek – v idealnem primeru samo za čas odpošiljanja enega samega paketa, kateremu potem v intervalih določenih s strmino *m2* sledijo ostali paketi.



Slika 27: povečanje zakasnitev s hierarhičnimi pravičnimi prenosnimi krivuljami

V primeru s slike 27 se dogaja nekaj podobnega. V začetnem *m1* delu je prenosna krivulja zelo položna (10 kbit/s) in ne zadostuje za nemoten pretok komunikacijskega toka 1. V čakalni vrsti se začne tvoriti čakalna vrsta. Zakasnitve se zelo strmo linearno povečujejo. Po intervalu $d = 0{,}1\ s$, kar ustreza malo več kot 10 medpaketnim intervalom, preide prenosna krivulja v znatno večjo strmino *m2* (100 kbit/s). Prepustnost je malo večja od potreb toka 1, zato se čakalna vrsta postopoma izprazni. Zakasnitve nekoliko počasneje linearno upadajo. Ko je čakalna vrsta prazna, delovanje ponovno preide v začetno *m1* strmino. Postopek se periodično ponavlja. Dobimo nestabilne vrednosti zakasnitev, ki ciklično linearno naraščajo in upadajo. Vrednosti zakasnitev bi se stabilizirale edino v primeru, če bi bil ponujan pretok natančno enak nastavljeni *m2* strmini prenosne karakteristike. Število prispelih in odposlanih paketov iz čakalne vrste bi moralo biti ves čas enako. V praksi je mogoče doseči konstantno vrednost zakasnitev edino na spodnji vrednosti, ki ga omejujejo zakasnitve samega omrežja in ob praznih čakalnih vrstah, ali na zgornji vrednosti, ki ga določa samoregulacijska lastnost uporabljenega TCP protokola ali kakršenkoli drug regulacijski mehanizem, ki sproti prilagaja ponujan pretok v okvirih, ki zagotavlja, da bi bile čakalne vrste ves čas enako zapolnjene.

Povečanje zakasnitev nastavljamo prvenstveno z manjšanjem začetne prepustnosti, strmine *m1*. Da se izognemo vplivu na pretok, poizkušamo začetni interval *d* obdržati na čim manjši vrednosti. Šele če s strmino *m1* ne moremo doseči zadovoljivih rezultatov, začnemo povečevati tudi interval *d*. Če s povečevanjem zakasnitev pretiravamo, začnejo nanje vplivati tudi samoregulacijske lastnosti TCP protokola. Spreminjati se začne ponujani pretok, občasno prihaja celo do ponavljanj paketov, ki sploh niso bili izgubljeni.

Slabosti razvrščanja s hierarhičnimi pravičnimi prenosnimi krivuljami

Razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami naj bi po besedah avtorjev [8] omogočalo neodvisno nastavljanje prepustnosti in zakasnitev. Kljub temu je še vedno večji poudarek pri nastavitvah prepustnosti, saj tudi zakasnitve še vedno nastavljamo posredno prek nastavljanja prepustnosti. Tako je pri izbranih nastavitvah zelo težko predvidevati, kakšen bo dejanski rezultat zakasnitev. Velja tudi obratno. Če želimo zakasnitve v danih mejah, ni mogoče trivialno nastaviti parametrov razvrščanja. Veliko bolj uporabno bi bilo, da bi prepustnost nastavljali v bit/s in zakasnitve kar v milisekundah.

Največkrat nas nominalne vrednosti zakasnitev ne zanimajo. Zagotoviti želimo le zgornjo mejo in medsebojna razmerja med različnimi toki. Vsaj slednje je mogoče dokaj enostavno nastavljati. Razredom, v katere razvrstimo toke, kar izkustveno v izbranih medsebojnih stopnjah podelimo različne *m1* strmine prenosnih karakteristik.

Dvostopenjsko krmiljenje pretoka

Zaradi omenjene pomanjkljivosti v praksi razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami le redko najde mesto pri upravljanju zakasnitev. Pogosteje se je pojavila drugačna uporaba. Primer prikazuje naslednji del nastavitev:

```
  ...

# spletni promet
tc class add dev eth0 parent 1:0 classid 1:500 hfsc \
  ls m1 10Mbit d 10s m2 2Mbit

tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
  match ip sport 80 0xffff flowid 1:500

  ...
```

Prikazano je dvostopenjsko krmiljenje pretoka spletnega prometa (izvorna vrata 80). Spletni promet, kakor tudi nekateri drugi, lahko prenaša tako interaktivne vsebine kakor tudi večje prenose podatkov. Podan primer omogoča večjo prepustnost 10 Mbit/s v prvih 10 sekundah. Če prenos traja dlje, kar je praviloma pri prenosih datotek, se pretok potem zmanjša na 2 Mbit/s, da preveč ne moti drugega interaktivnega prometa.

3.5.3 Hierarhično razvrščanje z vedri in žetoni (Hierachical token bucket – HTB)

V prejšnjem poglavju obravnavano razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami je nastalo kot nova in kar nekoliko revolucionarna oblika pravičnih razvrščanj, ki ločuje nastavitve prepustnosti od nastavitve zakasnitev – predvsem z namenom zagotavljanja majhnih zakasnitev šibkim komunikacijskim tokom, ki danes pripadajo IP telefoniji in podobnim storitvam delujočim v realnem času.

Če si podrobneje ogledamo delovanje, ki temelji na dejstvu, da začetnim paketom ali paketom po krajši prekinitvi komunikacijskega toka dodelimo večjo prepustnost izhodnega kanala, ugotovimo, da je to zelo podobno razvrščanju z vedrom in žetoni. Če nekaj časa paketi prihajajo z manjšo hitrostjo ali sploh ne, se vedro napolni z žetoni in nov rafal paketov zato hitro zdrvi skozi. Na ta način dobimo konkavnost prenosne funkcije podobno kot pri pravičnih prenosnih krivuljah in s tem manjše zakasnitve paketov.

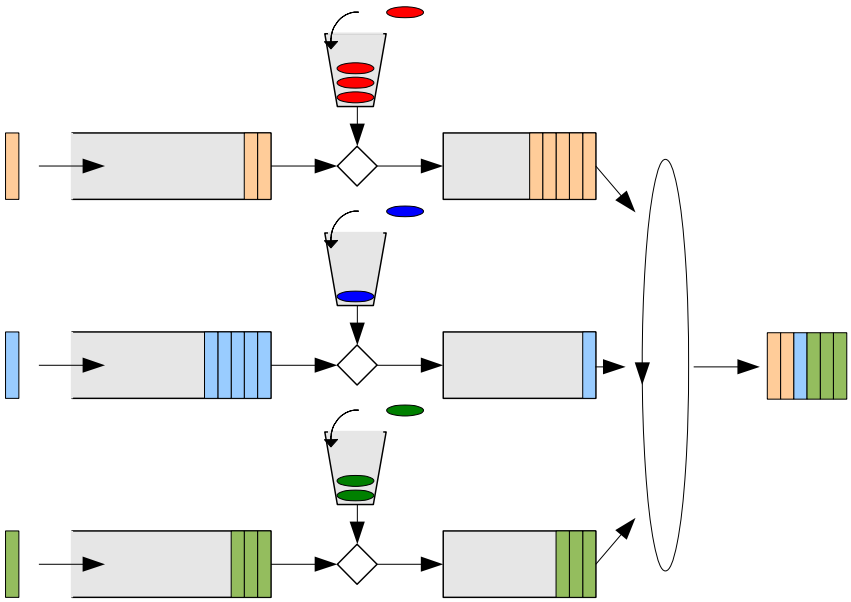
Z vedri in žetoni pa ne moremo upodobiti konveksne prenosne funkcije, kjer pakete na začetku nekoliko zadržimo in jih sprostimo šele po začetnem intervalu z namenom, da paketom zakasnitve povečamo.

Obstaja tudi podobnost med razvrščanjem z vedri in žetoni ter med krožnim razvrščanjem s primanjkljajem. Pri krožnem razvrščanju sistem po vsakem ciklu dobi dodatno kvoto, ki omogoča odpošiljanje novih paketov. Pri razvrščanju z vedri in žetoni sistem periodično dobiva žetone, ki omogočajo odpošiljanje novih paketov. Bistvena razlika med tema dvema postopkoma je v začetnem delovanju. Krožno razvrščanje ima na začetku ali po krajši prekinitvi toka vedno kvoto nič in mora počakati, da se ta napolni. Šele potem lahko začne odpošiljati pakete. Medtem ko razvrščanje z vedri in žetoni ob začetku ali po prekinitvi toka pričaka polno vedro žetonov, saj žetoni pritekajo tudi v neaktivnem času in se ne porabljajo. To omogoča ne le takojšnje odpošiljanje z nastavljenim pretokom, marveč celo krajši začetni rafal, ki običajni pretok presega. Razvrščanje z vedri in žetoni je torej veliko bolj strpno do občasnih rafalov in brez težav lahko zagotavlja majhne zakasnitve tudi šibkim

tokom.

Hierarhično razvrščanje z vedri in žetoni je v linux okolju prisotno že dolgo časa in je najbolj pogosto uporabljan način pravičnih razvrščanj.

Hierarhično razvrščanje z vedri in žetoni je hkrati tudi zanimiv primer uporabe tega postopka z namenom razdeljevanja skupnih omrežnih virov med več komunicirajočih subjektov. Postopke z vedri sicer pogosteje srečamo za omejevanje ali glajenje pretoka in ne kot postopek pravičnega razvrščanja.

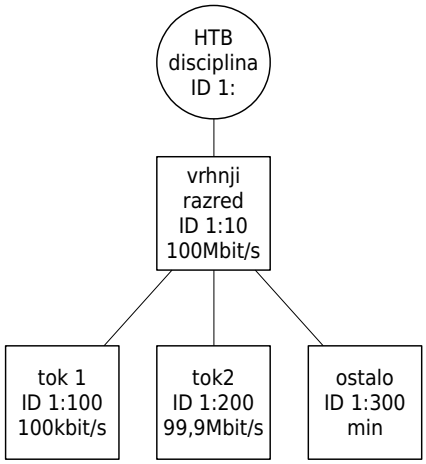


Slika 28: hierarhično razvrščanje z vedri in žetoni

Postopek spet temelji na razdelitvi omrežnega prometa v več čakalnih vrst s pomočjo filtrov. Ta del postopka na sliki ni predstavljen. Vsaka čakalna vrsta posebej je krmiljena s svojim vedrom in žetoni. Na koncu so komunikacijski toki v krožnem procesu spet združeni in poslani v izhodni kanal. S pomočjo parametrov krmilimo prepustnost, kot tudi zakasnitve vsake čakalne vrste posebej.

S poznavanjem teoretičnega ozadja razvrščanj z vedri in žetoni so nastavitve razmeroma preproste. Razvrščanje z manjšo razliko nastavimo podobno kot v prejšnjih primerih. Najprej nastavimo htb disciplino na izhodnem ethernet krmilniku eth0. Za tem naredimo skupni razred za vse komunikacijske toke. Ta korak ni obvezen, vendar omogoča medsebojno delitev omrežnih virov med podrejenimi razredi. Neizkoriščeni viri se namreč lahko prerazporedijo edino znotraj razredov. Brez hierarhične strukture bi bili viri nižje ležečih razredov med sabo popolnoma izolirani. Shemo načrtovane strukture razvrščanja z

vedri in žetoni prikazuje slika 29.



Slika 29: hierarhija razvrščanja z vedri in žetoni

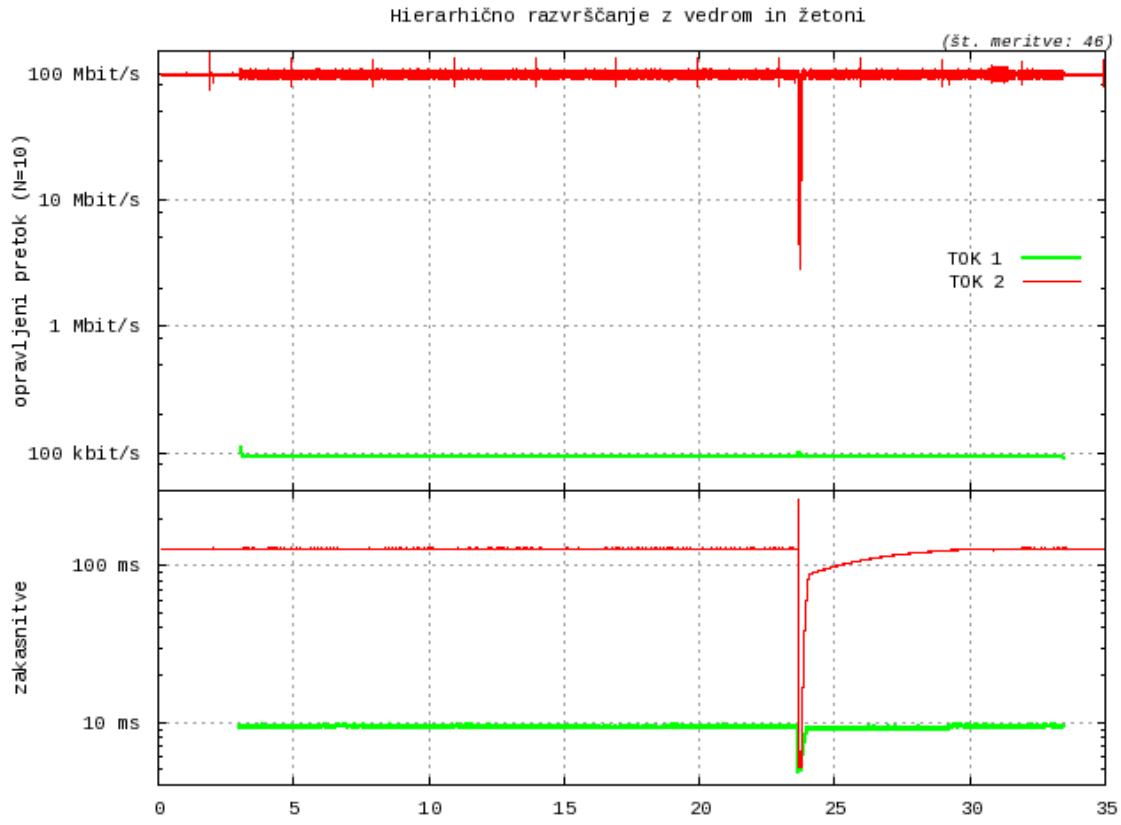
V resnici smo imeli povsem enako strukturo tudi v prejšnjem primeru pri razvrščanju s hierarhičnimi pravičnimi prenosnimi krivuljami, le da tam vrhnji razred nastane samodejno pri kreiranju hfsc discipline, s katero si tudi deli isto identifikacijsko oznako.

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

# nadrejeni skupni razred
tc class add dev eth0 parent 1:0 classid 1:10 htb rate 100Mbit

# definiramo tri razrede ...
# komunikacijski tok 1
tc class add dev eth0 parent 1:10 classid 1:100 \
  htb prio 1 rate 100kbit ceil 100Mbit burst 250

# komunikacijski tok 2
tc class add dev eth0 parent 1:10 classid 1:200 \
  htb prio 2 rate 99.9Mbit ceil 100Mbit

# ostalo
tc class add dev eth0 parent 1:10 classid 1:300 \
  htb prio 3 rate 8bit ceil 100Mbit

# ... in filtre prometa
# komunikacijski tok 1
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
  match ip dst 4.4.4.4/32 flowid 1:100 action skbedit queue_mapping 0

# komunikacijski tok 2
tc filter add dev eth0 protocol ip parent 1:0 prio 1 u32 \
  match ip dst 6.6.6.4/32 flowid 1:200 action skbedit queue_mapping 1

# ostalo
tc filter add dev eth0 protocol ip parent 1:0 prio 2 u32 \
  match ip dst 0.0.0.0/0  flowid 1:300 action skbedit queue_mapping 2
```

Pri krožnem in prioritetnem razvrščanju vrhnji razred za delitev virov med razredi ni potreben, saj že načina delovanja teh algoritmov samodejno prerazporejata neizkoriščene vire. Pod vrhnjim razredom naredimo tri spodnje razrede za posamezne tipe komunikacijskega prometa in s filtri vanje usmerimo pripadajoče toke.

Prvi razred 1:100 je namenjen šibkemu komunikacijskemu toku 1. Dodelimo mu potrebnih 100 kbit/s prepustnosti in vedro, ki dopušča 250 oktetov velik rafal, kar zadostuje za nekaj več kot dva paketa. Drugi razred 1:200 je namenjen komunikacijskemu toku 2. Dodelimo mu 99,9 Mbit/s prepustnosti. Velikosti vedra ne določimo, uporabi se privzeta velikost enega paketa. Tretji razred 1:300 za preostali promet nas posebej ne zanima, zato mu dodelimo minimalno prepustnost. Vsem razredom dovolimo, da v primeru neizkoriščenosti pretok povečajo do *ceil = 100 Mbit/s*, kar ustreza celotni prepustnosti izhodnega kanala.

Rezultate meritev prikazuje diagram na sliki 30. Dobimo pričakovan pretok komunikacijskega toka 1 približno 100 kbit/s z najmanjšimi možnimi zakasnitvami okoli 10 ms in pretok približno 100 Mbit/s z zakasnitvami 128 ms pri toku 2.



Slika 30: hierarhično razvrščanje z vedri in žetoni

Razvrščanje z vedri in žetoni nam da praktično enake rezultate kot razvrščanje s hierarhičnimi pravičnimi prenosnimi krivuljami opisanim v prejšnjem poglavju. Primerljiva je tudi stopnja težavnosti nastavitev. Ocenjujemo, da gre za dva dokaj enakovredna postopka iz družine pravičnih razvrščanj, le da ima hierarhično razvrščanje z vedri in žetoni že dolgo tradicijo.

Zaključek

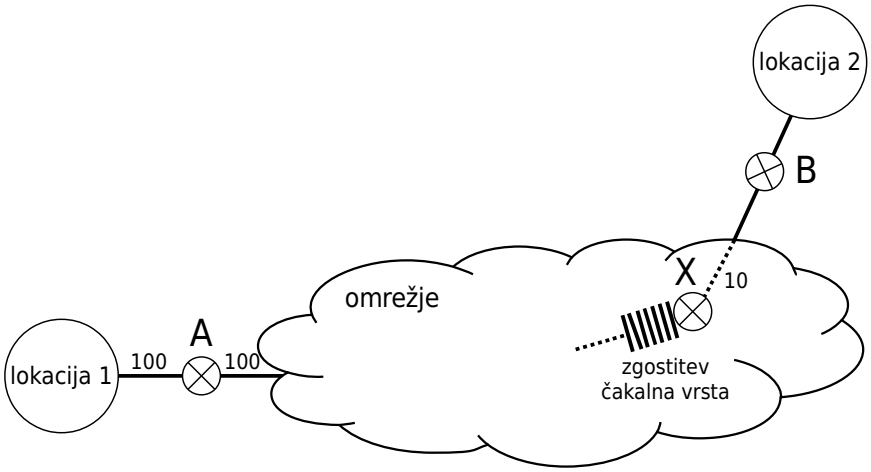
Tabela 6 prikazuje glavne lastnosti, prednosti in pomanjkljivosti posameznih, v tem delu obravnavanih razvrščanj paketov.

razvrščanje	***bistvene lastnosti***	***prednosti***	***slabosti***
v prihajajočem vrstnem redu – FIFO	*najenostavnejše razvrščanje, ki »se zgodi samo od sebe«*	*najenostavnejše,* *ob rafalih sebi blaži zakasnitve, saj jih prevali tudi na druge*	*veliki medsebojni vplivi tokov,* *nobenega nadzora*
puščajoče vedro	*strogo omejevanje pretoka,* *intenzivno razvrščanje*	*uravnavanje pretoka*	*vnaša dodatne zakasnitve*
z vedrom in žetoni – TBF	*omejevanje pretoka z dopustnimi rafali,* *intenzivno razvrščanje*	*uravnavanje pretoka in rafalov*	*vnaša dodatne zakasnitve*
po prioritetah	*razvršča toke glede na prioriteto*	*enostavna in učinkovita delitev virov*	*potencialna možnost, da višje prioritetni promet porabi vse vire (možnost zlorabe)*
uteženo pravično – WFQ	*razvršča toke po pripadajočem tokovnem modelu*	*pravično deli vire,* *zagotavlja omejene zakasnitve*	*zapleten postopek za izvedbo,* *veliko procesiranja*
krožno s primanjkljajem – DRR	*enostaven in točen postopek*	*pravično deli prepustnost*	*neizprosen, nobenih izjem in odstopanj*
hierarhične pravične prenosne krivulje – HFSC	*ločuje zagotovljeno prepustnost in omejevanje zakasnitev*	*šibkim tokom omogoča majhne zakasnitve,* *omogoča dvostopenjsko krmiljenje pretoka*	*zapleten in slabše dokumentiran postopek*
hierarhično z vedri in žetoni – HTB	*pravično razvrščanje z uporabo veder in žetonov*	*do neke mere ločuje prepustnost in zakasnitve,* *dokaj enostaven in učinkovit postopek*	*poseben, nestandarden postopek*
naključno zgodnje odmetavanje – RED	*naključno odmetava pakete, ko se pretok šele približuje zgornji meji prepustnosti*	*enostaven, učinkovit in procesno nezahteven postopek,* *primeren za zmogljiva hrbtenična omrežja*	*dodatne izgube paketov,* *pri protokolih brez samoregulacije ni učinka*
omrežni emulator – NETEM	*testno razvrščanje, oponaša delovanje realnih omrežij*	*bogat nabor nastavitev lastnosti in statističnih porazdelitev verjetnosti pojavov*	

Tabela 6: povzetek lastnosti razvrščanj

Namen pričujočega dela ni ukvarjati se direktno s postopki za zagotavljanje kakovosti storitev. Namen aktivnih razvrščanj v paketnih omrežjih je bolj osnoven: delitev skupnih omrežnih virov med posamezne komunikacijske toke, oziroma med sočasne komunicirajoče subjekte. Opisane postopke razvrščanj lahko uporablja in nastavlja administrator statično pri konfiguraciji omrežja. Lahko so izvajani tudi dinamično z namenskimi signalizacijskimi protokoli v upravljalski ravnini. V slednjem primeru opisana razvrščanja res predstavljajo osnovo načinov za zagotavljanje kakovosti storitev. Tako pridemo do aktivnega upravljanja komunikacijskih tokov in do rezervacijskih shem. V TCP/IP omrežjih so prvi poznani pod nazivom diferencirane in drugi pod integrirane storitve.

Splošno razširjeno mnenje je, da moramo za doseganje pričakovanih rezultatov pakete, ki tvorijo posamezne komunikacijske toke, razvrščati na isti način po celotni poti v vseh mrežnih elementih. To ni povsem točno. Tudi v našem testnem primeru se izkaže, da dosežemo vpliv na lastnosti pretoka paketov samo z razvrščanjem v usmerjevalniku R1 (slika 19 na strani 51). Pri neintenzivnih razvrščanjih velja pravilo, da razvrščamo samo v čakalni vrsti čakajoče pakete. Te najdemo samo v omrežnih elementih, kjer nastajajo zgostitve zaradi združevanja prometa iz več vhodnih kanalov ali zaradi manj prepustnih izhodnih linij. V praksi velikokrat naletimo na primer, ki ga prikazuje slika 31.



Slika 31: premik zgostitve paketov

Od lokacije 1 do lokacije 2 poteka več vrst komunikacijskih tokov, od katerih nekateri pripadajo bolj in drugi manj časovno zahtevnim storitvam. Da bi bolj zahtevnim storitvam

zagotovili krajše zakasnitve, na robnem usmerjevalniku A, nad katerim imamo popoln administrativni nadzor, vpeljemo razvrščanje po prioritetah, ki zahtevnejšim storitvam pripadajoče pakete obravnava z višjo prioriteto in manj zahtevne z nižjo. Taka rešitev, na prvi pogled nekoliko presenetljivo, ne da želenih rezultatov. Na robnem usmerjevalniku A imamo namreč eno vhodno in eno izhodno linijo z enako prepustnostjo 100 Mbit/s. Paketi skozenj potujejo brez čakanja in z minimalno zakasnitvijo. Nastavljeno razvrščanje po prioritetah zato nima nikakršnega učinka. Zgostitev prometa nastaja v povezovalnem omrežju in sicer v mrežnem elementu X, kjer se izhodna prepustnost zmanjša na 10 Mbit/s. Pravo mesto za razvrščanje čakajočih paketov, ki bi imelo učinek, bi bilo na tem mestu. Žal do naprav povezovalnega omrežja nimamo administrativnega dostopa, da bi to uresničili. Rešitev kljub temu obstaja.

Celotnemu komunikacijskemu prometu od lokacije 1 proti lokaciji 2 že v usmerjevalniku A omejimo pretok nekoliko pod pričakovano spodnjo vrednost, v našem primeru 10 Mbit/s, s postopki intenzivnega razvrščanja, na primer s puščajočim vedrom ali z vedrom in žetoni. S tem »preselimo« čakalno vrsto paketov iz omrežnega elementa X v usmerjevalnik A. V usmerjevalniku A lahko potem brez težav pred odpošiljanjem nad čakajočimi paketi izvajamo razvrščanje po prioritetah.

Razvrščanja paketov z neintenzivnimi postopki se izvajajo samo na mestih, kjer so zgostitve in samo tam je smiselna uporaba teh postopkov. Zgostitve lahko v nekaterih primerih tudi prestavljamo z intenzivnimi postopki razvrščanja omrežnih paketov.

Z različnimi postopki razvrščanja paketov na različne načine delimo omejene omrežne vire sočasnim komunikacijskim tokom. Od omrežnih virov je najpogosteje v ospredju zagotavljanje prepustnosti. Vzrok za privilegirano obravnavo prepustnosti je več. Zgodovinsko gledano je bila omrežna prepustnost v nekdaj šibkih paketnih omrežij res najpomembnejši vir za tradicionalne storitve prenosa podatkov, kot so prenosi datotek in elektronske pošte. Ostale lastnosti so bile drugotnega pomena. Prepustnost je tudi vir, katerega je mogoče enostavno in neodvisno upravljati na vsakem omrežnem elementu posebej. Tudi pri rezervacijskih shemah, kadar s pomočjo signalizacijskih protokolov neki storitvi zagotavljamo prepustnost na celotni prenosni poti, to v končni fazi še vedno nastavlja vsak omrežni element posebej.

Pri zakasnitvah je drugače. Če želimo paketom komunikacijskega toka zagotoviti

zgornjo mejo zakasnitev, ki je ne smejo preseči, si morajo omrežni elementi skupno zakasnitev med sabo dogovorno razdeliti. Če je v enem vozlišču zakasnitev večja od pričakovane, morajo ostala zato »malo pohiteti«, da na koncu skupna zakasnitev ostane v dovoljenih mejah. Morda ne bi bilo napak, da bi skupno zakasnitev kot največjo dovoljeno kvoto vgradili kar v komunikacijski protokol, od katere bi si vsako vozlišče odškrtnilo svoj košček pri prehodu skozenj, podobno kot imajo nekateri protokoli polje TTL (Time to Live) ali hop limit, le da bi bilo dejansko v časovnih enotah. Vsak mrežni element bi moral za vsak paket meriti čas od prispetja in umeščanja v čakalno vrsto do odpošiljanja ter to vrednost odšteti od kvote. Če bi kvote zmanjkalo, bi se paket zavrgel in poleg tega s signalizacijskim protokolom o tem obvestil ostale predhodne elemente na prenosni poti, da bi v bodoče poizkušali pakete tega komunikacijskega toka obdelati nekoliko hitreje. Verjetno bi bilo treba zahteve po manjših zakasnitvah vezati na ceno prenosa, da ne bi prihajalo do zlorab in bi uporabniki prometu vseh vrst zahtevali najkrajše zakasnitve.

Celo razvrščanja, ki so namenjena tudi upravljanju z zakasnitvami, na primer hierarhične pravične prenosne krivulje – HFSC, to največkrat dosegajo posredno prek povečanja prepustnosti – s strmejšo prenosno karakteristiko. Ob tem se vseeno lahko zgodi, da ima, kljub večji prepustnosti oziroma strmejši karakteristiki, komunikacijski tok z večjimi paketi večje zakasnitve od toka z manjšimi paketi. Pakete se namreč razvršča po končnih časih odpošiljanja v pripadajočem tokovnem modelu. Pri tem se lahko končni časi večjih paketov, ne glede na večji pretok, raztegnejo dlje v prihodnost od manjših paketov pri manjšem pretoku.

V zadnjem času odkrivajo možnost upravljanja kakovosti storitev predvsem strokovnjaki drugih strok. To je prodajno in računovodsko osebje komunikacijskih podjetij, ki opažajo upad prihodkov pri ponudbi klasičnih komunikacijskih storitev zaradi selitve uporabnikov k ceneni ali celo skoraj brezplačni uporabi paketnih omrežij. Zato so se že pojavile težnje po oviranju ali vsaj zmanjševanju prioritete komunikacijskega prometa določene vrste, na primer internetne telefonije in ponujanje boljših storitev za dodatno plačilo. Lahko gre celo oviranje konkurence, če sami ponujajo enako, vendar plačljivo storitev. Zadeve so prišle celo tako daleč, da so nekatere države na pritiske skupnosti uporabnikov že začele razmišljati o zakonski ureditvi, ki bi zagotavljala enakopravnost obravnave vseh komunikacijskih tokov, kar v resnici ni dobro za vse vrste storitev iz povsem tehničnih razlogov.

Na drugi strani tudi uporabniki v novejšem času uporabljajo čedalje več aplikacij, ki so omrežno zelo agresivne in izrazito neprijazne do pravičnega razvrščanja (P2P storitve, segmentiran prenos datotek, ...). Te aplikacije za svoje potrebe uporabijo več sočasnih vzporednih komunikacijskih tokov. Pri običajnih postopkih razvrščanja obravnavamo take toke neodvisno, zato dobi posamezna aplikacija več zagotovljene prepustnosti kot znaša njen pravični delež.

Viri:

[1] R. Jain, D. Chiu, and W. Hawe; *A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems, DEC Research Report TR–301*, September 1984, http://www.cse.wustl.edu/~jain/papers/fairness.htm (7. 1. 2012)

[2] J. Virtamo; *38.3141 Teletraffic Theory / Fairness*; http://www.netlab.tkk.fi/opetus/s383141/kalvot/E_fairness.pdf (7. 1. 2012)

[3] Vesa Timonen; *Static Fairness Criteria in Telecommunications*, November 2002, http://www.sal.tkk.fi/publications/pdf-files/etim02.pdf (7. 1. 2012)

[4] Jean-Yves Le Boudec; *SSC-D02 Congestion Control*; http://ica1www.epfl.ch/PS_files/ccC.pdf (7. 1. 2012)

[5] Viktoria Fodor; *2E1624 – Performance analysis of communication networks*; http://www.s3.kth.se/lcn/courses/2E1624/lecture7.pdf (7. 1. 2012)

[6] Zheng Wang; *Internet QoS : architectures and mechanisms for Quality of Service*; Morgan Kaufmann Publishers, 2001, ISBN 1–55860–608–4

[7] Sanjay Jha, Mahbub Hassan; *Engineering Internet QoS*; Artech House, Inc., 2002, ISBN 1–58053–341–8

[8] Ion Stoica, Hui Zhang, T. S. Eugene Ng; *A Hierarchical Fair Service Curve Algorithm for LinkSharing, RealTime and Priority Services*; http://trash.net/~kaber/hfsc/SIGCOM97.pdf (7. 1. 2012)

[9] Klaus Rechert, Patrick McHardy; *HFSC Scheduling with Linux*; http://linux-ip.net/articles/hfsc.en/ (7. 1. 2012)

[10] Injong Rhee, Lisong Xu; *CUBIC: A New TCP-Friendly High-Speed TCP Variant*; http://www4.ncsu.edu/~rhee/export/bitcp/cubic-paper.pdf (7. 1. 2012)

[11] Martin Devera; *HTB home*; http://luxik.cdi.cz/~devik/qos/htb/ (7. 1. 2012)

[12] B. Hubert, T. Graff, G. Maxwell, R. Mook, M. Oosterhout, P. B. Schroeder, J. Spaans, P. Larroy; *Linux Advanced Routing & Traffic Control HOWTO*; http://lartc.org/lartc.html (7. 1. 2012)

[13] Lucian Gheorghe; *Designing and Implementing Linux Firewalls and QoS using*

netfilter, iproute2, NAT, and L7-filter; Packt Publishing Ltd., 2006, ISBN 1–904811–65–5

[14] Kurt Wagner; *Short Evaluation of Linux's Token-Bucket-Filter (TBF) Queuing Discipline*, May 2001, http://www.docum.org/docum.org/docs/other/tbf02_kw.ps (12. 1. 2012)

[15] The Linux Fundation; *Netem*; http://www.linuxfoundation.org/collaborate/workgroups/networking/netem (7. 1. 2012)

Dodatek A

Linux in razvrščanje omrežnega prometa – povzetek »man« strani

tc qdisc, tc class in tc filter

Za razvrščanje mrežnega prometa na linux operacijskem sistemu se uporablja ukaz tc, ki je del paketa iproute2. Paket iproute2 je danes privzeto nameščen praktično na vseh linux distribucijah. Ves mrežni del kode je že vgrajen v samo jedro operacijskega sistema. Ukaz tc je le zunanje orodje za upravljanje.

Nastavitve potekajo na treh nivojih v naslednjem vrstnem redu:

- tc qdisc – pripne izbrano disciplino na mrežni krmilnik;
- tc class – glede na vrsto discipline zgradi hierarhično strukturo razredov, v katere se razvrščajo različni tipi mrežnega prometa;
- tc filter – glede na značilnosti prometa in nastavljene parametre umešča pakete v ustrezni razred.

Sintaksa

```
tc qdisc  [add|change|replace|link] dev DEV [parent qdisc-id|root]
   [handle qdisc-id] qdisc [qdisc specific parameters]


tc class  [add|change|replace] dev DEV parent qdisc-id [classid class-
id]
   qdisc [qdisc specific parameters]


tc filter [add|change|replace] dev DEV [parent qdisc-id|root]
   protocol protocol prio priority filtertype
   [filtertype specific parameters] flowid flow-id


tc [FORMAT] qdisc show [dev DEV]


tc [FORMAT] class show dev DEV
```

```
tc filter show dev DEV

FORMAT := {-s[tatistics]|-d[etails]|-r[aw]|-p[retty]|i[ec]}
```

Disciplina vrste (QDISC)

Poznavanje disciplin čakalnih vrst je osnova za razumevanje poteka mrežnega prometa. Ko linuxovo jedro želi odposlati paket skozi mrežni krmilnik, ga najprej uvrsti v čakalno vrsto v skladu z nastavljeno disciplino. Pravila discipline vrste potem določajo, kako in kdaj se posamezni paket odpošlje naprej v omrežje.

Najenostavnejša disciplina je »pfifo«, ki ne omogoča nikakršne obdelave, ampak enostavno odpošilja pakete v prihajajočem vrstnem redu.

Razredi

Nekatere discipline so več nivojske (hierarhične) in lahko vsebujejo razrede, ki potem lahko v nadaljevanju spet vsebujejo discipline. Ko linuxovo jedro želi odposlati paket, lahko ta pride iz kateregakoli razreda glede na nastavljena pravila.

Filtri

Filtri se uporabljajo za klasificiranje, to je razvrščanje prometa v različne razrede. Če imamo nastavljenih več razredov, moramo prispeli omrežni promet vanje ustrezno razvrstiti. V ta namen imamo na voljo več metod, ena od njih so filtri. Pri razvrščanju paketa se po vrsti uporabijo nastavljeni filtri, dokler en od njih ne razvrsti paketa. Če noben od filtrov paketa ne razvrsti, se lahko uporabijo še drugi kriteriji, ki so odvisni od posamezne discipline čakalne vrste – obstajajo lahko privzeti razredi, lahko se promet tudi enostavno zavrže.

Enostavne discipline (brez razredov)

Primeri enostavnih disciplin so:

- [p|b]fifo
  Najenostavnejša disciplina, ki posreduje pakete v prihajajočem vrstnem redu. Velikost je omejena v p-paketih ali b-oktetih.
- pfifo_fast
  Privzeta disciplina v linuxovih jedrih. Vsebuje tri čakalne vrste, v katere se paketi

razvrščajo po vnaprej določenih prioritetah glede na TOS polje v IP glavi paketa.

- red
  Naključno zgodnje odmetavanje umetno simulira obremenjeno omrežje, ki naključno odmetava pakete, ko se pretok približuje nastavljeni hitrosti. Enostaven algoritem in zato primeren za zelo hitra omrežja.
- sfq
  Stohastično pravično razvrščanje. Način krožnega razvrščanja, ki razvrsti komunikacijske toke v ločene čakalne vrste, iz katerih potem krožno odpošilja pakete.
- tbf
  Vedro z žetoni. Primeren za umirjanje prometa na nastavljeno hitrost. Zelo primeren za hitra omrežja.

Nastavitev enostavnih disciplin

Če ne uporabljamo disciplin z razredi, lahko enostavne discipline vključimo le neposredno na mrežne krmilnike. Sintaksa je naslednja:

```
tc qdisc add dev DEV root QDISC QDISC-PARAMETERS
```

To je tudi vse, kar je možno nastaviti, saj enostavne discipline ne vsebujejo razredov in potem ne rabimo niti filtrov za razvrščanje v razrede. Če mrežnim krmilnikom ne priredimo nobene discipline, je avtomatično privzeta pfifo_fast, ki je enostavnejša različica (brez razredov) prio discipline.

Hierarhične discipline z razredi

Primeri disciplin z razredi so:

- CBQ (Class Based Queuing)
  CBQ vsebuje bogato hierarhijo razredov za nastavitve deležev prepustnosti mrežne linije. Oblikovanje se izvaja z izračunavanjem deleža časa neaktivnosti linije z upoštevanjem povprečne velikosti paketov in kapaciteto izhodnega kanala.
- HTB (Hierarhical Token Bucket)
  HTB zagotavlja deleže prepustnosti razredom in hkrati omogoča podrobno določitev delitve tudi znotraj razredov. HTB vsebuje elemente oblikovanja prometa, ki temeljijo

na principu vedra z žetoni.

- PRIO
  PRIO disciplino sestavlja skupina razredov, iz katerih se jemlje pakete po prioritetnem vrstnem redu. To omogoča enostavno prioritiziranje prometa, kjer nižje prioritetni razredi odpošiljajo pakete samo takrat, ko višje prioritetni ne vsebujejo čakajočih paketov. Pri enostavnejši, privzeti nastavitvi je nastavljeno, da se za razvrščanje upošteva kar TOS polje v IP glavi paketov.
- HFSC (Hierarchical Fair Service Curve)
  HFSC disciplina ločuje lastnost prepustnosti od zakasnitev. Razredom lahko zagotovimo deleže prepustnosti in hkrati neodvisno tudi zakasnitve, ki so jih deleženi pripadajoči paketi.
- DRR (Deficit Round Robin)
  DRR pravično deli pretok med posamezne razrede. Pri tem upošteva tudi različne velikosti paketov. Vsakemu razredu lahko priredimo uteži, ki določajo razmerje delitve. Princip delovanja temelji na krožnem procesu s primanjkljajem.

Delovanje

Discipline imajo lahko nič ali več podrazredov, katerim pošiljajo promet. Ko paket vstopi v hierarhično disciplino z razredi, se ta s pomočjo filtrov klasificira in pošlje v enega od razredov.

Razredi tvorijo drevesno strukturo, kjer ima vsak razred natančno en nadrejen razred, lahko pa več podrejenih. Vsak razred ima prirejene disciplini ustrezne mrežne parametre. Razredi na skrajno spodnjem nivoju vsebujejo izhodne discipline, ki so privzeto pfifo, in jih je mogoče zamenjati z drugimi. Vsak spodnji razred lahko vsebuje samo eno izhodno disciplino. Tudi ta je lahko hierarhična in lahko spet vsebuje razrede.

Poimenovanje

Discipline in razredi imajo sklicevalne oznake, ki jih lahko dodelimo ali dobijo dodeljene samodejno. S sklicevanjem na oznake (podajamo oznako nadrejene komponente) povezujemo komponente skupaj ali vanje razvrščamo promet.

- OZNAKA
  je sestavljena iz glavnega in pomožnega števila, ki ju ločuje znak »:« (dvopičje).
- DISCIPLINA VRSTE

dobi dodeljeno le glavno število, ki ga imenujemo vzvod. Pomožno število se dodeljuje vsebovanim razredom. Primer vzvoda je lahko oznaka »10:«. Izbrane številke oznak so lahko poljubno izbrane.

- RAZREDI
  znotraj posamezne discipline od nje podedujejo glavno število, vsakemu potem izberemo svoje pomožno število, ki ga imenujemo znak razreda in ni direktno izpeljan ali povezan z nadrejenim razredom, ampak le iz nadrejene discipline.

Več o uporabi skupaj s praktičnimi primeri lahko najdemo v [11], [12], [13] in v »man« straneh na operacijskem sistemu linux.