S-38.201 ATM- JA MULTIMEDIASEMINAARI, KEVÄT -97

WFQ-jonotus ATM-verkon liikenteenhallinnassa

Sampsamatti Tanner
S 41820b

Sampsamatti.Tanner@hut.fi

Tiivistelmä

Tässä työssä on tutustuttu lyhyesti erilaisiin yleisnimellä Weighted Fair Queueing (painotettu reilu jonotus) tunnettuihin algoritmihin ja niiden ominaisuuksiin ATM-verkon liikenteenhallinnan näkökulmasta.
Alussa on yleisesti nimetty eräitä tunnetuimpia algoritmeja, tämän jälkeen on kuvattu näistä neljän: GPS:n, PGPS:n, Virtual Clockin ja Virtual Spacingin toimintaa hieman tarkemmin Lopussa on myös hieman pohdintaa algorimien tuomista eduista ja käytöstä.

1. Johdanto

Tietoverkot ja niissä käytetyt protokollat ovat perinteisesti olleet hyvin haavoittuvia, varsinkin jos verkon käyttäjissä on 'häirikkökäyttäjiä'. Esimerkiksi ATM-verkossa joku käyttäjä esimerkiksi päätelaitteen rikkouteessa saattaa lähettää dataa nopeammassa tahdissa (eli käyttää enemmän siirtokapasiteettia) kuin mihin hän on oikeutettu. Tällöin hän häiritsee kohtuuttomasti muiden verkonkäyttäjien saamaa palvelua, ellei tälläisiin tilanteisiin ole varauduttu.

Eräs ongelmatilanne syntyy, jos useammat käyttäjät lähettävät tietoa verkkoon yhteisen palvelimen kautta ja palvelimella on yksi yhteinen puskuri jota hallitaan FIFO-tyyppisellä algoritmilla. Näiden ongelmien vähentämiseksi on kehitetty erilaisia ns. painotettuja reiluja jonotusalgoritmeja. Tässä työssä käsitellään näistä eräitä.

Painotetuilla reiluilla jonotus –algoritmeilla on kaksi etua:

a) Ne takaavat kaistanleveyden reilun jakautumisen käyttäjien kesken
b) Ne takaavat rajoitetun päästä-päähän -viiveen istunnoille, joita valvotaan vuotavilla ämpäreillä

2. Reilu jonotus - algoritmeja

2.1. Reiluus

Ajateltaesssa siirtomediaa, jolla on rajoittu siirtokapasiteetti ja useampia käyttäjiä, tarkoittaa reiluus sitä, että kullakin käyttäjällä on (keskimäärin) käytettävissä se määrä siirtokapasiteettia niillä laatuparametreilla, joihin kukin on oikeuttu esim. liikennöintisopimuksen peruteella.

2.2. Painotettu reilu jonotus - Weighted Fair Queueing (WFQ)

Tässä työssä termiä WFQ käytetään yleisnimenä tarkoittamaan useita erilaisia painotuksilla varustettuja jonotus-algoritmeja, ei vain Demersin, Keshavin ja Shenkerin vuonna 1989 tällä nimellä esittelemää algoritmiä (käytetään nykyään yleisesti nimeä PGPS - käsitellään myöhemmin).

2.2.1. Generalized Processor Sharing (GPS)

Jos prosessorin (palvelimen/kytkimen) suorituskapasiteettia voitaisiin jakaa juoksevasti (ei-diskreetisti) eri käyttäjien kesken, jakaisi reilu jonotus sen tasan kaikkien niiden käyttäjien kesken, joilla on dataa lähetettävänään. Yleisestä prosessorin jakamisesta (GPS) puhutaan silloin kun eri lähteistä tulevilla tietovirroilla on painot ja prosessorikapasiteetti jaetaan näiden painojen osoittamassa suhteesssa /1/. Todellisuudessa ATM-kytkin käsittelee ATM-paketteja, eikä suinkaan bittivirtoja, joten GPS-algoritmia ei ole mahdollista toteuttaa, vaan käytetään algoritmeja, jotka mahdollisimman hyvin vastaavat sitä. Kts. 3.1

2.2.2. Packet-by-packet Generalized Processor Sharing (PGPS)

PGPS:ää pidetään yleisesti GPS:n parhaana approksimaationa pakettiverkoissa. Se tunnettiin alunperin nimellä Weighted Fair Queueing (kts. edellä). Ensimmäisenä nimeä PGPS käyttivät Parekh ja Gallager /2/. Algoritmin implementointi on kuitenkin sen verran monimutkaista, että yksinkertaisempia algoritmeja, joilla päästään lähes yhtä hyvään tulokseen on kehitetty. Kts 3.2

2.2.3. Frame-Based Fair Queueing /3/

Algoritmi, joka ainakin kehittäjiensä mukaan tarjoaa samat palvelut kuin PGPS, kuten päästä-päähän –viive, ilman että pitäisi suoritaa monimutkaista juoksevan prosessin mallinnusta. FBFQ on reilu siinä mielessä, että siinä ei rangaista lähdettä sen käyttämästä ‘ylimääräisestä’ kaistasta muiden lähteiden ollassa mykkinä. FBFQ kuuluu järjestelyalgoritmien yleiseen luokkaan, jota kutsutaan nimellä Suhteellisen nopeuden palvelimet (Rate-Proportional Servers)

2.2.4. Self Clocked Fair Queueinq (SCFQ)

Eräs PGPS:n muunnos. Ensimmäisen kerran sen esitteli Golestani v. 1994. Eräs toteutus nimeltään Virtual Spacing (Roberts 1994). Kts. 3.4

2.2.5. Virtual Clock

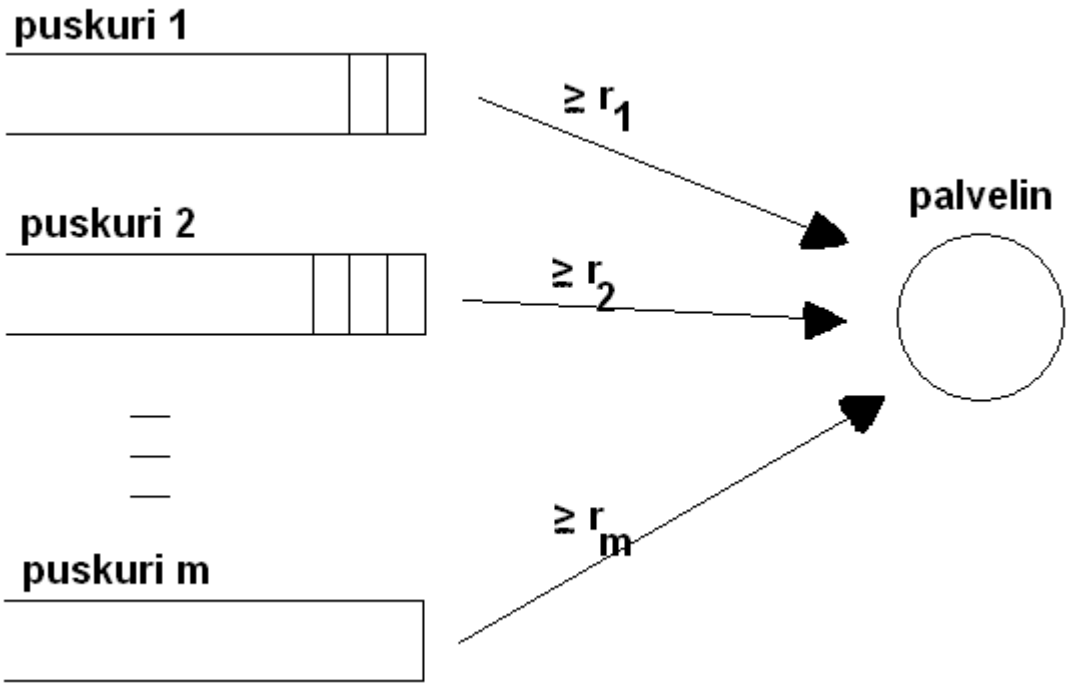
Myös eräs PGPS:n muunnos. Ensiesitys: Zhang 1990. Kts. 3.3.

2.2.6. Muita ‘reiluja’ algoritmeja

- Delay-Erliest-Due-Date (Delay-EDD)
- Weighted Round Robin
- Deficit Round Robin
- Hierarcical-Round-Robin (HRR)
- Stop-and-Go queueing
- Jitter-Earliest-Due-Date

3. Algoritmien toimintakuvaus

Kuvitellaan tilanne, jossa palvelin pystyy käsittelemään kiinteän määrän *c* paketteja aikayksikössä. palvelimeen tulee *m* kappaletta liikennevirtoja, joilla jokaisella on oma jononsa, joita hallitaan FIFO (first in - first out) - algoritmilla. Kuva 1 esitää tätä tilannetta. Jokaisen liikkennevirran voi ajatella kuvaavan tiettyä ATM-lähdettä, voidaan myöskin ajatella, että jotkut virroista saattaa kuvata joukkoja, jossa on liitetty useampi yhteys liikenteenksittelyä varten yhteen.



kuva 1

3.1 GPS

Generalized Processor Sharing on joksevava approksimaatio kuvan 1 jonotusjärjestelmälle, jossa palveluaste jokaiselle backlogged virralle *i* on suoraan verrainnollinen parametriin ϕ_i. Tässä esimerkissä jokaiselle tietovirralle *i*, ϕ_i on asetettu samaksi kuin virran luontainen solunopeus r_i ja toisaalta on myös voimassa ehto $\sum r_i \leq c$. Toisin sanoen palvelimen kapasiteetti on suurempi kuin virtojen yhteenlaskettu solunopeus. Parametrilla r_i tarkoitetaan täten minimikaistanleveyttä, joka yhteydelle *i* taataan. Tämä voi käytännössä tarkoittaa vakiokaistanleveyttä, joka on osoitettu esimerkiksi CBR-yhteydelle tai ns. vuotavan ympärin (kts. liite 1) läpi tulevalle, ylhäältä rajoitetulle, liikenteelle.

3.2 PGPS

Koska paketit (ATM-solut) eivät ole jaettavissa yksittäisiksi biteiksi- tai oikeammin niitä ei ole järkevä jakaa – on GPS:stä kehitetty algoritmeja, jotka käsittelevät kokonaisia paketteja. Näistä Packet-by-packet Generalized Fair Queueing (PGPS) suorituskyvyltään on lähinpänä ‘optimitilannetta’ eli varsinaista GPS:ää.

PGPS:ssä ideana on palvella (vaihtelevan pituisia) paketteja sellaisessa järjestyksessä - mahdollisuuksien mukaan - että ne tulevat palvelimelta ulos samassa järjestyksessä kuin ne tulisivat GPS:ää käytettäessä. Käytännössä tämä järjestys voidaan määrittää vain paketeille, jotka ovat paikalla kun edellisen paketin käsittely lopetetaan ja uusi paketti otetaan palveltavaksi: Palvelin valitsee aina paketin, joka läpäisisi GPS:n ensimmäisenä (käsittelyn aikana saapuvia paketteja ei huomioida). ATM-soluille on osoitettu että erotus PGPS:n ja GPS:n siirtoajoissa annetussa saapumisprosessissa on korkeintaan yhden solun siirtoaika ($1/c$).
Jotta voitaisiin laskea hypoteettinen päättymisaika (finishing time) – aika, jossa paketti tulisi ulos GPS:stä - on otettu käyttöön muuttuja virtuaaliaika (Virtual time). Tämän avulla PGPS huolehtii pakettien ajoittamisesta. Virtuaaliaika on oikean ajan t funktio, joka kasvaa kääntäen verrainnollisesti vastaavan GPS järjestelmän ruuhkaisten virtojen - virtojen, joilla on soluja jonossa - solunopeuksien r_i summaan verrattuna.

Merkitään $PGPS_i$:llä tietovirtaan i liittyvää muuttujaa, aikaleimaa. Kun PGPS:ää sovelletaan ATM-verkkoon, voidaan PGPS-palvelualgoritmiä kuvailla seuraavasti:

- Kun virran i solu saapuu

  (i) $PGPS_i \leftarrow \max\{Virtuaaliaika, PGPS_i\} + 1/r_i$

  (ii) Aikaleimataan solu $PGPS_i$:n arvolla

- Palvellaan soluja aikaleimojen kasvavassa järjestyksessä eli pienimmän PGPS:n omaava saa ensin palvelua.

On osoitettu /2/, että PGPS takaa tietyn minimisolunopeuden, sillä rajoituksella, että annetun virran jono missä tahansa aikavälissä poikkeaa korkeintaan yhden solun verran vastaavan GPS-järjestelmän vastaavasta arvosta. Raskaan PGPS-algoritmistä tekee virtuaaliajan laskeminen, sillä sen pyörittää muistissaan virtuaalista GPS-järjestelmää. Toisaalta juuri tämän ansiosta PGPS on niin hyvä approksimaatio GPS:lle.

3.3. Virtuaalikello (Virtual Clock)

Virtuaalikello on läheistä sukua PGPS:lle. Erona on lähinnä se että VC:ssä käytetään reaaliaikaa PGPS:n käyttämän virtuaaliajan asemesta.

Tässä algoritmissa tapahtumien kulku on seuraava (Nyt aikaleimaa merkitään VC:llä):

- Kun virran i solu saapuu

  (i) $VC_i \leftarrow \max\{t, VC_i\} + 1/r_i$

  (ii) Aikaleimataan solu VC_i:n arvolla

- Palvellaan soluja aikaleimojen kasvavassa järjestyksessä.

VC säätää työtä huomattavasti verrattuna PGPS:ään ja takaa silti keskimääräisen suoritustehon (troughput - läpäisyn) jokaiselle kytkennälle.

3.3.1. Esimerkki VC vs. PGPS

Lähde 1 lähettää viestin, jonka koko on 1 Mbit multiplekserille, jonka kapasiteetti c=100 Mbit/s; vaikka virralla on solunopeusparametri r_1 = 1 Mbit/s, sen siirto tapahtuu huippunopeudella, sen ollessa ainoa virta, joka lähettää kyseisellä hetkellä. Mutta, kuinka ollakaan, juuri ennen virtan 1 viimeisen paketin lähetystä, tulee toiseen virtaa kuuluva lähde aktiiviseksi ja alkaa lähettää dataa 100 Mbit/s

nopeudella. Ensimmäisen virran viimeisellä solulla on aikaleima, joka vastaa suurinpiirtein nykyistä ajanhetkeä *t* lisättynä yhdellä sekunnilla $\left(\frac{1Mbit}{1Mbit \ / \ s} \right)$. Olettaen että myös toisella virralla on samansuuruinen solunopeusparametri r_2 = 1 Mbit/s, viivästyy ensimmäisen virran viimeinen paketti sen ajan, että uuden virran soluista on kuljetettu noin 1 Mbit. Toisen virran kannalta on ihan sama olisiko ensimmäisen viimeinen paketti lähetetty ennen sitä vai sen solujen välissä, kuten kävi Virtuaalikelloa käytettäessä . PGPS olisi osannut lähettää ensimmäisen purskeen viimeisen solun ennen toisen purskeen välityksen aloittamista.

Yleisesti ottaen VC-algoritmi takaa käytettävissä olevan siirtonopeuden paljon huonommalla tarkuudella kuin PGPS. Kuten esimerkissä läpäisy voi olla huomattavasti r_i:aa heikompi merkitsevän pitkiäkin aikoja.

3.4. SCFQ/Virtuaalinen asettelu (Virtual Spacing)

Seuraavassa käsitellään virtaalista asettelua, joka on eräs SCFQ:n erikoistapaus. Virtuaalinen asettelu toimii samalla lailla kuin PGPS, paitsi että siinä käytetään yksinkertaisemmin laskettavaa asetteluaikaa (Spacing Time) virtuaaliajan sijasta. Asetteluaika on samansuuruinen kuin se aikaleima, joka oli solulla, jonka palvelin viimeksi käsitteli. Ruuhka-aikana tämä tarkoittaa parhaillaan käsiteltävän solun aikaleimaa.

Algoritmi; VS_i kuvaa virtaan *i* liittyvää aikaleimaa:

- Virran *i* solun saapuessa

  (i) $VS_i \leftarrow \max\{Asetteluaika, VS_i\} + 1/r_i$

  (ii) Aikaleimataan solu VS_i:n arvolla

- Palvellaan soluja nousevassa järjestyksessä

Koska asetteluaika ei voi olla suurempi kuin minkään solun leimaushetkellä jonossa olevan solun aikaleima, algoritmin askeleesta (i) voi päätellä, että ruuhkaisessa virrassa olevien solujen aikaleimat ovat aritmeettisesti jakautuneita intervallilla $1/r_i$. Asetteluaika puutuu saapuvan solun aikaleiman laskentaan ainoastaan solun saapuesssa ruuhkattomaan virtaan. Tällöin solu laitetaan sen avulla asiaankuuluvaan kohtaan siirtoaikataulussa.

4. Suorituskyky

Eri algoritmien suorituskyvyn, kuten viiveen raja-arvojen ja reiluuden, rajat on helpointa määritellä johtamalla ne idealisoidun GPS algoritmin ominaisuuksista.

4.1 Vuotavalla ämpärillä valvotut virrat

Oletetaan että yhteyttä *i* valvotaan vuotavalla ämpärillä, joka päästää liikennettä lävitseen solunopeudella r_i, ja jonka valtuusaltaan koko on b_i. Saapuva työ aikavälissä (s,t) on tällöin:

$$v_i(s,t) \leq b_i + r_i(t-s)$$

Merkitään V_{it}:llä työmäärää, joka liittyy yhteyteen *i* multiplekserin jonossa ajassa *t* (jonossa olevien solujen määrä sekä loppuosa soluista, joita parhaillaan liikutetaan). Koska palvelunopeus ei ole pienempi kuin r_i, saadaan Reich'in teoreema:

$$\begin{aligned} V_{it} &\le \sup_{s<t}\{v_i(s,t) - r_i(s-t)\} \\ &\le b_i . \end{aligned}$$

Mertkitään aikavälissä (t,u) yhteyteen *i* liittyvää työn määrää, joka poistuu multiplekseriltä $\xi(t,u)$. Tämä koostuu kolmesta osasta: solun lopusta, joka on jäljellä hetkellä, soluista, jotka saadaan kokonaan toimitettua ajassa (t,u) sekä solun siitä osasta, joka on keritty toimittaa ennen hetkeä *u*. Eli:

$$\xi(t,u) \le V_{it} + v_i(t,u)$$

josta saadaan:

$$\begin{aligned} \xi(t,u) &\le \sup_{s<t}\{v_i(s,t) - r_i(t-s)\} + v_i(t,u) \\ &= \sup_{s<t}\{v_i(s,u) - r_i(t-s)\} \\ &\le \sup_{s<t}\{r_i(u-s) + b_i - r_i(t-s)\} \\ &= r_i(u-t) + b_i . \end{aligned}$$

Viimeinen epäyhtälö osoitta, että multiplekserin ulostulo säilyttää vuotavan ämpärin takaaman purskeisuus raja-arvon. Tämä on erittäin haluttu ominaisuus, sillä se takaa, että jos kaikki multiplekserit **tietyn yhteyden** polulla takaavat minimi palvelunopeuden r_i, niin **solujen hukkuminen voidaan kokonaan** välttää valitsemalla puskurin koko samaksi kuin b_i. Lisäksi minkä tahansa solun viive on rajattu arvolla b_i / r_i jokaisessa multiplekserissä riippumatta muista yhteksistä. On jopa osoitettu, jos jätetään huomiotta vaikio prosessointi- ja kulkuaikaviiveet, että **viive koko tälläisen GPS-palvelinten verkon yli on rajoitettu b_i / r_i:llä.**

4.2. Päästä-päähän –viive

Kokonaisviiveraja b_i / r_i pätee vain GPS sovelluksille juoksevassa järjestelmässä, edellyttäen "cut through" kytkentää jokaisessa solmussa. Tämä tarkoittaa, että jokainen kytkin lähettää solun eteenpäin jo ennekuin se on kokonaan saapunut. Kuitenkin solmut toimivat yleensä store-and-forward –periaatteella (solun pitää olla kokonaan kytkimessä ennen eteenpäinlähettämistä) joten kokonaisviiveeseen pitää laskea mukaan myös nämä viiveet. Viive $D_i^{GPS}(K)$ K-vaiheisen verkon yli on täten:

$$D_i^{GPS}(K) \le \frac{b_i}{r_i} + \frac{K-1}{r_i}$$

ATM-solujen diskreetti luonne tuo palvelua kuvaavien muuttujien, kuten yllä kuvattuun viiveen, rajoihin ylimääräistä löyhyyttä. PGPS:lle saadaan viiveeksi:

$$D_i^{PGPS}(K) \le \frac{b_i}{r_i} + \frac{K-1}{r_i} + \sum_{k=1}^{K}\frac{1}{c_k}$$

missä c_k on linkin nopeus tasolla *k*. Yleensä c_k on huomattavasti suurempi kuin r_i, joten ero ero kahden edellisen GPS:n ja PGPS:n viiveillä ei ole kovin suuri.

Vastaava viive $D_i^{VS}(K)$ virtuaaliselle asettelulle:

$$D_i^{VS}(K) \leq \frac{b_i}{r_i} + \frac{K}{r_i} + \sum_{k=1}^{K} \frac{m_k - 1}{c_k}$$

missä m_k multipleksattujen virtojen määrä tasolla k. Jos kaikilla virroilla on sama nopeusparametri tulee summalausekkeen arvosta jälkimmäisessä lausekkeessa kutakuinkin sama kuin edellisessä.

4.2.1 Viiverajojen merkitys

Päästä-päähän -viiveiden merkitys yhteydellä riippuu suuresti yhteydelle vaadittavasta QoS:stä (Quality of Service) ja on luonnolisesti aivan eri reaaliaikaisille palveluille kuin muille. Jos yhteydeltä vaaditaan reaaliaikaisuutta, täytyy päästä-päähän -viiveen olla pieni ja lisäksi viiveenvaihtelun (jitter) olla tunnettu. Jälkimmäinen on tärkeä tieto mitoitettaessa vastaanottopään puskurin kokoa.

4.2.2. Vastaanottopuskurin mitoittaminen

Oletetaan, että vastaanottopurkuri toimii välikappaleena, joka lähetää soluja minimiaikavälillä $1/r_i$. Soluja ei hukata, jos puskuri on isompi kuin $r_i D_{max}$, missä D_{max} on WFQ-järjestelmän päästä-päähän -viiveen raja. Kun soluja ei hukata, solujen kokema kokonaisviive verkossa ja puskurissa on funktio solun järjestysnumerosta ja saavuttaa maksimiarvon, joka on pienempi, tai yhtäsuuri, kuin D_{max}.

Otaksutaan, esimerkkiä varten, että kaikki palvelimeen tulevat virrat ovat itsenäisiä 64 kbit/s CBR (vakio bittinopeus) yhteyksiä (CDV luokkaa b_i =2) ja että kaikki linkin kaistanleveys on käytössä (eli: Σ r_i = c). GPS:ssä jok'ikisen solun siirtoaika on tasan 6ms. PGPS ja VS toisaalta käyttäytyvät enemmän tavallisen FIFO-jonon tapaan: solun siirtoaika on yhtäkuin yksi palveluaika plus satunnainen jonotuksesta johtuva viive. Edellisessä kappaleessa estetyt viiveet olivat huonoimman vaihtoehdon mukaan laskettuja ja perustuivat viimeisen palvellun solun viiveeseen siinä tapauksessa, että kaikki yhteydet lähettävät soluja täsmälleen samalla hetkellä. Satunnaisen solun todellinen viive on yleensä huomattavasti pienempi, jolloin puskurin voi mitoittaa pienemmäksi. Näin on etenkin silloin kun multipleksataan virtoja, joiden purskeisuus on pientä (eli b_i:n arvo on pieni). Tarkastellaan seuraavaksi purskeisen liikenteen vaikutusta reaaliaikaisten yhteyksien viiveenvaihteluun.

CBR-virran solut saapuvat yleeensä tyhjiin puskureihin, sillä niiden viive on pieni. Ne saavat aikaleimakseen siis suoraan virtuaaliajan (tai asetteluajan jne.). Ruuhkaisten yhteyksien solujen aikaleimat sitä vastoin on jaettu tasasisesti kuhunkin yhteyteen liittyvän nopeusparametrin mukaan. nyt ajatellaan, että kaikilla virroilla on sama solunopeusparametri r mutta osa on purskeisia toisten ollessa CBR-yhteyksiä. Käytetään Virtuaalista asettelua. Kun CBR-solu saapuu se aikaleimataan arvolla *asetteluaika* $+1/r$. Nyt minkään ruuhkaisen (purskeisen) yhteyden seuraavana siirtovuorossa olevalla solulla ei voi olla suurempaa arvoa kun solulla, vaan se on todennäköisesti pienempi (niiden aikaleimat ovat määräytyneet edellisen kierroksen asetteluajan perusteella). Tilanteen ollessa näin, joutuu CBR-solu odottamaan yhden solun ajan jokaista ruuhkaista virtaa kohden. Näin se lisää CBR:n viivettä systemaattisesti ilman, että juuri hyödyttäisi ruuhkaisia virtoja.

Ratkaisuna on ruuhkattomien virtojen suosiminen virtuaalisen asettelun algoritmia muoivaamalla seuraavaan tapaan:

- Virran *i* solun saapuessa

  (i) $VS_i \leftarrow \max\{Asetteluaika, VS_i + 1/r_i\}$

  (ii) Aikaleimataan solu VS_i:n arvolla

- Palvellaan soluja nousevassa järjestyksessä

Samanlainen mukaelma sopii myös PGPS-algoritmille. Viiverajat pysyvät muutoksesta huolimatta ennallaan.
Koska viiverajat ovat hyvin löyhiä, kuvaa reiluus algoritmien ominaisuuksia usein paremmin.

4.3. Reiluus

Termi 'painotettu reilu jonotus' vihjaa käytettävän kaistanleveyden jakamiseen aktiivisten virtojen kesken, niiden solunopeusparametrien r_i määräämässä suhteessa. Tämä toteutuu täydellisesti vain teoreettisella GPS-algorimilla. Paketteihin perustuvissa algoritmeissa kaista voidaan jakaa vain pakettien siirtoajan määräämän 'rakeisuuden' tarkkuudella. Tässäreiluudella tarkoitetaan sitä, että mielivaltaisessa aikavälissä *(s,t]*, jossa virta *i* on ruuhkainen, on olemassa vakio *T(s,t)* siten, että virtaan *i* liittyvien palveltujen solujen määrä $\eta_i(s,t)$ ajassa *(s,t]* toteuttaa ehdot:

$$T(s,t)r_i - 1 < \eta(s,t) \le T(s,t)r_i + 1$$

Tästä huomataan, että jonosta otettujen solujen määrä on hyvin lähellä reilua osuutta kun aikaväli (s,t] kasvaa ($\eta_i(s,t) >> 1$). Tämä epäsuhta voidaan näyttää toteen virtuaaliselle asettelu -algoritmille seuraavasti.

4.3.1 Esimerkki virtuaaliselle asettelulle

Soluille, jotka saapuvat hetkellä *t* tai välittömästi ennen, annetaan aikaleima τ_t ja *s*:lle vastaavasti τ_s. Oletetaan että virta *i* on ruuhkainen aikavälillä *(s,t]* ja θ_i on ensimmäisen aikavälissä palvellun solun aikaleima. Koska virran *i* aikaleimat ovat jakautuneet $1/r_i$:llä, toteutuu:

$$\tau_s < \theta_i \le \tau_s + 1/r_i .$$

Koska aikaleimat virrassa *i* aikavälissä *(s,t]* ovat välttämättä pienempiä kuin τ_t (asetteluaika on ei-kasvava), saadaan:

$$\theta_i + (\eta_i(s,t) - 1)/r_i \le \tau_t < \theta_i + \eta_i(s,t)/r_i$$

Sijoittamalla tähän rajat θ_i:lle saadaan vakiolle $T(s,t) = \tau_t - \tau_s$.
PGPS:n reiluutta ei tässä osoiteta, mutta se seuraa sen läheisyydestä GPS:ään.

5. WFQ:n toteuttaminen ATM-kytkimessä

WFQ-algoritmien toteuttaminen ATM-kytkimissä on selvästi vaikeampaa kuin FIFO-algoritmien. Niiden tuoma hyöty - varsinkin isoissa kytkimissä - on kuitenkin sen verran iso, että algoritmi kannattaa toteuttaa.
Seuraavassa kahdessa esimerkissä on ajateltu lähtöpurkuroitua kytkinelementtiä, jossa palvelun ajoittaminen on toteutettu tarkoitukseen pyhitetyllä ohjaimella erikseen jokaista multipleksattua lähtöä kohden. Kytkimen rakenteen ei ajatella sinänsä vaikuttavan liikenteen ominaisarvoihin, kuten viiveeseen taikka solujen häviämiseen.

5.1 Jonon järjestely

Painotetussa reilussa jonotuksessa palvelin käsittelee soluja aikaleimojen määräämässä nousevassa järjestyksessä. Palvelimen pitää siis pystyä aina ottamaan käsittelyyn pienimmällä aikaleimalla

varustettu solu. Eräs mahdollisuus on laittaa solut sarjamuistiin nuosevaan järjestykseen; uudet solut laitetaan muistiin aina siihen väliin, johon ne aikaleiman perusteella kuuluvat. Toisin sanoen muistia joudutaan järjestämään tai ainakin tutkimaan jokaisen uuden solun saapuessa. Tälläiseen järjestelyyn tarkoitettuja lajittelufunktioita on kehittänyt mm. Chao. Vaihtoehtoinen malli edellä kuvatulle esitellään seuraavassa.

Koska palvelimen kannalta olennaista ei ole pitää kaikkia soluja 'aikajärjestyksessä', vaan saada aina pienimmällä aikaleimalla varustettu solu käsittelyyn, niin on turhaa pitää kaikkia soluja koko ajan täysin järjestyksessä. Seuraava algoritmi ajaa asian suorittamalla rinnakkaisia vertailu- ja vaihto-operaatioita viesteillä (jokaista solua kuvaa yksi viesti), jotka on järjestetty kateen m+1-kokoiseen yksiuloitteeseen taulukkoon, joita merkitään *A(j)* ja *B(j)*, $0 \le j \le m$. Jokainen sana taulukoissa *A* ja *B* on asettu joka maksimiarvoonsa (kaikki ykkösiä – muistipaikka tyhjä) tai vaihtoehtoisesti esittää viestiä ja sen aikaleimaa. Aikaleima varaa sanasta aina *k* vasemmanpuoleista bittiä lopun jäädessä viestille (tai yleensä osoittimelle, joka osoittaa solun sijainnin muistissa). Yksinkertaisuuden vuoksi oletetaan, että laskurin pituus on riittävä eli kello ei 'pyörähdä ympäri' ja että aikaleima yksiselitteisesti määrää viestien palvelujärjestyksen.
Aluksi kaikki sanat asetetaan oletusarvoonsa (bitit ykkösiksi). Lajittelualgoritmi on kaksiosainen; toisen avulla lisätään uusi solu muistiin, toinen on pienimmällä aikaleimalla varustetun solun poistoa varten.

1) Uuden solun lisäys:

- Viesti kirjoitetaan sanaan *A(0)*.
- Verrataan sanoja *A(j)* ja *B(j)*, $0 \le j \le m$ ja sisällöt vaihdetaan keskenään, jos *A(j)* < *B(j)*. Tästä seuraa että $B(j) \le A(j), 0 \le j \le m$.
- Siirretään $A(j) \leftarrow A(j-1), 1 \le j \le m$. (Tässä hukataan *A(m)* !)
- *A(0)* alustetaan uudelleen maksimiarvoon.

2) Solujen lähettäminen eteenpäin:

- Eteenpäin lähetettävä viesti on aina *B(0)*:ssa.
- A:n ja B:n vertailu ja vaihto kuten kodassa A
- Siirretään $B(j) \leftarrow B(j+1), 0 \le j \le m$.
- *B(m)* alustetaan

Algoritmi on ehdottu toteutettavaksi integroituna piirinä.

5.2. Virtuaalinen asettelu –lähestymistapa

Jokaiseen ulostuloon liitettyvään järjesteltävään jonoon voidaan laittaa tarpeeksi muistia, jotta kaikki solut mahtuisivat siihen kokonaisuudessaan, mutta on huomattavasti taloudellisempaa tallentaa itse solut yleiskäyttöiseen muistiin ja pitää järjestettävän jonon 'kirjanpidossa' ainoastaan solun aikaleimaa sekä tietoa solun sijainnista muistissa. Näin jokaista solua varten tarvitsee käydä vain kaksi kertaa varsinaisessa muistissa: kerran viemässä solu sinne ja toisen kerran kun solu lähetetään eteenpäin.
Kun käytetään tämän kaltaista lähestymistapaa, tarvitsee järjesteltävässä jonossa pitää itseasiassa vain sen verran jäseniä, kun palvelimeen tulee virtoja. Jonossa on vain kunkin virran seuraavaksi lähetettävään soluun liittyvät tiedot. Ja aina kun lähetetään solu eteenpäin korvataan tätä vastaava paikka jonossa saman virran seuraavana lähdössä olevalla solulla. Tämä on mahdollista Virtuaaliasettelussa, sillä siinä ruuhkaisen yhteyden solujen aikaleimat voidaan määrätä niinkin myöhään kuin siinä vaiheessa kun edellinen solu lähetetään eteenpäin. Ruuhkattoman yhteyden solujen aikaleimat määräytyvät sen sijaan suoraan asetteluajan avulla ja ne pääsevät suoraan jonoon.

6.Yhteenveto

ATM-verkon pitää taata käyttäjilleen tiettyä 'laatua', riippuen yhteyden vaatimasta QoS:stä. Verkoissa on monenlaisia yhteyksiä, jotka ovat hyvinkin erilaisia luonteeltaan (reaaliaikavaatimukset, purkeisuus ym.). Yhteydet saattavat häiritä toisiaan, ellei jo etukäteen oteta huomioon mahdollisia häiriötilanteita ja verkon mahdollisissa pullonkauloissa (mm. kytkimet ja multiplekserit) varauduta niihin ja pyritä kehittämään ratkaisuja, joilla haitat minimoidaan.
Painotettu reilu jonotus (Weighted fair queueing) on hyvä tapa saada häiriköt kuriin ja taata kullekin yhteydelle sille kuuluva osa kaistasta. WFQ:n avulla voidaan myös määrittää rajat päästä-päähän -viiveille ja lisäksi voidaan parametrit sopivasti valitsemalla huolehtia, ettei soluja pääse siirrossa katoamaan.
Haittana WFQ-algoritmeissa on niiden vaatima prosessointiteho ja -aika, jonka vuoksi niitä ei juuri ole nykyisiin ATM-kytkimiin toteutettu. Toinen haitta on se että algoritmit eivät voi taata tiettyä viivettä yhteyden päästä-päähän eikä solujen hukkumattomuutta ellei yhteyden kaikissa kykimissä ole käytössä WFQ-algoritmia.
WFQ-algoritmeja on useita, mutta niiden kaikkien yleinen idea on sama; ne yrittävät jakaa kapasiteetin mahdollisimman oikeudenmukaisesti mahdollisimman pienellä laskentatyöllä.

Lähteet

/1/ James Roberts, Ugo Mocci, Jorma Virtamo (eds.): Broadband Network Teletraffic, Performance Evaluation and Design of Broadband Multiservice Network. Lecture Notes in Computer Science 1155, Final Report of Action COST 242. Luku 6: Weighted Fair Queueing. Springer.

/2/ Abhay k. Parekh & Robert G. Gallager: A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-Node Case. IEEE/ACM Transactions on Networking, Vol. 1 No. 3, June 1993.

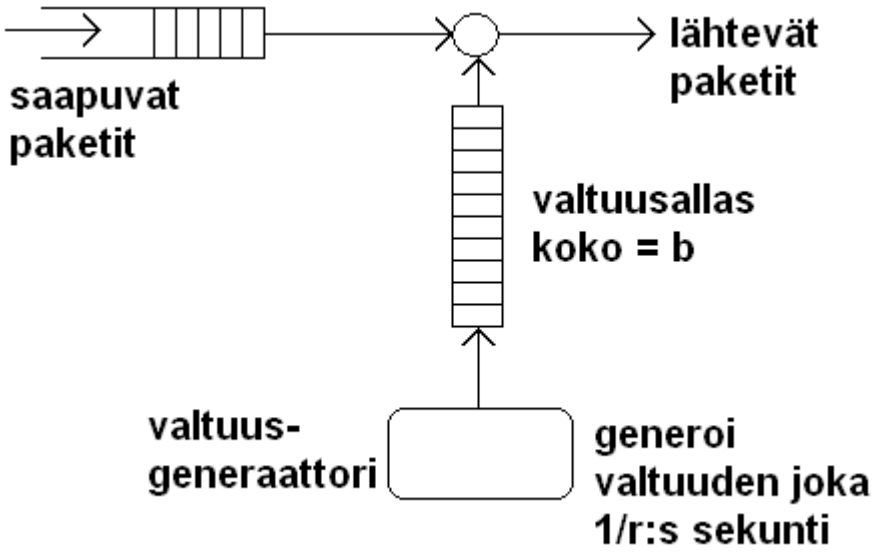
/3/ Dimtrios Stiliadis, Anujan Varma: Frame-Based Fair Queueing: A New Traffic Scheduling Algorithm for Packet-Switched Networks. University of California, June 1995.

/4/ http://kolkata.unl.edu/~sarit/courses/cs462-862/flow-cntrl/node29.html

Liite 1: Vuotava ämpäri – leaky bucket

Algorimin kulku kuvan 1 mukaisessa järjejestelmässä /4/:

- Jos on olemassa valtuus, paketti voidaan ottaa vastaan.
- Paketti ottaa valtuuden mukaansa ja jatkaa matkaa.
- Paketteja synnytetään nopeudella *r* altaaseen, jonka koko on *b*.
- Tämän jälkeen valtuudet hylätään
- *b*:n arvo on kriittinen:
  - pieni *b*: purskeet viivästyvät
  - iso *b*: tungos, puskuri vuotaa yli



Kuva 1.