S-38.201 ATM- JA MULTIMEDIASEMINAARI kevät -97

TCP Vuonohjaus

Ville Häkkinen, 41679p
vhakkine@cc.hut.fi
Sähkö- ja tietoliikennetekniikan osasto
Teknillinen korkeakoulu

Tiivistelmä

Seminaariesitelmän aiheena on TCP:n vuonohjaus. Esitelmän tavoitteena on selvitettää peruskäsitteitä TCP:stä ja TCP -protokollalle ominaisista vuonohjausmekanismeista. Esitelmässä vertaillaan eri toteutusten suorituskykyä ja käsitellään niihin liittyviä ongelmia.

TCP, lyhenne sanoista transmission control protocol, on yhteydellinen tiedonsiirtoprotokolla, jota käytetään yhdessä alemman tason IP -protokollan kanssa maailman suurimmassa tietoverkossa, Internetissä. TCP tarjoaa käyttäjälleen luotettavan informaationsiirtopalvelun. TCP:n tiedonsiirto tapahtuu segmenteissä. Segmentit sisältävät otsikko-osan ja datakentän.

Vuonohjaus, eli flow control, on virhevalvonnan (error control) ohella tiedonsiirtoprotokollan tärkeimpiä tehtäviä. Vuonohjaus pyrkii kontrolloimaan lähetysnopeutta siten, ettei vastaanottajan puskurissa esiinny ylivuotoa. Esitelmässä esitellään kaksi TCP:n vuonohjausmekanismia, TCP Reno ja TCP Vegas. Lisäksi esitellään TCP Vegas*, joka on Vegasin parannettu versio slow-start -ominaisuuden osalta. Se on muilta ominaisuuksiltaan täsmälleen sama kuin TCP Vegas.

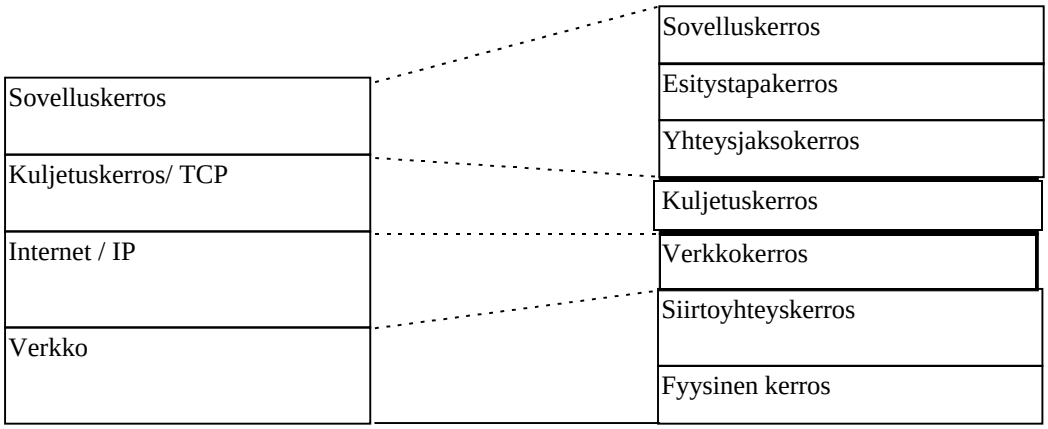
Vertailtaessa Reno- ja Vegas -toteutuksia, yleinen huomio on, että Vegas on huomattavasti tehokkaampi käyttämään kaistanleveyden hyväkseen ja välttämään ruuhkatilanteita kontrolloimalla lähteen lähetysnopeutta. Numeroina Vegasin läpäisy on 37 % parempi kuin Renolla ja pakettien hävikki alle puolet Renon vastaavasta. Yleisesti protokollina TCP Renon ja Vegasin etuja ovat niiden yksinkertaisuus ja riippumattomuus verkon muista komponenteista.

1. Johdanto

Tämä dokumentti on Teknillisen korkeakoulun kurssin S-38.201 ATM- ja Multimediaseminaari kirjallinen osa. Seminaariesitelmän aiheena on TCP:n vuonohjaus. Esitelmässä selvitetään aluksi peruskäsitteitä TCP:stä ja vuonohjauksesta. Sen jälkeen keskitytään juuri TCP:lle ominaisiin vuonohjausmekanismeihin, vertaillaan niiden suorituskykyä ja käsitellään niihin liittyviä ongelmia. Lopuksi esitellään hieman TCP:n vuonohjausta ATM-verkossa. Päälähteenä on käytetty IEEE Journal on Selected Areas in Communications -lehden artikkelia, *TCP Vegas: End to End Congestion Avoidance on a Global Internet* [5]. Artikkelin ovat kirjoittaneet Tohtori Lawrence S. Brakmo Arizonan yliopistosta ja Professori Larry L. Peterson myös Arizonan yliopistosta. Muina lähteinä on käytetty Network Working Groupin Request for Comments (RFC) -dokumentteja ja muita aihetta käsitteleviä IEEE Magazine -artikkeleita. Erilaiset IEEE:n julkaisut sisältävät pääosin eri yliopistojen tutkijoiden raportteja.

2. TCP -protokolla

TCP, lyhenne sanoista Transmission Control Protocol, on yhteydellinen tiedonsiirtoprotokolla, jota käytetään yhdessä alemman tason IP -protokollan kanssa maailman suurimmassa tietoverkossa, Internetissä. TCP tarjoaa käyttäjälleen luotettavan informaationsiirtopalvelun. Palvelu on samankaltainen kuin ISO:n OSI-mallin neljännen kerroksen tehtävissä on määritelty, esimerkiksi OSI TP4 -kuljetuspalvelu. Koska TCP/IP yhdistelmä on vanhempi kuin OSI -malli, kerrokset eivät protokollissa osu aivan kohdalleen. [1] Kuva 2-1 selvittää TCP:n ja OSI -mallin kerrosten vastaavuutta.



Kuva 2-1: Vasemmalla TCP/IP ja oikealla ISO:n OSI -mallin arkkitehtuuri. [8]

Yhteys TCP:ssä muodostetaan kolmivaiheisen kättelyn avulla. Lähetystä suunnitteleva lähde lähettää viestin vastaanottajalle. Viestin mukana välitetään muunmuassa järjestysnumero, josta eteenpäin paketteja aiotaan numeroida. Vastaanottaja kuittaa ja hyväksyy, lähettäen samalla oman vastaavan järjestysnumeronsa. Lähde kuittaa vielä tämän ja yhteys on selvä. Lähetyksen aikana odotetaan joka paketista kuittaus ennen kuin lähetetään uusi paketti. Jos kuittausta ei tule tietyn ajan kuluessa, paketti lähetetään uudestaan. Vastaanottaja kerää paketit puskuriin ja toimittaa ne sieltä ylemmille protokollille ja edelleen sovelluksille.

Paketit koostuvat TCP:n tapauksessa segmenteistä. Segmenttien maksimikoko on määrätty tietylle TCP -toteutukselle. Erään lähteen mukaan oletusarvoinen käyttäjän datan määrä segmentissä on 536 tavua [3]. Segmentit sisältävät lisäksi otsikko-osan ja datakentän. Otsikko-osassa siirretään muunmuassa tietoa segmentin kohteesta, lähteestä, prioriteetista, järjestysnumerosta ja käytetystä sekä pyydetystä ikkunan koosta. [1]

3. Vuonohjaus

Vuonohjaus, eli flow control, on virhevalvonnan (error control) ohella tiedonsiirtoprotokollan tärkeimpiä tehtäviä. Vuonohjaus kontrolloi lähetysnopeutta siten, ettei vastaanottajan puskurissa esiinny ylivuotoa. Ruuhkan välttämisen tärkein syy on turvallisen tiedonsiirron toteuttaminen. Vuonohjauksen epäonnistuessa tehtävässään, datasegmenttejä pudotetaan siirtokanavalta, jolloin ne joudutaaan lähettämään uudestaan, mikä aiheuttaa edelleen verkon ylimääräistä kuormittumista. Jos verkon kuormitus ei ole kriittinen, vuonohjaus pyrkii ensisijaisesti käyttämään verkon siirtokapasiteetin mahdollisimman tarkasti.

Verkon kyky välittää liikennettä riippuu monesta tekijästä. Siirtoteiden nopeudet ja puskurien (buffers) kapasiteetit ovat vuonohjauksen kannalta olennaisimmat tekijät. Juuri puskurien täyttyminen kytkimissä ja päätelaitteissa, ja sitä seuraava

ilmoitus pakettien pudottamisesta yhteydeltä, on vuonohjauksen tarkkailema signaali.

Vuonohjauksen yleiset vaihtoehdot TCP:ssä ovat X-On/ X-Off -tekniikka ja Sliding windows -tekniikka. X-On/ X-Off -tekniikka on niin sanottu “in-band-flow-control” -tekniikka ja sitä käytetään terminaalien ja serverien välisessä merkkipohjaisessa liikennöinnissä. Se ei ole olennainen tekniikka TCP -protokollassa.

Sliding window -tekniikka sen sijaan on käytössä TCP:ssä. Tekniikka perustuu informaatioon siitä, kuinka monta ikkunaa lähettäjä voi lähettää ilman kuittausta vastaanottajalta. Vastaanottaja välittää pyytämänsä tavujen määrän ikkunassa datasegmentin otsikkotietueessa, kentässä window. Ikkuna voi sisältää useita datasegmenttejä. Jos vastaanottaja ei pysty vastaanottamaan saapuvia paketteja, se ei kuittaa niitä ja lähettäjä on tällöin pakotettu lopettamaan lähetyksen toistaiseksi. Ongelman ratkaisut kulkevat nimen, Congestion Avoidance Mechanism (CAM), alla. [1]

4. TCP vuonohjaus

4.1 Perusongelmat

TCP:n vuonhallinnassa on muutama perusongelma. Pienten pakettien ongelma syntyy kun dataa lähetetän liian pienissä ikkunoissa. Tälloin kapasiteettia ei käytetä tehokkaasti. Kun esimerkiksi lähetetään TCP:n avulla yksittäisiä merkkejä näppäimistöltä verkkoon, paketit ovat kooltaan 41:n tavun kokoisia, joissa yksi tavu on hyötykuormaa ja loput otsikkoa. Varsinkin raskaasti kuormitetuissa verkoissa tälläinen 4000 prosentin ylijäämä (overhead) ei ole hyväksyttävää [3]. Ratkaisuna tähän ongelmaan on ehdotettu muunmuassa menetelmää, jossa lähettäjä viivästää ikkunan lähetystä, kunnes sen koko on jokin ennalta määrätty prosenttiosuus linkille ominaisesta maksimaalisesta ikkunan koosta. [2]

Source Quench -ongelma on toinen yleinen vaikeus TCP:ssä. Source Quench -ongelmaan liittyy Retransmission Mechanism, joka esiintyy seuraavissa luvuissa. Source Quench -viesti lähetetään kun yhteydeltä joudutaan pudottamaan segmenttejä. Viestin lähettää reititin jossain yhteyden välillä. Koska TCP:n periaatteisiin kuuluu, ettei dataa tulisi joutua hylkäämään normaalissa liikenteessä, Source Quench sanoma tulisikin lähettää aikaisemmin, esimerkiksi kun jonkin reitittimen puskuri on puolillaan. Näin lähettäjä voisi pienentää ikkunan kokoa ennen kuin puskurissa tapahtuu ylivuotoa ja paketteja joudutaan pudottamaan yhteydeltä. [3]

Seuravassa kappaleessa esitetyt TCP vuonohjauksen versiot yrittävät ratkaista muunmuassa näitä ongelmia erilaisin menetelmin. TCP:n retransmission ja congestion control -mekanismit perustuvat alkujaan niinsanottuun Jacobsonin paperiin. [4]

4.2 TCP Reno

TCP Reno on TCP:n versio, joka käyttää kahta mekanismia havaitakseen uudelleenlähetyksen tarpeen ja sitten suorittaakseen uudelleenlähetyksen. Alkuperäinen mekanismi on TCP:n spesifikaatioissa määritelty Retransmit timeout. Tämä mekanismi perustuu informaatioon pakettien läpimenoajasta, RTT Round Trip Time. RTT on se aika, joka kuluu paketin lähetyksestä kuittauksen (ACK) saapumiseen. Aikaa mittaa laskuri, joka lisää lukemaansa, joka 500:s millisekunti. Eräissä Internetissä suoritetuissa testeissä Reno odotti keskimäärin 1100 millisekuntia, ennenkuin se luokitteli paketin menetetyksi ja suoritti uudelleenlähetyksen. Tällainen keskimääräinen viive koettiin aivan liian suureksi ja niinpä esiteltiin kaksi uutta tekniikkaa, jotka käyttivät kuittauksia paremmin ja ilmoittivat lähettäjälle myös silloin jos saivat paketteja väärässä järjestyksessä. Uudelleenlähetys suoritettiin, kun oli vastaanotettu *n* kappaletta virhekuittauksia. Luku *n* on yleensä kolme. Lisäksi RTT:n aikaa mitattiin tarkemmalla laskurilla, joka lisäsi lukemaansa aina 300:n millisekunnin välein. Uudet tekniikat olivat nimeltään Fast Retransmit and Fast Recovery. Näiden tekniikoiden avulla uudelleenlähetysten määrää tippui puoleen.

Renon ruuhkan havaitsemistekniikka perustuu pudotettujen pakettien tarkkailuun. Kun saadaan Source Quench -signaali, todetaan että verkossa on ylikuormitus. Reno siis tarvitsee hävikkiä pystyäkseen kontrolloimaan vuota. Tämä aiheuttaa kapasiteetin ylittämisiä säännöllisesti, josta puolestaan seuraa se tosiasia, että uudelleenlähetysten määrä kasvaa ja verkon hyödyntäminen kärsii. Resurssien käyttöä saadaan kyllä parannettua, jos Fast Retransmit ja Fast Recovery -tekniikat pystyvät menestyksekkäästi huolehtimaan pakettien uudelleenlähetyksistä. Ruuhkan aiheuttanut yhteys ei kuitenkaan ole ainoa, jonka paketteja voidaan pudottaa, vaan riippuen verkon kytkimien ja reitittimien ruuhkanhallintaparametreistä, saattavat muutkin, niinsanotut syyttömät, yhteydet menettää paketteja ja joutua uudelleenlähettämään niitä. Uudelleenlähetykset kumuloitujat tällaisissa tilanteissa.

Useita ratkaisuja on ehdotettu parannuksiksi tällaisille ei-ennustaville ruuhkanhavaitsemistekniikoille. On ehdotettu suurempia puskureita verkon pullonkauloihin, mutta silloin läpimenoajat pitenisivät merkittävästi. Wang ja Crowford kehittivät DUAL -algoritmin, joka tarkastaa joka toisen RTT:n ja vertaa sitä suurimmasta ja pienimmästä RTT:stä laskettuun keskiarvoon. Jos uusin RTT on suurempi kuin tuo keskiarvo DUAL -algoritmi pienentää seuraavan ikkunan kokoa yhdellä kahdeksasosalla edellisestä ikkunasta.

Toinen esitetty parannus on Jainin CARD -lähestyminen. CARD päätää ikkunan koon laskemalla joka toisen RTT:n avulla seuraavan lausekkeen

$$(WindowSize_{Current} - WindowSize_{Old}) * (RTT_{Current} - RTT_{Old}) \qquad (1).$$

Jos lausekkeen arvo on positiivinen, pienennetään ikkunan kokoa seuraavaan lähetykseen yhdellä kahdeksasosalla edellisestä lähetyksestä. Jos lausekkeen arvo on negatiivinen tai nolla, kasvatetaan ikkunan kokoa seuraavaan lähetykseen yhdellä segmentillä. Ikkunan koko huojuu optimiarvonsa molemmin puolin.

Tri-S tekniikka suurentaa ikkunaa jokaisessa lähetyksessä yhdellä segmentillä ja sitten vertailee mitattua uusinta RTT:tä edellisen yhtä segmenttiä pienemmän ikkunan RTT:hen. Jos erotus on pienempi kuin puolet yhden segmentin lähetyksen RTT:stä, pienennetään ikkunaa yhdellä segmentillä. Tri-S laskee läpäisyn jakamalla lähetyksen tavujen lukumäärän läpäisyajalla, RTT.

Slow-start on mekanismi, joka asteittain lisää liikenteessä olevan datan määrää. Slow-start aloittaa toimintansa ainoastaan silloin, kun yhteydellä ei ole paketteja. Retransmit timeoutin jälkeinen lähetys ja yhteyden muodostusvaihe ovat tällaisia vaiheita. Jos kyseessä on jo olemassa oleva yhteys, protokollalla on tietoa edellisissä lähetyksissä käytetyista ikkunoiden suuruuksista. Reno aloittaa slow-start lähetyksen ikkunalla, jonka koko on puolet sen ikkunan koosta, joka viimeksi lähetettiin ennen timeout -tilaa. Yhteyden alussa slow-start aloittaa lähettämällä yhden segmentin ikkunassa. Seuraavassa lähetyksessä se lähettää yhden segmentin lisää jokaista edellisen lähetyksen segmenttiä kohden eli käytännössä se tuplaa aina kuittauksen saatuaan segmenttien määrän. Lähetysnopeuden kasvu slow-start tilassa on siis eksponentiaalinen. Yhteyden alussa tapahtuva slow-start eroaa edellisestä ainoastaan siitä että mekanismilla ei ole mitään tietoa yhteyden kapasiteetista, joten alustavan ikkunan koon valinnassa joudutaan arvailemaan. Jos ikkuna valitaan liian pieneksi, kestää ikkunakoon kasvattaminen optimiin liian kauan. Jos taas ikkuna on alkujaan liian suuri se aiheuttaa estoa liian aikaisin, eikä yhteys pääse aloittamaan toimivaa lähetystä vähään aikaan. [5]

4.3 TCP Vegas

TCP Vegas on TCP:n versio, jonka uudelleenlähetysmekanismia on hieman kehitetty Reno -versiosta ja jonka vuonohjaus perustuu informaatioon pakettien läpimenoajasta. Ensinnäkin, Vegas lukee ja tallentaa kellonajan jokaisen lähetetyn segmentin kohdalla ja aina kuittauksen saapuessa. Se laskee RTT:n ja käyttää sitä apuna päättäessään uudelleenlähetyksestä. Uudelleenlähetyksen mahdollisuudet voi jakaa tässä kahteen kategoriaan

- Vastaanotetaan virhekuittaus. Vegas tarkastaa paketin lähdösta kuluneen ajan ja vertaa sitä timeout -aikaan. Jos timeout -aika on ylitetty Vegas suorittaa uudelleenlähetyksen, odottelematta *n* kappaletta virhekuittauksia saapuvaksi vastaanottajalta.
- Vastaanotetaan normaali kuittaus. Jos se on ensimmäinen tai toinen uudelleenlähetyksen jälkeen, Vegas suorittaa jälleen aikavertailun, ja sen perusteella suorittaa uudelleenlähetyksen, jos tarpeellista.

Vegasin ruuhkanhallinta perustuu odotetun läpimenon ja todelliseen läpimenon suhteeseen. Ikkunan kokoa muutetaan tietyllä määrällä perustuen tämän laskutoimituksen tuloksiin. Kun ikkunaa kasvatetaan, läpimenoajan odotetaan kasvavan, koska siirrettävää on enemmän, jos taas läpimenoaika pienenee, voidaan olettaa verkossa olevan enemmän resursseja ja ikkunaa kasvatetaan seuraavan paketin lähetykseen. Vegasin käyttämää algoritmia on kuvattu seuraavassa kappaleessa.

Odotettu läpimeno lasketaan seuraavalla kaavalla:

odotettu, expected, läpimeno = ikkunan koko / base RTT *(2).*

Kaavassa 1, base RTT tarkoittaa yhteydellä havaittua pienintä kiertoaikaa. Seuraavaksi Vegas laskee todellisen läpimenon.

todellinen, actual, läpimeno = segmentissä lähetetyt tavut / RTT *(3).*

Tuloksia verrataan toisiinsa kaavalla.

Diff = Actual - Expected *(4).*

Vuonohjaus perustuu *Diff* arvoon ja kahteen erilliseen parametriin α ja β. α ja β määräävät rajat, joiden suhde *Diff* -muuttujaan ratkaisee seuraavan lähetyksen ikkunan koon. Käytännössä α ja β ilmoitetaan yleensä puskurikokoina.

- Jos *Diff* < α suurennetaan ikkunaa seuraavaan lähetykseen
- Jos *Diff* >β pienennetään ikkunaa seuraavaan lähetykseen
- Jos α < *Diff* >β lähetetään edelleen samalla ikkunakoolla

Vegas ei tarvitse hävikkiä pystyäkseen kontrolloimaan vuota. Vegasin vuonohjaus reagoi läpimenoaikojen muutoksiin ja pyrkii siten ennakoimaan mahdolliset ruuhkatilanteet. Erityisesti Vegas rajoittaa ikkunoiden nopeaa kasvamista sellaisilla yhteyksillä, joiden RTT:t ovat pieniä. Vegasin etu Renoon nähden on lisäksi se, että se ennemminkin käyttää olemassaolevan kaistan paremmin kuin että veisi kaistaa toisilta ja aiheuttaisi muille yhteyksille hävikkiä.

Vegasin slow-start tekniikka on suurelta osin sama kuin Reno -toteutuksella. Vegas tosin kasvattaa ikkunaa eksponentiaalisesti ainoastaan joka toisen RTT -jakson jälkeen. Näin Vegas voi laskea vuonohjauksessa käyttämänsä *Diff* -arvon. Vegas tarkkailee slow-start -tilassakin *Diff* -arvoa. Kun arvo muuttuu positiiviseksi, Vegas lopettaa slow-start -moodin ja siirtyy normaaliin vuon tarkkailuun. Vegasin heikkoudet ovat samankaltaisia verrattuna Renoon. Vegasin tapauksessa ongelmat voi kuvata seuraavasti

- Ylilähetyksen ongelma tulee kun kuittausten taajuudesta riippumatta lähetetään liian paljon dataa. Reititin saattaa joutua puskuroimaan jopa puolet lähetettyjen ikkunoiden informaatiosta. Hävikki tulee tällöin nopeasti mahdolliseksi.
- Yhteyden alussa kaistanleveysarvaus on liian suuri. Reitittimen puskurin tulisi silloin olla riittävän suuri, jotta Vegas ehtii havaita uhkaavan ylikuormituksen ja laskea lähetysnopeutta. [5]

*4.4 TCP Vegas **

TCP Vegas* on Vegasin parannettu versio, slow-start -ominaisuuden osalta. Se on muilta ominaisuuksiltaan täsmälleen sama kuin TCP Vegas. Slow-start -moodissa Vegas* tarkkailee ikkunan kokoa neljän RTT -jakson välein. Vegas* ajoittaa ikkunan koon modifioinnin tulevaisuuteen riippuen RTT:stä, linkin

segmenttien maksimikoosta ja kaistanleveysarviosta. Jos esimerkiksi yhteyden tunnusluvut ovat:

RTT = 100 ms
Maximum segment size = 1 KB
Bandwith estimate = 200 KB/s
Odotettujen kuittausten lukumäärä RTT:n aikana = 4

Näillä arvoilla Vegas* laskee kuittausten odotetun aikavälin.

Maximum segment size / Bandwith estimate = 5 ms *(5).*

Kun kaikki kuittaukset on vastaanotettu seuraava lähetys ajoitetaan ajan *t* päähän

*t = Maximum segment size / Bandwith estimate * Odotettujen kuittausten lukumäärä RTT:n aikana* *(6).*

Tässä tapauksessa Vegas* lähettäisi seuraavan yhdellä maksimisegmentillä kasvatetun ikkunan 20:n millisekunnin päästä. Tämän menettelyn ansiosta Vegas* -toteutus tarvitsee huomattavasti vähemmän puskuritilaa reitittimissä kuin Reno tai tavallinen Vegas. [5]

5. TCP Reno ja Vegas Internetissä

5.1 Yleinen suorituskyky

Vertailtaessa Reno ja Vegas toteutuksia, yleinen huomio on, että Vegas on huomattavasti tehokkaampi käyttämään kaistanleveyden hyväkseen ja välttämään ruuhkatilanteita kontrolloimalla lähteen lähetysnopeutta. Numeroina Vegasin läpäisy on 37 % parempi kuin Renolla ja pakettien hävikki alle puolet Renon vastaavasta.

IEEE Magazinen artikkelissa Internetissä TCP:n eri toteutuksilla suoritettut mittaukset tehtiin seuraavin parametrein. Molempia toteutuksia käyttäen lähetettiin samat aineistot seitsemän kertaa. Tiedostojen koot olivat 1 MB, 512 KB ja 128 KB. Vegasin parametrit ilmoitetaan sen nimessä. Esimerkiksi Vegas-1,3 tarkoittaa Vegas -toteutusta, jossa α on yksi ja β vastaavasti kolme. Nämä arvot ovat yleisimmät Vegasille. Taulukoissa 1 esitetään mittauksen tuloksia 1 MB:n tiedostolle. Kuten taulukosta voidaan nähdä Vegas on suorityskykyisempi. [5]

Taulukko 5-1: Mittaustulokset Internetissä. Paketin koko 1 MB. [5]

	Reno	Vegas-1,3	Vegas-2,4
Läpäisy (KB/s)	53,00	72,50	75,30
Läpäisysuhde verrattuna Renoon	1,00	1,37	1,42
Uudelleenlähetykset (KB)	47,80	24,50	29,30
Uudelleenlähetyssuhde varrattuna Renoon	1,00	0,51	0,61
Timeout -tilanteet	3,30	0,80	0,90

Taulukossa 2 on vertailtu tiedoston koon vaikutusta samoihin testiarvoihin kuin taulukossa 1. Tiedoston koon pienentyessä Vegas lisää eroaan Renoon.

Taulukko 5-2: Mittaustulokset Internetissä vaihtelevalla paketin koolla. [5]

	1024 KB		512 KB		128 KB	
	Reno	Vegas	Reno	Vegas	Reno	Vegas
Läpäisy (KB/s)	53,00	72,50	52,00	72,00	31,10	53,10
Läpäisysuhde verrattuna Renoon	1,00	1,37	1,00	1,38	1,00	1,71
Uudelleen-lähetykset (KB)	47,80	24,50	27,90	10,50	29.90	4,00
Uudelleenlähetys-suhde varrattuna Renoon	1,00	0,51	1,00	0,38	1,00	0,17
Timeout -tilanteet	3,30	0,80	1,70	0,20	1,10	0,20

5.2 Reiluus

Jos linkillä on useampia yhteyksiä samanaikaisesti on resurssit jaettava jotenkin niiden välillä. Normaalisti haluttaisiin kaikkien yhteyksien saavan yhtä paljon palvelua. TCP ei pysty tähän ilman reitittimien apua. Yleisesti Renon reiluutta pidetään riittävänä. Vertailussa käytettiin Jainin reiluusindeksiä .

Kokeilut tehtiin kahden, neljän ja 16:n yhtäaikaisen yhteyden kanssa. Kahden ja neljän yhteyden tapauksessa lähteet lähettivät 8 MB dataa. Reno osoittautui hieman reilummaksi kuin Vegas, kun kaikilla yhteyksillä oli sama etenemisviive, mutta Vegas oli reilumpi kun puolella yhteyksistä oli suurempi etenemisviive. Kun lähteitä oli 16 ja jokainen lähetti 2 Mb dataa, Vegas oli reilumpi, riippumatta viiveistä. [5]

5.3 Stabiilisuus

Internet protokollan tulisi olla vakaa kaikissa tilanteissa, koska yhtäaikaisia yhteyksiä on aktiivisessa tilassa useita joka hetkellä. Vegas on paljon stabiilimpi niin kauan kuin ikkunoiden koot ovat suurempia kuin yksi maksimisegmentti. Jos linkki on niin ruuhkaisa, ettei keskimääräinen ikkuna sisällä kuin yhden maksimisegmentin dataa Vegas alkaa käyttäytyä kuin Reno. Yleisesti ottaen Reno ei ehkäise ruuhkaa, vaan luo sitä säännöllisesti. Vegasin slow-start ominaisuudet ovat paremmat ja se ennaltaehkäisee ruuhkia. [5]

6. TCP ATM-verkossa

Monissa tutkimuksissa on tultu siihen tulokseen, että TCP suoriutuu huonosti ATM-verkoissa. TCP toimii hyvin ainoastaan, jos verkko pudottaa vain yhden paketin, kun TCP lisää ikkunan kokoaan yhdellä segmentillä. Koska ATM -kytkimet yleensä pudottavat tarpeen mukaan yksittäisiä soluja, jotka saattavat olla eri paketeista, TCP joutuu lopettamaan lähetyksensä hetkeksi sen sijaan, että se vain pienentäisi ikkunan kokoa seuraavassa lähetyksessä. Tämä johtuu informaation häviämisestä useassa ikkunassa, koska ATM -solut ovat paljon

pienempiä kuin TCP -ikkunat. Tämä ongelma ratkeaisi, jos TCP pystyisi toipumaan monen paketin pudottamisesta yhtä nopeasti kuin yhden. [6]

TCP:n suorituskyvyn parantamiseksi ATM -verkossa on kehitetty ainakin kaksi yksinkertaista kontrollimenetelmää. Partial Packet Discard, PPD ja Early Packet Discard, EPD. Kun solu pudotetaan ATM -kytkimessä PPD tekniikka aiheuttaa koko siihen liittyvän suuremman kokonaisuuden, segmentin, hylkäämisen. PPD nostaa TCP:n suorituskykyä tiettyyn rajaan saakka. Vielä pareman suorituskyvyn saa aikaan EPD. Kun käytetään EPD:tä ja ATM -kytkimen puskurissa olevien solujen määrä ylittää tietyn kynnysarvon, kokonaisia TCP -ikkunoita pudotetaan. [7]

7. Yhteenveto

Internetin ja verkoittumisen valtava kasvunopeus tulee vielä asettamaan suuria vaatimuksi erilaisten verkkojen suorituskyvylle. TCP:n asema Internet-standardina on vankka. Kasvun johdosta on erittäin tärkeää jatkuvasti pyrkiä parantamaan TCP:n suorituskykyä. ATM -verkkojen todennäköinen nouseminen valta-asemaan aiheuttaa yhä suurempia paineita saada TCP toimimaan liukkaasti ATM -verkon päällä. TCP Renon ja Vegasin rakenteen etuja ovat kuitenkin niiden yksinkertaisuus ja riippumattomuus verkon muista komponenteista. Suorituskykyä parantaisi tietysti vielä edelleen reitittimien ja vuonohjausmekanismien laajempi kanssakäyminen, esimerkiksi linkin kaistanleveyden kertova ilmoitus reitittimeltä, kun se havaitsee uuden lähteen linkkiensä päässä parantaisi huomattavasti TCP:n slow-start ominaisuuksia. Tässä dokumentissa esitettyjen ratkaisujen osaksi jää luultavasti vain olla välietappeja matkalla kohti edelleen tehokkaampia ja monipuolisempia vuonohjausmekanismeja.

8. Lähteet

/1/ Fred Halsall: Data Communications, Computer Networks and Open systems, Fourth Edition, Addison - Wesley, 1996

/2/ DARPA Internet Program, Protocol Specification: Transmission Control Protocol, RFC 793, syyskuu 1981

/3/ John Nagle: Congestion control in IP/TCP Internetworks, RFC 896, 6. tammikuuta, 1984

/4/ V. Jacobson: Congestion Avoidance and Control, SIGCOMM '88 symposium, elokuu 1988

/5/ Lawrence S. Brakmo, Larry L. Peterson: TCP Vegas: End to End Congestion Avoidance on a Global Internet, IEEE Journal on Selected Areas in Communications, vol. 13, lokakuu 1995

/6/ Detailed Behaviour of TCP over ATM, http://www.eecs.harvard.edu/~rtm/tcp-atm/tcp-atm-1.html, 27.1. 1997

/7/ Allyn Romanov: Dynamics of TCP Traffic Over ATM Networks, IEEE Journal on Selected Areas in Communications, vol. 13, huhtikuu 1995

/8/ Teletekniikan laboratoriotyöt II: Teknillisen korkeakoulun opetumoniste, Otatieto Oy, kevät -97