Lasse Ropponen

Ruuhkanhallinta-algoritmien toiminta haasteellisissa tietoverkoissa

Elektroniikan, tietoliikenteen ja automaation tiedekunta

Diplomityö, joka on jätetty opinnäytteenä tarkastettavaksi diplomi-insinöörin tutkintoa varten Espoossa 7.5.2010.

Työn valvoja:

Prof. Jukka Manner

Työn ohjaaja:

FM Sebastian Siikavirta

Aalto-yliopisto
Teknillinen korkeakoulu

AALTO-YLIOPISTO
TEKNILLINEN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Lasse Ropponen

Työn nimi: Ruuhkanhallinta-algoritmien toiminta haasteellisissa tietoverkoissa

Päivämäärä: 7.5.2010 Kieli: Suomi Sivumäärä:7+47

Elektroniikan, tietoliikenteen ja automaation tiedekunta

Tietoliikenne- ja tietoverkkotekniikan laitos

Professuuri: Tietoverkkotekniikka Koodi: S-38

Valvoja: Prof. Jukka Manner

Ohjaaja: FM Sebastian Siikavirta

Tämä diplomityö käsittelee ruuhkanhallinta-algoritmien toimintaa erilaisilla pakettien hukkatodennäköisyyksillä ja kauttakiertoajoilla. Tutkimuksen kohteena oli seitsemän eri kuljetuskerroksen protokollaa, joista viisi kuuluvat TCP- ja kaksi DCCP-protokolliin. Tutkimusmenetelmänä käytettiin systemaattista simulointia, joka toteutettiin ns2-ohjelmistolla.

Ruuhkanhallinta-algoritmien tarkoituksena on välttää ruuhkautumistilanteita ja saada verkon resurssit mahdollisimman tehokkaasti käyttöön. Tämän lisäksi ruuhkanhallinnalta odotetaan tasapuolista suhtautumista muita verkon käyttäjiä kohtaan. Itsekäs ja aggressiivinen verkkoprotokolla saattaa näyttää kyseisen verkkosovelluksen näkökulmasta varsin tehokkaalta, mutta kokonaistehokkuuden kannalta kyseisenlainen toiminta ei ole järkevää.

Ruuhkanhallinta perustuu verkon ruuhkautumisen toteamiseen ja tästä aiheutuvaan lähetysnopeuden säätelyyn. Algoritmeilla on erilaisia menetelmiä ruuhkautumisen toteamiseksi, mutta kaikkia niitä yhdistää kyvyttömyys erottaa verkkoliikenteen häiriintymisen syy. Yleisin indikaatio verkon ruuhkautumisesta on pakettien katoaminen, joka saattaa johtua monestakin eri syystä. Ruuhkautumisen tai verkkohäiriön takia aiheutuneisiin katoamisiin reagoidaan kaikkiin yleensä samalla tavalla, joka saattaa johtaa joissakin tapauksissa verkon resurssien tehottomaan käyttöön. Näin käy esimerkiksi ruuhkattomissa langattomissa verkko-olosuhteissa, joissa esiintyy runsaasti pakettihukkaa.

Tutkimuksen simulaatiot jakautuivat kahteen eri osaan, joista ensimmäisessä tutkittiin eri algoritmien suorituskykyä ruuhkattomassa verkossa, jonka pullonkaulaksi muodostui pakettihukkaa aiheuttava linkki. Simulaatioiden toisessa osiossa pyrittiin selvittämään tutkittujen algoritmien tasapuolisuutta pullonkaulalinkillä, jolle ohjattiin kaksi datavuota, joista molemmat olivat eri TCP- tai DCCP-versioita.

Avainsanat: Ruuhkanhallinta-algoritmi, TCP, DCCP, SCTP, simulointi, ns2, pakettihukka, läpivienti

AALTO UNIVERSITY
SCHOOL OF SCIENCE AND TECHNOLOGY

ABSTRACT OF THE
MASTER'S THESIS

Author: Lasse Ropponen

Title: Congestion Control Algorithms in Challenging Networks

Date: 7.5.2010 Language: Finnish Number of pages:7+47

Faculty of Electronics, Communications and Automation

Department of Communications and Networking

Professorship: Computer Networks Code: S-38

Supervisor: Prof. Jukka Manner

Instructor: FM Sebastian Siikavirta

The focus of this Master's thesis is to research the behavior of different congestion control algorithms by different packet drop and latency values. Seven different transport layer protocol variants were researched. Five of them were TCP variants and two DCCP variants. Used research method was systematic simulation that was performed by Ns2-simulator.

Congestion control algorithms try to avoid congestion and permit as efficient usage of the network as possible. Congestion control algorithms are also responsible about the fairness between different network users. Selfish and aggressive network protocol may seem efficient from its own perspective but for the overall efficiency such a behavior is questionable.

Congestion control is based on noticing and reacting for congestion by tuning the transfer rate. There are many different methods to become aware of congestion but all of them are incapable to distinguish the reason for the interference at the transmission. The most common indication about congestion is a packet loss. Unfortunately there is no mechanism to know exactly the reason for the packet loss. All indications about congestion cause same congestion control behavior and therefore algorithms may sometimes react incorrectly. A good example about this is an uncongested wireless network which has high packet drop rate.

The simulations were divided into two parts. At the first part the goodput of different TCP and DCCP variants through an uncongested bottleneck link was researched by computer simulations. At the second part of the simulations the fairness of different TCP and DCCP variants at a bottleneck link were researched.

Keywords: Congestion Control, Congestion Avoidance, TCP, DCCP, Simulation, ns2, Throughput

Esipuhe

Tämä diplomityö on tehty Aalto-yliopiston Teknillisen korkeakoulun tietoliikenne- ja tietoverkkotekniikan laitoksella.

Haluan kiittää työni valvojana toiminutta professori Jukka Mannerta diplomityöaiheen antamisesta sekä työhön liittyneistä erinomaisista kommenteista. Kiitän myös työn ohjaajana toiminutta FM Sebastian Siikavirtaa hyvästä tuesta työn tekemisen aikana.

Suuri kiitos myös läheisilleni sekä etenkin vaimolleni Sannalle kärsivällisyydestä ja tuesta opiskeluaikanani.

Espoo, 7.5.2010

Lasse Ropponen

Sisältö

Lyhenteet

ACK	Acknowledgement
AIAD	Additive Increase, Additive Decrease
AIMD	Additive Increase, Multiplicative Decrease
CWND	Congestion Window
DACK	Dublicate Acknowledgement
DCCP	Datagram Congestion Control Protocol
DupACK	Duplicate Acknowledgement
FACK	Forward Acknowledgement
IETF	Internet Engineering Task Force
IP	Internet Protocol
MIAD	Multiplicative Increase, Additive Decrease
MIMD	Multiplicative Increase, Multiplicative Decrease
RFC	Request For Comments
RTO	Retransmission Timeout
RTT	Round Trip Time
RWND	Receiver's Advertised Window
SACK	Selective Acknowledgement
SCTP	Stream Control Transmission Protocol
SMSS	Sender Maximum Segment Size
SSTHRESH	Slow Start Threshold
TCP	Transmission Control Protocol
TFRC	TCP Friendly Rate Control
UDP	User Datagram Protocol

1 Johdanto

Ruuhkanhallinta on ollut osana tietoverkkojen kehityksessä lähes nykyaikaisten tietoverkkojen syntyajoista lähtien. Jo 1980-luvulla Van Jacobson totesi ruuhkautumisen aiheuttavan todellisen riskin tietoverkoille. Hänen mukaansa ruuhkautuminen ja etenkin käyttäjien välinpitämättömyys ruuhkaa kohtaan saattavat aiheuttaa koko verkkoliikenteen romahtamisen. Tietoverkkojen käyttäjämäärien valtavan kasvun myötä verkkoliikenteen säätely ruuhkautumistilanteissa on tullut välttämättömäksi.

Ruuhkanhallinta-algoritmien päämääränä on mahdollistaa tietoverkkojen mahdollisimman tehokas ja tasapuolinen käyttö. Ruuhkanhallinta ominaisuus sijoittuu kuljetuskerroksen protokollien tehtäviin, mutta esimerkiksi UDP-protokollassa ei käytetä minkäänlaista ruuhkanhallintaa. Ruuhkanhallinta-algoritmien tehtävänä on säädellä tietoliikenneyhteyden lähetysnopeutta verkon ruuhkautumisasteen mukaisesti. Algoritmit pyrkivät arvioimaan verkon ruuhkautumista esimerkiksi pakettien katoamisten, viiveen vaihtelun sekä toteutuneen läpiviennin perusteella.

Nykyisin yleisimmät käytössä olevat kuljetuskerroksen protokollat on suunniteltu ennen langattomien verkkotekniikoiden aikakautta. Protokollat käyttävät pääsääntöisesti pakettihukkaa indikaationa verkon ruuhkautumisesta. Kyseinen oletama toimii varsin hyvin perinteisissä kiinteissä tietoliikenneverkoissa, mutta langattomien verkkojen yhteydessä pakettien katoaminen ei ole läheskään aina merkki ruuhkautumisesta, sillä suuri osa pakettien epäonnistuneista siirroista johtuu radiotiellä olevista häiriötekijöistä. Näin ollen yleisimmät käytössä olevat kuljetuskerroksen protokollat käyttävät ruuhkanhallintamekanismejaan usein virheellisesti tiputtaessaan lähetysnopeuttaan radikaalisti myös häiriöllisten langattomien verkkojen yhteydessä.

Tämän tutkimuksen päämääränä oli selvittää viiveen ja pakettihukan vaikutusta eri ruuhkanhallinta-algoritmien suoritustasoon. Tutkimuksen kohteeksi valittiin viisi eri TCP- ja kaksi DCCP-versiota, joiden tehollisten läpivientien tasoja vertailtiin käyttäen systemaattista simulointia tutkimusmenetelmänä. Tutkimuksen tarkoituksena oli selvittää mikä ruuhkanhallinta-algoritmi pystyy tarjoamaan parhaan tehollisen läpivientitason ruuhkautumattomassa, mutta paketteja hukkaavassa verkossa. Tutkimus tarkastelee olosuhteita, joissa pakettihukka vaihtelee välillä 0,1-10,0 %, eli tutkitun pakettihukan alueelle sijoittuu mobiiliverkkojen pakettihukkatodennäköisyydet. Tutkimuksen tulosten perusteella pystytään määrittelemään mikä ruuhkanhallinta-algoritmi pystyy tarjoamaan parhaan tehollisen läpiviennin erilaisilla viiveen ja pakettihukan yhdistelmillä. Tämän lisäksi tutkimuksessa selvitettiin kuinka eri ruuhkanhallinta-algoritmit suoriutuvat tasapuolisuuden suhteen. Näiden kahden eri ominaisuuden perusteella voidaan arvioida eri ruuhkanhallinta-algoritmien suorituskykyä.

2 Ruuhkanhallinta tietoverkoissa

Tietoverkot koostuvat reitittimistä ja niitä yhdistävistä linkeistä, joiden molempien kapasiteetit ovat rajallisia. Kun verkon resursseihin nähden liian suuri määrä liikennettä pyrkii samanaikaisesti tietyn verkon osan läpi, syntyy ruuhkaa. Ruuhka aiheuttaa reitittimien puskurien ylivuotoa, jolloin osa paketeista joudutaan tiputtamaan pois. Verkon ruuhkautumiseen tulee reagoida, jotta pystytään takaamaan verkon stabiilisuus, verkon mahdollisimman tehokas käyttö sekä resurssien reilu jakaminen käyttäjien kesken. Mikäli ruuhkautumiseen ei reagoida ollenkaan tai tarpeeksi voimakkaasti, seurauksena voi olla jopa verkon totaalinen romahtaminen.

Ruuhkautumisen havainnointia varten on kehitetty useita erilaisia mekanismeja. Tässä luvussa perehdytään niiden toimintaan sekä erilaisiin tapoihin reagoida ruuhkautumistilanteisiin. Tämän lisäksi tutustutaan teoreettisen maksimiläpivienninmäärittelyyn sekä siihen vaikuttaviin seikkoihin.

2.1 Tietoverkkojen ruuhkautuminen

Tietoliikenneverkkojen kuormittaminen verkon resursseja suuremmalla volyymillä aiheuttaa ruuhkautumista. Se ilmenee käyttäjälle yhteysnopeuden pienentymisenä ja vasteajan kasvuna. Verkkotasolla ruuhkautuminen aiheuttaa verkon reitittimien puskureissa odottavien pakettien määrän kasvua. Mikäli verkon käyttäjät eivät pienennä lähetysnopeuksiaan puskurien jonojen kasvaessa, on seurauksena pakettien tippumista pois ylivuotavista puskureista. Paketti voi kadota matkalla myös muistakin syistä kuin ruuhkautumisen takia, esimerkiksi laite- tai linkkivian seurauksena paketti voi korruptoitua tai kadota.

Internet palveluntarjoajat (ISP) eivät aina mitoita verkkojaan sen mukaan että kaikille sen asiakkaille pystyttäisiin takamaan liittymäsopimuksen mukaisen maksimisiirtonopeuden samanaikaisesti. Sen sijaan esimerkiksi tuhatta neljän megabitin liittymää kohden saatetaan joissakin tapauksissa varata runkoverkon kapasiteettia yhden gigabitin verran, eli vain neljäsosa kaikkien liittymien muodostamasta maksimikapasiteetista. ISP säästää tällä tavoin rahaa käyttämällä pienempikapasiteettista linkkiä asiakkaidensa Internet -yhdyskäytävään [2]. Näin voidaan toimia, koska asiakkaat kuormittavat verkkoa hyvin harvoin täydellä kapasiteetilla samanaikaisesti. Kyseisenlainen runkoverkon alimitoittaminen aiheuttaa kuitenkin ennemmin tai myöhemmin tilanteita, jolloin verkon kapasiteetti ei riitä tyydyttämään siltä vaadittua suoritustasoa ruuhkapiikeissä. Edellä kuvailtu tapa mitoittaa runkoverkko säästösyihin vedoten on kuitenkin jäänyt suurelta osin historiaan. Nykyisin ISP:den kannalta tehokkain tapa on ylimitoittaa (overprovisioning) runkoverkon linkit, eli resursseja on riittävästi vaikka kaikki asiakkaat samanaikaisesti käyttäisivät liittymiään täydellä kapasiteetilla. Syitä ylimitoitukseen siirtymiseen ovat runkoverkon linkkikapasiteetin halpa hinta, ylimitoitetun verkon ylläpidon helppous sekä verkon helpompi muuntuminen tulevaisuuden haasteisiin. Runkoverkkojen ylimitoitus on tarkoittanut myös ruuhkautumisen siirtymistä pois runkoverkon linkeiltä asiakkaiden ja ISP:n yhdyskäytävän välille [2].

Runkoverkon yli- tai alimitoituksen tila ovat vaihdelleet eri aikoina riippuen ai-

na sen hetkisistä runkoverkon ja toisaalta liityntäverkon teknologioista. Esimerkiksi ISDN-yhteyksien yleistyttyä oli runkoverkoissa käytetyt frame relay -verkot selvästi alimitoitettuja liityntäverkkojen nopeuksiin verrattuna [2]. Nykyiset optiset runkoverkot pystyvät hyvin tyydyttämään resurssitarpeet, mutta uusien nopeiden liityntäverkkojen myötä saatetaan taas joku päivä siirtyä tilanteeseen, jossa runkoverkoja joudutaan alimitoittamaan.

2.2 Ruuhkan- ja vuonhallinta

Ruuhkanhallinta ja vuonhallinta ovat tietoliikennetekniikan mekanismeja, joiden avulla pyritään estämään verkon komponenttien ylikuormittumista. Ruuhkanhallinnan päämääränä on suojella verkon ruuhkautumista sen perusteella, minkälaista palautetta verkosta tulee esimerkiksi kadonneiden pakettien tai viiveajan kasvun perusteella. Vuonhallinnalla pyritään puolestaan estämään vastaanottajan resurssien ylikuormittumista, kun vastaanottaja ilmoittaa palautteella kuinka nopeaan vastaanottonopeuteen se pystyy.

Verkkosovelluksen olisi hyvä suojella sekä verkkoa että vastaanottajaa ylikuormittumiselta samanaikaisesti, joten lähettäjän tulisi säätää maksimi lähetysnopeus sen mukaan kumpi vuon- vai ruuhkanhallintamekanismi antaa pienemmän sallitun lähetysnopeuden.

2.3 Palautetyypit

Systeemiä joka käyttää saamaansa palautetta toimintansa säätelyyn kutsutaan niin sanotuksi closed-loop -kontrolloiduksi systeemiksi. Vastaavasti open-loop kontrolloiduissa systeemeissä ei saada järjestelmän tilasta mitään palautetta. Kaikki tässä diplomityössä tutkittavat tietoliikenneprotokollat käyttävät closed loop -kontrollia. Open loop -systeemeissä voidaan määritellä järjestelmän toimintaa jo ennalta tiedossa olevien ominaisuuksien perusteella. Esimerkiksi jotkin sovellukset määrittelevät lähetys- ja vastaanottonopeuksia sen perusteella kuinka nopeaksi Internet-liittymä on määritelty sovelluksen asennuksen yhteydessä. Tällaisen sovelluksen voidaan katsoa käyttävän alkeellista open loop -ruuhkanhallintaa.

Tietoverkkoon lähetetty paketti voi vastaanottajan näkökulmasta tulla normaalisti perille, kadota, tulla perille epänormaalin viiveen jälkeen tai korruptoitua. Katoaminen voi johtua esimerkiksi ruuhkautumisesta, laiteviasta tai häiriöistä radiotiellä. Samoin viiveelle voi olla useita syitä.

Palaute on yksinkertaisimmillaan binaarista, eli palautteella pystytään ilmoittamaan kaksi eri tilaa. Esimerkiksi binäärinen palaute tietoverkosta sisältää kaksi eri tilaa: 0 paketti meni perille, 1 paketti kadonnut. Palautteen perusteella tehdään tulkinta, josta seuraa mahdolliset jatkotoimet palautteesta riippuen. Esimerkiksi paketin katoaminen voidaan tulkita verkon ruuhkautumiseksi, jonka perusteella lähetysnopeutta pienennetään verkon pahemman ruuhkautumisen välttämiseksi. Binäärinen palaute voi usein aiheuttaa myös virhetulkintoja verkon tilasta, jolloin sen aiheuttama vaste saattaa olla tilanteeseen sopimaton. Tästä hyvänä esimerkkinä on

TCP-ruuhkanhallinnan reagointi tilanteeseen, jossa paketti katoaa mobiiliverkossa heikon SINR-tason vuoksi.

2.4 Kiertoajan arviointi

Round Trip Time (RTT) eli kiertoaika kertoo kuinka kauan signaalin kestää kulkea tietoliikenneyhteyden päästä päähän ja takaisin. RTT määrittelee minimiajan mikä kuluu paketin lähettämisestä ACK-kuittauksen vastaanottoon. Useat ruuhkanhallintamekanismit käyttävät hyödykseen kiertoajalle laskemaansa arviota pystyäkseen mahdollisimman tehokkaasti huomaamaan pakettien katoamiset. Mikäli lähetetylle paketille ei saada kuittausta tietyn ajan kuluessa, lähettäjä toteaa sen kadonneeksi ja uudelleenlähettää sen. Kyseistä mekanismia kutsutaan nimellä Automatic Repeat Request (ARQ).

Retransmission Timeout (RTO) on aika joka odotetaan ennen kuin paketti todetaan kadonneeksi. Sen mahdollisimman tarkka määrittely tasoon, jossa se rajaa mahdollisimman tarkasti ajanhetken joka erottaa onnistuneet ja epäonnistuneet siirrot, on äärimmäisen tärkeää systeemin tehokkuuden kannalta. RTO:n määrittely liian pieneksi aiheuttaa useiden pakettien virheellisen tulkinnan kadotetuiksi, jonka seurauksena aiheutuu mahdollisesti turhia uudelleenlähetyksiä ja käytettävän ruuhkanhallintamekanismin mukaisia lähetysnopeuden pienennyksiä. Vastaavasti RTO:n määrittely liian suureksi aiheuttaa suoritustason laskua, koska pakettien katoaminen todetaan optimaalista hetkeä myöhemmin [3].

2.5 AIMD, AIAD, MIMD ja MIAD

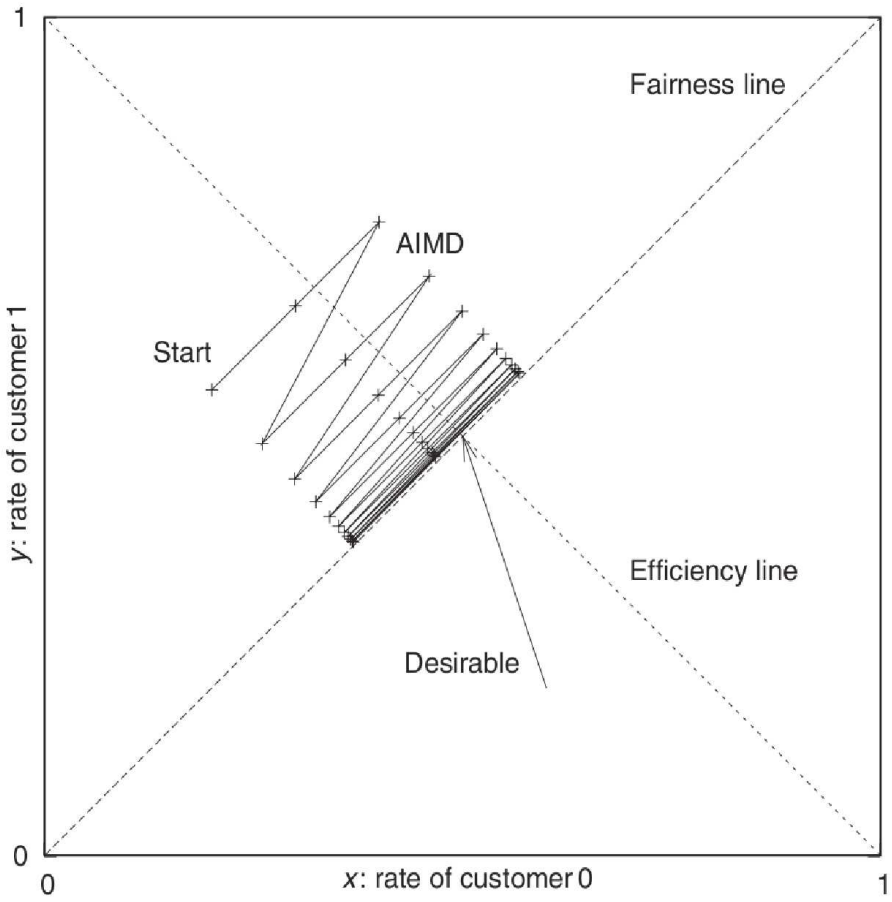
Lähetysikkunaa käyttävät kuljetuskerroksen protokollat jaetaan eri ryhmiin niiden ikkunan säätelymekanismien perusteella. Nämä niin sanotut palautekontrollialgoritmit (feedback control algorithms) koostuvat additiivisesta tai multiplikatiivisesta ikkunan suurentamisesta sekä pienentämisestä. Esimerkiksi TCP-protokollissa yleisesti käytetyn mekanismin nimi Additive Increse Multiplicative Decrease (AIMD) kuvastaa hyvin kuinka lähetysikkunan kokoa säädellään.

Tyypillisesti lähetysikkunan kokoa kasvatetaan aina kun vastaanotetaan kuittauksia perille menneistä segmenteistä, eikä ruuhkautumisesta ole mitään merkkejä. Vastaavasti lähetysikkunan kokoa pienennetään mikäli verkko tulkitaan ruuhkautuneeksi. Lähetysikkunan additiivinen kasvatus tarkoittaa ikkunan koon kasvattamista yhden segmentin verran jokaista vastaanotettua ACK-viestiä kohden. Multiplikatiivinen lähetysikkunan pienennys tarkoittaa esimerkiksi TCP-Renon yhteydessä ikkunan koon puolittamista. AIMD-algoritmille tunnusomaista on ikkunan koon sahalaitainen käyttäytyminen.

Kuvissa 1 ja 2 nähdään vektoridiagrammina kahden saman linkin jakavan yhteyden resurssiosuuksien muutoksia eri algoritmeilla. X ja Y -akselit kuvaavat kahden eri käyttäjän saamaa osuutta yhteisen linkin kapasiteetista. Vasemmasta alanurkasta oikeaan ylänurkkaan menee niin sanottu tasavertaisuus linja, eli kyseisellä linjalla käyttäjät saavat yhtä suuren osuuden resursseista. Vasemmasta ylänurkasta oikeaan alanurkkaan menevä tehokkuuslinja kuvastaa tasoa, jolla kahden käyttäjän yhteen-

laskettu kuorma hyödyntää koko linkin kapasiteetin. Mitä lähempänä kyseistä linjaa ollaan, sitä parempi on systeemin hyötysuhde [3].

Kuvasta 1 voidaan huomata kuinka perinteinen AIMD-algoritmi hakeutuu kohti tasavertaisuuslinjaa. Tämän jälkeen siirrytään edestakaisin tehokkuuslinjan eri puolin ja eikä stabiilia tilaa saavuteta koskaan. Tämä johtuu palautteen binaarisesta luonteesta.

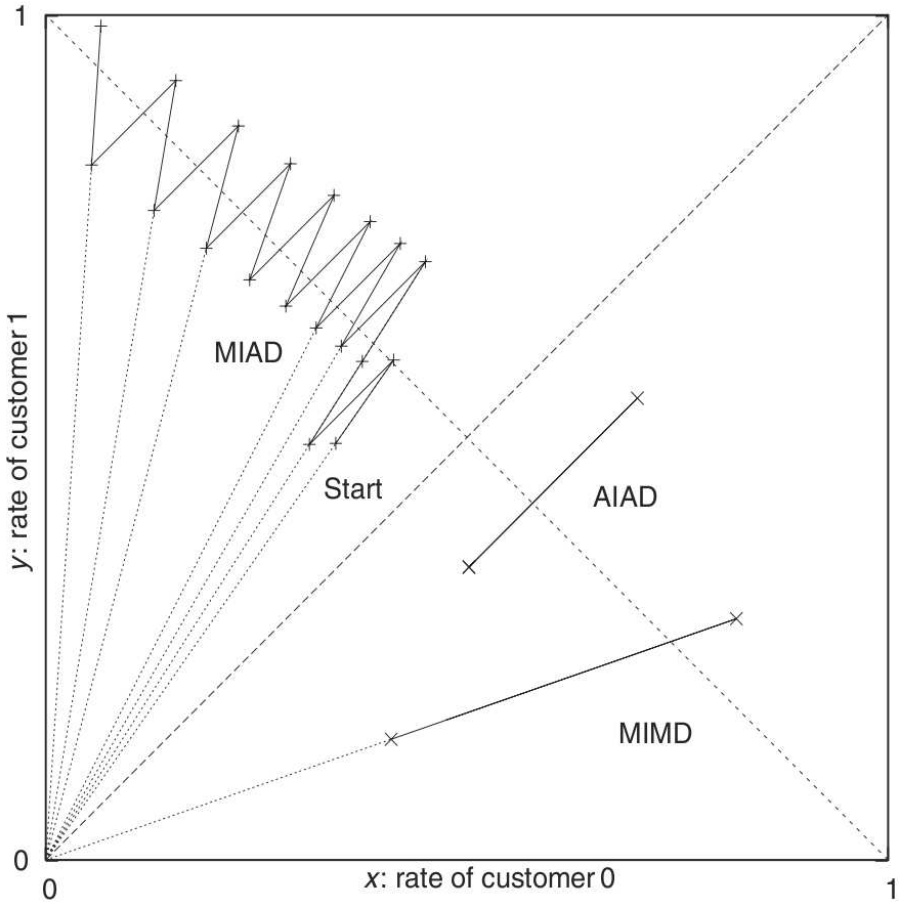


Kuva 1: AIMD [2]

Ensimmäinen yleisesti tunnettu AIAD-algoritmiä käyttävä kuljetuskerroksen protokolla on TCP Vegas. Kuvasta 2 voidaan huomata kuinka AIAD:n vektori on 45 asteen kulmassa, jossa tila menee edes takaisin kyseistä linjaa. Ilmiö johtuu käyttäjien lineaarisesta ikkunan koon kasvattamisesta ruuhkautumistilanteeseen asti. Tämän jälkeen ikkunan kokoa pienennetään samassa suhteessa kuin sitä kasvatettiinkin. Sama ilmiö toistuu MIMD:n tapauksessa, mutta tällä kertaa kuljetaan vektoria, jonka jatke lävistää origon. Origon lävistävillä ns. equi fairness -linjoilla käyttäjien saama suhteellinen osuus pysyy koko ajan samana. AIAD:n vektori ei osoita origoon ja näin ollen kuvan 2 tapauksessa x-akselin saama suhteellinen osuus kasvaa mitä lähempänä origoa ollaan.

Kaikista neljästä algoritmista epäreiluin on MIAD, joka lähestyy kohti tilannet-

ta jossa alunperin suuremman osuuden saanut vie pian kaikki resurssit. Tilanne on juuri päinvastainen verrattuna AIMD:hen, jossa kaikista alkutilanteista hakeudutaan kohti tehokkuus- ja tasavertaisuuslinjojen leikkauspistettä.



Kuva 2: AIAD, MIAD ja MIMD [2]

Edellä käsiteltyjen algoritmien matemaattinen kuvaus binäärisen palautteen vallitessa ja mikäli vain yhtä komponentin arvoa voidaan muuttaa kerrallaan onnistuu seuraavasti. Yhtälö 1 määrittelee lähetysnopeuden x(t) ajan hetkellä t. y(t) kuvastaa verkosta saatua binaarista palautetta verkon tilasta, jonka arvo nolla kuvastaa ruuhkautumatonta tilaa ja arvo yksi ruuhkaa. Vakiot a_i, b_i, a_d ja b_d ovat vakioita, joiden avulla määritellään funktion additiiviset ja multiplikatiiviset ominaisuudet.

$$x(t+1) = \begin{cases} a_i + b_i x(t) & \text{if } y(t) = 0 \\ a_d + b_d x(t) & \text{if } y(t) = 1 \end{cases} \tag{1}$$

- $a_i > 0;\ a_d = 0;\ b_i = 1;\ 0 < b_d < 1$
  Additive Increase, Multiplicative Decrease (AIMD)
- $a_i > 0;\ a_d < 0;\ b_i = 1;\ b_d = 1$
  Additive Increase, Additive Decrease (AIAD)

- $a_i = 0;\ a_d < 0;\ b_i > 1;\ b_d = 1$
  Multiplicative Increase, Additive Decrease (MIAD)

- $a_i = 0;\ a_d = 0;\ b_i > 1;\ 0 < b_d < 1$
  Multiplicative Increase, Multiplicative Decrease (MIMD)

2.6 Hidas aloitus ja ruuhkan välttely

Hidas aloitus (Slow Start) -algoritmi on yksi Van Jacobsonin vuonna 1988 esittelemistä ruuhkanhallintatoiminnoista[10]. Se perustuu TCP-yhteyden segmenttien lähetysnopeuden säätelyyn saapuvien ACK -kuittauksien perusteella. Hidas aloitus määrittelee lähettäjälle uuden lähetysikkunan, joka on nimeltään congestion window (cwnd). Cwnd on lähettävän pään rajoitin datan määrälle, joka verkkoon voidaan lähettää ennen kun on vastaanotettu kuittauksia perille menneistä segmenteistä. Vastaavasti vuonhallinnassa käytetty receiver's advertised window (rwnd) on vastaanottavan pään ilmoittama rajoitin lähettäjälle, jolla se kertoo kuinka nopeasti se on valmis vastaanottamaan dataa. TCP-lähettäjä määrittelee lähetysikkunan kooksi sen muuttujista cwnd tai rwnd kumpi niistä saa pienemmän arvon.

$$wnd = min(cwnd, rwnd) \tag{2}$$

Cwnd:n alkuarvo (initial value, IW) on korkeintaan kaksi kertaa SMSS:s koko tavuina eikä se saa olla enempää kuin kaksi segmenttiä. On olemassa joitain epästandardeja TCP-versioita, joissa IW voi saada suurempia arvoja. TCP-yhteyden muodostuksen jälkeen cwnd:n arvoa kasvatetaan jokaista vastaanotettua ACK -viestiä kohden yhdellä segmentillä. Lähetysikkuna kasvaa tässä vaiheessa hyvin nopeasti aina niin kauan kunnes vastaanottaja ei saa ACK-kuittausta kaikista lähetysikkunan segmenteistä.

Uudelleenlähetysajastimen laukeaminen tulkitaan signaaliksi verkon ruuhkautumisesta. Tämä aiheuttaa cwnd:n arvon palauttamisen takaisin alkuarvoonsa. Tästä alkaa cwnd:n kasvatus uudelleen yhdellä jokaista vastaanotettua ACK-viestiä kohden.

TCP-yhteyden muodostuksen alussa ei lähettäjällä ole mitään tietoa verkon ominaisuuksista johon dataa ollaan lähettämässä. Hitaan aloituksen tapa aloittaa datan lähetys hyvin pienellä lähetysikkunan koolla mahdollistaa lähettäjän sopeutumisen verkon resursseihin siten ettei mahdollisesti ruuhkauteta verkkoa liian aggressiivisella lähetysnopeudella. Myös verkon ruuhkaannuttua ja RTO:n lauettua cwnd:n tiputtaminen takaisin yhteen segmenttiin estää lähettäjää kuormittamasta ruuhkautunutta verkkoa entisestään.

Ruuhkan välttelyä käytetään cwnd arvon ollessa suurempi kuin slow start threshold (ssthresh). Ssthresh määrittelee onko TCP-yhteys hitaan aloituksen vai ruuhkan välttelyn tilassa. Ssthresh parametrin alkuarvo on 64 kilotavua.

Ruuhkan välttely tilassa cwnd:n kokoa ei enää kasvateta yhtä aggressiivisesti kuin hitaan aloituksen tilassa ja cwnd:tä kasvatetaan korkeintaan yhdellä segmentillä yhden RTT:n aikana.

$$cwnd = cwnd + SMSS * SMSS/cwnd/BaseRTT \quad (3)$$

Kun TCP protokolla huomaa segmentin kadonneen tulee ssthresh määritellä uudelleen, jolloin sen maksimiarvo määritellään kaavan x mukaisesti. Flight size on lähetettyjen pakettien määrä joita ei ole vielä kuitattu.

$$ssthresh = max[2 * FlightSize, 2 * SMSS] \quad (4)$$

Segmenttien katoaminen aiheuttaa cwnd:n asettamisen takaisin alkuarvoonsa myös siinä tapauksessa että TCP-yhteys on ruuhkan välttely moodissa.

2.7 Nopea uudelleenlähetys ja toipuminen

Perinteinen TCP-algoritmi käyttää pakettien katoamisten havaitsemiseen niin sanotun uudelleenlähetysajastimen laukeamista (retransmission timeout, RTO). TCP asettaa jokaiselle lähetetylle paketille ajan, jonka kuluessa lähettäjä odottaa saavansa kuittauksen paketin perille menosta. TCP ei tiedä pakettien mahdollisista katoamisista mitään ennen kuin RTO laukeaa ja näin ollen jokaisen katoamisen toteamiseen kuluu paljon aikaa.

RTO:n laukeamiseen perustuva uudelleenlähetysmekanismi toimii tehottomasti etenkin suurilla lähetysikkunoilla. Tämän takia TCP-algoritmiin on lisätty niin sanottu nopea uudelleenlähetys -mekanismi (fast retransmit), joka käyttää pakettihukan toteamisessa hyödykseen saapuvia ACK-viestejä. TCP-yhteyden vastaanottaja kuittaa jokaisen vastaanotetun segmentin ACK-viestillä, jossa se ilmoittaa seuraavaksi odottamansa segmentin numeron. Näin ollen paketin katoaminen verkossa saa vastaanottajan kuittaamaan kadonneesta paketista seuraavat segmentit kadonneen paketin segmenttinumerolla. Tätä tietoa hyväksi käyttäen lähettäjä voi huomata paketin kadonneen tai niiden saapuneen perille poikkeavassa järjestyksessä. Jotta paketin katoaminen voitaisiin erottaa järjestyksen muutoksesta, vastaanottaja odottaa useamman duplikaatin ACK-viestin vastaanottamista ennen kuin paketti todetaan kadonneeksi. Nopea uudelleenlähetys ei aina pysty toteamaan paketin katoamista RTO:n laukeamista aiemmin ja näin ollen molempia mekanismeja käytetään samanaikaisesti. Esimerkiksi pienillä lähetysikkunan koilla nopea uudelleenlähetys ei toimi ja tällöin joudutaan toimimaan täysin RTO:n varassa.

Nopea uudelleenlähetys aiheuttaa TCP Tahoessa siirtymisen hitaaseen aloitukseen sekä cwnd:n pienentämisen alkuarvoonsa, eli täysin samat toimenpiteet jotka seuraavat RTO:n laukeamisesta. Kyseisillä toimilla tyhjennetään yhteys lähetyksessä olevista paketeista, joka aiheuttaa turhan voimakkaan suoritustason laskun. TCP Reno:ssa sekä sitä uudemmissa TCP-varianteissa käytetään kolmen duplikaatin ACK-viestin vastaanottamisen jälkeen nopean toipumisen -mekanismia (fast recovery). Nopeassa toipumisessa lähetysikkunaa ei pienennetä yhteen vaan sen sijaan toimitaan seuraavasti:

1 Asetetaan ssthresh = cwnd/2

2 Lähetetään puuttuva segmentti uudelleen

3 Asetetaan cwnd = ssthresh +3 (kolme segmenttiä on mennyt perille)

4 Kasvatetaan cwnd:tä yhdellä jokaista duplikaattia ACK:ta kohden

5 Kun vastaanotetaan kuittaus uudelleenlähettämättömästä segmentistä, asetetaan cwnd = ssthresh

2.8 TCP-ystävällisyys

Nykyajan tietoverkoissa protokollan tasapuolisuus määritellään usein niin sanotulla TCP-ystävällisyydellä (TCP friendliness). Käytännössä tämä ominaisuus kertoo kuinka aggressiivisesti kyseinen protokolla kohtelee muita TCP:n tavoin ruuhkautumiseen reagoivia datavirtoja. Mikäli protokolla on TCP-vihamielinen, voi seurauksena olla muiden TCP:n tavoin käyttäytyvien yhteyksien tukahtuminen pois kyseisen röyhkeän verkkosovelluksen tieltä. Esimerkiksi User Datagram Protocol (UDP) on hyvin yleinen kuljetuskerroksen protokolla, joka ei säätelee lähetysnopeuttaan ruuhkautumistilanteissa millään tavalla. Seurauksena voi olla TCP:n totaalinen tukahtuminen UDP-yhteyksien tieltä ja täten verkon toiminnan vakava häiriintyminen.

2.9 Teoreettinen maksimiläpivienti

Tietoliikenteessä käytetään linkin liityntänopeudesta termiä kaistanleveys (bandwidth). Se kuvastaa kanavan siirtokapasiteetin maksimia ja se ilmoitetaan yleensä muodossa bittiä sekunnissa (bps).

Läpiviennillä (throughput) tarkoitetaan verkon läpi kuljetetun fyysisen datan siirtonopeutta. Läpivienti pitää sisällään kaiken mitä verkon läpi kulkee, eli varsinainen lähetettävä data sekä esimerkiksi protokollien kehykset ja uudelleenlähetetyt segmentit. Läpivienti ei voi ylittää kaistanleveyden nopeutta.

Tehollinen läpivienti (goodput) määrittelee kuinka paljon alkuperäistä lähetettävää tietoa saadaan siirrettyä yhteyden yli aikayksikköä kohden. Yleisin tapa ilmoittaa tämäkin siirtosuure on muodossa bittiä sekunnissa.

Analogisen tietoliikennekanavan maksimaalisen läpiviennin perusyhtälönä voidaan pitää niin sanottua Shannon-Hartley -teoreemaa, joka määrittelee kuinka paljon virheetöntä tietoa on mahdollista kuljettaa yhteyden läpi tietyllä kaistanleveydellä ja signaali-kohina suhteella (Signal to noise ratio, SNR). Shannonin teoreeman antamaa tulosta ei pystytä ylittämään missään olosuhteissa ja on kyse systeemin tehokkuudesta kuinka lähelle sen arvoa käytännössä päästään. Yhtälössä 5 V_s on bittikaistanleveys (bittiä/sekunti), B on kaistanleveys (Hz) ja S/N on signaali-kohina suhde.

$$V_s = B * log_2(1 + S/N) \tag{5}$$

Kaistanleveys-viive tulo (Bandwidth-delay Product, BDP) kertoo kuinka paljon esimerkiksi TCP yhteydelle mahtuu kuittaamattomia segmenttejä, eli kuinka suuri on maksimaalinen datamäärä mitä verkossa voi olla samanaikaisesti. Saavuttaakseen

mahdollisimman suuren hyötysuhteen tulee verkko olla mahdollisimman suuren osan ajasta niin sanotusti täynnä. Hyvin suuria BDP arvoja saavia linkkejä kutsutaan termillä "long fat pipes"(LFP). Hyvä esimerkki tällaisesta on satelliittilinkit, joilla RTT sekä kaistanleveys voivat saada hyvinkin suuria arvoja.

$$BDP = Kaistanleveys * RTT \tag{6}$$

Ikkunointiin perustuvaa vuon- ja ruuhkanhallintaa käyttävien protokollien maksimiläpivientiin vaikuttaa maksimi ikkunan koko. Esimerkiksi TCP-protokollan maksimi vastaanottoikkunan koko on 65535 tavua, joka määritellään 16-bittisessä TCP Window Size -kentässä. Nykyisin on mahdollista käyttää myös suurempia rwnd:n arvoja käyttämällä skaalautuvia ikkunoita (RFC1323). Perinteinen 64 kilotavun vastaanottoikkuna rajoittaa esimerkiksi 10 millisekunnin viiveen omaavan linkin maksimi läpivienniksi 52 Mbps kaavan 7 mukaisesti. Näin ollen kauttakiertokulkuajan kasvun voidaan todeta pienentävän suurimman mahdollisen läpiviennin arvoa linearisesti.

$$Maksimilapivienti \leqslant \frac{RWIN}{RTT} \tag{7}$$

Pakettien katoaminen aiheuttaa läpiviennille myös ylärajan. Niin sanottu Mathiksen approksimaatio (kaava 8) antaa arvion TCP:n läpiviennistä tietyllä pakettihukan todennäköisyydellä P_l ja vakiolla C, joka saa tyypillisesti arvokseen $\sqrt{3/2b}$. Yhtälössä käytetty b on vakio, joka kertoo kuinka monta segmenttiä yksi ACK-viesti kuittaa kerralla. Perinteinen TCP kuittaa yhden segmentin jokaista ACK-viestiä kohden, mutta uudemmat viivästetyn kuittauksen algoritmit voivat saada b:lle arvon kaksi. Mathiksen malli ottaa huomioon TCP ruuhkan välttely ja nopean uudelleenlähetyksen mekanismit[7].

$$B(RTT, p_l)_M = \frac{C}{RTT\sqrt{P_l}} \tag{8}$$

Mathiksen approksimaatiota monimutkaisempi ja tarkempi malli on Padhye malli (kaava 9). Se ottaa huomioon myös nopean uudelleenlähetyksen sekä RTO:n ruuhkanvälttämis mallissaan[8].

$$B(RTT, p_l)_P = min\{\frac{W_m}{RTT}\ , \frac{1}{RTT\sqrt{\frac{2bp_l}{3}}\ + T_0 min(1, 3\sqrt{\frac{3bp_l}{8}}\ )p_l(1 + 32p_l^2)}\ \} \tag{9}$$

2.10 Ruuhkanhallinta-algoritmit langattomissa verkoissa

Kuljetuskerroksen TCP -protokolla on kehitetty kauan ennen nykyisiä langattomia lähiverkkoja ja mobiileja verkkoyhteyksiä. TCP:n ruuhkanhallintaominaisuudet on suunniteltu toimimaan optimaalisesti kaapeleiden välityksellä tapahtuvassa tiedonsiirrossa. Tiedonsiirrossa tapahtuva segmenttien katoaminen on varsin harvinaista kiinteissä verkoissa verrattuna langattomiin verkkoihin, joissa vaimentuminen, häipyminen (fading), varjostuminen (shadowing) ja muut radiotien ilmiöt aiheuttavat

segmenttien korruptoitumista ja katoamista. Perinteiset ruuhkanhallinta-algoritmit eivät pysty erottamaan radiotiellä sattuvia siirtovirheitä ruuhkautumisesta ja näin ollen ne virheellisen tulkinnan takia aiheuttavat vastaavat toimet kuin ruuhkautuminen. Radiotien erilaisuus siirtotienä aiheuttaa linkkikapasiteetin alikäyttöä, koska esimerkiksi TCP-yhteys tulkitsee verkon ruuhkautuneeksi jo pienestäkin pakettihukasta [3]. Kyseisen ilmiön vaikutuksen suuruutta eri kuljetuskerroksen algoritmeihin tutkitaan tarkemmin tämän diplomityön simulaatioissa.

3 Kuljetuskerroksen verkkoprotokollat

Tietoliikennetekniikassa kuljetuskerroksella tarkoitetaan joukkoa protokollia ja mekanismeja, joiden tehtävänä on kapseloida sovellustason data segmentteihin ja datagrammeihin ja näin mahdollistaa tiedon siirron sovellukselta toiselle. Kuljetuskerroksen tehtäviin kuuluvat myös esimerkiksi ruuhkan- ja vuonhallinta sekä taata mahdollisesti luotettava tiedonsiirto yhteyden päästä päähän.

Yleisesti tunnettuja kuljetuskerroksen protokollia ovat muun muassa Transmission Control Protocol (TCP), User Datagram Protocol (UDP) sekä Datagram Congestion Control Protocol (DCCP). Nykyisissä tietoverkoissa käytetään pääasiassa TCP- ja UDP-protokollia [11]. Tämän tutkimuksen kohteena ovat ruuhkanhallinnallisia ominaisuuksia sisältävät TCP sekä DCCP.

Taulukko 1: Kuljetuskerroksen protokollat

	TCP	UDP	DCCP
Paketin tyyppi	Segmentti	Datagrammi	Datagrammi
Luotettava siirto	Kyllä	Ei	Ei
Vuonhallinta	Kyllä	Ei	Ei
Ruuhkanvälttely	Kyllä	Ei	Kyllä

Tässä luvussa perehdytään TCP- ja DCCP-protokollien toimintaan sekä viiden eri TCP- ja kahden DCCP-version ominaisuuksiin. Tutkimuksen kohteeksi on valittu yleiset TCP-variantit Tahoe, Reno, New Reno, Sack sekä Vegas. DCCP-variantteja on kehitetty useita erilaisia ja ne nimetään ruuhkanhallinnallisten ominaisuuksiensa perusteella eri Congestion Control Identifier (CCID) -tunnuksella. Tähän tutkimukseen on valittu CCID-2 (TCP Like) sekä CCID-3 (TFRC).

3.1 TCP

Transmission Control Protocol (TCP) on yksi alkuperäisistä Internetin protokollista. Se takaa luotettavan (reliable) tiedonsiirron epäluotettavan (unreliable) IP-pohjaisen verkon yli, eli se huolehtii siirrossa kadonneiden pakettien lähettämisestä uudelleen kunnes ne saadaan onnistuneesti perille[18]. TCP-protokollasta on kehitetty useita eri versioita, jotka poikkeavat toisistaan esimerkiksi ruuhkanhallintamekanismiensa osalta.

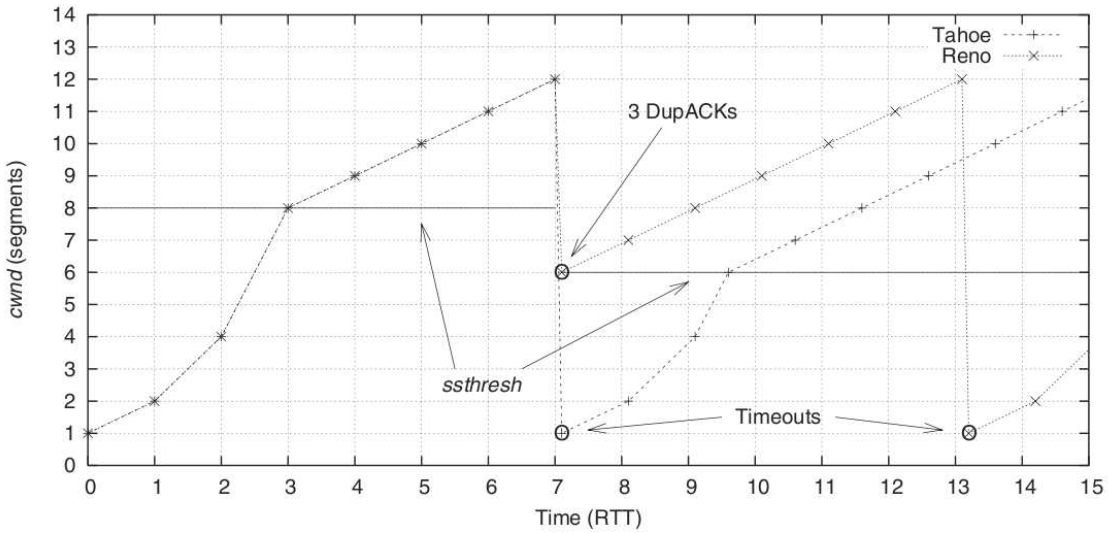
3.1.1 Tahoe

TCP Tahoe käyttää hidasta aloitusta, ruuhkan välttämistä sekä nopeaa uudelleenlähetystä. Ruuhkaikkunaa kasvatetaan eksponentiaalisesti kunnes sen koko saavuttaa ssthresh:n arvon. Tämän jälkeen siirrytään ruuhkan välttely -tilaan, jossa cwnd:n koko kasvaa yhdellä jokaista RTT:tä kohden. TCP-yhteys siirtyy nopean uudelleenlähetyksen -tilaan, mikäli lähettäjä vastaanottaa kolme samaa kuittausta. Sen aikana lähetetään kadonnut segmentti uudelleen ja siirrytään takaisin hitaaseen aloitukseen sekä asetetaan cwnd takaisin alkuarvoonsa, joka on yleensä yksi. Ssthreshin koko asetetaan puoleen cwnd:n arvosta ajanhetkellä kun verkon ruuhkautuminen havaittiin [19].

3.1.2 Reno

TCP Reno poikkeaa Tahoesta siihen lisätyn nopean toipumisen -mekanismin osalta. TCP-lähettäjän vastaanottaessa tietyn määrän (yleensä 3) duplikaatteja kuittauksia se siirtyy nopean toipumisen tilaan hitaan aloituksen sijaan. Nopea toipuminen aloitetaan asettamalla ssthreshin arvoksi puolet cwnd:stä ja cwnd saa arvokseen uuden ssthreshin lisättynä vastaanotettujen duplikaattien ACK-viestien määrällä. TCP Reno pysyy nopean toipumisen -tilassa, niin kauan kunnes nopean uudelleenlähetyksen laukaissut segmentti saadaan kuitattua perille menneeksi.

Vastaanottajan saadessa uusia ACK-viestejä se siirtyy nopean toipumisen -tilasta takaisiin ruuhkan välttelyyn ja cwnd muutetaan samaan arvoon ssthreshin kanssa. Nopean toipumisen avulla cwnd pysyy keskimäärin suurempana kuin TCP Tahoella, jonka myötä yhteyden suorituskyky on parempi.



Kuva 3: Cwnd:n kehittyminen TCP Tahoella ja Renolla [1]

TCP Reno toimii hyvin, mikäli segmenttejä katoaa melko harvoin. Yhteyden laadun ollessa hyvin huono paketteja voi kadota kuitenkin useampiakin saman lähetysikkunan aikana. Tällöin TCP Renon suorituskyky ei ole hyvä, sillä se voi huomata

vain yhden segmentin katoamisen kerralla. Yhden paketin kadottua lähettäjä vastaanottaa useampia kuittauksia samalla segmenttinumerolla. Mikäli kadoksissa olevan segmentin ja jo vastaanotettujen segmenttien välistä puuttuu yksi tai useampi segmentti, niin tästä saadaan tietä vasta sen jälkeen kun ensimmäisenä kadonneesta segmentistä saadaan kuittaus. Näin ollen toisena kadonneesta segmentistä saadaan tieto vähintään RTT:n pituisen viiveen jälkeen. Usean segmentin katoaminen samasta lähetysikkunasta voi aiheuttaa myös cwnd:n pienentämisen useammin kuin yhden kerran. Tämä nopean toipumisen ja nopean uudelleenlähettämisen toistuva vaihtelu huonontaa TCP Renon toimintaa hyvin radikaalisti .

TCP Reno tarvitsee nopean uudelleenlähetyksen toimintaan tarpeeksi suuren lähetysikkunan, koska lähettäjän tulee vastaanottaa tarpeeksi monta duplikaattia ACK-viestiä huomatakseen jonkin segmentin kadonneen. Näin ollen pienen lähetysikkunan tapauksessa lähettäjä joutuu odottamaan RTO:n laukeamista ennen kuin se toteaa segmentin kadonneeksi. Tämä ominaisuus yhdistettynä cwnd:n mahdolliseen pienentämiseen useita kertoja yhden ikkunan aikana aiheuttaa sen että TCP Reno ei toimi tehokkaasti mikäli segmenttejä katoaa hyvin paljon [3].

3.1.3 New Reno

New Reno on paranneltu versio TCP Reno:sta. New Reno:ssa siirrytään Renon tapaan nopeaan uudelleenlähetykseen, mikäli vastaanotetaan tarpeeksi monta duplikaattia ACK-viestiä. Uutena ominaisuutena New Reno:ssa poistutaan nopeasta toipumisesta vasta kun kaikkiin segmentin katoamisen aikana lähetettyinä olleisiin segmentteihin saadaan kuittaukset. New Renon nopea toipuminen siirtyy ruuhkan välttely -tilaan, mikäli lähettäjä vastaanottaa ACK-viestin, joka kuittaa kaikki lähetyksessä olleet segmentit. Mikäli vastaanotettu ACK on niin sanottu osittainen kuittaus, niin seuraavaksi lähetetään ensimmäinen kuittaamaton segmentti. Tällöin myös cwnd:tä pienennetään ACK:n kuittaamalla datamäärällä ja erotukseen lisätään vielä yksi SMSS. Tämän jälkeen jatketaan toipumisvaihetta. Tällä mekanismilla vältytään Reno:ssa esiintyvältä ongelmalta, jossa cwnd:tä saatetaan pienentää useita kertoja saman lähetysikkunan aikana[21].

New Renon ongelmana on että jokaisen kadotetun paketin huomaaminen kestää yhden RTT:n verran. Vasta sen jälkeen kun ensimmäinen kadotettu paketti saadaan uudelleenlähetettyä perille, voidaan huomata mikä on seuraava mahdollisesti kadotettu segmentti.

3.1.4 Sack

TCP with Selective Acknowledgements (SACK) käyttää ns. SACK-optiota ACK-viesteissään, jonka avulla lähettäjälle ilmoitetaan pakettihukan tai muun syyn takia epäjärjestyksessä saapuneista segmenteistä. Selektiivinen kuittaus mahdollistaa usean puuttuvan segmentin identifioinnin kullakin toistokuittauksella. Lähettäjä ylläpitää taulukkoa, joka sisältää tiedon normaalisti vastaanotetuista sekä epäjärjestyksessä vastaanotetuista segmenteistä. Taulukon sisältämän tiedon perusteella uudelleen lähetetään kuittaamattomat segmentit odottamatta RTO:n laukeamista.

Vasta sen jälkeen kun taulukossa olevat kuittaamattomat aukot on täytetty, voidaan lähettää täysin uusia segmenttejä [23].

TCP SACK ylläpitää kuittaamattomien segmenttien määrää muuttujassa nimeltään *pipe*.Nopean toipumisen aikana lähetetään uusia tai uudelleen lähetettäviä segmenttejä vain jos *pipen* arvo on cwnd:tä pienempi. *Pipen* arvoa kasvatetaan yhdellä segmentin lähetyksen yhteydessä ja pienennetään yhdellä kun vastaanotetaan duplikaatti ACK-viesti SACK-optiolla. Osittaisen ACK:n vastaanotto pienentää *pipen* arvoa kahdella. Fast recovery tilasta poistutaan kun vastaanotetaan ACK-viesti, joka kuittaa kaikki lähetettyinä olleet segmentit siirryttäessä nopean toipumisen -tilaan.

TCP SACK eroaa TCP Reno:sta/New Reno:sta lähinnä nopean toipumisen -mekanismin osalta. TCP SACK pystyy käsittelemään useiden pakettien hukkumisen samasta lähetysikkunasta New Renoa paremmin[4].

3.1.5 Vegas

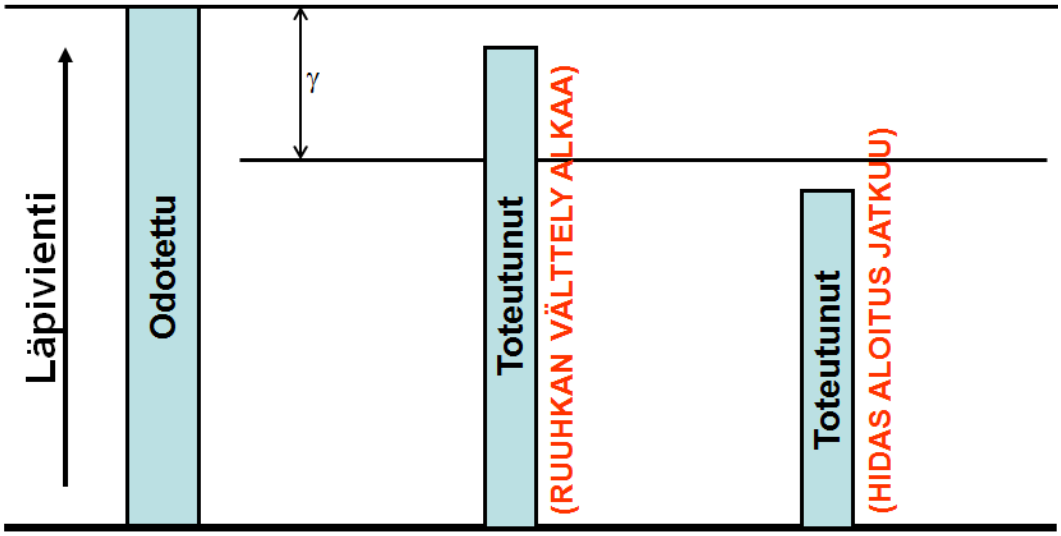
TCP Vegas esiteltiin jo vuonna 1994, eli ennen New Reno:n ja SACK:n kehittämistä[5]. TCP Vegas poikkeaa kaikista muista TCP-varianteista mekanismilla jolla se toteaa pakettien hukkuneen. Vegas ei odota RTO:n laukeamista pienentääkseen cwnd:tä vaan se tarkkailee odotetun ja toteutuneen läpiviennin suhdetta. Mikäli verkkoon alkaa kehittyä ruuhkaa, niin toteutunut läpivienti alkaa saada odotettua läpivientiä pienempiä arvoja. Odotettu läpivienti lasketaan kaavan 10 mukaisesti, jossa BaseRTT on pienin mitattu RTT:n arvo.

$$Odotettu\ lapivienti = \frac{cwnd}{BaseRTT} \tag{10}$$

Toteutunut läpivienti lasketaan mittaamalla segmentin RTT ja laskemalla kyseisenä aikana lähetettyn datan määrä. Toteutuneen ja odotetun läpiviennin erotusta varten käytetään muuttujaa *Diff* jonka arvoa vertaillaan parametreihin α ja β. Mikäli *Diff* on pienempi kuin α, niin cwnd:tä kasvatetaan lineaarisesti seuraavan RTT:n aikana. Vastaavasti jos *Diff* on arvoltaa yhtä suuri tai suurempi kuin β, niin cwnd:tä pienennetään lineaarisesti. Parametrit α ja β asetetaan yleensä arvoihin 2 ja 4, jotka määrittelevät kuinka paljon verkossa on kuittaamattomia segmenttejä[4].

TCP Vegas käyttää muunneltua hitaan aloituksen -algoritmia. Alkuperäinen hidas aloitus ja ruuhkan välttely -mekanismit tarvitsevat segmenttien katoamisia todetakseen ruuhkautumisen alkaneen. TCP Vegasin hidas aloitus pyrkii löytämään oikean cwnd:n koon ennen kuin segmenttien katoaminen alkaa. Tämä toteutetaan lisäämällä cwnd:n kokoa eksponentiaalisesti joka toisella RTT:llä ja mittaamalla muuttujan *Diff* arvo korotusten välissä. Vegas siirtyy hitaasta aloituksesta ruuhkan välttely tilaan ennen kuin toteutunut läpivienti saavuttaa odotetun läpiviennin arvon.

Vegas käyttää myös hieman muunneltua segmenttien uudelleenlähetysstrategiaa. Segmentti lähetetään uudelleen jo yhden duplikaatin ACK-viestin jälkeen mikäli RTT:n estimaatio on suurempi kuin RTO. Tämä auttaa niissä tilanteissa jolloin vastaanottaja ei milloinkaan vastaanottaisi kolmea duplikaattia ACK-viestiä, eli esi-



Kuva 4: TCP Vegasin hidas aloitus ja ruuhkan välttely

merkiksi silloin kun lähetysikkunasta katoaa useita segmenttejä tai ikkunan koko on hyvin pieni.

3.2 DCCP

Kuljetuskerroksen protokollat voidaan jaotella luotettaviin ja epäluotettaviin protokolliin, joista jälkimmäisenä mainitut eivät takaa pakettien perille menoa. On kuitenkin olemassa useita verkkopalveluita, joiden käyttö on enemmän riippuvainen pakettien nopeasta perille viennistä pienellä viiveellä kuin siitä että kaikki paketit saadaan toimitetuksi. Esimerkiksi IP-puheluiden ja videoiden streamauksen palvelutaso eivät juurikaan kärsi vaikka osa yhteyden paketeista katoaisikin.

User Datagram Protocol (UDP) on yleisesti käytetty kuljetuskerroksen protokolla multimediasovelluksille. Se ei tarjoa minkäänlaisia ruuhkanhallintaominaisuuksia, joten sovelluksen konservatiivinen (ns. TCP-friendly) käyttäytyminen on täysin kiinni sovelluksen kehittäjästä. UDP kehitettiin aikoinaan hyvin lyhyiden datavoiden käyttöön, kuten esimerkiksi DNS-kyselyihin. Tästä johtuen ruuhkanhallintaa ei otettu aikoinaan huomioon kyseistä protokollaa kehitettäessä. UDP on hyvin paljon käytetty protokolla esimerkiksi VoIP, musiikki- ja video-sovellusten yhteydessä. UDP:n reagoimattomuus verkon ruuhkautumiseen aiheuttaa sen että TCP-yhteydet jäävät oman ruuhkanhallintansa myötä UDP-liikenteen dominoimaksi.

IETF on kehittänyt UDP:lle korvaajaa, joka täyttäisi nykyisten UDP:ta käyttävien sovellusten vaatimukset, mutta samalla sisältäisi myös ruuhkanhallinnallisia ominaisuuksia. Työn tuloksena on syntynyt Datagram Congestion Control Protocol (DCCP) [12].

DCCP:n voidaan katsoa olevan käytännössä UDP:n kaltainen protokolla, johon on lisätty ruuhkanhallinta, kättely ja ACK-kuittaukset. DCCP aloittaa session aina yhteyden muodostuksella (setup) ja päättää sen yhteyden purkamiseen (teardown). Näin toimitaan siitäkin huolimatta että DCCP ei takaa luotettavuutta, koska tällä tavoin mahdollistetaan läpipääsy useista palomuureista, jotka normaalisti hylkäävät UDP-liikenteen [2].

DCCP:tä varten on kehitetty joukko erilaisia ruuhkanhallintamekanismeja, joille on annettu tunnisteeksi niin sanottu Congestion Control Indentification (CCID). Tällä hetkellä on olemassa muutamia eri CCID-mekanismeja, joista tämän työn piiriin on valittu CCID-2: TCP-Like sekä CCID-3: TCP Friendly Rate Control (TFRC). Näiden lisäksi IETF on standardoinut CCID-4:n joka on nimeltään VoIP Variant of TFRC.

3.2.1 TCP-like Congestion Control

TCP-Like Congesstion Control (CCID-2) on nimensä mukaisesti hyvin TCP:n kaltainen ruuhkanhallintamekanismi ja se käyttää AIMD-tyylistä säätelyä lähetysikkunan kokoon. Suurimpana erona TCP:n ja CCID-2:n välillä voidaan pitää sitä ettei TCP-Like koskaan uudelleenlähetä paketteja. Tämän lisäksi protokollan parametrit, kuten ikkunoiden koot, määritellään CCID-2:ssa paketteina eikä tavuina kuten TCP:n yhteydessä tehdään.

TCP-Like soveltuu erityisen hyvin sovelluksiin, joiden liikenne on hyvin purskeista ja vastaanotettu data puskuroidaan ennen sen käyttöä sovellustasolla [13]. Esimerkkeinä tällaista sovelluksista mainittakoon tilausvideo-palvelut (VoD) ja Internet radiot.

3.2.2 TCP Friendly Rate Control

TCP Friendly Rate Control on ruuhkanhallintamekanismi, jota voidaan käyttää eri kuljetuskerroksen protokollien kanssa. DCCP:n ja TFRC:n yhdistelmää kutsutaan CCID-3:si. TFRC pyrkii olemaan UDP:tä tasapuolisempi verkon TCP-yhteyksiä kohtaan. Tähän pyritään säätelemällä lähetysnopeutta Padhyen -yhtälön mukaisesti (yhtälö 9).

TFRC:ssä vastaanottajan tehtävänä on laskea niin sanottu "loss event rate", joka tarkoittaa todennäköisyyttä että RTT:n aikana sattuu vähintään yhden paketin katoaminen. TFRC:n yhteydessä lähettäjä kasvattaa jokaisen lähetettävän paketin järjestysnumeroa yhdellä. Näin toimitaan myös niillä protokollilla joilla uudelleenläetykset ovat mahdollisia. Vastaanottaja tulkitsee paketin kadonneeksi mikäli se vastaanottaa paketin, jonka järjestysnumero on vähintään kolme numeroa suurempi kuin vielä vastaanottamaton mahdollisesti kadotetun paketin järjestysnumero. Vastaanottaja tehtävänä on määritellä pakettihukan perusteella loss event rate, jota varten sen tulee tietää RTT:n saama arvo mahdollisimman tarkasti. Loss event rate saa pakettihukkaa pienempiä arvoja ja tämä ero kasvaa pakettihukan kasvun myötä. Tämä johtuu siitä että usean paketin hukkuminen yhden RTT:n aikana tulkitaan yhdeksi hukkatapahtumaksi. RTT määritellään yhteyden lähettäjän päässä, josta se sitten toimitetaan säännöllisin päivitysvälein vastaanottajalle [14].

4 Simuloinnit

Simulointi on tutkimusmenetelmä jonka tarkoituksena on suorittaa todellista systeemiä kuvaavalla mallilla koe, josta voidaan tehdä johtopäätöksiä todellisen systeemin toiminnasta.

Suorituskykymittauksia voidaan tehdä kolmella eri tavalla: simuloimalla, analyyttisillä malleilla sekä mittauksilla. Simulointi on yleinen tapa tutkia algoritmien ja protokollien suorituskykyä ja toimintaa. Sen etuina muihin menetelmiin nähden voidaan pitää seuraavia ominaisuuksia[4]:

- Mikäli varsinaista tutkittavaa verkkoa ei ole vielä toteutettu tai sitä ei muuten päästä mittaamaan, niin simuloinnin avulla pystytään kuitenkin tehokkaasti arvioimaan sen ominaisuuksia.

- Simuloinnin avulla voidaan helposti ja tehokkaasti testata verkon toimintaa eri kuormituksilla ja verkko-olosuhteissa.

- Simulointi mahdollistaa eri verkon ominaisuuksien pysymisen täsmälleen samana kerrasta toiseen mikä ei aina ole mahdollista verkon suorassa mittauksessa. Tämä on eduksi eri algoritmien ja protokollien vertailussa.

- Simuloinnin avulla saadaan enemmän yksityiskohtaista tietoa verrattuna analyyttiseen mallinnukseen.

- Analyyttisten mallien tulokset voidaan usein vahvistaa simuloimalla.

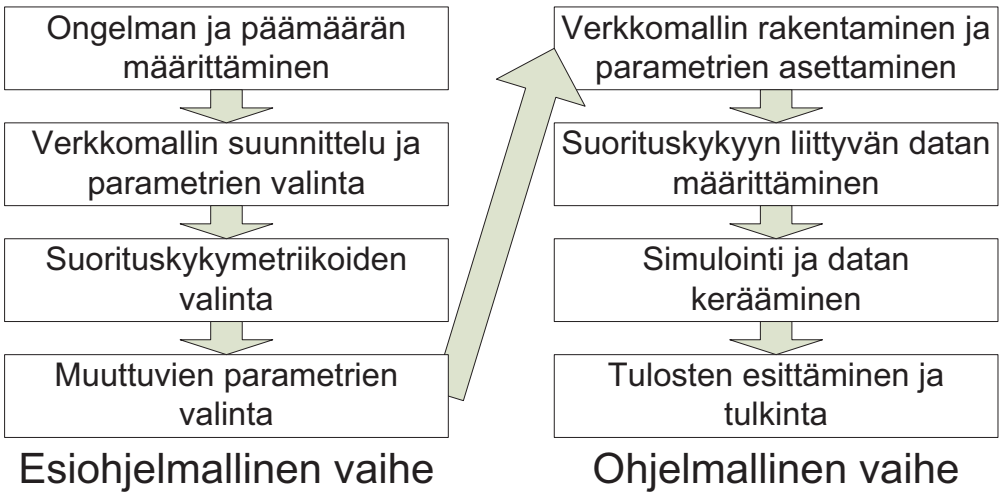
Systemaattista simulointia varten on useita eri tapoja suorittaa tehtävä. Niin sanottu Jain-malli on suorituskykymittauksia varten kehitetty projektimalli. Kyseinen malli jakaa tehtävän kahdeksaan osavaiheeseen (kuva 5), joista puolet kuuluvat niin sanottuun esiohjelmalliseen vaiheeseen ja toinen puolisko ohjelmalliseen vaiheeseen [4].

4.1 ns2-simulaattori

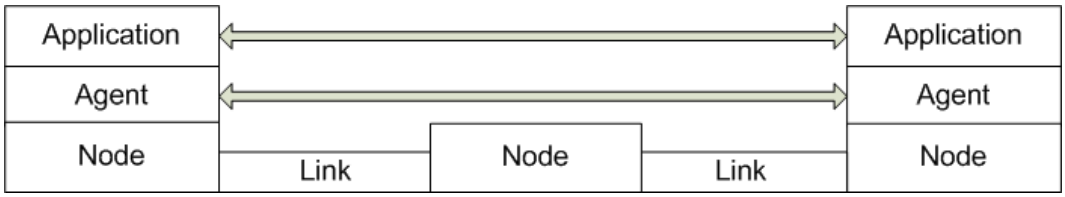
Simulaatiot suoritettiin ns2-simulaattorilla, joka on C++ sekä Obejct Oriented TCL:llä (oTcl) toteutettu simulaatio-ohjelma [25]. C++ käytetään protokollien toteutuksiin ja uusien moduulien luontiin. Varsinainen simulaation konfigurointi tehdään oTcl:llä. Ns2:lla pystytään simuloimaan esimerkiksi tietoverkkojen reitityksiä ja kuljetuskerroksen protokollia sekä langallisissa että langattomissa ympäristöissä.

4.2 Simulaation topologia ja parametrit

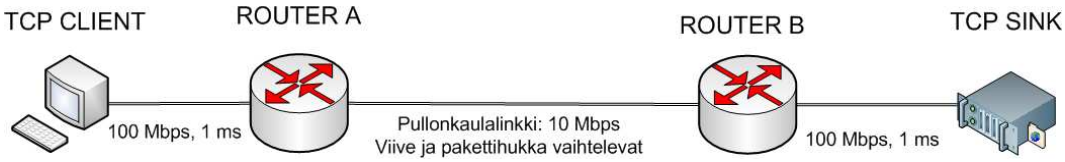
Simuloitu verkko koostuu reitittimistä, palvelimista sekä linkeistä, joista yksi on muita pienempi kapasiteettinen pullonkaulalinkki. Simulaatioiden ensimmäisessä vaiheessa tutkitaan yksittäisten protokollien suorituskykyä pullonkaulalinkin yli olosuhteissa joissa ne eivät joudu jakamaan verkon resursseja muiden käyttäjien kesken (kuva 7). Simulaatioiden jälkimmäinen osuus koostuu eri protokollien testaamisesta



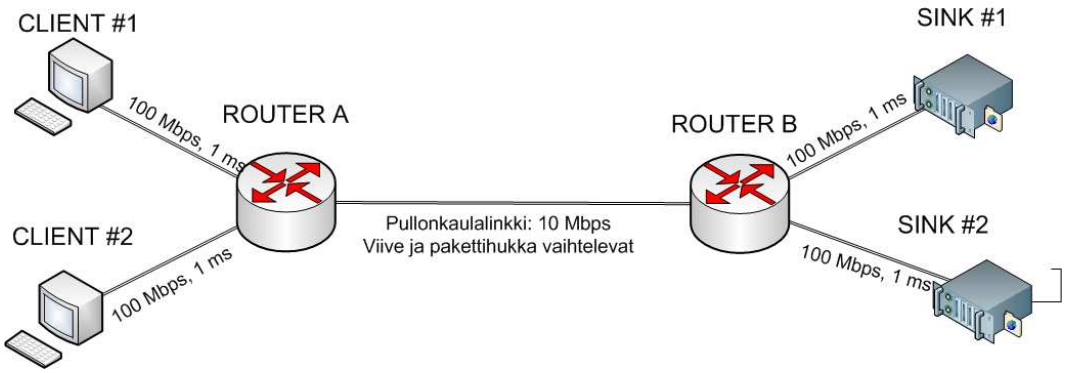
Kuva 5: Jain-mallin mukainen systemaattinen simulointi



Kuva 6: Ns2-simulaattorin objektit ja niiden yhteydet



Kuva 7: Simulaation topologia



Kuva 8: Simulaation topologia vertailtaessa protokollien oikeudenmukaisuuksia

pullonkaulalinkillä kun kahden eri kuljetuskerroksen protokollan yhteydet jakavat saman pullonkaulalinkin resurssit (kuva 8).

Simulaatiota varten pitää määritellä useita parametrejä, jotka pysyvät vakioina kaikissa simulaatioissa. Yksi simulaation tärkeimmistä ominaisuuksista on sen kesto. Liian lyhyt simulaatioaika aiheuttaa tuloksien epäluotettavuutta, sillä tällöin jo pienet sattumat aiheuttavat suuria muutoksia tuloksissa. Toisaalta liian pitkä simulaatioaika aiheuttaa turhaa simulaatiotyön pitkittymistä tuomatta kuitenkaan mitään lisäinformaatiota lopputuloksiin. Tätä työtä varten simulaatioajaksi valittiin 600 sekuntia, jossa ajassa pienet satunnaisuudet tasoittuvat ja varsinainen simulaatio saadaan ajettua läpi tehokkaasti. Tätä työtä varten suoritettiin lähes 100 000 simulaatioajoa, joten tehokkuus oli varsin tärkeässä asemassa.

Verkon liityntälinkkien kaistanleveydeksi valittiin 100 Mbps ja viiveeksi yksi millisekunti. Pullonkaulalinkin kaistanleveys oli 10 Mbps ja viive sekä pakettihukka olivat dynaamisia muuttujia. Verkossa olevien puskurien koko vaikuttaa paljon sen suorituskykyyn. Hyvin suuret puskurit aiheuttavat ylikuormitustilanteissa pitkiä viiveitä, mutta toisaalta niiden avulla pakettihukka saadaan pidettyä pienenä mikäli ylikuormitus on purskeluonteista. Suuret puskurit hidastavat myös kuljetuskerroksen protokollien reagointikykyä ruuhkautumistilanteissa [2]. Yleisenä sääntönä pidetään että puskurit olisi hyvä pitää melko pienenä. Tarkempaa määritystä varten löytyy useampiakin eri suosituksia, mutta yksi yleisimmistä ohjeista on että puskurin tulisi olla kaistanleveys-viive -tulon kokoinen. Simuloitavan verkon tapauksessa tämä tarkoittaa että esimerkiksi kymmenen millisekunnin viiveellä puskurin koon tulee olla noin 10 kilotavua. Näin ollen simulaation puskurien vakiokooksi valittiin kymmenen pakettia.

Puskurin tapa käsitellä ylikuormitustilanteita tulee myös määritellä suorituskykysimulointia varten. NS2-simulaattori tukee useita eri jononhallinta-algoritmejä, kuten drop-tail, Random Early Detection (RED) sekä Random Early Detection with Explicit Congestion Notification (RED-ECN). RED on aktiivinen jononhallinta-algoritmi, joka pudottaa paketteja tietyn todennäköisyysmallin mukaisesti jo ennen kuin puskuri vuotaa yli. Näin pyritään säätelemään puskuriin saapuvan datan määrää ja välttämään pahat ruuhkautumistilanteet. Näin ollen RED on myös ruuhkanvälttämisalgoritmi. RED:n yhteydessä voidaan käyttää myös ECN-mekanismia, jolloin pakettien tiputtamisen sijaan RED merkkaa kyseisten muutoin pudotettavien pakettien IP-kehyksien ECP-kenttään tiedon että verkko saattaa olla ruuhkautumassa. Näin pystytään informoimaan lähettäjää ja vastaanottajaa mahdollisista ylikuormitustilanteista jo ennalta ilman että joudutaan tiputtamaan paketteja. Drop-tail on hyvin yleisesti reitittimissä käytetty jononhallinta-algoritmi joka nimensä mukaisesti pudottaa kaikki puskuriin saapuvat paketit, jotka eivät sinne enää mahdu. Drop-tail kohtelee kaikkia paketteja samanarvoisesti ja se käyttää niin sanottua first in - first out (FIFO) jonotusmallia. Tämän diplomityön simulaation jononhallintaan valittiin käytettäväksi drop-tail.

Simulointityössä tarvitaan vakioparametrien lisäksi muuttuvia tekijöitä, joiden vaikutusta systeemiin tutkitaan. Tämän diplomityön tavoitteiden saavuttamiseksi muuttuviksi parametreiksi valittiin verkon pullonkaulalinkin viive, pakettihukka sekä kuljetuskerroksen protokolla.

Taulukko 2: Simulaation vakioparametrit ja niiden arvot

Simulaatioaika	600 s
Linkkien kaistanleveys	100 Mbps
Pullonkaulalinkin kaistanleveys	10 Mbps
Linkkien viive	1 ms
Puskurien koko	10 pakettia
Jononhallinta-algoritmi	Drop-tail
Max. cwnd	64 KB
DCCP paketin koko	1000 B

Taulukko 3: Simulaation muuttuvat parametrit ja niiden arvot

RTT	1 ms -> 100 ms, resoluutio 10 ms
Pakettihukka	0.000 -> 0.150, resoluutio 0.001
Protokollat	TCP: Tahoe, Reno, NewReno, Sack, Vegas DCCP: TFRC, TCP-Like

4.3 Pakettihukan vaikutus

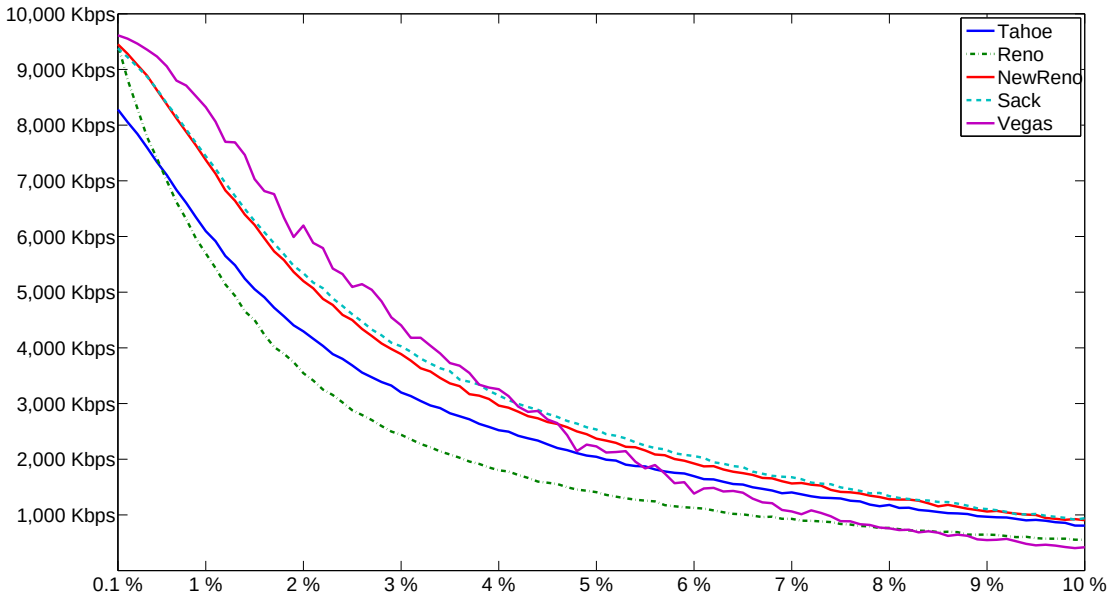
Pakettien hukkuminen aiheuttaa TCP:n lähetysikkunan koon pienentymistä ja pakettien uudelleen lähetyksiä. Eri TCP-versiot reagoivat eri tavoin pakettihukkaan. Tämän simulaation tarkoitus oli tutkia kuinka eri algoritmit suoriutuvat läpiviennin osalta kun pakettihukka vaihtelee välillä 0.01 ja 0.10, mutta RTT pysyy vakioarvossa 50 ms.

Kuvan 9 perusteella voidaan todeta TCP Vegasin tehollisten läpivientiarvojen olevan vertailujoukon suurimpia pakettihukan ollessa alle 4 %. Tätä huonommissa olosuhteissa TCP Vegasin suoritustaso heikkenee dramaattisesti muihin TCP-variantteihin verrattuna. TCP Renon ja Tahoen järjestys läpivientitason osalta vaihtuu pakettihukan kasvaessa yli puolen prosentin. Myös TCK Sack ja New Reno vaihtavat paremmuusjärjestystään tämän simulaation perusteella.

4.4 Kiertokulkuajan vaikutus

RTT vaikuttaa suoraan teoreettisen läpiviennin suuruuteen kaavan 7 mukaisesti. Eri ruuhkanhalinta-algoritmien reagointikyky verkon ominaisuuksiin riippuu pakettihukan lisäksi RTT:n suuruudesta. Simulaation avulla tutkittiin algoritmien läpivientiä kun RTT vaihteli välillä 1 ja 100 millisekuntia. Pakettihukka asetettiin tässä simulaatiossa vakioksi, joka oli 0,1 %.

Kuvan 10 perusteella voidaan todeta TCP Renon saavan muista poikkeavia arvoja pienillä RTT:n arvoilla. TCP Renon ruuhkanhallinta toimii tunnetusti huonosti olosuhteissa, joissa saman lähetysikkunan aikana katoaa useita paketteja. Myös cwnd:n hyvin pieni koko aiheuttaa ongelmia TCP Renon nopealle uudelleenlähetykselle, sillä pieni ikkunan koko estää tarpeeksi usean duplikaatin ACK:n vastaanoton. TCP:n lähetysikkuna on erittäin pienillä RTT:n arvoilla hyvin pieni. Kaavojen 11 ja 12 perusteella voidaan arvioida cwnd:n keskimääräinen koko tietyillä tehollisen



Kuva 9: Läpivienti pakettihukan arvoilla 0,1 -> 10 prosenttia

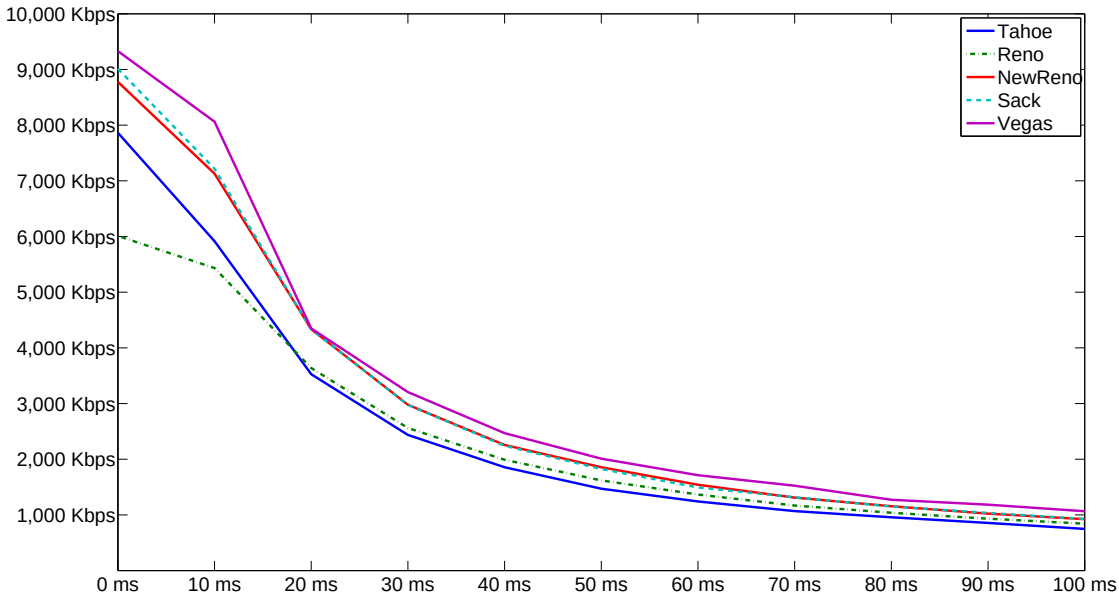
läpiviennin ja RTT:n arvoilla. TCP Renon cwnd keskimääräinen koko kahden millisekunnin RTT:llä on vain 1.5 kilotavua. Näin ollen nopea uudelleenlähetys ei voi toimia halutulla tavalla ja pakettihukka todetaan aina RTO:n laukeamisesta.

$$TCP\ siirtonopeus = \frac{cwnd}{RTT}\ (pakettia/s) \tag{11}$$

$$cwnd = RTT * TCP\ siirtonopeus \tag{12}$$

4.5 TCP-varianttien vertailu

Edellisten lukujen simulaatioiden perusteella voidaan todeta että TCP-variantit suoriutuvat eri tavoin riippuen mm. verkon pakettihukasta ja kauttakiertoajasta. Kuvan 9 perusteella huomataan että TCP Vegas saa parhaat läpivientiarvot aina reiluun neljän prosentin pakettihukkaan asti. Tämän jälkeen sen suorituskyky huononee merkittävästi muihin variantteihin verrattaessa. Kuvassa 10 nähdään kuinka RTT:n vaikuttaa eri varianttien läpivienteihin pakettihukan pysyessä yhden prosentin vakio arvossa. Kuvan perusteella voidaan todeta Vegasin pärjäävään edelleen kaikkein parhaiten, mutta samalla voidaan todeta RTT:n kasvun muuttavan TCP Tahoen ja Renon tehollisten läpivientien paremmuusjärjestystä kun RTT kasvaa yli 20 millisekuntiin. Vastaavasti New Reno ja Sack vaihtavat paremmuusjärjestystään. Näin



Kuva 10: Läpivienti RTT:n suhteen eri TCP-varianteilla

ollen herääkin kysymys mikä varianteista pystyy tarjoamaan parhaan läpiviennin erilaisten verkko-olosuhteiden vallitessa.

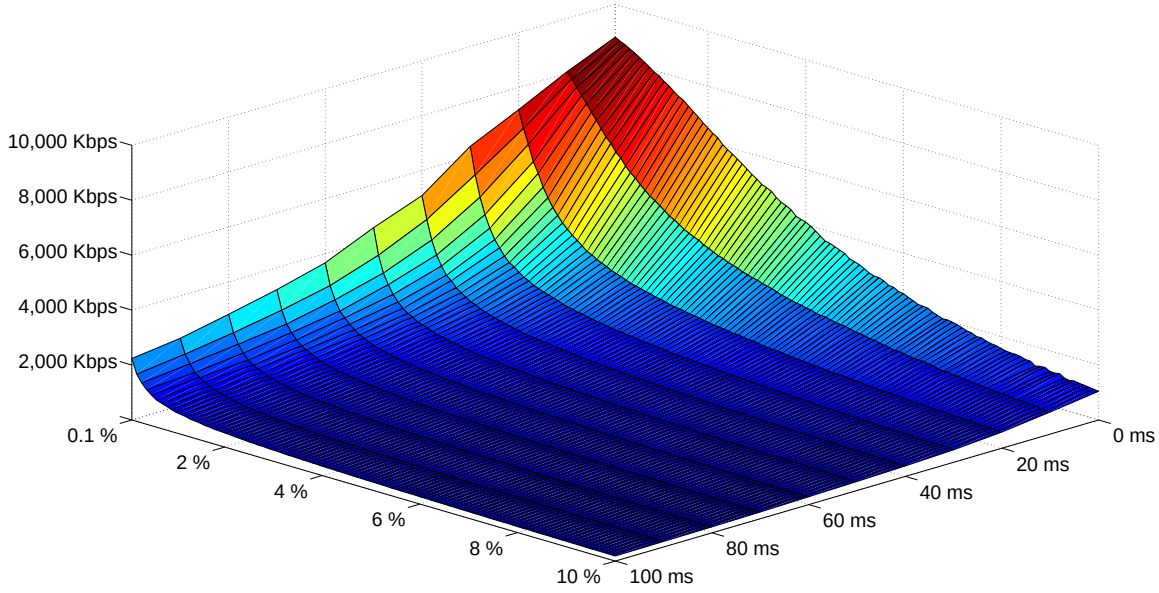
Tutkimuksen kohteena oleville viidelle TCP-variantille suoritettiin simulaatiot käyttäen pakettihukalle arvoja 0,1-10,0 % sekä RTT:lle arvoja 1-100 ms. Simulaatioiden resoluutio pakettihukan osalta oli 0,1 prosenttia ja RTT:llä kymmenen millisekuntia. Simulaatiotoistoja suoritettiin tutkimuksessa 10-40 kertaa jokaista skenaariota kohden. Näin ollen yksittäisiä simulaatioajoja tehtiin tätä osuutta varten lähes 100 000 kappaletta.

Simulaatiosta saaduista trace-tiedostoista käy ilmi kaikkien verkossa liikkuneiden pakettien liikkeet solmujen välillä. Tämän informaation avulla pystytään analysoimaan simulaatiossa tapahtuneita ilmiöitä ja laskemaan esimerkiksi läpivienti, jitter ja keskimääräinen viive. NS2-simulaattori ei tarjoa valmista työkalua simulaatiodatan käsittelyyn, joten sitä varten täytyy käyttää muita käytettävissä olevia työkaluja tai tehdä ne itse. Tämän työn simulaatioaineiston käsittely toteutettiin käyttämällä awk-skriptiä, jonka avulla tiedostosta parsittiin erilleen vastaanotetut segmentit, joiden joukosta poistettiin protokollien hylkäämät sekä duplikaatit segmentit. Jäljelle jääneiden pakettien määrällä kerrottiin yhden paketin hyötykuorman koko ja saatu tulo jaettiin simulaatioajalla. Näin saatiin selville simulaatioajon protokollien teholliset läpiviennit eli nopeus jolla hyötykuormaa saatiin siirrettyä verkon läpi sekuntia kohden.

Trace-fileistä selvitetyt teholliset läpivientiarvot siirrettiin edelleen Matlab-ohjelmaan, jossa luotiin 11x101 matriisi (11 eri viivettä, 101 eri pakettihukkaa) jokaiselle tutkitulle protokollalle. Jokaiselle matriisin solulle asetettiin arvoksi sen sijainnin kuvaa-

mien parametrein tuottama läpivienti kyseisellä protokollalla. Esimerkiksi matriisin solu $Vegas_{10,25}$ pitää sisällään tiedon TCP Vegasin tuottamasta tehollisesta läpiviennistä simuloidussa topologiassa RTT:n arvolla 10 ms ja 2,5 prosentin pullonkaulalle lisätyllä pakettihukalla.

Kuvassa 11 nähdään TCP Tahoen tehollisia läpivientiarvoja simulaation perusteella. Pakettihukan vaikutus näkyy tasaisena läpivientiarvojen alenemana siirryttäessä kohti suurempaa pakettihukkaa. Vastaavasta Z/X-tasolla nähdään viiveen vaikutus pakettihukan ollessa 0,1 prosenttia. Kuvaajan lopputuloksena on keskeltä hieman alas painunut liukumäki.

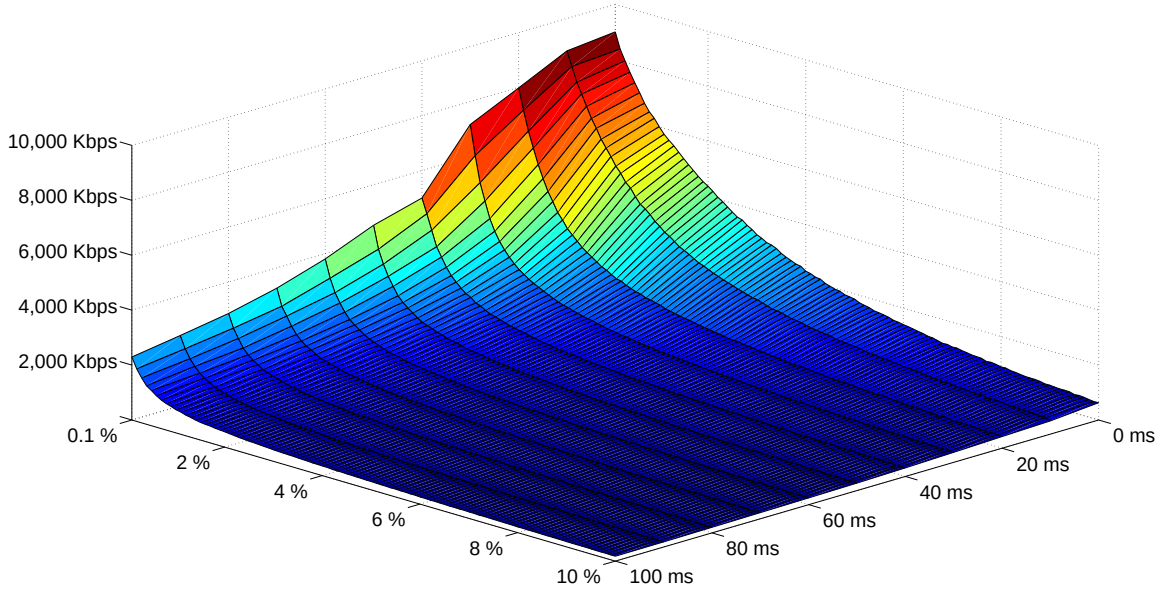


Kuva 11: TCP Tahoe

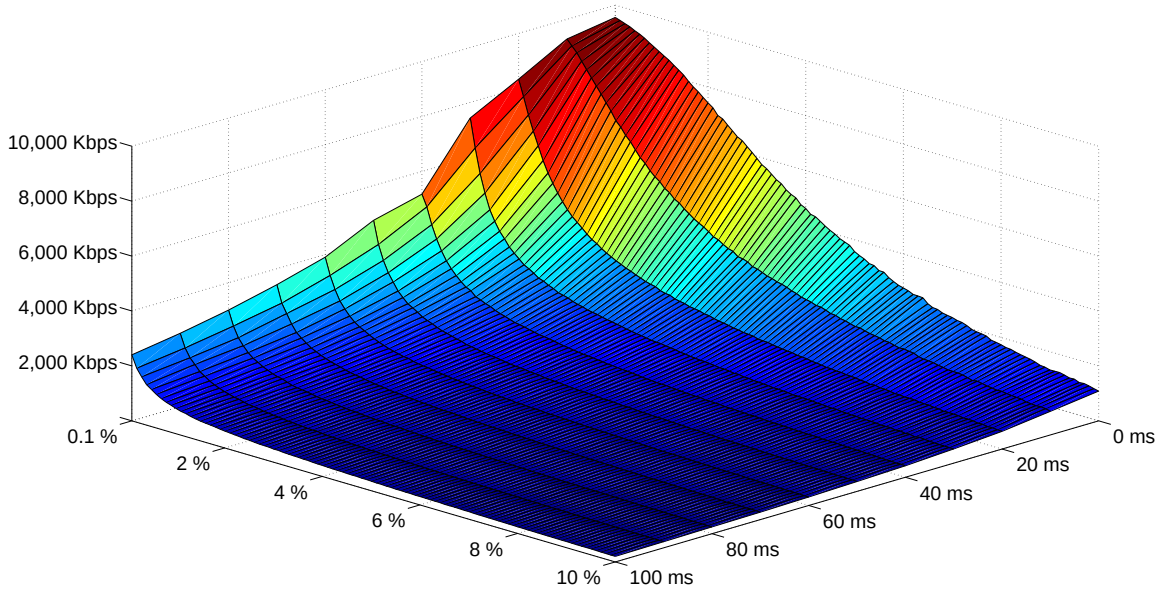
TCP Renon ruuhkanhallintamekanismin suurimpia ongelmakohtia on sen selviytyminen tilanteista, joissa paketteja katoaa useampia kappaleita saman lähetysikkunan aikana. Kyseinen ilmiö tulee hyvin esille kuvasta 12, jossa Renon läpivientiarvot laskevat hyvin jyrkästi siirryttäessä kohti suuria pakettihukka-arvoja. Kuvaajan perusteella voidaan toisaalta todeta Renon toimivat varsin tehokkaasti pienillä pakettihukilla verrattuna TCP Tahoeen.

TCP New Renossa on Renoon verrattuna paranneltu nopean toipumisen -mekanismi, jonka ansiosta cwnd:tä ei puoliteta suurenkaan pakettihukan sattuessa kahteen kertaan yhden lähetysikkunan aikana. TCP New Renon kuvaajan perusteella voidaan todeta sen saavan selvästi parempia läpivientiarvoja kauttaaltaan, mutta etenkin suuren pakettihukan aiheuttama dramaattinen suorituskyvyn heikentymä on korjaantunut täysin verrattuna TCP Renoon.

TCP Sack käyttää nimensä mukaisesti selektiivistä segmenttien kuittausta. Tällä tavoin lähettäjällä on koko ajan tiedossaan kaikki perille menneet segmentit mukaan

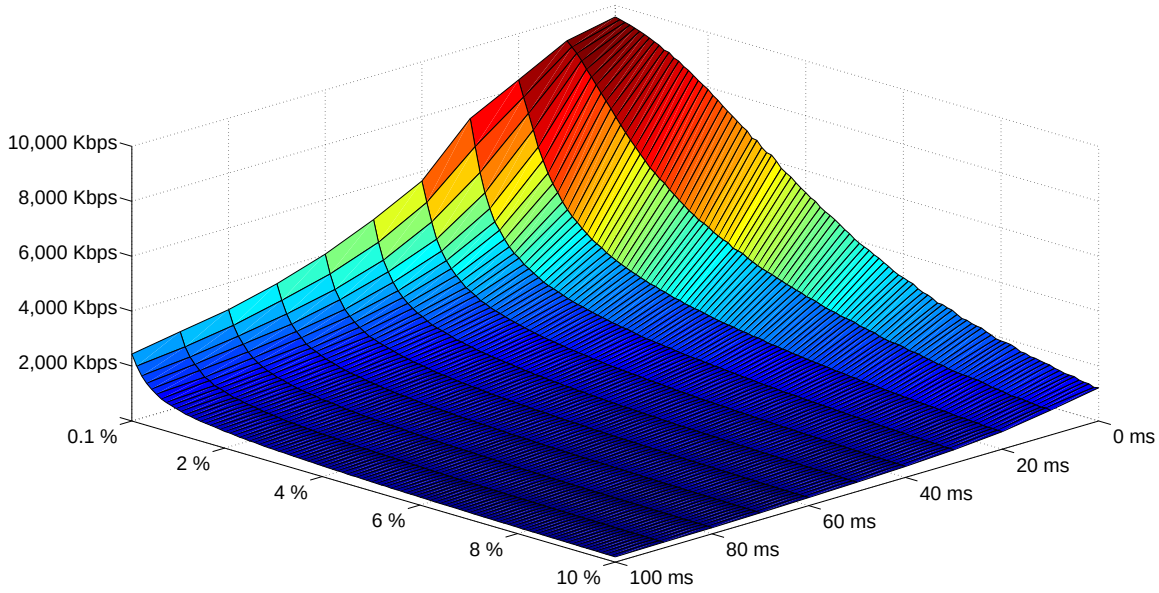


Kuva 12: TCP Reno



Kuva 13: TCP NewReno

lukien kadonneen paketin jälkeen perille menneitä segmenttejä. Tällä ominaisuudella pyritään parantamaan TCP-yhteyden mahdollisuutta uudelleenlähettää enemmän kuin yksi segmentti yhden RTT:n aikana. Kuvan 14 perusteella pystyy näkemään Sack:n saavan ainakin silmämääräisesti parempia läpivientiarvoja ainakin pienillä RTT:n arvoilla. Tarkempiin vertailuihin ryhdytään myöhemmin tässä dokumentissa.

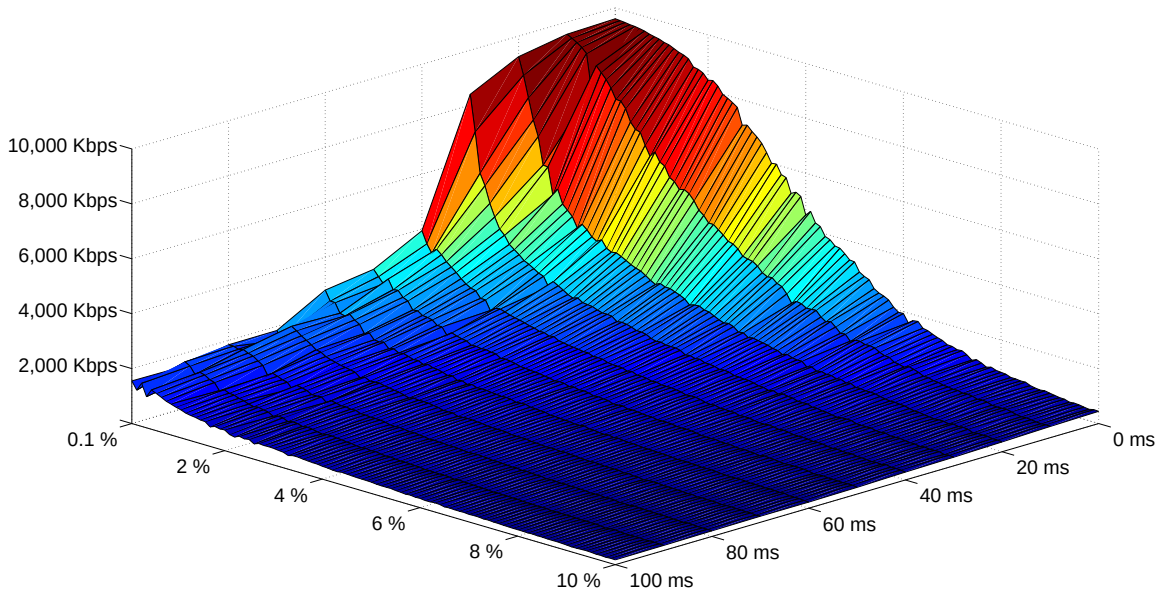


Kuva 14: TCP Sack

TCP Vegas poikkeaa useiden ominaisuuksiensa perusteella muista TCP-varianteista. Se käyttää AIAD-palautekontrollialgoritmiä kun muut tässä työssä tutkitut protokollat käyttävät AIMD:tä. TCP Vegas käyttää ruuhkanhallinnassaan myös muista poikkeavaa mekanismia, jossa seurataan odotetun ja toteutuneen läpiviennin suhdetta. Kuvan 15 perusteella voidaan huomata kuinka TCP Vegas tarjoaa erittäin suuria läpivientiarvoja pakettihukan pysyessä alle viiden prosentin ja RTT alle 50 millisekunnin. Tätä huonommissa verkko-olosuhteissa löytyy parempia suoritusarvoja tarjoavia TCP-versioita.

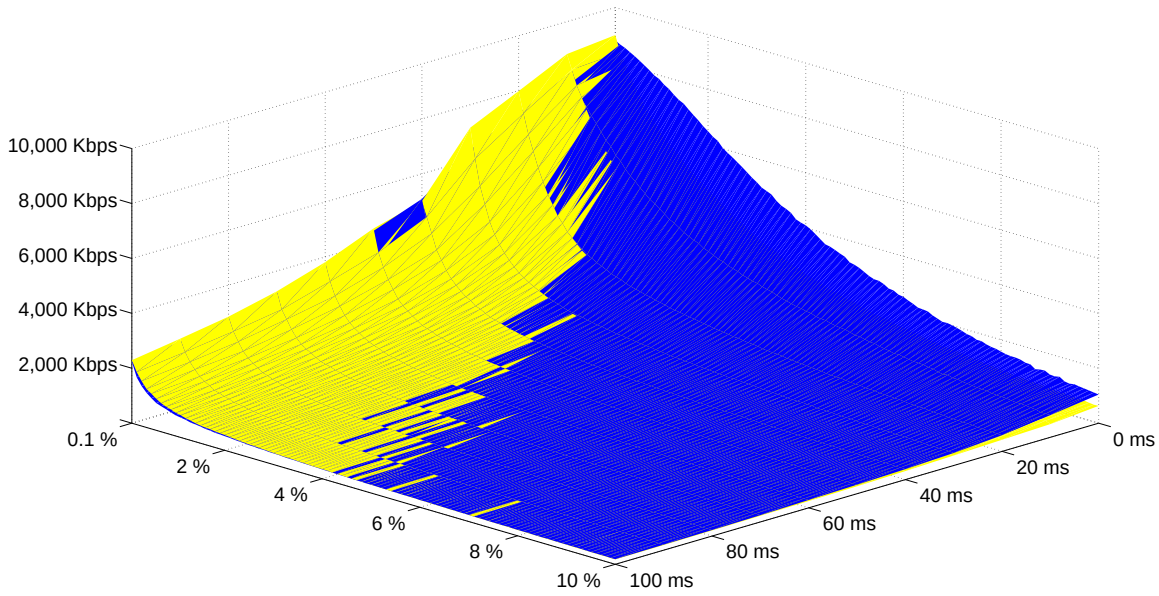
TCP-protkollien keskinäinen vertailu ei onnistu kovin tarkasti suoraan kuvien -perusteella. Heikompien verkko-olosuhteiden läpiviennit ovat pieniä verrattuna suotuisten olosuhteiden tapauksiin ja näin ollen kyseisten arvojen erot on vaikea hahmottaa samasta kuvaajasta.

Kuvaajissa 16-18 on esitetty mikä TCP-varianteistä sai simuloinneissa suurimman läpivientiarvon koko simulaatiomuuttujien kentässä. Kuvassa 16 voidaan selvästi nähdä TCP Renon kykenemättömyys selviytyä hyvin suurista pakettihukista. Kuvaajan avulla voidaan lisäksi huomata kuinka kyseinen ilmiö on sitä radikaalimpi mitä pienempiä arvoja RTT saa. Voidaan siis todeta että TCP Renon suorituskykyongelmat Tahoeen verrattuna koskevat erityisesti suuren pakettihukan sekä



Kuva 15: TCP Vegas

pienehkön pakettihukan ja pienen viiveen olosuhteita.



Kuva 16: TCP Tahoe (sininen) ja TCP Reno (keltainen)

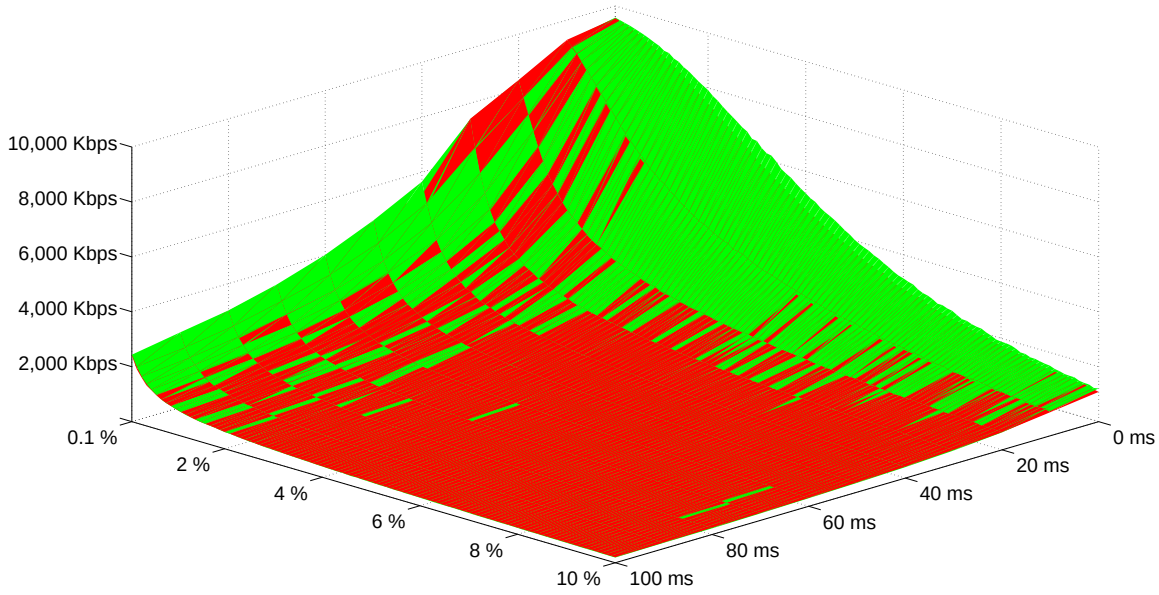
Kuvaajissa 19-22 vertaillaan TCP Renon, New Renon, Sack:n ja Vegasin tehollisia läpivientejä suhteessa TCP Tahoen läpivientiin kyseisen x-y -koordinaatin määrittelemillä pakettihukka ja RTT -arvoilla. Arvot on laskettu kaavan 13 mukaisesti.

$$Vertailtu\ TCP\ vs\ TCP\ Tahoe\ (x,y) = \frac{VertailtuTCP(x,y) - Tahoe(x,y)}{Tahoe(x,y)} \quad (13)$$

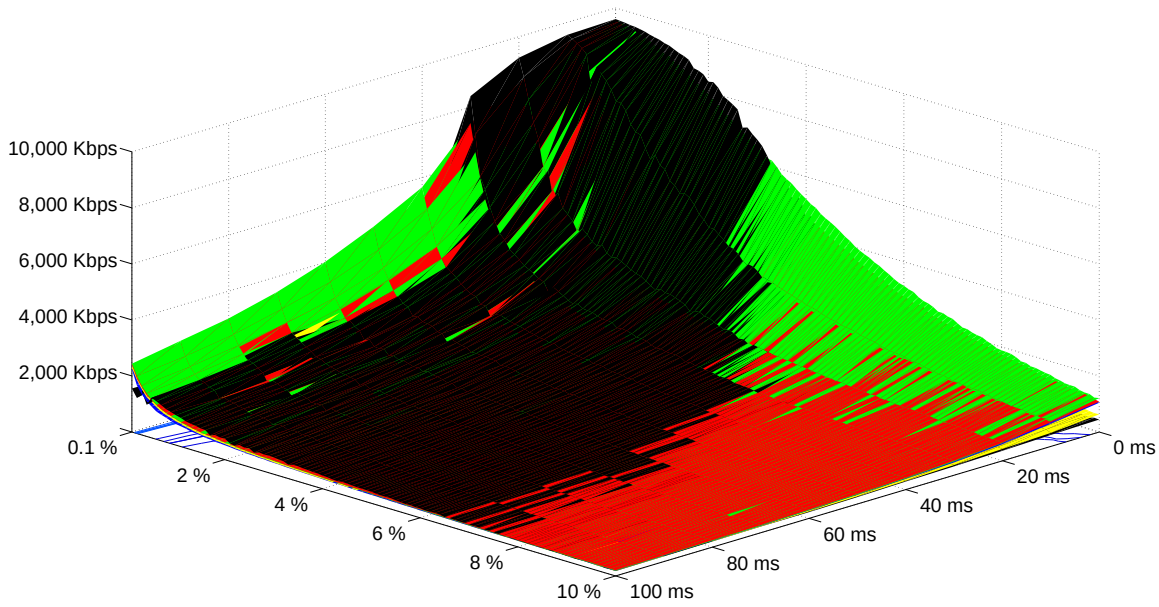
TCP Renon tehollinen läpivienti on pienillä RTT:n arvoilla jopa 60 prosenttia TCP Tahoeta pienempi. Erittäin pienellä lähetysikkunan koolla ilmenevä TCP Renon nopean uudelleenlähetyksen ongelmat katoavat simulaation perusteella RTT:n ollessa yli 40 millisekuntia. Pienillä pakettihukkatodennäköisyyksillä TCP Reno suoriutuu kauttaaltaan TCP Tahoeta paremmin.

TCP New Reno on paranneltu versio TCP Renosta. New Renon paranneltu nopean toipumisen mekanismit mahdollistavat usean kadonneen paketin huomaamisen samasta lähetysikkunasta. Kuvasta 20 nähdään kuinka TCP Renon kanssa ilmenneet ongelmat pienillä RTT:n arvoilla ovat kadonneet lähes kokonaan ja läpivientiarvot ovat kauttaaltaan 10-20 prosenttia suurempia kuin TCP Tahoella.

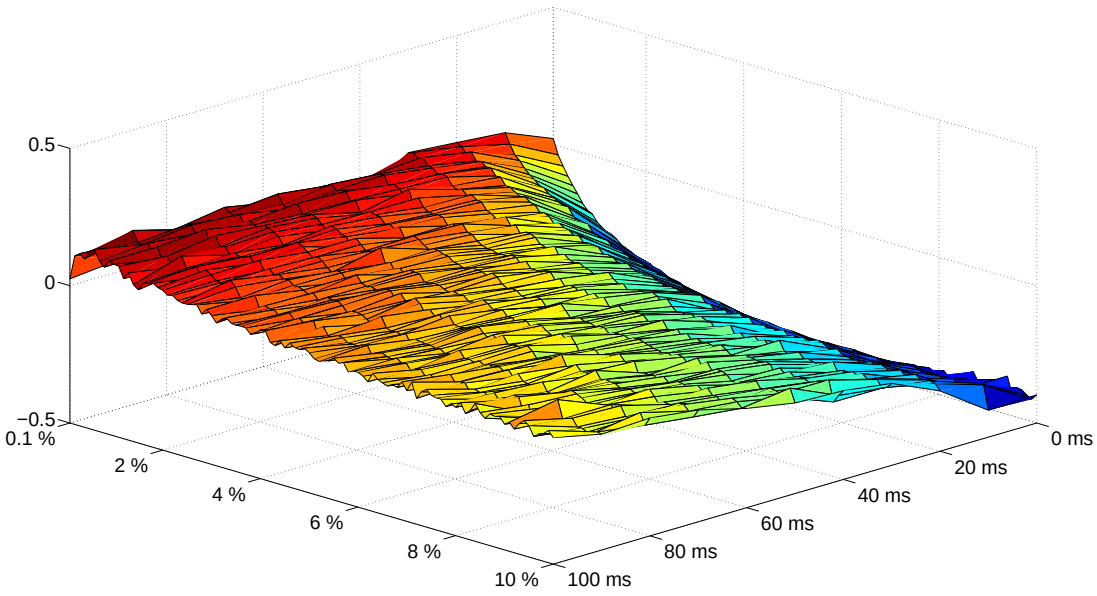
TCP Sack pystyy Reno:sta ja New Reno:sta poiketen huomaamaan useamman kuin yhden segmentin katoamisen samasta lähetysikkunasta. Tämän lisäksi Sack pystyy uudelleenlähettämään useita segmenttejä yhden RTT:n aikana. Selektiivisten ACK-viestien avulla saavutetaan huomattavasti TCP New Renoa suurempia tehollisia läpivientiarvoja RTT:n pienillä arvoilla. Simulaatioiden perusteella TCP



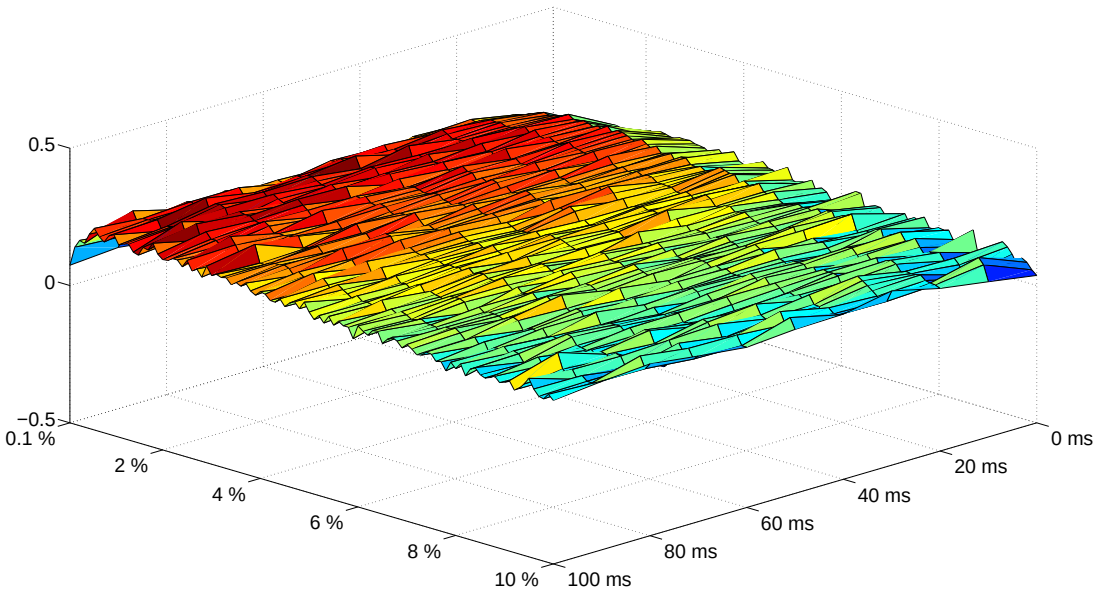
Kuva 17: TCP Sack (vihreä) ja TCP NewReno (punainen)



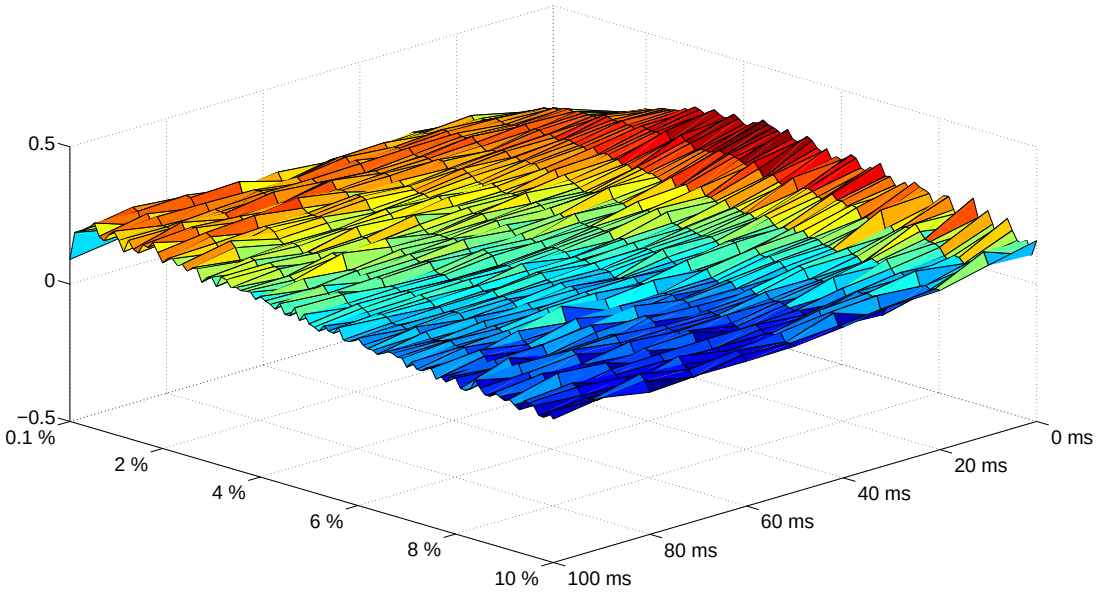
Kuva 18: TCP Vegas (musta), TCP Sack (vihreä) ja TCP NewReno (punainen)



Kuva 19: TCP Reno

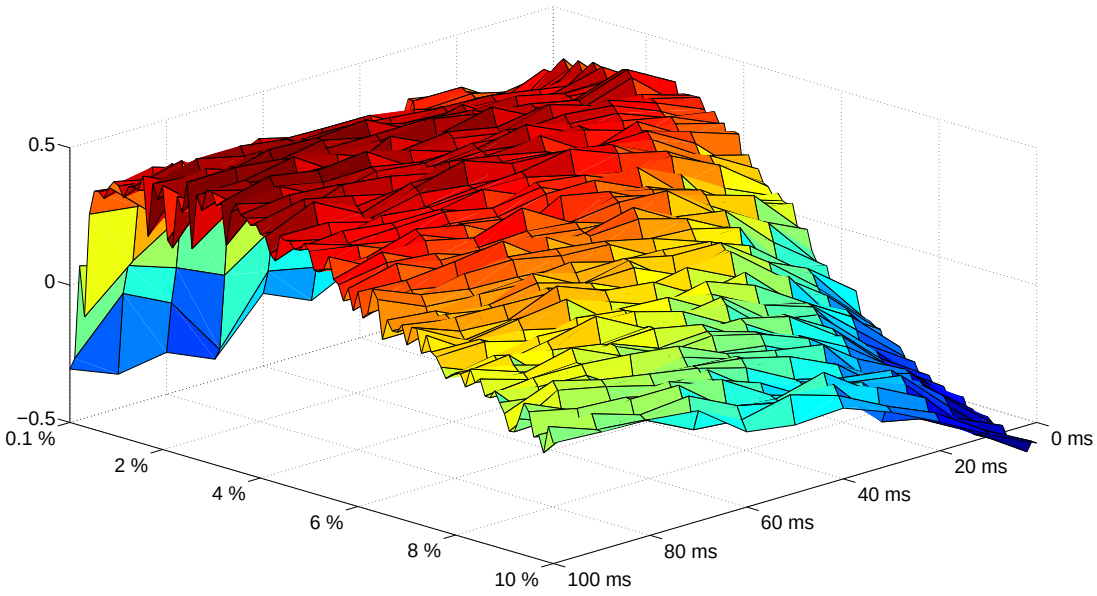


Kuva 20: TCP NewReno



Kuva 21: TCP Sack

Sack saavuttaa koko tutkitulla alueella noin 10-20 prosenttia parempia läpivientiarvoja TCP Tahoeen verrattuna.



Kuva 22: TCP Vegas

TCP Vegas saa simulaatioiden perusteella kaikkein suurimpia läpivientiarvoja suurimmassa osassa simulaatioita. Vegas saavuttaa 1-4 prosentin pakettihukan arvoilla jopa 50 prosenttia suurempia läpivientejä kuin TCP Tahoe. Kuvan 22 perusteella voidaan todeta TCP Vegasin suorituskyvyn huonontuvan merkittävästi pakettihukan ollessa erittäin suuri ja RTT:n ollessa samalla hyvin pieni.

Taulukko 4: TCP-varianttien tehollisia läpivientiarvoja eri olosuhteissa

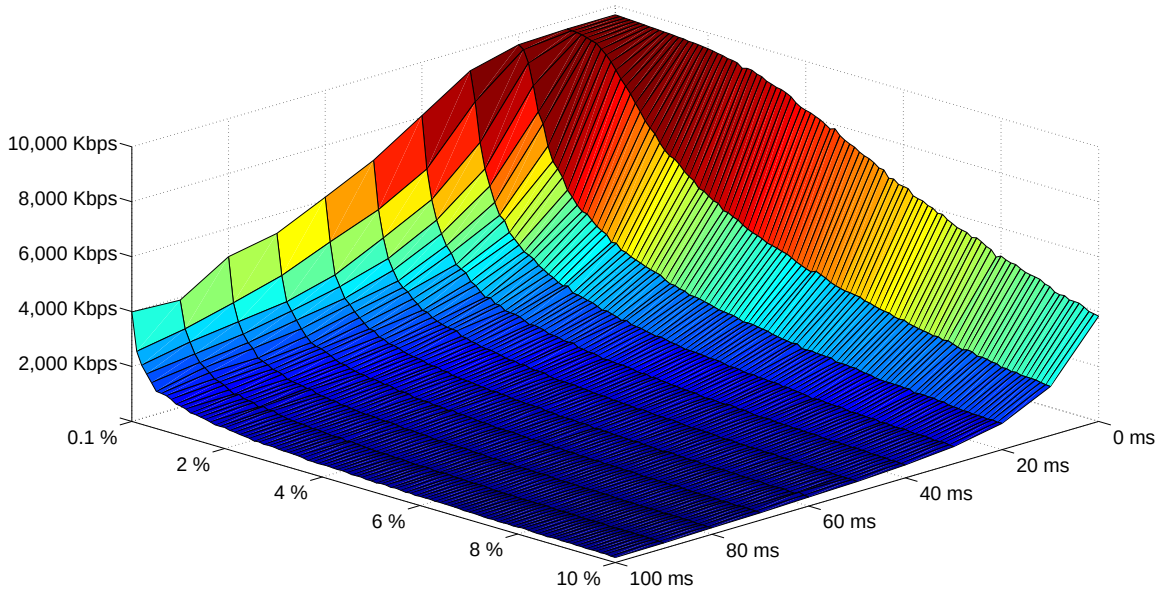
RTT	10 ms			100 ms		
Pakettihukka	0.2 %	3.0 %	6.0 %	0.2 %	3.0 %	6.0 %
Tahoe	7788 kbps	3001 kbps	1690 kbps	1773 kbps	355 kbps	234 kbps
Tahoe	100	100	100	100	100	100
Reno	106	77	65	105	107	102
New Reno	116	117	114	105	121	112
Sack	115	126	117	106	119	108
Vegas	126	143	93	106	171	139

4.6 DCCP-simulaatiot

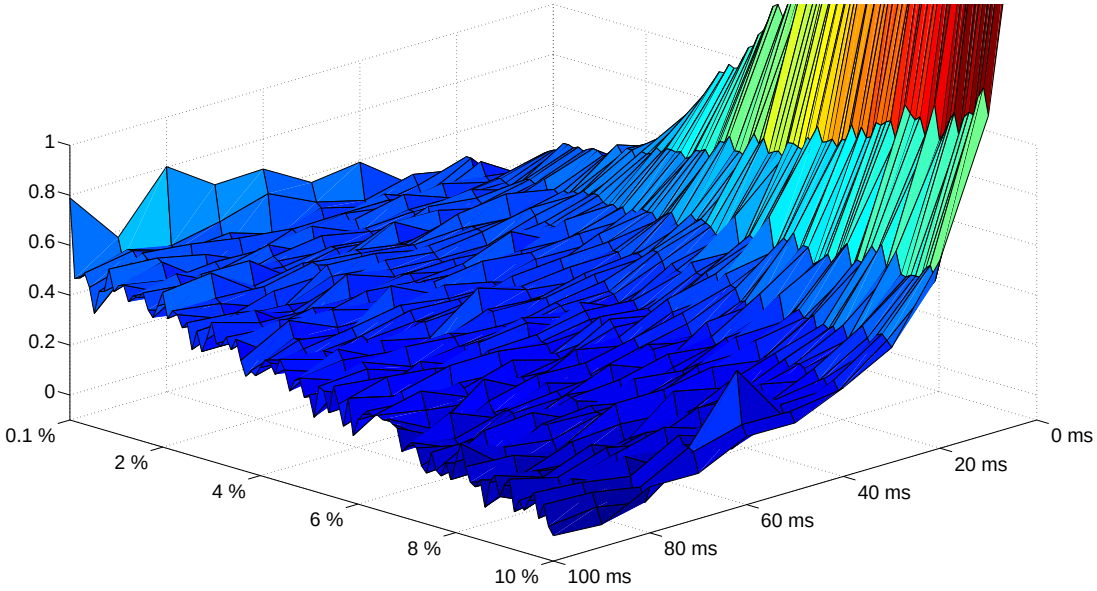
Ns2-simulaattori ei tue suoraan DCCP-protokollaa, mutta siihen on saatavilla Nils-Erik Matssonin tekemä laajennus, joka tukee TCP-like (CCID 2)sekä TFRC (CCID-3) -ruuhkanhallintamekanismeja [15]. Matssonin moduli on tehty ns2:n versiolle 2.26, mutta siitä on saatavilla myös Wassim Ramadanin tekemä päivitetty versio uusimmalle ns2:n versiolle[16].

DCCP:n CCID-2 ja CCID-3 -algoritmeille suoritettiin samanlaiset simulaatiotestit kuin TCP-algoritmeille. DCCP TFRC:n läpivientitasot suhteessa TCP Tahoeen olivat kauttaaltaan useita kymmeniä prosentteja suurempia (kuva 24). TFRC:n tehollinen läpivientitaso oli erityisen suuri pienillä RTT:n arvoilla, jolloin läpivienti oli jopa 150 % TCP Tahoeta suurempi. DCCP ei kuitenkaan takaa luotettavaa tiedonsiirtoa, mutta sovelluksissa joissa pakettihukka ei aiheuta kriittisiä ongelmia on sen suorituskyky erittäin hyvä.

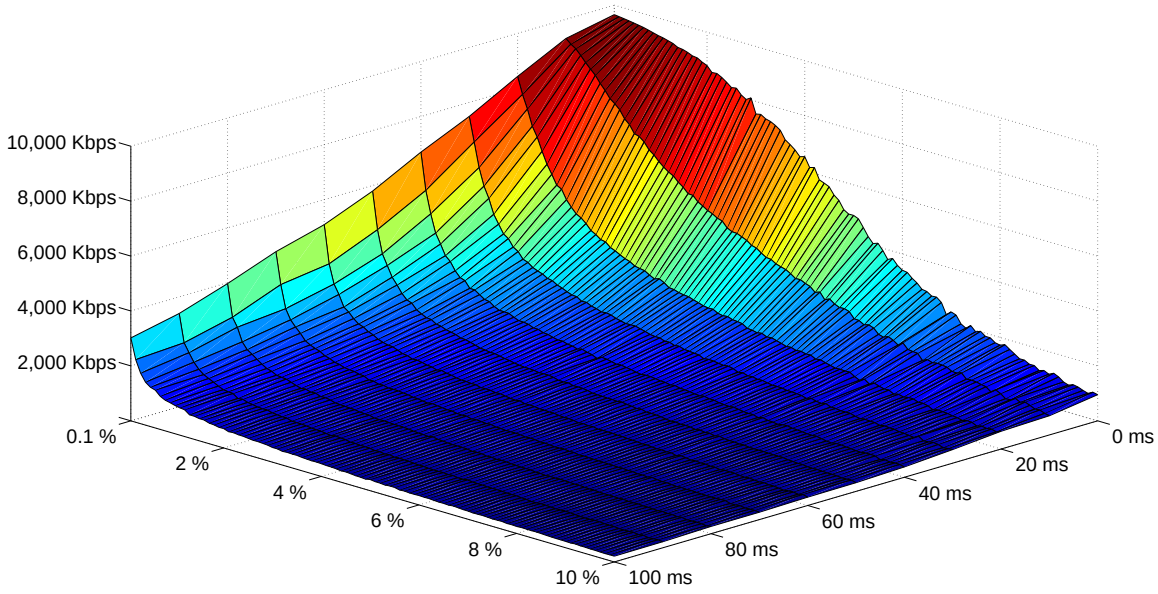
DCCP TCP Like:n läpivientiarvot (kuvat 25 ja 26) ovat hyvin lähellä TCP Vegasin suoritustasoa. TCP Like saavuttaa parhaimmillaan noin 40 % TCP Tahoe:n läpivientiä suurempia arvoja pakettihukan ollessa alle 6 %. TCP Liken suoritustaso romahtaa TCP Vegasin tavoin pakettihukan saadessa erittäin suuria arvoja ja RTT:n ollessa samalla hyvin pieni.



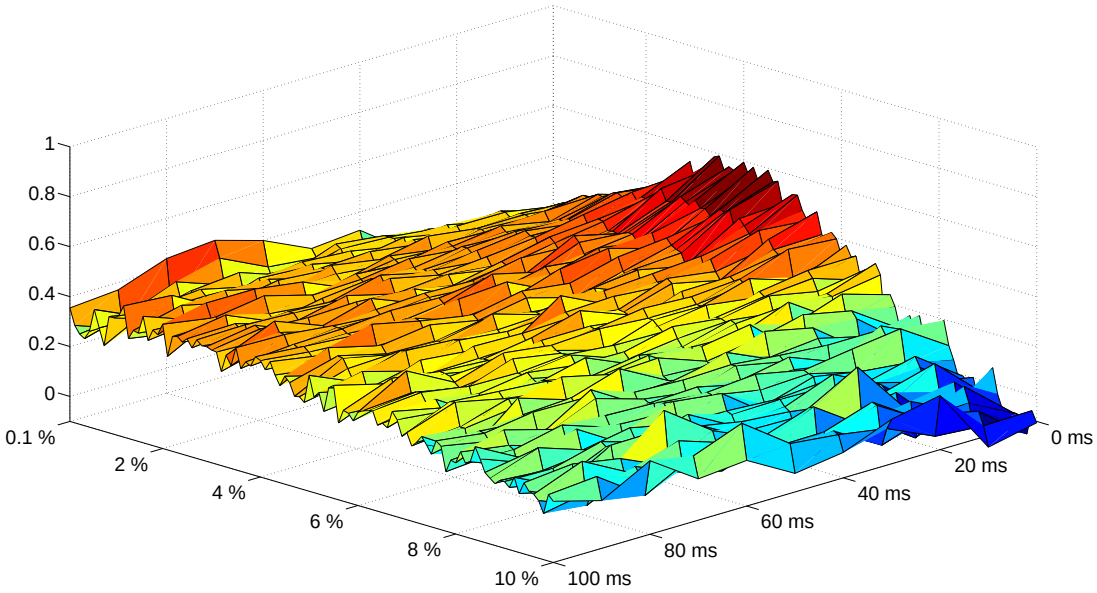
Kuva 23: DCCP TFRC



Kuva 24: DCCP TFRC



Kuva 25: DCCP TCP Like



Kuva 26: DCCP TCP Like

4.7 Protokollien tasapuolisuus

Ruuhkanhallinnan yksi tärkeimmistä tehtävistä on verkkoresurssien mahdollisimman tehokkaan hyödyntämisen mahdollistaminen. Tämän lisäksi protokollien tasapuolisuus muita verkon käyttäjiä kohtaan on tärkeää verkkoliikenteen sujuvuuden kannalta. Edellisen osion TCP- ja DCCP-protokollien simulaatiot suoritettiin simulaatioympäristössä, jossa verkon kaikki resurssit olivat vain yhden käyttäjän käytössä. Kyseisten tulosten perusteella ei pystytä vielä arvioimaan eri protokollien tasapuolisuutta verkon muita käyttäjiä kohtaan.

Tasapuolisuussimulaatiot suoritettiin Ns2-simulaattorilla kuvan 8 mukaisessa verkkoympäristössä, jossa pullonkaulalinkkiin oli yhdistetty kaksi FTP-asiakas ja kaksi FTP-palvelin solmua. Simulaatioissa verkon pullonkaulalinkille ohjattiin kaksi eri kuljetuskerroksen protokollaversiota käyttävää vuota. Pullonkaulalinkin kaistanleveydeksi määriteltiin 1,0 Mbps, jotta TCP- ja DCCP-yhteydet joutuivat rajoittamaan lähetysnopeuksiaan pullonkaulalla syntyvän ruuhkan takia. Pullonkaulalinkille oli myös tässä simulaatiossa lisätty verkon virheilyä kuvastavaa pakettihukkaa, jonka arvo vaihteli välillä 0,1 ja 4,0 prosenttia. RTT oli tutkimuksen tässä osiossa vakio 50 millisekuntia kaikissa skenaarioissa. Tutkimuksen aikaisempien TCP- ja DCCP-simulaatioiden läpivientiarvojen perusteella voidaan todeta pullonkaulalinkin ruuhkautuvan tässä simulaatiossa.

Taulukko 5: Protokollien oikeudenmukaisuus 1 Mbps:n pullonkaulalinkillä

	0,1 %	1 %	2 %	3 %	4 %
Tahoe / Reno	0.38 / 0.62	0.46 / 0.54	0.49 / 0.51	0.51 / 0.49	0.51 / 0.49
Tahoe / NewReno	0.37 / 0.63	0.41 / 0.59	0.43 / 0.57	0.44 / 0.56	0.46 / 0.54
Tahoe / Sack	0.35 / 0.65	0.43 / 0.57	0.44 / 0.56	0.45 / 0.55	0.47 / 0.53
Tahoe / Vegas	0.48 / 0.52	0.54 / 0.46	0.51 / 0.49	0.49 / 0.51	0.49 / 0.51
Tahoe / TFRC	0.43 / 0.57	0.42 / 0.58	0.42 / 0.58	0.42 / 0.58	0.43 / 0.57
Tahoe / TCP Like	0.94 / 0.06	0.77 / 0.23	0.62 / 0.38	0.52 / 0.48	0.48 / 0.52
Reno / NewReno	0.48 / 0.52	0.45 / 0.55	0.44 / 0.56	0.44 / 0.56	0.44 / 0.56
Reno / Sack	0.50 / 0.50	0.46 / 0.54	0.45 / 0.55	0.45 / 0.55	0.45 / 0.55
Reno / Vegas	0.60 / 0.40	0.56 / 0.44	0.51 / 0.49	0.49 / 0.51	0.47 / 0.53
Reno / TFRC	0.50 / 0.50	0.45 / 0.55	0.43 / 0.57	0.42 / 0.58	0.41 / 0.59
Reno / TCP Like	0.95 / 0.05	0.80 / 0.20	0.63 / 0.37	0.53 / 0.47	0.44 / 0.56
NewReno / Sack	0.51 / 0.49	0.51 / 0.49	0.50 / 0.50	0.50 / 0.50	0.50 / 0.50
NewReno / Vegas	0.63 / 0.37	0.61 / 0.39	0.58 / 0.42	0.55 / 0.45	0.53 / 0.47
NewReno / TFRC	0.51 / 0.49	0.50 / 0.50	0.47 / 0.53	0.47 / 0.53	0.47 / 0.53
NewReno / TCP Like	0.97 / 0.03	0.85 / 0.15	0.70 / 0.30	0.60 / 0.40	0.53 / 0.47
Sack / Vegas	0.61 / 0.39	0.60 / 0.40	0.57 / 0.43	0.55 / 0.45	0.53 / 0.47
Sack / TFRC	0.47 / 0.53	0.47 / 0.53	0.48 / 0.52	0.48 / 0.52	0.46 / 0.54
Sack / TCP Like	0.97 / 0.03	0.85 / 0.15	0.70 / 0.30	0.58 / 0.42	0.52 / 0.48
Vegas / TFRC	0.31 / 0.69	0.23 / 0.77	0.29 / 0.71	0.34 / 0.66	0.38 / 0.62
Vegas / TCP Like	0.45 / 0.55	0.43 / 0.57	0.42 / 0.58	0.43 / 0.57	0.45 / 0.55
TFRC / TCP Like	0.83 / 0.17	0.65 / 0.35	0.56 / 0.44	0.53 / 0.47	0.50 / 0.50

Tasapuolisuussimulaatioita suoritettiin 20-40 kappaletta viidellä eri pakettihukan arvolla kaikille 21 eri protokollaparille. Pullonkaulalinkin rajoittaman verkkokapasiteetin suhteellinen jakautuminen eri skenaarioissa löytyy taulukosta 5. Kokonaisläpivienti vaihteli noin 960 kpbs:n arvosta alle 800 kbps:n tasoon riippuen pullonkaulalinkille lisätyn pakettihukan arvosta.

TCP Vegas oli kaikkein suorituskykyisin TCP-variantti tutkimuksen luvussa 4.5 tehtyjen suorituskykymittausten perusteella. Tasapuolisuussimulaatiot osoittavat TCP Vegasin toimivan kuitenkin varsin konservatiivisesti mikäli se joutuu jakamaan ruuhkautuneen verkon resursseja muiden käyttäjien kesken. Sen saamat osuudet pullonkaulalinkin läpi menevästä liikenteestä olivat suurimmassa osassa skenaarioita luokkaa 40-45%. TCP Vegasin ja Renon jakaman osuuden muuttuminen siirryttäessä kohti suurempia pakettihukan arvoja kuvastaa hyvin TCP Renon ongelmia suurilla pakettihukan arvoilla.

DCCP TCP Like sai erittäin pieniä läpivientiarvoja kaikkien muiden paitsi TCP Vegasin kanssa. Sen saama osuus pullonkaulan kokonaisläpiviennistä jäi useiden protokollien parina alle 10 prosenttiin, mikäli lisätyn pakettihukan arvo oli pieni. TCP Liken läpivienti kasvoi selvästi pakettihukan kasvun myötä. Neljän prosentin lisätyllä pakettihukalla se sai puolet pullonkaulan verkkoresursseista käyttöönsä. Ilmiö selittynee osin suuren pakettihukan aiheuttamista ruuhkanhallinnan mekanismeista, jotka rajoittavat kilpailevan yhteyden lähetysnopeutta. Voidaan kuitenkin todeta että esimerkiksi TCP Vegas saavuttaa kyseisissä 50 millisekunnin RTT:n ja 4% pakettihukan olosuhteissa yli 900 kbps:n läpiviennin mikäli yhteys on yksin sen käytettävissä.

DCCP TFRC suoriutui tutkimuksen ensimmäisen osuuden simulaatioissa kaikkein parhaiten tehollisen läpiviennin osalta. Tasapuolisuussimulaatioiden perusteella TFRC käyttäytyy hyvin TCP-ystävällisesti. Taulukon 5 perusteella voidaan todeta kuinka TFRC:n saavuttama osuus pullonkaulalinkin resursseista vaihtelee välillä 49-59% kaikkien muiden paitsi TCP Vegasin ja TCP Liken kanssa. TCP Vegas saavutti selvästi muita pienempiä läpivientiarvoja TFRC:tä vastaan, jääden jopa vain 23 % osuuteen kokonaisläpiviennistä. TCP Vegas jäi myös DCCP TCP-Liken kanssa suoritetuissa simulaatioissa muiden suoritustason alapuolelle.

TCP Vegasin simulaatiotulokset vaihtelivat myös tässä tutkimuksen osassa selvästi eniten. Tulosten riittävän luotettavuustason saavuttamiseksi TCP Vegasille suoritettuja simulaatioita jouduttiin toistamaan yleensä kaikissa tapauksissa muita enemmän. Tasapuolisuusmittauksissa joitakin skenaarioita toistettiin 40 kertaa, jotta TCP Vegasin tulos voitiin kelpuuttaa riittävän tarkaksi. Simulaatioiden luotettavuustasosta löytyy lisäinformaatiota seuraavassa luvussa.

4.8 Tulosten luotettavuustaso

Simulaatiotutkimuksen tarkoituksena on saada mahdollisimman tarkka arvio tietystä ulostulomuuttujasta. Simulaattori käyttää syötteenä satunnaislukua, jonka perusteella simulaatiotapahtumat määritellään. Mitä lyhyempi käytetty simulaatioaika on, sitä suurempi vaikutus tuloksiin on yksittäisillä simulaatiotapahtumilla. Simulaatiotutkimuksen kannalta on erittäin tärkeää optimoida simulaatioaika riittävän pitkäksi, jotta saavutetaan riittävä luotettavuustaso. Tehokkaan simuloinnin ja tutkimuksen saavuttamiseksi tulee kuitenkin välttää liian pitkän simulaatioajan käyttöä, sillä tietyn luotettavuustason saavuttamisen jälkeen ei simulaatioajan pidennys tuo tutkimukseen enää lisäarvoa. Tulosten luotettavuutta voidaan lisätä suorittamalla useita simulaatioajoja samoja parametrejä käyttäen, mutta eri satunnaislukugeneraattorin siemenarvolla.

Tietyn simulaatioskenaarion ajojen perusteella voidaan laskea tunnusluku $\bar{X}(n)$, joka on n näytettä sisältävän otannan laskennallinen keskiarvo (14). Simuloitujen läpivientiarvojen tarkka keskiarvo tai keskihajonta eivät ole tiedossa. Näin ollen keskihajonta nσ korvataan muuttujalla s, joka on estimoitu keskihajonta eli standardivirhe. Näin määritelty jakauma voidaan kuvata t-jakaumana, jonka keskiarvo on $\bar{X}(n)$ ja keskihajonta $\frac{s}{\sqrt{n_f}}$. Keskihajonnassa käytetty n_f on vapausasteiden määrä, joka on yhtä kuin näytteiden määrä vähennettynä yhdellä.

Kaavassa 15 on luotettavuustaso keskiarvolaskelman $\bar{X}(n)$ mukaan, jossa δ on luotettavuusvälin pituuden puoliväli. Näin ollen $100(1-\alpha)$ luotettavuus sijaitsee $\bar{X} \pm \delta$ välillä.

$$\bar{X}(n) = \frac{1}{n}\sum_{i=0}^{n} x_i \tag{14}$$

$$P(|\bar{X}(n) - \mu| < \delta) = 1 - \alpha,\ 0 < \alpha < 1 \tag{15}$$

$$s^2[\bar{X}(n)] = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X}(n))^2 \tag{16}$$

Tämän tutkimuksen simulaatioissa käytettiin 10-40 toistokertaa jokaiselle skenaariolle. T-jakauman määrittelemä 0.025 kriittinen arvo yhdeksällä vapausasteella ja 95% luotettavuustasolla on $t^*_{0.025} = 2.26$ [24]. Kaava 17 määrittelee 95% luotettavuusalueen, joka riippuu t-jakauman kriittisen arvon lisäksi estimoidusta keskihajonnasta sekä simulaatiokerroista.

$$\bar{X}(n) \pm t^*_{0.025}\frac{s}{\sqrt{n}} \tag{17}$$

Simuloinnin suhteellinen tarkkuus kertoo kuinka paljon tulokset korkeintaan poikkeavat keskiarvosta tietyllä todennäköisyydellä (18). Yleisesti simulointien suhteelliseksi tarkkuudeksi valitaan 0,05 arvioidusta arvosta 95% luotettavuustasolla,

eli tulos poikkeaa todellisesta arvosta yli viisi prosenttia korkeintaan viiden prosentin todennäköisyydellä.

$$\varepsilon = \frac{s}{\bar{X}(n)} \tag{18}$$

Ensimmäisen osion simulaatioskenaarioiden keskimääräinen suhteellinen tarkkuus oli 3,78% kun käytössä oli 95% luotettavuustaso. TCP Vegasin tulokset vaihtelivat selvästi eniten ja kymmenen simulaatioajon perusteella sen suhteellinen tarkkuus on 7,6% kun kaikkien muiden protokollien tuloksien suhteellinen tarkkuus oli alle 3%. TCP Vegasin suhteellisen tarkkuuden tasoa voisi parantaa suorittamalla lisää simulaatioita, mutta tarkkuuden parannus ei muuttaisi tutkittujen ilmiöiden tuloksia. Tämän tutkimustyön päämääränä ei ole löytää mahdollisimman tarkkoja yksittäisiä arvioita tietyistä olosuhteista vaan tutkia protokollien käyttäytymistä laajalla parametriskaalalla.

Taulukko 6: TCP varianttien suhteelliset tarkkuudet ensimmäisen osion tuloksissa

RTT	4ms	10ms	20ms	30ms	40ms	50ms	60ms	70ms	80ms	90ms	100ms
Tahoe	2,3%	2,1%	2,1%	2,2%	2,1%	2,1%	2,4%	2,6%	2,7%	2,8%	2,8%
Reno	2,7%	2,7%	2,5%	2,6%	2,6%	2,5%	2,9%	2,9%	2,7%	2,9%	3,0%
Newreno	2,6%	2,5%	2,3%	2,2%	2,2%	2,3%	2,3%	2,5%	2,6%	2,7%	2,7%
Sack	2,5%	2,1%	2,1%	2,2%	2,2%	2,3%	2,6%	2,6%	2,7%	2,9%	2,9%
Vegas	7,8%	7,9%	7,3%	7,0%	6,3%	6,5%	6,4%	6,4%	6,9%	6,1%	7,5%

Tasapuolisuussimulaatioissa pyrittiin saavuttamaan vähintään viiden prosentin suhteellinen tarkkuus 95% luotettavuustasolla. Simulaatiotoistoja jouduttiin tekemään 20-40 kappaletta jokaista algoritmiparia kohden. Näin pystyttiin saavuttamaan tavoiteltu suhteellinen tarkkuus pois lukien DCCP TFRC ja TCP Like -pari, jonka simulaatiotuloksissa jäätiin 6,1% suhteelliseen tarkkuuteen 95% luotettavuustasolla.

Taulukko 7: Tasapuolisuussimulaatioiden suhteelliset tarkkuudet

	Vegas	Sack	New Reno	Reno	TFRC	TCP Like
Tahoe	1,9%	1,7%	2,0%	1,8%	3,0%	3,1%
TCP Like	4,7%	3,4%	4,1%	3,3%	6,1%	
TFRC	4,7%	3,6%	2,5%	2,6%		
Reno	2,5%	1,8%	2,0%			
New Reno	2,4%	1,9%				
Sack	2,3%					

4.9 Vertailu aikaisempiin tutkimuksiin

Kuljetuskerroksen protokollia on tutkittu paljon aikaisemmissa tutkimuksissa erilaisten analyyttisen mallien sekä simulaatioiden avulla.

Vuonna 2006 tehdyssä tutkimuksessa Alaa Seddik-Ghaleb et al. tutkivat kuuden eri TCP variantin energiatehokkuutta sekä läpivientiä ad hoc -verkoissa [26]. Kyseisen tutkimuksen tuloksia voidaan osittain verrata myös tämän tutkimuksen tuloksiin. Seddik-Ghaleb et al. totesivat TCP New Renon tarjoavan suurimman läpivientitason hyvin suurilla pakettihukan arvoilla. Sama ilmiö on havaittavissa tämän tutkimuksen kuvasta 8, jonka perusteella TCP New Reno saavuttaa parhaan läpivientiarvon pakettihukan ollessa yli 8 %. Seddik-Ghalebin tutkimuksen mukaan pienillä pakettihukan arvoilla TCP Vegas tarjoaa parhaan läpivientitason. Tulos on yhtenevä tässä tutkimustyössä saatuihin tuloksiin.

TCP Tahoen ja Renon läpivientitasoista on tehty useita tutkimuksia, joista esimerkiksi Anurag Kumarin tekemä *Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link* perehtyy Tahoen ja Renon läpivientieroihin simuloimalla sekä analyyttisin mallein [27]. Kumarin tutkimuksen mukaan TCP Renon suorituskyky on Tahoeta pienempi mikäli lähetysikkunan koko on hyvin pieni tai pakettihukka kasvaa suureksi. Kyseinen ilmiö tuli hyvin havainnollistettua myös tämän tutkimuksen simulaatioissa, joiden perusteella pystytään huomaamaa hyvin sekä pakettihukan että RTT:n vaikutus TCP Renon ja Tahoen paremmuusjärjestykseen.

Tämän tutkimuksen tasapuolisuussimulaatiossa todettiin TCP Vegasin suorituskyvyn kärsivän sen joutuessa kilpailemaan pullonkaulalinkin resursseista jonkin toisen variantin kanssa. Osakan yliopistossa vuonna 2000 tehdyssä tutkimuksessa todetaan TCP Vegasin kärsivän etenkin drop-tail jononhallinta-algortitmiä käyttävissä ruuhkaisissa verkoissa [28]. Kyseisessä tutkimuksessa oli testattu etenkin TCP Vegasin ja Renon tasapuolisuutta. Tämän diplomityön tulosten perusteella voidaan todeta ongelman ilmenevän myös TCP New Renon, Sack:n sekä DCCP TRRC:n kanssa. Corkin yliopistossa vuonna 2003 tehdyssä tutkimuksessa todettiin TCP Vegasin suorituskyvyn ongelmat TCP Renoa ja TCP Sack:ta kohtaan tutkimuksessa, jossa perehdyttiin eri TCP varianttien soveltuvuuteen SIP-liikenteen kuljetuskerroksen protokollaksi [29]. TCP Renon yleisyys Internetissä aiheuttaa heidän mukaansa todella suuren haasteen TCP Vegasin mahdolliselle käytölle.

DCCP-varianttien tasapuolisuudesta on tehty joitakin tutkimuksia, joista mainittakoon *Transport Protocol Throughput Fairness* vuodelta 2009. Kyseisessä tutkimuksessa S. Bhatti sekä M. Bateman selvittivät CCID-3:n tasapuolisuutta muutamia TCP-variantteja kohtaan. Heidän tutkimuksensa perusteella DCCP TFRC kohtelee TCP-yhteyksiä varsin tasapuolisesti, mutta heidän mukaansa tasapuolisuus on riippuvainen yhteyden viiveestä. Tämän tutkimuksen TFRC-tasapuolisuussimulaatioiden tulokset tukevat heidän tuloksiaan.

Tässä työssä ilmenneille CCID-2:n suurille ongelmille tasapuolisuussimulaatioissa ei löydy tukea aiemmista tutkimuksista. Esimerkiksi vuonna 2008 Brasiliassa tehdyn tutkimuksen *On the performance of TCP, UDP and DCCP over 802.11 g networks* mukaan kyseistä ilmiötä ei esiintynyt ainakaan heidän testeissään [31].

4.10 Johtopäätökset

Tutkimuksen päämääränä oli selvittää kuinka yleiset ruuhkanhallintaa käyttävät kuljetuskerroksen protokollat selviytyvät olosuhteissa, joissa verkon häiriötekijöiden takia pakettihukka on suurta. Ruuhkanhallinta-algoritmien reagointi aina samalla tavalla riippumatta paketin katoamisen syystä aiheuttaa suorituskykyongelmia pakettihukan johtuessa esimerkiksi huonolaatuisesta radiotiestä.

Simulaatioiden ensimmäisen osuuden perusteella voidaan määritellä kuinka seitsemän eri tutkittua algoritmia suoriutuvat pakettihukan ja RTT:n eri arvoilla verkossa, jossa ei esiinny muun liikenteen aiheuttamaa ruuhkaa. TCP Renon suorituskykyongelmien taso ja riippuvuus RTT:n ja pakettihukan suhteesta pystyttiin hyvin tuomaan esille simulaatioissa. TCP New Renon ja Sack:n hyvin tasavertainen ja hyvä suoritustaso tuli hyvin havainnollistettua. TCP Vegasin poikkeava luonne muihin algoritmeihin verrattuna näkyi simulaatiotuloksissa suoritustason hyvin voimakkaana vaihteluna. TCP Vegasin läpivienti oli simulaatioissa DCCP-algoritmien lisäksi kaikkein vaihtelevinta ja kyseisille algoritmeille jouduttiinkin suorittamaan eniten simulaatiotoistoja riittävän luotettavuustason saavuttamiseksi.

Simulaatioiden toisessa osuudessa tutkittiin eri ruuhkanhallinta-algoritmien tasapuolisuutta pullonkaulalinkillä. Tulosten perusteella voidaan todeta linkille lisätyn pakettihukan vaikuttavan merkittävästi algoritmien keskinäiseen tasapuolisuuteen. Esimerkiksi TCP Reno pärjäsi hyvin pienellä pakettihukalla, mutta pakettihukan kasvun myötä sen saama suhteellinen osuus linkin läpiviennistä laski merkittävästi. TCP Vegas osoittautui tasapuolisuussimulaatioissa yllättävän passiiviseksi skenaarioissa, joissa pullonkaulalinkin pakettihukka oli pieni.

Tulosten perusteella voidaan todeta TCP Vegasin tarjoavan parhaan tehollisen läpivientitason ruuhkattomassa langattomassa verkossa, jossa pakettihukka on 1-5 % välillä. Tätä suuremmilla pakettihukan arvoilla TCP New Reno suoriutuu paremmin. Suuren pakettihukan ja pienen viiveen olosuhteissa TCP Sack osoittautui parhaaksi TCP-variantiksi tehollisen läpiviennin osalta. Samanlaisissa olosuhteissa DCCP TFRC pystyi saavuttamaan kuitenkin huomattavasti TCP Sack:ta suurempia läpivientiarvoja, joten mikäli verkkosovellukselta ei edellytetä ehdotonta luotettavuutta on DCCP TFRC läpivientitehokkuuden kannalta paras valinta. Langallisessa verkkoliikenteessä valitsevissa pienen pakettihukan ja kohtuullisen viiveen olosuhteissa TCP Vegas suoriutui parhaiten.

Taulukossa 8 on simulaatioiden ensimmäisen osion perusteella arvioitu parhaimman tehollisen läpiviennin tarjoava TCP-variantti verkon erilaisilla pakettihukka ja RTT arvoilla.

Taulukko 8: Optimaalinen TCP-variantti eri pakettihukan ja RTT:n arvoilla

Olosuhteet		Suurin läpivienti
Pakettihukka	RTT	Variantti
Pieni	Pieni	Vegas
Pieni	Keskisuuri	Vegas
Pieni	Suuri	Sack
Keskisuuri	Pieni	Vegas
Keskisuuri	Keskisuuri	Vegas
Keskisuuri	Suuri	Vegas
Suuri	Pieni	Sack
Suuri	Keskisuuri	New Reno
Suuri	Suuri	New Reno

5 Yhteenveto

Tämän tutkimuksen päämääränä oli perehtyä ruuhkanhallinta-algoritmien toimintaan verkoissa joiden pakettihukka on suuri. Työn kirjallisuuskatsauksessa tutkitaan verkon ruuhkautumisen syitä sekä kuinka ruuhkautuminen on havaittavissa. Yleisin indikaatio ruuhkautumisesta on lähetettyjen pakettien katoaminen verkossa. Lähettäjä ei yleensä tiedä miksi paketti katoaa ja pakettihukkaan reagoidaan yleensä samalla tavalla. Näin ollen todellinen ruuhkautuminen ja verkon virheilystä tai radiotien häiriöistä johtuvat pakettien katoamiset ja korruptoitumiset aiheuttavat molemmat lähetysnopeuden pienentämisen. Kyseisen toimintamallin takia lähetysnopeuden pienentäminen ruuhkattomassa mutta virheilevässä verkkoympäristössä aiheuttaa verkkoresurssien alikäyttöä ruuhkanhallinta-algoritmien tarpeettoman suuren vasteen takia.

Kuljetuskerroksen protokollat ovat vastuussa tietoliikenneyhteyksien ruuhkanhallinnasta ja kyseinen ominaisuus löytyy TCP, SCTP sekä DCCP -protokollista. Kirjallisuuskatsauksessa perehdytään viiden eri TCP-variantin sekä kahden DCCP-version ruuhkanhallinnallisiin ominaisuuksiin. Tutkimukseen valittiin TCP:n versiot Tahoe, Reno, New Reno, Sack sekä Vegas. TCP ja DCCP -protokollat eroavat toisistaan useiden ominaisuuksiensa osalta ja tärkeimpänä erona voidaan pitää TCP:n luotettavaa tiedonsiirtoa verrattuna DCCP:n varmistamattomaan tiedonsiirtoon. DCCP-protokollasta on suunniteltu ilman ruuhkanhallinnallisia ominaisuuksia olevan UDP:n mahdollista korvaajaa sovelluksiin, joissa tiedonsiirron tehokkuus on luotettavuutta tärkeämpää.

Diplomityössä käytettiin tutkimusmenetelmänä systemaattista simulointia, jonka niin sanotussa Jain-mallissa tutkimus jaetaan kahdeksaan eri vaiheeseen. Esiohjelmallinen vaihe aloitetaan määrittelemällä tutkimusongelma sekä päämäärä. Tämän lisäksi verkkomallin suunnittelu, dynaamisten sekä staattisten parametrien valinta sekä mitattavien suorituskykymetriikoiden määrittely kuuluvat esiohjelmalliseen osuuteen.

Systemaattisen simuloinnin ohjelmallinen vaihe sisältää simuloitavan verkkomallin rakentamisen, varsinaisen simuloinnin sekä tulosten keräämisen ja tulkinnan. Tämän tutkimuksen simulaatiot toteutettiin ns2-simulaattorilla ja tulosten analysoinnissa ja esittämisessä käytettiin itse koodattuja työkaluja sekä Matlab-ohjelmistoa.

Tämä tutkimuksen simulointityö jakautuu kahteen eri osioon, joista toisessa tutkittiin tehollisen läpiviennin tasoja ruuhkautumattomassa verkossa ja toisessa läpivientien suhdetta kahden eri kuljetusprotokollan kilpaillessa pullonkaulalinkin resursseista. Simulaatioskenaarioissa käytettiin dynaamisina parametreinä RTT:tä sekä pullonkaulalinkin pakettihukkaa. RTT sai simulaatiossa arvoja 2-100 ms ja pakettihukka 0,1-10,0 %. Viiveen resoluutio oli 10 ms ja pakettihukan 0,1 % ja näin ollen erilaisia viive-pakettihukka pareja oli 1100 kappaletta.

Simulaation ensimmäisen osan tuloksena syntyi useita kolmiulotteisia kuvaajia eri TCP- ja DCCP-versioiden tehollisten läpivientien arvoista virheilevän 10 Mbps-linkin läpi. Kuvaajien perusteella pystyttiin analysoimaan protokollien suorituskykyä erilaisilla viive-pakettihukka -yhdistelmillä verkossa, jossa ei esiintynyt muusta liikenteestä aiheutuvaa ruuhkaa. Tulosten perusteella tutkitiin esimerkiksi TCP Ta-

hoen ja TCP Renon sekä TCP Sack:n ja TCP New Renon suorituskykyä toisiinsa nähden. TCP Renon ongelmat suuren pakettihukan olosuhteissa pystyttiin hyvin todentamaan sekä rajaamaan, missä olosuhteissa ongelma aiheuttaa suorituskyvyn laskun alle TCP Tahoen (Kuva 16). TCP Renon suorituskyky oli heikkoa mikäli pakettihukan arvo oli suuri, mutta myös RTT:n vaikutus oli merkittävää ilmiön esiintymiselle. Syy tähän on lähetysikkunan koon suhde pakettihukkaan eri viiveillä.

TCP Sack ja TCP New Reno olivat tutkimuksen kaikissa osissa erittäin tasavertaisia keskenään. TCP Sack pärjäsi kuitenkin hieman paremmin viiveen tai pakettihukan ollessa pieni, mutta mikäli molemmat dynaamiset parametrit saivat suuria arvoja, niin TCP New Reno pärjäsi vertailussa paremmin.

TCP Vegas poikkeaa tutkimuksen muista TCP-varianteista käyttämänsä ruuhkantunnistuksen sekä siihen reagointinsa osalta. Simulaation ensimmäisen osion tuloksissa TCP Vegas sai kokonaisuutena selvästi suurimpia läpivientiarvoja pakettihukan pysyessä alle kuuden prosentin (kuva 18). TCP Vegasin läpivientikuvaajan perusteella voidaan todeta sen tarjoaman läpiviennin olevan kuitenkin varsin herkkä suurelle viiveelle ja pakettihukalle.

DCCP-simulaatioissa perehdyttiin TFRC:n ja TCP Like:n teholliseen läpivientiin täysin samoilla verkkoparametreilla kuin TCP:lle suoritetuissa simulaatioissa. Molemmat datagrammipohjaiset protokollat osoittautuivat hyvin suorituskykyisiksi ruuhkautumattomassa verkossa. Erityisesti TFRC:n saamat arvot suuren pakettihukan ja pienen viiveen yhdistelmillä olivat äärimmäisen suuria verrattuna muihin (kuva 25). Myös TCP Like sai lähes koko tutkiskelualueella noin 20 % TCP Tahoeta suurempia läpivientiarvoja (kuva 26).

Simulaatioiden toisessa osuudessa selvitettiin tutkittujen seitsemän protokollan tasapuolisuutta 1 Mbps:n pullonkaulalinkin läpi, jonka dynaamiset parametrit olivat pakettihukan osalta samat kuin ensimmäisessä osuudessa, mutta RTT:n arvoksi määriteltiin vakio 50 ms. Seitsemän eri protokollan joukosta muodostettiin 21 kahden protokollan paria, joiden suhteellista osuutta pullonkaulalinkin läpiviennistä tutkittiin. Näin pystyttiin arvioimaan protokollien suorituskykyä sekä tasapuolisuutta verkossa, jossa pakettihukkaa esiintyy ruuhkautumisen sekä jonkin muun syyn takia.

TCP Tahoen sekä Vegasin saavuttama suhteellinen osuus kasvoi lisätyn pakettihukan kasvun myötä (taulukko 4). TCP Vegasin saamat arvot olivat noin 40 % luokkaa 0,1 % pakettihukalla, mutta paranivat pakettihukan kasvun myötä. Sama ilmiö on havaittavissa ensimmäisen osion tuloksista. TCP Reno saavutti hyviä suhteellisia osuuksia pienellä pakettihukalla, mutta pakettihukan kasvun myötä Renon ongelmat näkyivät myös tässä simulaatiossa.

DCCP TFRC kohteli muita protokollia pääsääntöisesti hyvin tasapuolisesti ja sen saavuttama suhteellinen osuus vaihteli pääsääntöisesti välillä 50-55%. Mielenkiintoinen ilmiö löytyi TFRC:n aiheuttamista ongelmista TCP Vegasille, joka jäi joissakin olosuhteissa vain neljänneksen osuuteen kokonaisläpiviennistä. TFRC käyttää lähetysnopeuteen perustuvaa ruuhkanhallintaa lähetysikkunan sijaan. TCP Like saavutti erittäin pieniä osuuksia kokonaisläpiviennistä pakettihukan pienillä arvoilla. TCP Like:n suhteellinen osuus kasvoi lähelle puolta pakettihukan arvolla 4%.

Simulaatiotulosten luotettavuudeksi pyrittiin saamaan 5 % suhteellinen tarkkuus 95 % luotettavuustasolla. Suurin osa skenaarioista saavutti 3 % suhteelli-

sen tarkkuuden tavoitellulla luotettavuustasolla, mutta TCP Vegasin sekä DCCP -simulaatioiden osalta tarkkuus jäi hieman tavoitellusta suhteellisen tarkkuuden jäädessä kuitenkin kaikissa tapauksissa reilusti alle 10 %. DCCP-simulaatioita toistettiin joissakin tapauksissa 40 kertaa yhtä skenaarioita kohden, jotta saavutettiin tulosten kelvollinen luotettavuustaso.

Simulaatiotulosten perusteella pystytään määrittelemään tutkittujen TCP- ja DCCP-versioiden suorituskykyä erilaisilla viiveillä ja pakettihukan määrillä. Tämän tutkimustyön perusteella pystytään määrittelemään mikä ruuhkanhallinta-algoritmi tulisi valita erilaisten verkko-olosuhteiden perusteella.

Viitteet

[1] Mamatas, L. Harks, T Tsaoussidis, V. Approaches to Congestion Control in Packet Networks. *Journal of Internet Engineering, Vol 1, No. 1* Tammikuu 2007.

[2] Welzl, M. *Network Congestion Control - Managing Internet Traffic* , Wiley, 2005.

[3] Guizani, M. *Wireless Communication Systems and Networks* , Kluwer, 2004.

[4] Hassan, M. Jain, R. *High Performance TCP/IP Networking* , Pearson Prentice Hall, 2004.

[5] Brakmo, L.S. et al. TCP Vegas: New Techniques for Congestion Detection and Avoidance. *Proceedings of the ACM SIGCOMM* s. 24-35, 1994.

[6] Brakmo, L.S. et al. TCP Vegas: End to End Congestion Avoidance on global Internet. *IEEE Journal on Selected Areas in Communications, 13(8)* s. 1465-1480, 1995.

[7] M. Mathis, J. Semke, J. Mahdavi and T. Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communication Review 27* 1997.

[8] J. Padhye, V. Firoiu, D. Towsley and J. Kurose. TModeling TCP throughput: A simple model and its empirical validation. *Proceedings of the ACM SIGCOMM* 1998.

[9] Mathis, M. Mahdavi, J. Forward acknowledgement: Refining TCP congestion control. *Proceedings of the ACM SIGCOMM* s. 281-291, 1996.

[10] Jacobson, V. Congestion avoidance and control. *Symposium Proceedings on Communications Architectures and Protocols* , s. 314-329, ACM Press, 1988.

[11] Tanenbaum, A.S. *Computer Networks, 4th Ed.* , Prentice Hall, 2002.

[12] Kohler e. et al. Datagram Congestion Control Protocol (DCCP), Internet Engineering Task Force, RFC 4340, 2006.

[13] Floyd S. et al. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control, Internet Engineering Task Force, RFC 4341, 2006.

[14] Floyd S. et al. Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC), Internet Engineering Task Force, RFC 4342, 2006.

[15] Matsson, Nils-Erik, A DCCP module for ns-2. Examensarbete. Luleå Tekniska Universitet

[16] Ramadan, Wassim, Ns2 patch from 2.31 ported to 2.34. Viitattu 11.4.2010. Saatavissa: `http://eugen.dedu.free.fr/ns2/dccp-ns2.34.patch`

[17] Stewart R. et al. Stream Control Transmission Protocol, Internet Engineering Task Force, RFC 2960, 2000.

[18] Transmission Control Protocol, Internet Engineering Task Force, RFC 793, 1981.

[19] Postel J. User Datagram Protocol, Internet Engineering Task Force, Internet Engineering Task Force, RFC 768, 1980.

[20] Allman M. et al. TCP Congestion Control, Internet Engineering Task Force, RFC 2581, 1999.

[21] Floyd S. et al, The NewReno Modification to TCP's Fast Recovery Algorithm, Internet Engineering Task Force, RFC 3782, 2004.

[22] Mathis M. et al. TCP Selective Acknowledgement Options, Internet Engineering Task Force, RFC 2018, 1996

[23] Floyd s. et al. An Extension to the Selective Acknowledgement (SACK) Option for TCP, Internet Engineering Task Force, RFC 2883, 2010.

[24] Kreyszig. *Advanced Engineering Mathematics, 7th ed.* , Wiley, 1993.

[25] The Network Simulator - ns2. Viitattu 12.1.2010. Saatavissa: `http://www.isi.edu/nsnam/ns`

[26] Seddik-Ghaleb A. et al, *A performance study of TCP variants in terms of energy consumption and average goodput within a static ad hoc environment* , Proceedings of the 2006 international conference on Wireless communications and mobile computing, 2006.

[27] Kumar A., *Comparative Performance Analysis of Versions of TCP in a Local Network with a Lossy Link* ,IEEE/ACM Transactions on Networking (TON) archive, Volume 6 , Issue 4, 1998.

[28] Kurata K., Hasegawa Go, *Fairness Comparisons between TCP Reno and TCP Vegas for Future Deployment of TCP Vegas* , Proceedings of INET 2000, 2000.

[29] Lulling M. Vaughan J., *An investigation of TCP throughput variation for SIP traffic* ,ACM International Conference Proceeding Series; Vol. 49, 2003.

[30] Bhatti S., Bateman M., *Transport Protocol Throughput Fairness* ,Journal of Networks, Vol. 4, no. 9, November 2009.

[31] de Sales L. M. et al. *On the performance of TCP, UDP and DCCP over 802.11 g networks* ,Proceedings of the 2008 ACM symposium on Applied computing, 2008.