Quality of Service -sovelluskehyksen suunnittelu

Sami Venetjoki

Tampereen Yliopisto
Tietojenkäsittelytieteiden laitos
Pro gradu –tutkielma
Lokakuu 2003

Tampereen Yliopisto
Tietojenkäsittelytieteiden laitos
Sami Venetjoki: Quality of Service -sovelluskehyksen suunnittelu
Pro gradu –tutkielma, 82 sivua
Lokakuu 2003

Quality of Service (QoS) –tekniikoita esitetään ratkaisuksi tietoverkkojen ruuhkautumiseen ja kriittisen liikenteen kulkemisen takaamiseksi. Tekniikat palvelun tason ja laadun takaamiseksi ovat kuitenkin periaatteiltaan ja toteutukseltaan hyvin erilaisia ja sovelluksien kehittäjien täytyy rakentaa sovellukset yleensä tiettyä tekniikkaa silmälläpitäen. Tällä tavalla toimiminen ei tarjoa hedelmällistä maaperää uusien, QoS-tietoisten sovellusten kehittämiselle, koska asiakkaat eivät ole valmiita ostamaan sovelluksen mukana erillistä QoS-ratkaisua, varsinkin jos käytössä on jo jokin QoS-tekniikka, jota voitaisiin käyttää hyväksi. Tutkin tässä pro gradu -tutkielmassa mahdollisuuksia löytää eri QoS-tekniikoille yhteisiä nimittäjiä siten, että niistä voidaan johtaa yleisesti käytettävä sovelluskehys, jota käyttäen voidaan luoda ja ottaa käyttöön QoS-tekniikoita hyödyntäviä sovelluksia.

1. Johdanto

Quality of Service (QoS) on ollut eri muodoissaan käytössä jo vuosikymmeniä. Yleisimmissä verkkoprotokollissa, kuten esimerkiksi IP:ssä (Internet Protocol), on varauduttu liikenteen luokitteluun sen varalta, että eri paketteja tarvitsisi kohdella verkon solmuissa eri tavalla. IP-protokollassa tähän luokitteluun on varattu erityiset TOS-bitit (Type Of Service) IP-paketin otsikkokentissä. Aiemmin tietoverkoissa ei ole kuitenkaan ollut suurta tarvetta kohdella eri dataliikennettä eri tavalla, koska kapasiteetti ei ole ollut suurena ongelmana, eivätkä palvelun laatua parantavat tekniikat ole yleistyneet odotetulla tavalla. Tänä päivänä kuitenkin liikennemäärät ovat kasvaneet sellaisiin mittoihin, että pelkällä verkkojen kapasiteetin kasvattamisella ei pystytä takaamaan verkon käyttäjille toimivaa palvelua [Shepard, 2000]. Toki tulevaisuudessa yhä tehokkaammat liikennöintiratkaisut tarjoavat enemmän kapasiteettia ja tätä myös tarvitaan, mutta on vain ajan kysymys kunnes alati kasvava datamäärä verkoissa kuluttaa tämänkin loppuun. Aina kapasiteetin lisääminen ei ole edes taloudellisesti kannattavaa, vaan samaan palvelun tason ja laadun nousuun voidaan päästä ilman kalliita investointeja. Tiedon määrä verkoissa joka tapauksessa kasvaa, ja rajusti yleistyvät laajakaistaliittymät rohkaisevat käyttäjiä siirtämään yhä suurempia ja suurempia tietomääriä siirtonopeuksien kasvaessa.

Tietoverkkojen rakentajilla ja sovellusten tekijöillä onkin syytä miettiä miten verkoissa liikkuva kriittinen data saadaan kulkemaan luotettavasti, kun toissijaisen liikenteen (Internet-selailu, videokuva tms.) määrä kasvaa. Erilaiset QoS-teknologiat antavat mahdollisuuden taata eri liikennetyypeille riittävän palvelun vaikka tietoverkon liikenne olisikin ruuhkautunut, mutta teknologioita on paljon erilaisia ja sovellusten kehittäjien näkökulmasta olisi kaupallisesti suotavaa, että sovellus voisi toimia erilaisissa QoS-ympäristöissä ilman erillistä räätälöintiä kullekin QoS-tekniikalle erikseen. QoS-tekniikan valitseminen sovellusten mukaan (ja toisinpäin) ei tyydytä maksavaa asiakasta ja palvelujen leviäminen ei tule yleistymään ilman tekniikan ja sen käyttäjien yhteistyötä.

Tässä pro gradu -tutkielmassa pyrin selvittämään miten sovellusten ja QoS-tekniikoiden välinen alue voitaisiin toteuttaa siten, että sovellukset voitaisiin rakentaa ilman riippuvuuksia tiettyihin QoS-tekniikoihin. Aluksi käsittelen tutkielmassani yleisimpiä QoS-tekniikoita ja niiden periaatteita (luvut 2-3), tämän jälkeen selvitän näille tekniikoille yhteiset vaatimukset (luku 4) ja lopuksi tuotan vaatimuksista sovelluskehyksen, jota käyttäen voidaan rakentaa QoS-tietoisia sovelluksia ilman sitoutumista mihinkään tiettyyn QoS-tekniikkaan (luvut 5-6).

Tämän pro gradu -tutkielman tuloksena olen hahmotellut joukon rajapintoja ja palveluja, joiden avulla geneerinen QoS-parametrien varastointi, hallinta ja jakelu voidaan hoitaa riippumatta käytettävästä QoS-tekniikasta. Rajapintoja ja niiden käyttöä havainnollistetaan esimerkinomaisesti tutkielman lopussa C++-koodin muodossa.

2. Alueen kuvaus

2.1 QoS:n periaatteet

QoS:n perusideana on jollain menetelmällä taata käyttäjälle tietty palvelun taso verkkoresursseja käytettäessä. QoS-tekniikat poikkeavat normaalisti verkoissa käytettävästä ns. best effort –palvelusta siten, että käytössä olevista verkkoresursseista pidetään kirjaa ja tietyt liikenneluokat asetetaan prioriteetiltaan eri tasoille palvelutason mukaan. Best effort –palvelulla tarkoitetaan, että verkko välittää liikennettä sen hetkisellä maksiminopeudella. Tämä nopeus saattaa vaihdella rajusti riippuen verkon kuormituksesta. Käytännössä kyse on siis tietoliikenteen ohjaamisesta aivan kuten normaalissa tieliikenteessäkin: vilkkaasti liikennöidyt reitit rakennetaan leveämmiksi ja prioriteetiltaan kiireellisempi liikenne on etuoikeutettua muuhun normaaliin liikenteeseen verrattuna (vrt. busseille ja takseille varatut kaistat tai hälytysajo).

QoS-tekniikoiden ideana on siis asettaa tietoverkkojen liikenne erilaisiin kategorioihin siten, että sovelluksille voidaan antaa takeita yhteyden toimivuudesta kaistanleveyden, viiveen, luotettavuuden ja hinnan suhteen. Eri QoS-tekniikat käyttävät erilaisia lähestymistapoja asiaan.

Toiset tekniikat luokittelevat liikenteen pakettikohtaisesti siten, että eri verkon solmut käsittelevät datapaketit eri prioriteeteilla. Suuremmalla prioriteetilla leimattu paketti käsitellään nopeammin kuin pienemmällä prioriteetilla oleva paketti. Pienemmällä prioriteetilla oleva paketti voidaan myös jättää käsittelemättä, jos verkon laitteiden kapasiteetti ei pysty käsittelemään kaikkia tietoverkossa liikkuvia paketteja. Tällaiset tekniikat eivät välttämättä vaadi sovellukselta mitään erityistä, vaan datapaketit voidaan leimata sen jälkeen kun sovellus on ne verkkoon lähettänyt. Nämä tekniikat toteutetaan pääosin verkon reitittimissä, joissa eri leimoilla varustetut paketit ohjataan prioriteeteiltaan erilaisiin jonoihin. Yleensä näillä tekniikoilla toteutetaan vain yksittäisiä verkkosegmenttejä, eikä tekniikka soveltuisikaan isompaan mittakaavaan, koska eri palveluntarjoajat eivät noudata mitään yleistä periaatetta pakettien leimaamisessa. Tällaiset verkot onkin varustettu erityisillä verkkosolmuilla, jotka käsittelevät verkkoon ulkopuolelta tulevan ja verkosta lähtevän liikenteen siten, että tulevan liikenteen datapaketit leimataan käytetyn luokitteluperiaatteen mukaan ja ulospäin kulkevat datapaketit taas riisutaan verkossa käytetyistä leimoista, jotta liikenne muissa verkoissa ei häiriintyisi. Esimerkkeinä tällaisista tekniikoista ovat DiffServ (Differentiated Services) ja hieman laajempaan mittakaavaan tarkoitettu MPLS (Multi-Protocol Label Switching). Näitä kahta tekniikkaa käsitellään tässä tutkielmassa tarkemmin aliluvuissa 3.3 ja 3.6.

Toiset tekniikat taas perustuvat siihen, että sovellus pyytää verkkoinfrastruktuurilta tiettyä kaistaa verkon kapasiteetista siten, että palvelun vaatima dataliikenne voidaan pitää vakiona. Näissä tekniikoissa siis verkon infrastruktuurin täytyy pitää jonkinlaista tilastoa verkon varatusta kapasiteetista, jotta sovelluksille voidaan kertoa, onko niiden pyytämä kapasiteetti taattavissa. Jotta kapasiteetti voitaisiin taata, on verkon tuettava ko. teknologiaa kaikissa solmuissa lähettäjältä vastaanottajalle. Jos välissä on laitteita jotka eivät ole QoS-tietoisia, niin kapasiteettia ei voida tällöin varmasti taata. Tekniikat eivät välttämättä vaadi, että sovellukset varaavat juuri tietyn osan verkon kapasiteetista, vaan sovellukset voivat antaa raja-arvot, joiden mukaan verkko varaa resurssit. Kaistanleveys voidaan antaa ala- ja ylärajojen avulla, jolloin verkko voi varata kapasiteetin riittäessä resurssit läheltä ylärajaa

ja tarvittaessa vähentää sovellukselle varattua kaistaa, jos varausten määrä kasvaa eikä aiemmin myönnetty kaista enää ole taattavissa. Tekniikat siis toimivat kaksisuuntaisesti siten, että verkon varausten muuttumisesta voidaan tiedottaa varauksia tehneille sovelluksille, jotka voivat edelleen itse ratkaista voivatko ne toimia muuttuneessa ympäristössä sen tarjoamalla kapasiteetilla. Tekniikoilla voidaan tehdä verkkoihin kiinteitä varauksia, jolloin kaikesta kapasiteetista tietty kiinteä osa varataan tietyille sovelluksille. Tällä voidaan vähentää sovelluksille tarjottavien kaistojen muuttumisia ja vähentää siten ongelmia, joita sovelluksille voi muutoksista tulla. Esimerkkeinä tällaisista tekniikoista ovat ATM (Asynchronous Transfer Mode) ja RSVP (Resource Reservation Protocol). Aliluvuissa 3.2 ja 3.4 käsitellään näiden tekniikoiden ominaisuuksia tarkemmin.

2.2 Sovelluskehysten idea

Sovelluskehysten (framework) ideana on kasata tiettyyn sovellusalueeseen liittyvät yleiset asiat yhdeksi kokonaisuudeksi siten, että uusien, samoja asioita hyödyntävien sovellusten kehittämisessä voidaan hyödyntää jo hyväksi havaittua menettelyä [Fayed et al, 1999]. Sovelluskehysten avulla saadaan kasvatettua projektien luotettavuutta, nopeutettua sovelluskehitystä ja helpotettua testausta, koska alueeseen liittyvä perustoiminnallisuus on jo aiempien projektien aikana hiottu toimivaksi.

Sovelluskehyksiä ei yleensä voida rakentaa ennen kuin ensimmäistäkään sovellusta on tehty, koska jostain on lähdettävä liikkeelle ja tunnistettava yleiset osat jotka sovelluskehyksen tulisi toteuttaa. Sovelluskehysten kehitys onkin iteratiivinen prosessi, ne kehittyvät ensimmäisestä kehitysvaiheestaan aina edelleen kun uusia projekteja valmistuu ja aiemmissa iteraatiovaiheissa havaitsematta jääneet puutteet tulevat esille. Ensimmäisellä iteraatiokierroksella ei siis päästä vielä lähellekään optimitilannetta, vaan hyvä sovelluskehys vaatii useita iteraatioita.

Sovelluskehys koostuu tavallisesti abstraktista ja konkreettisesta osasta, eli joukosta toimintaohjeita, tapoja ja tietoja, joita alueen sovellusten rakentamisessa tarvitaan, sekä valmiista rakennuspalikoista, joiden avulla voidaan toteuttaa tietyt osat toiminnallisuudesta ilman uudelleenkoodausta.

Nämä rakennuspalikat voivat olla esimerkiksi valmiita luokkarakenteita, valmiiksi käännettyjä binäärisiä moduleita, tai vaikkapa velho-tyyppinen toiminne sovelluskehitysympäristössä, jonka avulla voidaan generoida valmis koodirunko sovellukselle. Velhon tuottamassa koodissa kaikille samantyyppisille sovelluksille yhteinen perustoiminnallisuus generoidaan automaattisesti, käyttäjä vain lisää oman sovelluksensa vaatimat erityistoiminnallisuudet velhon generoiman koodin joukkoon.

Sovelluskehyksistä saadaan siis monenlaisia hyötyjä: toimintatavat ja ohjeet kasaavat dokumentaatiotyyppistä tietoa sovellusten toiminnasta ja rakentamisesta, valmiit rakennuspalikat lisäävät järjestelmän luotettavuutta, kun jo sovelluksen rakennusvaiheessa iso osa toiminnallisuudesta on todettu toimivaksi, ja lopuksi velho-tyyppiset toiminnot ohjaavat sovellusten koodin rakennetta ylläpidettävämpään muotoon, joka on tärkeää esimerkiksi tilanteessa, jossa sovelluksen ylläpito- ja kehitysvastuu siirtyy toiselle henkilölle. Jos uusi vastuullinen on jo ollut tekemisissä saman sovelluskehitysympäristön kanssa, hänen on helpompi siirtyä hoitamaan uuden vastuualueen tehtäviä sovellusten rakenteiden ollessa samankaltaisia.

Sovelluskehysten käyttöön liittyy myös riskejä. Ihmiset tekevät virheitä ja samaa koodipohjaa käytettäessä voi tulla tilanteita, joissa jokin virhetilanne ei tule esiin normaaleissa testaustilanteissa. Tällöin piilevä vika saattaa vaikuttaa useaan sovellukseen ja korjaustoimenpiteet vaativat paljon työtä. Onkin siis erittäin tärkeää, että sovelluskehykset ja niitä käyttävät sovellukset testataan huolellisesti jokaisen sovelluskehyksen iterointikierroksen jälkeen, jotta tällaisilta tilanteilta vältyttäisiin. Monimutkaisen sovelluskehyksen oppimiskynnys voidaan myös nähdä riskitekijänä, koska opetteluun kuluva aika saattaa pidentää myös sovellusten valmistumisaikataulua. Tällainen riskitekijä on mahdollinen erityisesti uuden henkilön tullessa mukaan sovelluksen kehitykseen tai ylläpitoon.

2.3 QoS-sovelluskehyksen idea ja sisällön kuvaus

QoS:n ja sovelluskehysten perusperiaatteiden ja hyötyjen esittelyn jälkeen onkin syytä pohtia, mitä sitten saavutettaisiin näiden yhdistämisellä. QoS-kenttä on ehkä vielä vailla selkeää konsensusta siitä, miten asiat olisi hyvä

tehdä, jotta sovellukset saataisiin toimimaan tekniikoiltaan erilaisissa QoS-ympäristöissä. Sovelluskehysten avulla taas tällaisia tilanteita pyritään ratkaisemaan, joten tuntuu loogiselta yrittää yhdistää tätä menettelyä QoS-tekniikoiden toteuttamiseen.

QoS-alueelta löytyy paljon yhteisiä asioita tekniikoista riippumatta. Yhteyksien rakentaminen, verkkojen varaustilanteen muuttuminen ja siitä johtuva yhteyksien uudelleenneuvottelu, sekä itse datan käsittely ovat sovellusten kehittäjän näkökulmasta sama asia riippumatta siitä, mitä QoS-tekniikkaa ollaan käyttämässä, lähinnä vain itse QoS-ympäristön käyttäminen poikkeaa eri sovellusten tapauksessa.

Mitä sitten QoS-sovelluskehyksellä voitaisiin saavuttaa ja mikä olisi sovelluskehyksen sisältö? Kuten sovelluskehyksillä yleensä, QoS-sovelluskehyksellä voitaisiin saavuttaa etua sovelluskehityksessä tuotantosyklien nopeutuessa ja tehostuessa valmiiden rakennuspalikoiden myötä. Lähinnä sovelluskehyksen voisi ajatella konkretisoituvan valmiin koodikirjaston muodossa, jossa valmiita luokkia ja binäärisiä rakennuspalikoita voitaisiin käyttää hyväksi sovellusten rakentamisessa. Lisäksi ns. know-how –alue olisi oleellisessa asemassa, jotta hyviksi todetut menettelytavat saataisiin dokumentoitua siten, että niitä voidaan hyödyntää muissa projekteissa.

QoS-sovelluskehys voitaisiin käsittää myös laajempana kokonaisuutena kuin pelkkänä joukkona rakennuspalikoita ja –ohjeita, joiden avulla voidaan rakentaa yksittäisiä sovelluksia. Sovelluskehystä voitaisiin laajentaa myös ns. välitason ohjelmiston (middleware) puolelle, eli kun QoS-tekniikoita käyttävät sovellukset asennetaan asiakkaan verkkoympäristöön, niin välitason ohjelmiston avulla voitaisiin hoitaa sovellusten konfigurointi siten, että ne osaisivat käyttää verkossa tarjolla olevaa, yhtä tai useampaa, QoS-tekniikkaa. Tällöin QoS-sovelluskehys määrittelisi vain rajapinnan, jonka avulla QoS-tekniikkaan liittyvät resurssien varaukset, uudelleenneuvottelut, sekä datan lähetys ja vastaanotto hoidettaisiin ja välitason ohjelmisto tarjoaisi sitten valmiit modulit, jotka voitaisiin ladata sovelluksen käynnistyessä käytettävän QoS-tekniikan mukaan. Kuva 1 esittää tapaa, jolla sovellukset toteutetaan QoS-sovelluskehyksen mukaan valmiiksi määriteltyjä QoS-rajapintoja käyttäen.

Kuva 1: QoS-middleware –ympäristö.

Välitason ohjelmisto hoitaa kaiken QoS-tekniikoihin liittyvän toiminnallisuuden, sekä selvittää sovelluksen käynnistämisvaiheessa konfiguraatiosääntökannasta mitä QoS-tekniikkaa käytetään, ja mitkä ovat esimerkiksi sovellukselle määritellyt valmiit yhteysparametrit. Välitason ohjelmiston avulla voidaan siis määritellä koko putki asiakas- ja palvelinosapuolien välillä, toisaalta taas palvelinpuolen osalta välitason ohjelmistoa ei ole välttämätöntä käyttää, esimerkiksi silloin jos palvelinosa on toteutettu eri osapuolen toimesta kuin asiakassovellus, eikä palvelinpuolella ole tällöin samaa välitason ohjelmisto / sovelluskehys-konseptia käytössään.

2.4 Muut tutkimukset.

Columbian yliopistossa on tehty tutkimus [Wang et al. 2000] geneerisistä QoS-rajapinnoista QoS-palveluja tarvitsevien sovellusten käyttöön. Tutkimuksessa kehitellään geneeristä rajapintaa ja välitason ohjelmistoa, jolla QoS-resurssien hallintaa ja monitorointia tehdään. Tutkimuksen yhteydessä on tehty prototyyppi Corba-teknologiaa käyttäen.

Frankfurtin yliopistossa [Becker ja Geihs, 2000] on kehitetty QoS-resurssien hallintaan soveltuvaa arkkitehtuuria Corba-teknologiaa käyttäen.

Karr et al. [2001] tutkivat QuO (Quality Objects) –konseptin sopivuutta miehittämättömän lentokoneen kuvaaman videon reaaliaikaiseen välitykseen. Tutkimus keskittyy lähinnä resurssien saatavuuden vaihtelun aiheuttamien ongelmien reaaliaikaiseen ratkaisemiseen. Tuloksena on reaaliaika-CORBA -määritystä käyttävä järjestelmä, joka sopeutuu tietoliikenneympäristön resurssitilanteen vaihteluun. Tähän tutkimukseen liittyy toinen, osin samojen kirjoittajien kirjoittama artikkeli [Krishnamurthy et al., 2001] jossa kuvataan yleisemmin QuO-konseptia.

Matsui et al. [1999] esittelevät QoSPath-konseptiaan, jonka avulla sovellukset voivat esittää QoS-vaatimuksensa geneerisesti ja joka tarjoaa myös modulaarisuuden kautta laajennettavan QoS-menettelyvalikoiman.

3. QoS-toteutustekniikat

Tässä osiossa käydään läpi erilaisia yleisesti käytössä olevia QoS-toteutustekniikoita. QoS-tekniikoiden määrittelyjen laajuuden vuoksi tässä osiossa käsitellään vain oleellisimmat osat eri QoS-tekniikoiden ominaisuuksista ja hallinnasta.

QoS-tekniikoista voidaan yleisesti mainita se, että palvelujen luokittelu eri tasoisiksi tapahtuu OSI-mallin [Black, 1999] alimmilla kerroksilla. Itse luokittelun vaatima älykkyys sijaitsee reitittimissä jotka sijaitsevat OSI-mallin verkkokerroksella.

3.1 Ethernet (IPv4 ja IPv6)

Tämä aliluku Ethernet-verkoista perustuu Kaarion [2002] kirjaan TCP/IP-verkoista. Ethernet-verkoissa käytettävissä IP-protokollissa on IPv4-protokollasta lähtien ollut palvelun laatuun käytettävä kenttä, Type Of Service (TOS). Kuva 2 esittää IPv4-otsikon rakennetta, TOS-kenttä on 8-bittinen osa IP-paketin otsikkoa:

<table>
<tr><td>Versio (4)</td><td>IHL (4)</td><td>TOS (8)</td><td colspan="2">Datagrammin pituus (16)</td></tr>
<tr><td colspan="3">Datagrammin osan tunniste (16)</td><td>Liput (3)</td><td>Datagrammin osan siirtymä (13)</td></tr>
<tr><td colspan="2">Eloaika (8)</td><td>Protokolla (8)</td><td colspan="2">Otsikon tarkistussumma (16)</td></tr>
<tr><td colspan="5">Lähdeosoite (32)</td></tr>
<tr><td colspan="5">Kohdeosoite (32)</td></tr>
<tr><td colspan="5">Optiot ja tarvittaessa täytebittejä</td></tr>
</table>

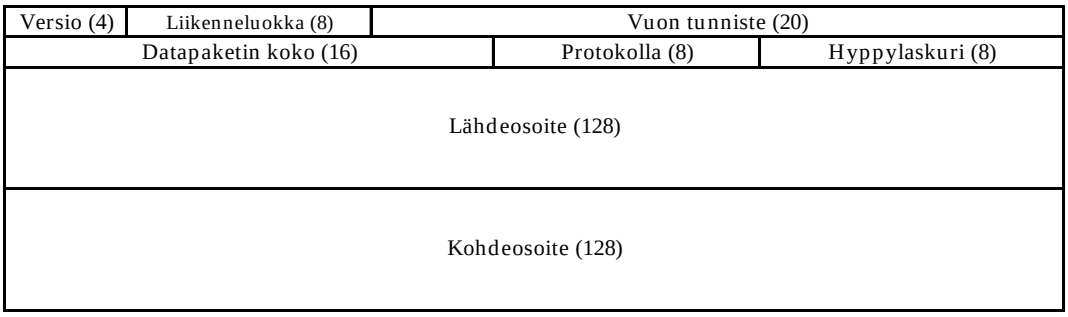
Kuva 2: IPv4-otsikon rakenne.

TOS-kentästä käytetään varsinaisesti vain neljää bittiä (bitit 4-7) palvelun laadun kuvaamiseen. Nämä bitit tarkoittavat järjestyksessä seuraavien yhteyden ominaisuuksien minimointia tai maksimointia: viive, läpäisy, hinta ja luotettavuus. IP-standardi määrää, että vain yksi biteistä saa olla kerrallaan asetettuna, esimerkiksi viiveen minimointi samanaikaisesti luotettavuuden maksimoinnin kanssa on standardin vastaista. Verkon laitteista riippuen tällainen standardin vastainen useampien TOS-bittien käyttö voi johtaa

odottamattomiin seurauksiin. Tuloksena voi olla jomman kumman ominaisuuden toteutuminen reititystasolla, tai laitteen sekoaminen.

Normaalissa IP-verkossa TOS-kentän bittejä ei käytetä, mutta edistyneemmät tekniikat, kuten myöhemmissä luvuissa käsiteltävät DiffServ (ks. aliluku 3.3) ja RSVP (ks. aliluku 3.2), voivat käyttää kenttää hyväkseen.

IPv6 ei varsinaisesti tuo paljoa uutta QoS:n saralla IPv4:ään verrattuna. IPv4:n TOS-bitit on korvattu prioriteettibiteillä (4kpl), joiden arvo määrää eri pakettien prioriteetin. Pelkän prioriteetin perusteella pakettien käsittely on selkeämpää kuin IPv4:n TOS-biteillä. Selkeänä muutoksena on vuon tunnistekenttä, jonka avulla reitittimissä voidaan ohjata paketteja vuon tunnisteen mukaan, eikä paketin kohdeosoitteen perusteella tehtävää seuraavan solmun hakemista tarvitse tehdä. Lähinnä IPv6:ssa paketin otsikkoa on yksinkertaistettu IPv4:sta. Pääasiassa IPv6:lla haetaan ratkaisua alati vähentyviin IPv4-osoitteisiin, IPv6:ssa osoitteet ovat 128-bittisiä verrattuna IPv4:n 32 bittiin. Kuva 3 esittää IPv6-otsikon rakenteen.



Kuva 3: IPv6-otsikon rakenne.

3.2 IntServ ja RSVP

Tämän aliluvun teksti perustuu Blackin [1999] kirjaan. IntServ (Integrated Services) –arkkitehtuurin periaate on taata palvelun laatu yhteyskohtaisesti siten, että tietty palvelun taso säilyy koko yhteyden olemassaolon ajan. Konkreettinen teknologia IntServ-arkkitehtuurissa on RSVP (Resource Reservation Protocol), jolla yhteyden resurssien varaukset tehdään dynaamisesti. Dynaamisuus tarkoittaa sitä, että yhteyden ominaisuuksia voidaan muuttaa tarvittaessa yhteyden aikana ilman koko varausoperaation tekemistä uudestaan. RSVP toimii IPv4- ja IPv6-verkoissa, oletus kuitenkin on,

että TCP (Transmission Control Protocol) -protokollan sijasta käytetään RTP (Real-time Transfer Protocol) –protokollaa, joka tarjoaa paremman tuen QoS:n käytölle.

RSVP:ssä yhteys ja resurssien varaus kahden osapuolen välillä luodaan erillisillä sanomilla. Yhteyden aloittaja lähettää *Path*-sanoman vastaanottajalle. Tämän sanoman kulkiessa verkon läpi, verkon RSVP-kykyiset laitteet liittävät sanomaan dataa, joka kertoo minkälaista liikennöintiä kyseinen laite voi tarjota. Sanoman lopullinen vastaanottaja tutkii Path-sanomasta minkälaista palvelun laatua verkko voi tarjota ja minkälaista dataa yhteyden avulla halutaan siirtää. Tämän jälkeen vastaanottaja lähettää *Resv*-sanoman takaisin *Path*-sanoman lähettäjälle. Resv-sanoman kulkiessa verkossa takaisin *Path*-sanoman lähettäjälle, varaavat reitin varrella olevat RSVP-kykyiset laitteet yhteydelle sanomassa kuvatun määrän resursseja.

Yhteyden aloittajan lähettämässä *Path*-sanomassa kulkevat *AdSpec*- ja *TSpec*-kentät. *AdSpec*-kenttään kerätään tietoa verkon laitteista, kun *Path*-sanoma kulkee lähettäjältä vastaanottajalle. *AdSpec*-kenttään kerätyn datan avulla saadaan selville esimerkiksi reitin vuonleveys (bandwidth) ja se, tukeeko reitti lähettäjältä vastaanottajalle yleensä kokonaisuudessaan RSVP:n käyttöä. *TSpec*-kenttä sisältää tiedon datasta jota lähettäjä haluaa lähettää. Tiedot sisältävät esimerkiksi lähetettävän datan pakettien koon ja maksimimäärän.

Yhteyden resurssien varaus tehdään siis *Resv*-sanomalla, joka kulkee *Path*-sanoman vastaanottajalta takaisin sen lähettäjälle samaa reittiä. *Resv*-sanoma sisältää saman *TSpec*-kentän kuin *Path*-sanomakin, sekä lisäksi *RSpec*-kentän, joka sisältää vaatimukset resurssien varaukselle. *RSpec*-kentän mukaan verkon laitteet varaavat tarvittavat resurssit sitä mukaa, kun *Resv*-sanoma etenee verkossa. Yhteyden varauksen jälkeen on huomattava, että vuo toimii vain lähettäjältä vastaanottajan suuntaan. Jos halutaan kaksisuuntainen yhteys, on sama resurssien varaus tehtävä myös toiseen suuntaan.

RSVP:tä käyttävän sovelluksen on säännöllisesti tehtävä Path/Resv-sanomanvaihtoa, jotta yhteys säilyisi. Ilman tällaista yhteyden päivitystä resurssit tullaan vapauttamaan järjestelmän toimesta. Sanomien avulla saadaan näin samalla tieto mahdollisesta verkon kapasiteetin muuttumisesta.

RSVP tarjoaa siis yhteyskohtaisesti mahdollisuuden varata tietyt resurssit datan välittämiseen. Tämä vahvuus on samalla myös heikkous, sillä lyhytkestoisten ja purskeisten yhteyksien perustamiseen se ei sovi yhteyden perustamiseen kuluvan ajan vuoksi. Lisäksi RSVP sopii parhaiten lähinnä pieniin Intranet-ympäristöihin, joissa varattujen yhteyksien määrä on pieni.

3.3 DiffServ

Tämän aliluvun teksti perustuu Kilkin [1999] kirjaan. DiffServ (Differentiated Services) on palveluiden luokitukseen perustuva menetelmä, jossa verkossa liikkuviin paketteihin liitetään palveluluokkatieto, ja tämän tiedon perusteella paketit käsitellään verkon reitittimissä eri tavalla. DiffServ ei siis perustu yhteydellisyyteen kuten esim. IntServ/RSVP vaan kaikki verkossa liikkuvat paketit käsitellään samojen sääntöjen perusteella. Säännöt määritellään hyppykohtaisesti, eli paketin kulkiessa verkon laitteen (reititin, kytkin) läpi, sen käsittelyyn sovelletaan PHB (Per Hop Behaviour) –sääntöä. PHB on kuvaus, jonka perusteella paketit käsitellään (lähetetään edelleen, luokitellaan, järjestetään tai jätetään lähettämättä). PHB:n perusteella paketit voidaan esimerkiksi ohjata kytkimissä eri prioriteeteilla käsiteltäviin jonoihin, tai näissä jonoissa eri järjestykseen. Tärkeiksi luokitellut paketit ohjataan prioriteetiltaan korkeampaan jonoon, josta paketit lähetetään edelleen verkossa nopeammin kuin prioriteetiltaan alemmista jonoista.

DiffServ käyttää IPv4-otsikon TOS-kenttää (IPv6:ssa liikenneluokka-kenttää) palveluluokituksen kuljettamiseen, kenttä on nimetty uudelleen DS-tavuksi. TOS/liikenneluokka-kentästä käytetään vain kuutta (6) ensimmäistä bittiä.

DiffServ ja siihen liittyvät PHB-säännöt toimivat aina tietyn verkkosegmentin sisällä. Tämä tarkoittaa sitä, että PHB-sääntöjen mukainen pakettien käsittely tapahtuu vain tietyn sisäverkon alueella. Verkon ulkoreunoilla olevat solmut tekevät ”raskaimman” työn, eli ne asettavat pakettikohtaisesti tarvittavat DS-bitit oikeisiin arvoihin, jotta käsittely verkon sisäsolmuissa tapahtuu sääntöjen ja liikennöintisopimuksen mukaisesti. Samaten verkosta poistuvat paketit riisutaan DiffServ:iin liittyvistä tiedoista. Erillisten verkkojen välillä kulkevissa paketeissa ei liiku palveluluokkatietoa.

Tämä siksi, että kytkimet, jotka eivät tue palveluluokitusta, eivät osaisi käsitellä paketteja oikein. Lisäksi, vaikka paketit lähetettäisiin toiseen verkkoon jossa on myös DiffServ käytössä, niin luokitus ei välttämättä ole molemmissa verkoissa sama ja tämä voisi johtaa odottamattomaan käyttäytymiseen verkon laitteissa.

DiffServ ei siis tarjoa ratkaisua päästä päähän ulottuvaan QoS-palveluun, vaan sen avulla voidaan verkon sisäisesti ohjata pakettien käsittelyä siten, että tärkeämmät paketit käsitellään tehokkaammin ja nopeammin. Tällainen menettely on yksinkertaista toteuttaa eikä se vaadi erillistä signalointia osapuolien välillä resurssien varaamiseen. Menettely sopii hyvin mm. lyhytkestoisiin lähetyksiin, joissa täytyy välttää raskaita yhteydenluontiprosesseja.

3.4 ATM

ATM:ssä palvelun laatuun on kiinnitetty huomiota jo arkkitehtuurin [ATM Forum] suunnittelussa. Yhteyteen liittyy aina QoS-vaatimuksia, vaikka sovelluksella ei olisi mitään erityisiä tarpeita palvelun laadun suhteen.

ATM on yhteydellinen arkkitehtuuri, jossa kahden osapuolen välille muodostetaan virtuaalinen kanava (Virtual Circuit, VC). ATM tukee kytkettyjä (Switched) ja pysyviä (Permanent) virtuaalikanavia. Yhteyden muodostuksen jälkeen kaikki liikenne kulkee verkossa muodostettua kanavaa pitkin. Kytkeminen on tällöin nopeaa, koska verkon laitteiden täytyy vain tarkistaa ATM-paketin vuon tunnisteen perusteella mihin paketti on ohjattava.

Yhteyden avauksessa määritellään QoS-parametrit, joiden perusteella resurssit varataan. Jos verkko ei pysty tarjoamaan vaatimuksia vastaavia resursseja, yhteyden muodostus epäonnistuu. Yhteyden muodostuksessa käytetään PNNI (Private Network – Network Interface) –signalointiprotokollaa, jossa kuvataan laatuparametrit. Näitä laatuparametreja ovat: solun siirtoviive (Cell Transfer Delay, CTD), solujen viiveen vaihtelu (Cell Delay Variation, CDV), soluvirhesuhde (Cell Error Ratio, CER), soluhukkasuhde (Cell Loss Ratio, CLR), solujen virhelisäysnopeus (Cell Misinsertion Rate, CMR) ja solulohkovirhesuhde (Severely Errored Cell Block Ratio, SECBR). Lisäksi liikennelähteelle määritellään liikenneparametrit eli huippusolunopeus (Peak Cell Rate, PCR), keskimääräinen solunopeus (Sustained Cell Rate, SCR),

purskeen maksimipituus (Maximum Burst Size, MBS) ja minimisolunopeus (Minimum Cell Rate, MCR).

Virtuaalisia kanavia ei ole aina pakko varata yhteyden luonnissa, vaan verkkoon voidaan konfiguroida pysyviä virtuaalikanavia (Permanent Virtual Circuit, PVC). Pysyvien kanavien tapauksessa ei käytetä yhteyden rakennusvaiheessa signalointia ollenkaan, joten yhteyden avaaminen on nopeampaa, mutta sovellukset eivät pysty määrittelemään omia palvelun laadulle asettamiaan kriteereitä. Yleensä pysyvät virtuaalikanavat tarjoavatkin vain best-effort –palvelun, mutta tämäkin on huomattavasti parempi kuin esimerkiksi normaalissa ethernetissä, koska yhtä virtuaalikanavaa pitkin kuljetetaan vain tietynlaista liikennettä. Virtuaalikanavien kapasiteetti mitoitetaan käyttötarpeen mukaan.

ATM-protokollan määrittelyssä on neljä eri palveluluokkaa: *Constant Bit Rate (CBR), Real-Time & Non-Real-Time Variable Bit Rate (rt/nrt-VBR)*, *Available Bit Rate (ABR) ja Unspecified Bit Rate (UBR)*. CBR-palveluluokkaa käytetään tapauksissa, joissa yhteys vaatii jatkuvasti tietyn huippusolunopeuden ja yhteyden käyttäjä voi missä tahansa vaiheessa yhteyden aikana käyttää joko koko kapasiteetin, tai vain osan siitä. Tyypillisiä käyttökohteita CBR:lle ovat esimerkiksi videoneuvottelut, joissa vaaditaan jatkuvasti tiettyä kapasiteettia kuvan ja äänen laadun takaamiseksi. Laatuparametreista CBR:ään kuuluvat viiveeseen liittyvät solun siirtoviivettä ja viiveen vaihtelua ilmaisevat arvot.

VBR-palveluluokkia käytetään purskeisen liikenteen välitykseen, rt-VBR asettaa tiukemmat vaatimukset pakettien viiveelle ja viiveen vaihtelulle kuin nrt-VBR, mutta molemmissa tapauksissa muut yhteydelle määriteltävissä olevat liikennöintiparametrit ovat samat, eli huippusolunopeus, keskimääräinen solunopeus ja purskeen maksimipituus. Laatuparametrien suhteen rt-VBR lupaa pienen solujen siirtoviiveen, mutta vastaavasti alemman solujen hukkasuhteen, koska kauemmin verkossa viipyvien pakettien ei taata pääsevän perille. Vastaavasti nrt-VBR taas lupaa suuremman luotettavuuden matalalla solujen hukkasuhdearvolla, mutta viiveen suhteen ei olla niin kriittisiä, solujen siirtoviive on suurempi ja lähinnä yhteydeltä voidaan odottaa vain, että paketit saadaan välitettyä eteenpäin annetun siirtoviiveen puitteissa.

ABR-palveluluokka on tarkoitettu yhteyksiin, joilla ei ole kovin tarkkoja vaatimuksia yhteyden laadun suhteen, vaan ne tulevat toimeen vaihtelevissa olosuhteissa verkon kapasiteetin suhteen. Yhteyden perustamisvaiheessa määritellään maksimi- ja minimisolunopeus, joiden välille verkon tarjoaman yhteyden nopeuden tulee asettua. Arvot siis merkitsevät pakettien maksimi- ja miniminopeutta, joiden avulla dataa lähettävä sovellus vielä pystyy toimimaan normaalisti, miniminopeus voidaan määritellä alkaen arvosta 0. ABR-palveluluokkaan liittyen ATM:ssä käytetään RM (Resource Management) -soluja ilmaisemaan muutoksia verkon tarjoamassa palvelun tasossa. RM-solujen CCR-kenttään (Current Cell Rate) asetetaan ACR-arvo (Actual Cell Rate), joka kuvaa vuohon liittyvää todellista solunopeutta ja ER-kenttään (Explicit Rate) asetetaan vaadittu solunopeus. RM-solujen kulkiessa verkossa, kytkimet säätävät ER-kentän arvoa ja jos verkossa on ruuhkaa, voivat kytkimet asettaa RM-solun CI- (Congestion Indication) tai NI-bitin (No Increase) arvoon 1. RM-solujen palatessa lähettäjälle näiden kenttien perusteella voidaan tehdä tarpeen vaatiessa yhteyden parametrien säätämistä. Maksimitaajuus, jolla RM-soluja voidaan lähettää, on 10 solua sekunnissa. RM-solujen määrää on rajoitettu, jotta ne eivät haittaisi varsinaista dataliikennettä. ABR-palveluluokka ei varsinaisesti sisällä mitään QoS-takeita, mutta oletuksena on että solujen hukkasuhde on pieni. Yhteyden voidaan odottaa olevan luotettava, mutta viiveen suhteen ei voida antaa mitään takuita, eikä ABR näin ollen sovellu ainakaan reaaliaikaisuutta vaativiin tarkoituksiin.

UBR-palveluluokka ei anna mitään takeita palvelun laadun suhteen, siihen ei liity mitään laatu- tai liikennöintiparametreja, eikä yhteyden käyttäjä voi näin ollen tehdä mitään oletuksia vuonleveyden tai viiveen suhteen. UBR katsotaankin kuuluvaksi best effort –palvelujen luokkaan, jotka kuljettavat liikennettä täysin sen hetkisen verkon tilanteen mukaan, pakettien hukkuminen tai viivästyminen on mahdollista ja todennäköistä verkon ollessa ruuhkautunut.

3.5 Verkon ylirakentaminen

Tietoverkkojen ylimitoittaminen on ehkä yksinkertaisin keino parantaa palvelun laatua tietoverkoissa. Verkko mitoitetaan tällöin suurempaa

liikennemäärää varten, kuin mitä tavallisesti normaalitilanteissa tarvittaisiin. Samalla verkkojen rakentamisen hinta kuitenkin kasvaa, eikä ylimitoittamisella voida järkevästi ratkaista WAN (Wide Area Network) –liikenteen ongelmia, koska näiden linkkien rakentaminen on erityisen kallista pitkien etäisyyksien vuoksi.

Pienissä Intranet-verkoissa ylirakentaminen voi olla helpoin tapa ratkaista palvelun laatuun liittyviä ongelmia. Tällaisissa verkoissa liikenteen määrä on ennakoitavissa ja tällöin verkko voidaan mitoittaa siten, että kapasiteetti riittää tilanteessa kuin tilanteessa.

Verkon ylirakentamista ei voida kuitenkaan pitää yleispätevänä ratkaisuna verkon ruuhkautumisen helpottamiseksi. Tietyn palvelun tai palvelimen ollessa ruuhkautunut, ei verkon kapasiteetin kasvattamisesta ole kuin haittaa, koska palvelut ja palvelimet vain ruuhkautuvat entistä enemmän. Tieliikenneanalogiaa käyttäen asia on sama kuin ruuhkia helpotettaisiin kaistoja lisäämällä, kun ongelma on tien päässä olevassa kapeassa risteyksessä. Verkkojen ylirakentamisessa tulee siis ottaa huomioon myös kasvavan kuorman jako, jotta ongelmat eivät vain siirry yhdestä paikasta toiseen. Tietoliikenteen luonteeseen kuuluu myös, että kaikki käytössä oleva kapasiteetti käy jossain vaiheessa riittämättömäksi. Näin käy viimeistään siinä vaiheessa, kun käyttäjien tottumukset ja odotukset verkon palvelun tasosta nousevat kasvatetun kapasiteetin tasolle ja sen ylitse [Bragg, 1999].

3.6 MPLS

Multiprotocol Label Switching (MPLS) ei varsinaisesti ole QoS-teknologia vaan reititysratkaisu, jonka avulla useat eri QoS-teknologiat saadaan reititettyä verkoissa rinnakkain. MPLS mahdollistaa myös tietoliikenteen reitityksen suunnittelun siten, että eri liikenneluokat voidaan reitittää aina samaa reittiä, jolloin lähestytään ATM:n VC/VP –konseptia ja tällöin voidaan antaa takeita verkon tarjoaman kapasiteetin suhteen.

Tietoliikenteen reititys perustuu MPLS:ssä pakettien leimaamiseen (labeling) ja ohjaamiseen näiden leimojen perusteella. MPLS-verkko koostuu ingress- ja egress-solmuista, sekä MPLS-kykyisistä reitittimistä (LSR, label switched router). Ingress-solmuissa MPLS-verkkoon sisään tulevat datapaketit

leimataan kohdeosoitteen mukaan, egress-solmuissa taas paketit poistuvat MPLS-verkosta ja tällöin leimat poistetaan paketeista. Jokainen MPLS-verkon reititin tietää ns. leimapinon, jonka perusteella paketit osataan reitittää tiettyä reittiä kohdeosoitteeseen. Leimapinot rakennetaan MPLS-verkon käynnistysvaiheessa, jolloin reitittimet kommunikoivat keskenään LDP-protokollan avulla (Label Distribution Protocol) ja reitit (leimapinot) eri kohdeosoitteiden välillä rakennetaan. LSR-reitittimissä leimatut paketit käsitellään leimakannan (LIB, label information base) perusteella siten, että leimakannasta haetaan vastaanotetun paketin sisältämä leima ja sitä seuraava leima korvataan datapakettiin. Lopuksi paketti lähetetään leiman mukaiselle reitittimelle. MPLS mahdollistaa liikenteen tunneloinnin ja esimääritellyt reitit datapakettiin liitettävän leimapinon avulla, tällöin LSR-reitittimet eivät hae leimoja leimakannasta, vaan paketin päällimmäinen leima poistetaan ja paketti lähetetään edelleen seuraavana pinossa olevan leiman mukaan. Tunneloinnin avulla voidaan ohjata tietyn tyyppinen liikenne omaa reittiään ja näin tarjota parempaa suorituskykyä, eli antaa QoS-takeita tietyn liikenteen osalta.

MPLS toimii nopeasti, koska pakettien otsikoista ei tarvitse tulkita kohdeosoitetta ja tälle oikeaa reittiä, vaan pelkän leiman perusteella saadaan selville seuraava reitityskohde. Kovin laajaan verkkoon MPLS ei sovellu, koska leimapinojen ylläpitäminen kävisi mahdottomaksi. Sen sijaan LAN-ympäristössä ja yritysten intranet-verkoissa MPLS on hyvä vaihtoehto.

4. QoS-sovelluskehyksen vaatimusten kartoitus

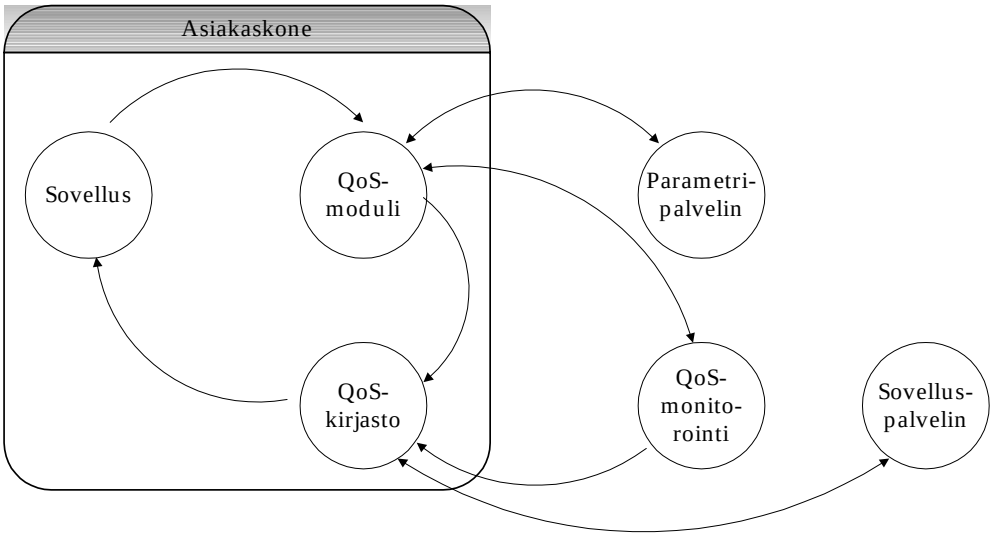
Tässä luvussa kartoitetaan operatiivisia vaatimuksia QoS-sovelluskehykselle, eli toimintoja, joita sovelluskehyksen täytyisi tarjota sovelluksille, jotta QoS-tarpeiden esittäminen ja itse resurssien hallinta olisi mahdollista toteuttaa geneerisesti.

QoS-sovelluskehyksen vaatimukset on kerätty käyttötapausten (use case – menetelmän) avulla, alkuperäiset käyttötapaukset on listattu tutkielman lopussa. Käyttötapaukset sisältävät yleisen kuvauksen, esiehdot, osallistujat, toiminnan kuvauksen askeleittain, jälkiehdot sekä mahdolliset poikkeukset.

4.1 Sovelluskehyksen arkkitehtuuri

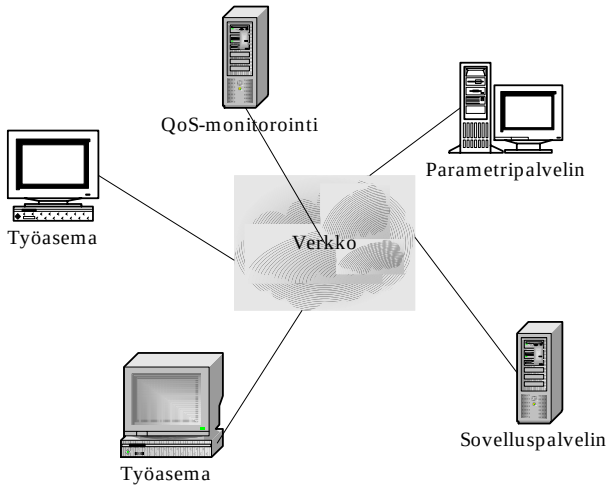
Seuraavissa luvuissa käsitellään operaatioita itse sovelluksen ja sovelluskehyksen eri osien välillä. Tätä varten tehdään joitain oletuksia sovelluskehyksen arkkitehtuurista, seuraavat oletukset pohjautuvat tutkielman alussa (aliluku 2.3) hahmoteltuun välitason ohjelmisto –malliin, jossa sovelluslogiikan ja QoS-tekniikoiden toteutuksen välille sijoittuu ohjelmisto, joka hoitaa QoS-tekniikoihin liittyvät operaatiot.

QoS-sovelluskehyksessä on tunnistettavissa palvelua käyttävän sovelluksen puolelle sijoittuvassa välitason ohjelmistossa seuraavat osat: geneeriset toiminnot (esimerkiksi parametrien hakemisen) hoitava olio, nämä toiminnot eivät liity suoranaisesti itse QoS:n hallintaan, sekä olio, joka sisältää QoS-menetelmäkohtaiset menettelyt yhteyksien hallintaan. QoS-sovelluskehyksen geneerisiä toimintoja hoitavaa oliota kutsuttakoon *QoS-moduliksi*, ja QoS-menetelmäkohtaista oliota *QoS-kirjastoksi*. QoS-moduli hoitaa siis kommunikoinnin keskitetyn parametripalvelimen kanssa ja tekee päätöksen myös tarpeenmukaisen QoS-kirjaston lataamisesta parametrien hakemisen jälkeen. QoS-kirjasto sisältää QoS-menetelmäkohtaiset menettelyt yhteyksien resurssien hallintaan, eri QoS-menetelmät toteutetaan eri QoS-kirjastomodulin avulla. Palvelua käyttävän sovelluksen ulkopuolella sovelluskehys sisältää QoS-parametripalvelimen, jonne sovelluskohtaiset QoS-parametrit on talletettu, sekä QoS-monitorointisovelluksen, jonka avulla aktiivisia yhteyksiä ja verkon resursseja monitoroidaan. Näillä neljällä osalla pystytään kuvaamaan järjestelmän eri osien välinen kommunikaatio, todellinen arkkitehtuuri voi toteutustavasta riippuen olla jaettuna useampiinkin osiin. Kun tekstissä mainitaan QoS-sovelluskehys, tarkoitetaan sillä kaikkia järjestelmän osia yleisesti. Kuva 4 esittää QoS-sovelluskehyksen loogisia osia ja näiden välistä kommunikointia.



Kuva 4: QoS-sovelluskehyksen loogiset osat.

Asiakaskoneella sijaitsevat itse sovellus, QoS-moduli, sekä QoS-kirjasto. QoS-parametripalvelin, mahdollinen QoS-monitorointisovellus ja itse sovelluspalvelin johon asiakassovellus on yhteydessä, sijaitsevat jossain toisaalla verkossa. Kuva 5 tarkentaa järjestelmän eri osien sijaintia verkossa.



Kuva 5: Verkkoarkkitehtuuri.

Järjestelmän osista siis QoS-moduli ja QoS-kirjasto sijaitsevat fyysisesti samalla koneella (työasemat), ja QoS-monitorointi, parametripalvelin ja sovelluspalvelin sijaitsevat jossain toisaalla verkossa.

Järjestelmän eri osien välillä tapahtuva kommunikaatio pitää sisällään seuraavat operaatiot: yhteysparametrien haku QoS-parametripalvelimelta yhteyden avausta varten, yhteyden avaus asiakaskoneelta sovelluspalvelimelle, yhteyden sulkeminen, synkroninen datan lähetys asiakaskoneelta palvelimelle

synkronisesti (vastaus halutaan heti lähetyksen jälkeen) ja asynkroninen datan lähetys (vastaus tulee joskus myöhemmin), palvelimen lähettämän datan vastaanotto, sovelluksen käyttämän yhteyden parametrien muuttaminen keskitetysti QoS-parametripalvelimella, verkon palvelutason muuttuminen ja yhteyden odottamaton katkeaminen. Nämä operaatiot ja niihin liittyvät rajapinnat käsitellään yksityiskohtaisemmin seuraavissa luvuissa.

4.2 Yhteysparametrien haku

QoS-tekniikoissa yhteyden avaamiseen tarvitaan joukko erinäisiä tietoja, jotka kuvaavat avattavalle yhteydelle asetettavia ominaisuuksia yhteyden avaukseen ja tietyn palvelun tason ylläpitoon liittyen. Normaalisti tällaiset tiedot sijaitsevat jossain sovelluskohtaisissa konfiguraatiotiedoissa, esimerkiksi koneen kovalevyllä sijaitsevassa tiedostossa, tietokannassa, tai vaikkapa suoraan koodattuna sovelluksen toimintalogiikkaan. Yksittäisille sovelluksille tällainen menettely riittää, mutta jos halutaan hallita verkon resursseja kuluttavien sovellusten yhteyksien varaamista keskitetysti laajemmassa mittakaavassa, on tarpeen toteuttaa jonkinlainen keskitetty mekanismi, jolla tarpeelliset parametrit voidaan jakaa niitä tarvitseville sovelluksille.

4.2.1 Parametrien varastointi

Jotta keskitetty parametrien jakelu voitaisiin toteuttaa, on ensimmäisenä vaatimuksena tietysti tehtävään pyhitetty palvelinkone johon kaikilla QoS-parametreja tarvitsevilla, kyseistä QoS-sovelluskehystä käyttävillä sovelluksilla on pääsy. Toisin sanoen asiakaskone ja parametripalvelin sijaitsevat samassa verkossa ja ovat yhteydessä toisiinsa. Palvelin varastoi eri sovellusten QoS-parametrit tietovarastoon ja tarjoaa palvelut tietojen hakemiseen. Parametrien hakeminen edellyttää luonnollisesti myös sen, että arvot on syötetty palvelimelle esimerkiksi sovelluksen asennushetkellä. Palvelin ja sen saavutettavuus käsitetään tässä esiehtoina, joiden täytyy toteutua yhteysparametrien hakemisen mahdollistamiseksi.

4.2.2 Hakurajapinnan määrittely

Ensimmäisenä varsinaisena vaatimuksena parametrien jakelulle on määriteltävä rajapinta, jonka avulla sovelluskohtaiset parametrit voidaan

kysellä palvelimelta. Rajapinnan määrittelyn tulee sisältää tarpeelliset hakuparametrit, joiden avulla QoS-parametrit identifioidaan palvelimella, sekä paluuarvot, joissa itse QoS-parametrit siirretään palvelimelta takaisin asiakaskoneelle. Tätä rajapintaa kutsuttakoon operaationimellä *HaeParametrit*. Asiakassovellus on velvollinen täyttämään hakuehtoina vaadittavat parametrit, joiden avulla tietyn sovelluksen käyttämät QoS-parametrit haetaan. Sovelluksen tunnistamisessa täytyy varautua tilanteisiin, joissa tunnisteet osuvat päällekkäin jo palvelimelle tallennettujen tietojen tunnisteiden kanssa. Käytännössä sovellukset kannattaa tunnistaa esim. sovelluksen nimen sijasta jollakin globaalisti yksilöllisellä tunnisteella, joka generoidaan sovelluskohtaisesti. Yksi tällainen menetelmä on mm. Windows-maailmassa käytetyt GUID:it (Globally Unique Identifier), jotka ovat 128-bittisiä tunnisteita ja ne generoidaan mm. ajankohdan ja koneen verkkokortin tunnisteen avulla. Sovelluksen yksilöllisen tunnisteen lisäksi QoS-parametrien hakemisessa tarvitaan kohdeosoitetta johon liikennöinti halutaan kohdistaa, koska eri kohdeosoitteisiin voidaan tarjota tai tarvita erilaisia QoS-menettelyjä.

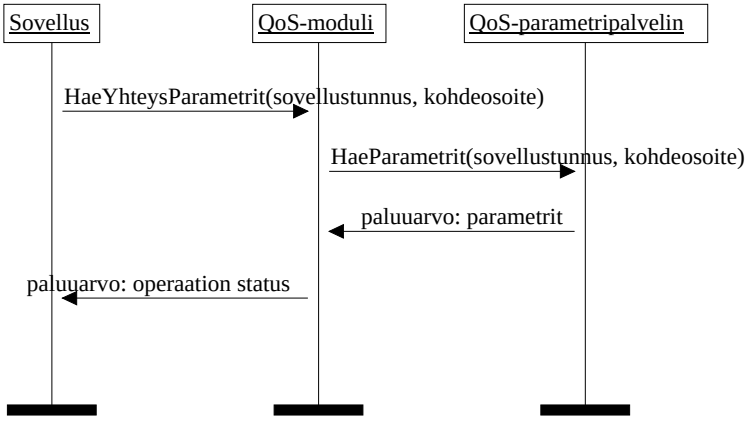
4.2.3 QoS-parametrien haku ja käsittely

Sovellus täyttää hakuparametrit *HaeParametrit*-operaatioon, jonka QoS-moduli toteuttaa välttämättömänä osana QoS-sovelluskehyksen alustustoimia. Sovelluksen kutsuessa itse metodia, QoS-moduli avaa yhteyden parametripalvelimelle ja välittää kutsun edelleen palvelimelle.

Parametripalvelimen käsitellessä *HaeParametrit*-operaatiota, se käyttää syöteparametreina saamaansa sovelluksen tunnistetta tietojen hakemiseen tietovarastosta. Tietovarastoon on oltava mahdollista tallettaa muuttuva määrä yhteysparametreja sovelluksen tarpeen mukaan, koska sovelluskohtaisesti eri parametrit ovat merkityksellisiä. Tietovarastosta haetut parametrit palautetaan asiakassovellukselle *HaeParametrit*-operaation paluuarvoina. Tietokannan tietueet sisältävät sovelluskohtaisesti seuraavat tiedot: sovellukselle osoitettu QoS-menetelmä, vuonleveyden minimi- ja maksimiarvot QoS-menetelmäkohtaisin arvoin, sekä viivevaatimukset. Lisäksi parametrit voivat sisältää tiedon miten esimerkiksi yhteyden uudelleenneuvottelussa menetellään, jos verkon muuttunut kapasiteetti ei mahdollista

minimikapasiteetin määrittelemää liikennöintikapasiteettia, sekä liikennöinnin kohdeosoitteen, jos saman sovelluksen halutaan käyttävän eri QoS-menetelmiä eri kohdeosoitteisiin.

Asiakassovelluksen saatua yhteysparametrit se tallettaa ne paikallisesti käytön nopeuttamista varten. Tämän jälkeen parametrit ovat QoS-modulin käytettävissä kun varsinaista yhteyttä avataan. Kuva 6 havainnollistaa operaatioon liittyvää liikennettä järjestelmän eri osien välillä.



Kuva 6: Yhteysparametrien hakeminen.

4.2.4 QoS-parametrien haun epäonnistuminen

On mahdollista, että parametrien haku epäonnistuu esimerkiksi verkko-ongelmien takia tai sovelluspalvelimelle ei ole syötetty sovelluksen vaatimia parametreja. Ensimmäinen toipumistoimenpide on hakea asiakaskoneelta mahdolliset paikallisesti varastoidut yhteysparametrit ja avata yhteydet näitä käyttäen. Toisena mahdollisuutena (jos parametreja ei löydy asiakaskoneelta) on avata yhteydet best effort –menetelmää käyttäen, eli avataan yhteydet ilman erillisiä QoS-vaatimuksia ja yritetään tulla toimeen niillä resursseilla mitä verkolla on kyseisellä hetkellä tarjota. Verkon resurssien kannalta on kuitenkin riskialtista avata yhteydet best effort –menetelmää käyttäen, koska useiden sovellusten yhtäaikainen käynnistyminen saattaa aiheuttaa kovan kuormituksen sen vuoksi, että best effort –menetelmällä yritetään ottaa sen hetkinen maksimaalinen verkon kapasiteetti käyttöön. Tämän vuoksi mielestäni paras tapa toimia parametrien haun epäonnistumisen jälkeen on estää sovellusten pääsy verkkoon kokonaan.

4.3 Yhteyden avaus

Erilaiset QoS-tekniikat toteuttavat yhteyksien avauksen eri tavoilla. Jotkut menetelmät vaativat joukon erilaisia parametreja yhteyden eri ominaisuuksien määrittelyyn (esim. RSVP, ATM), toiset taas toimivat samalla tavoin kuin esimerkiksi normaalit socket-yhteyksien avaukset (esim. DiffServ). QoS-sovelluskehyksen avulla sovelluksen ei tarvitse tietää tarkemmin mitä parametreja yhteyden avaus vaatii, tai sen puoleen edes mitä QoS-menetelmää käytetään, koska sovelluskehys peittää nämä tiedot alleen. Tämä on mahdollista, koska tarpeelliset parametrit saadaan keskitetysti parametripalvelimelta eikä sovelluksen tarvitse käsitellä näitä tietoja suoraan.

4.3.1 Yhteyden avauksessa käytettävä rajapinta

QoS-modulin täytyy tarjota rajapinta, jonka avulla sovellukset voivat pyytää yhteyden alustusta parametripalvelimelta ladattujen QoS-parametrien mukaan. Tarvittavat parametrit ovat jo sovelluskehyksen tiedossa, joten sovelluksen ei tarvitse niitä käsitellä. Mahdollisten verkossa tapahtuvien muutosten (esim. yhteyksien uudelleenneuvottelu resurssitilanteen muuttumisen jälkeen) ja QoS-sovelluskehyksen havaitsemien virheiden varalta on oltava mekanismi, jolla sovellusta voidaan informoida esimerkiksi katkenneesta yhteydestä. Tähän tarkoitukseen voidaan käyttää ns. takaisinkutsumekanismia, jossa sovellus alustaa QoS-sovelluskehyksen tunteman rajapinnan toteuttavan objektin (esim. C++-luokka), jonka avulla QoS-kirjasto voi raportoida tarpeen mukaan sovellukselle yhteystilanteiden muutoksista, tai datasta joka on vastaanotettu kohdepalvelimelta. Sovellus voi sitten tarpeen mukaan käyttää QoS-kirjastolta saamaansa informaatiota hyväkseen ja esimerkiksi informoida käyttäjää verkossa tapahtuneista muutoksista. Jotta QoS-kirjasto voisi informoida sovellusohjelmaa muutoksista, on sovelluksen välitettävä yhteyden avausmetodissa viite (esim. osoitin) takaisinkutsuobjektiin. Tämä viite välitetään yhteyden avausmetodin parametrina, kutsuttakoon yhteyden avausmetodia nimellä *AvaaYhteys*.

Toinen oleellinen tieto yhteyden avauksessa on kohdeosoite, johon yhteys halutaan rakentaa. Kohdeosoitteen ollessa myös hakuehtona QoS-parametreille, voidaan samasta sovelluksesta avata yhteyksiä eri kohdeosoitteisiin erilaisia

QoS-menetelmiä käyttäen. Tämä ei ole kuitenkaan kannattavaa, koska usean QoS-menetelmän samanaikainen tukeminen tietoverkossa tulee kalliiksi esimerkiksi laite- ja sovellushankintojen lisääntymisen ja verkon hallintakulujen kasvamisen myötä.

4.3.2 Yhteyden avaaminen QoS-sovelluskehyksessä

QoS-moduli tarkistaa ensimmäiseksi, että tarvittavat yhteysparametrit on haettu. Jos näin ei ole tehty, on mahdollista palauttaa sovellukselle virhestatus, tai vaihtoehtoisesti ladata QoS-parametrit parametripalvelimelta ja jatkaa yhteyden avausta. QoS-parametrien ollessa selvillä QoS-moduli lataa parametrien mukaisen QoS-kirjaston, joka toteuttaa toimintalogiikan tietyn QoS-menettelyn osalta. Latauksen yhteydessä QoS-yhteysparametrit välitetään QoS-kirjastolle, kuin myös sovellukselta saatu viite takaisinkutsuobjektiin, joka tallennetaan QoS-kirjaston muistiin myöhempää käyttöä varten. QoS-kirjasto tulkitsee yhteysparametrit käytetyn tekniikan mukaan ja avaa yhteyden kohdepalvelimelle. Yhteyden avauksen jälkeen QoS-kirjasto palauttaa operaation statuksen sovelluskehykselle ja edelleen sovellukselle. QoS-kirjasto siis sisältää kaiken QoS-menetelmäkohtaisen toiminnan ja piilottaa sen sovelluksen toteutukselta. Yhteys on onnistuneen avauksen jälkeen sovelluksen käytettävissä datan lähetykseen ja vastaanottoon.

4.3.3 Poikkeustilanteet yhteyden avaamisessa

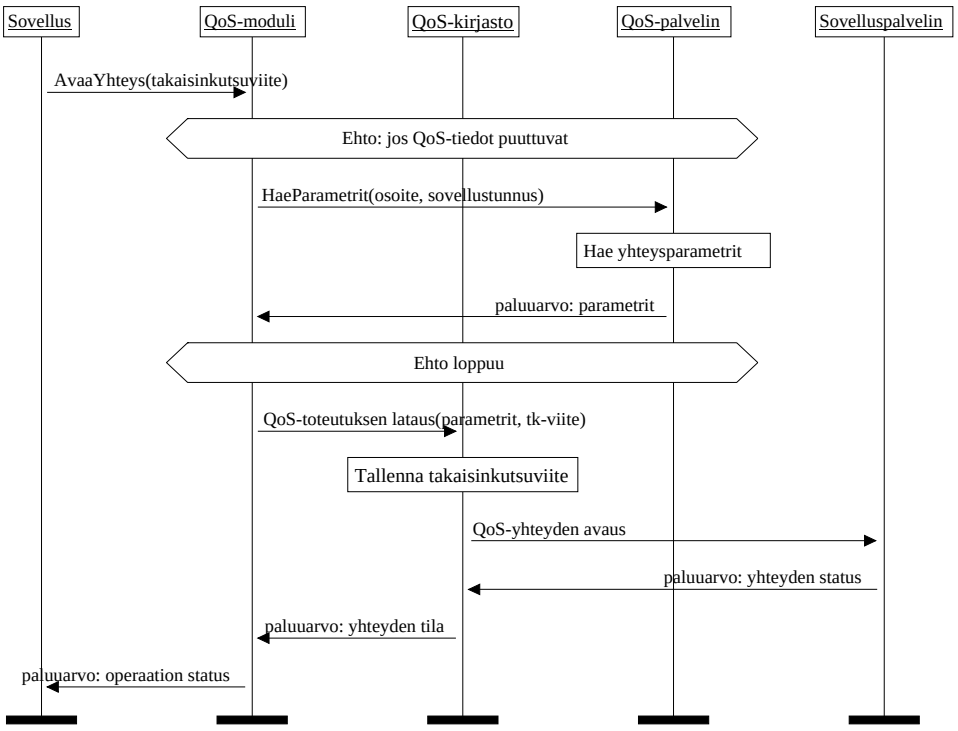
Yhteyden avaamisen epäonnistuessa sovellus ei pysty lähettämään tai vastaanottamaan dataa. Sovellus havaitsee yhteyden avauksen epäonnistumisen operaation tuloksena palautuvasta statuksesta.

Parametrien puuttuessa ja parametrihaun epäonnistuessa on mahdollista turvautua best effort –liikennöintiin, mutta koska tämä saattaa laajamittaisessa käytössä aiheuttaa huomattavaa verkon kuormitusta, ei sitä voi pitää parhaana vaihtoehtona. Näissä tapauksissa sovelluksen käynnistyminen on selkeintä estää, kunnes parametrihakuun liittyvät ongelmat on korjattu.

Itse yhteyden avaamisen epäonnistuminen voi johtua monista syistä riippuen käytetystä QoS-menetelmästä. Jos kyseessä on verkkoresurssien riittämättömyys, niin toipumista voidaan tehdä laskemalla tilapäisesti

verkkoresursseille asetettuja vaatimuksia. Tällainen toipumismenettely pitäisi voida jotenkin kuvata geneerisissä QoS-parametreissa, koska sovellus tai sovelluksen käyttäjä ei välttämättä edes tiedä, että jotain QoS-menettelyä käytetään.

Kuva 7 esittää operaatiot, joita sovelluksen ja eri sovelluskehyksen osien välillä tehdään, lisäksi myös eri osien tallettamat tiedot käyvät ilmi kaaviosta.



Kuva 7: Yhteyden avaus.

4.4 Yhteyden sulkeminen

Yhteyden sulkeminen on tavallisesti triviaali toimenpide, toisaalta, riippuen QoS-menetelmästä, toimintoon saattaa liittyä myös monimutkaisempia toimintoja. Resursseja vapautettaessa menetelmissä, jotka varaavat verkon laitteilta tietyn kaistan, on oltava huolellinen että resurssit vapautetaan oikein. Tämä on tärkeää siksi, että vapauttamattomat resurssit saattavat haitata verkossa toimivia muita sovelluksia ainakin jonkun aikaa, kunnes mekanismin hallinta siivoaa käyttämättömät resurssit pois.

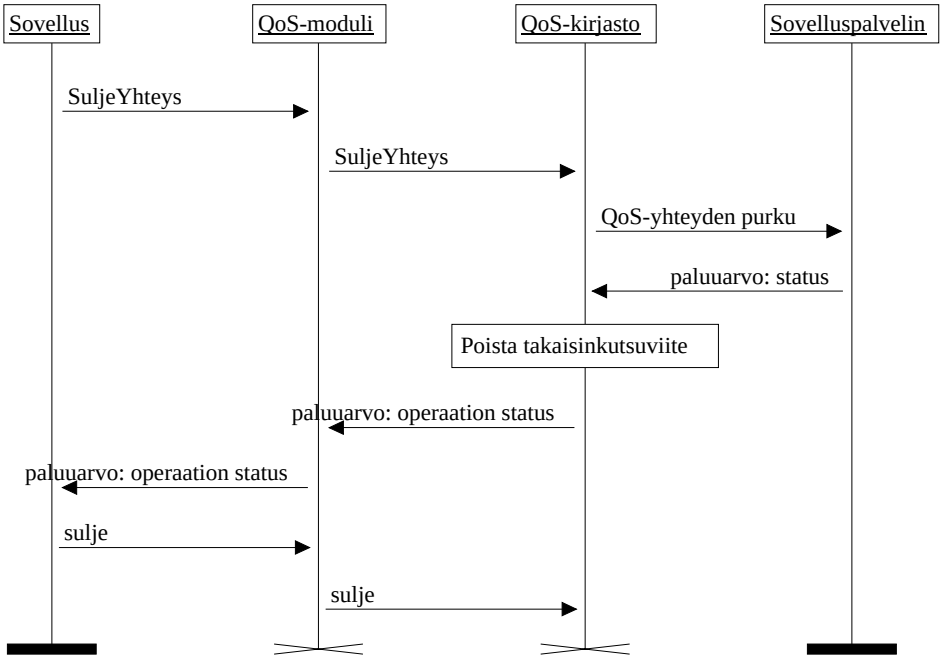
4.4.1 Rajapinta yhteyden sulkemiseen

Yhteyden sulkemiseen käytettävä rajapinta on yksinkertainen, koska sulkemisessa ei tarvita mitään erityisiä parametreja. Lähinnä riittääkin, että sulkeminen osataan kohdistaa oikeaan yhteyteen, jos sovelluksella on niitä auki useampia. Oletuksena on, että sovelluskehyksen tiedoissa yksittäiset yhteydet tunnistetaan siten, että sovelluksen ei tarvitse pitää tallessa mitään sovelluskehyksen generoimaa tunnistetietoa, vaan pelkkä yhteyden sulkemisoperaation kutsuminen riittää. Näin ollen parametreja ei tarvita ja operaation paluuarvo kertoo miten yhteyden sulkeminen onnistui. Yhteyden sulkemiseen käytettävää rajapintaa kutsuttakoon nimellä *SuljeYhteys*.

4.4.2 Resurssien vapautus sovelluskehyksessä

QoS-modulin välitettyä yhteyden sulkupyynnön QoS-kirjastolle, kirjasto sulkee ensin yhteyden QoS-menettelyn vaatimalla tavalla. Tämän jälkeen sovelluksen QoS-sovelluskehyksen käyttöön tarjoama viite takaisinkutsuobjektiin vapautetaan toteutustekniikan vaatimalla tavalla. QoS-kirjasto kutsuu ennen viitteen vapautusta takaisinkutsuobjektin toteuttamaa operaatiota, jolla yhteyden ilmoitetaan sulkeutuvan. Tällöin takaisinkutsuobjekti voi tehdä tarvittavat toimenpiteet omien resurssiensa vapauttamiseksi. Yhteyden sulkemisen ja takaisinkutsuobjektin viitteen vapauttamisen jälkeen QoS-kirjasto palauttaa operaation statuksen QoS-modulille, joka välittää tämän tiedon sovellukselle QoS-kirjaston vapauttamisen jälkeen. Yhteyden sulkemisen jälkeen kirjasto ei siis ole enää latautuneena muistiin, eikä yhteyttä voida uudelleen avata ilman QoS-parametrien hakemista parametripalvelimelta uudelleen.

Kuva 8 havainnollistaa toiminnot sovelluksen ja eri QoS-sovelluskehyksen osien välillä yhteyden purkamisvaiheessa. Yhteyden sulkemisen jälkeen QoS-kirjaston elinkaari loppuu, eikä se enää ole tämän jälkeen käytettävissä ilman yhteyden rakentamista uudelleen alusta lähtien.



Kuva 8: Yhteyden sulkeminen.

4.5 Datan lähetys synkronisesti

Datan synkroninen lähetys tarkoittaa operaatiota, jossa lähetettyyn dataan saapuvaa vastausta jäädään odottamaan kohdepalvelimelta, ennen kuin lähetysoperaatio palaa takaisin sovelluksen puolelle. Tällöin siis operaatiota suorittavan datan lähettäjäprosessin säikeen täytyy odottaa suorituksen valmistumista. Vastausdatan saavuttua kohdepalvelimelta se liitetään synkronisen datan lähetysmetodin paluuarvoksi, jolloin sovellus saa datan käyttöönsä. Tässä tapauksessa sovelluksen toteuttamaa takaisinkutsurajapintaa ei siis käytetä. Synkroninen datanlähetystoiminto on tarpeen esimerkiksi käyttöliittymien toteutuksessa, kun käyttäjän tekemään toimintoon halutaan vastaus ennen kontrollin antamista takaisin käyttäjälle, käytännössä siis toiminnon oletetaan suoriutuvan tietyssä ajassa.

4.5.1 Rajapinta synkroniseen datan lähetykseen

Synkronisessa datan lähetyksessä rajapinnan tulee sisältää parametrit sekä lähetettävälle datalle että paluuarvoille. Paluuarvot ovat tässä tapauksessa vastausdata ja operaation status, jolla ilmaistaan operaation onnistuminen. Lähetettävä data, kuten myös vastausdata, on ennen lähetystä muunnettava

puskuriksi, jonka välitys onnistuu geneerisesti. Tällä saavutetaan rajapinnan soveltuvuus useiden sovellusten tarpeisiin. Myös vastausdata siirretään takaisin sovellukselle geneerisessä formaatissa, jonka sovellus osaa edelleen muuntaa tarpeen vaatiessa sopivampaan muotoon. Datan tulee olla lähetysvaiheessa muodossa, joka siirtyy muuntumatta verkon ylitse, käytännössä tämä tarkoittaa tavuista muodostuvaa puskuria. Synkronisessa kutsutavassa on mahdollista, että vastausta joudutaan odottamaan kauan ja kohdepalvelimella tapahtuvissa virhetilanteissa saatetaan päätyä tilanteeseen, jossa vastausta ei saada koskaan. Tällaisten tilanteiden varalta rajapinnassa täytyy olla mahdollisuus määritellä maksimikesto, jonka verran vastauksen saapuminen kohdepalvelimelta saa enimmillään kestää. Synkroniseen datan lähetykseen käytettävää rajapintaa kutsuttakoon nimellä *SynDatanLahetys*.

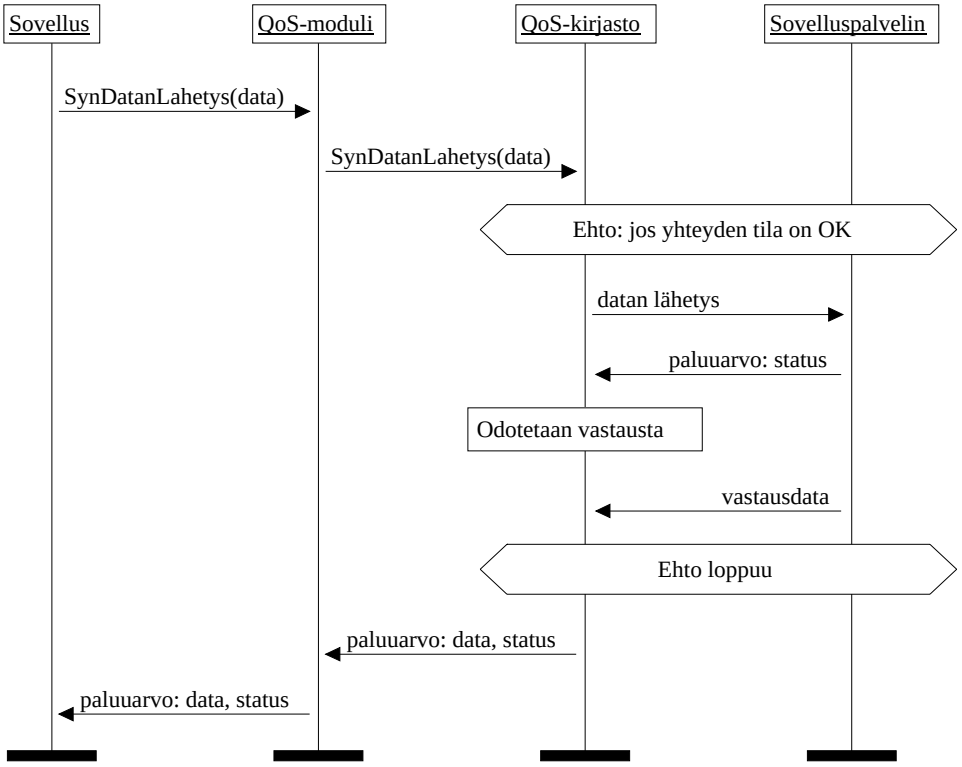
Sovelluksen toteuttamisen helpottamiseksi datanlähetysrajapinnassa voitaisiin käyttää jotain rajapinnan määrittelykieltä, (IDL, Interface Definition Language), Corba- tai COM-ympäristöjen tapaan. Tällöin sovelluskohtaisesti voitaisiin määritellä mahdollisesti useita datanlähetysrajapintoja siten, että lähetettävää dataa ei tarvitsisi muuntaa sovelluksen koodissa, vaan muunnos tapahtuisi IDL-kääntäjän tuottamassa ns. stub-koodissa, joka suoritetaan ennen kuin kutsu siirtyy rajapinnan toteuttavan objektin käsiteltäväksi. Tällaista menettelytapaa voidaan pitää suositeltavana, koska se vähentää virhemahdollisuuksia sovelluksen toteutusvaiheessa. Tällainen toiminta edellyttää, että sekä asiakas- että palvelinpäässä käytetään samaa rajapinnan IDL-määrittelyä, koska määrittelyn perusteella tehtävä automaattinen datan muunnos vaikuttaa itse datan esitystapaan verkossa siirrettävässä muodossa.

4.5.2 Synkroninen datan lähetys sovelluskehyksessä

Sovelluskehyksessä QoS-moduli välittää sovelluksen tekemän datanlähetyskutsun parametreineen QoS-kirjastolle, jonka ensimmäinen tehtävä on tarkistaa yhteys kohdeosoitteeseen. Jos yhteyttä ei ole, ei datan lähetystäkään voida suorittaa.

QoS-kirjasto lähettää datan käytössä olevaa QoS-ratkaisua käyttäen kohdepalvelimelle ja saa varmistuksen lähetyksen onnistumisesta. Tämän jälkeen QoS-kirjasto jää odottamaan kohdepalvelimelta saapuvaa vastausta

esimerkiksi säikeessä, jossa samalla tarkastellaan, että sovelluksen antama aikaraja ei täyty vastausdataa odotellessa. Vastauksen saatuaan säie lopettaa aikavalvonnan ja tallettaa vastaanotetun datapuskurin. Datapuskuri välitetään takaisin sovellukselle rajapinnan paluuparametreina operaation tulosta ilmaisevan status-tiedon kanssa. Status-tiedolla ei ole yhteyttä itse palvelimella tehdyn operaation tulokseen, vaan se ilmaisee pelkästään datan lähetyksen ja vastaanoton onnistumista käytetyn datanvälitysprotokollan tasolla. Kuva 9 havainnollistaa synkroniseen datan lähetykseen liittyviä operaatioita QoS-sovelluskehyksessä.



Kuva 9: Synkroninen datan lähetys.

4.5.3 Poikkeustilanteet synkronisessa datan lähetyksessä

Synkronisessa datan lähetyksessä tapahtuvat virheet aiheuttavat sovelluksen kannalta operaation epäonnistumisen riippumatta siitä, missä vaiheessa operaatio epäonnistuu. Poikkeustilanteeksi katsotaan yhteyden tilaan ja datan siirtoon liittyvät ongelmat, itse siirrettävän datan mahdollisesti sisältäviin virheisiin ei kiinnitetä huomiota sovelluskehyksen puolella, vaan näiden virheiden havaitseminen on sovelluksen vastuulla.

Jos yhteyttä ei ole avattu ennen datan lähetysvaihetta, katsotaan operaatio epäonnistuneeksi. Yhteyden avaamista uudelleen ei voida tehdä, koska rajapinnassa tulisi tällöin määritellä myös yhteyden avaamiseen liittyvät parametrit. Tässä tapauksessa sovellukselle palautetaan välittömästi statustieto, josta ilmenee, että yhteys ei ollut avoinna.

Datan lähetykseen liittyvät virheet yritetään korjata uudelleenlähetyksellä. Ennen uudelleenlähetystä tarkistetaan yhteyden tila, jos käytetty QoS-tekniikka tämän mahdollistaa. Tapauksessa, jossa myös uudelleenlähetys epäonnistuu, palautetaan sovellukselle datan lähetyksen epäonnistumista merkitsevä statustieto.

Rajapinnassa määritellyn aikarajan kuluttua umpeen ei vastausta enää odoteta kohdepalvelimelta. Ongelmaksi tässä tapauksessa tulee mahdollinen viivästynyt vastaus kohdepalvelimelta myöhemmissä datan lähetysoperaatioissa. Mahdollisesti myöhästynyttä vastausta ei saa sotkea myöhempien datanlähetysten vastausten kanssa. Samaan datanvaihto-operaatioon liittyvät kyselyt ja vastaukset tulisi siis jotenkin tunnistaa. Tunnisteen perusteella QoS-sovelluskehys voi hylätä mahdolliset aiempiin kyselyihin myöhässä tulevat vastaukset.

4.6 Datan lähetys asynkronisesti

Asynkronisella datan lähetyksellä tarkoitetaan tilannetta, jossa vastausdataa ei jäädä odottamaan kohdepalvelimelta, vaan vastaus saapuu joskus myöhemmin. Tällaista operaatiota tarvitaan tilanteissa, joissa vastauksen saapumisaikaa ei tiedetä. Esimerkkinä voisi olla säätilaa tarkkaileva sovellus, joka haluaa tiedon palvelimelta lämpötilan laskiessa tietyn rajan alle. Säätilan ollessa vaikeasti ennustettavissa, ei rajan alitusajankohtaa voida etukäteen tietää. Toimintona asynkroninen datan lähetys on hyvin samankaltainen, kuin synkroninenkin datan lähetys, toiminta QoS-sovelluskehyksessä vain loppuu sen jälkeen, kun data on saatu välitettyä kohdepalvelimelle. Asynkronisen datan lähetyksessä vaatimuksena on, että sovellus toteuttaa takaisinkutsurajapinnan, jonka avulla kohdepalvelimelta saapuva data voidaan välittää sovellukselle käsiteltäväksi.

Samaan kohdeosoitteeseen ei samanaikaisesti ole suositeltavaa käyttää synkronista ja asynkronista datan lähetystä samaa yhteyttä käyttäen, koska synkronisen lähetyksen vastausdatan odottelu ja mahdolliset asynkroniset vastaukset voisivat mennä sekaisin. Sovellus voi käyttää samanaikaisesti kahta yhteyttä, joista toinen toimii synkronista datan lähetystä varten ja toinen hoitaa asynkronisen datan vastaanottamisen.

4.6.1 Rajapinta asynkronista datan lähetystä varten

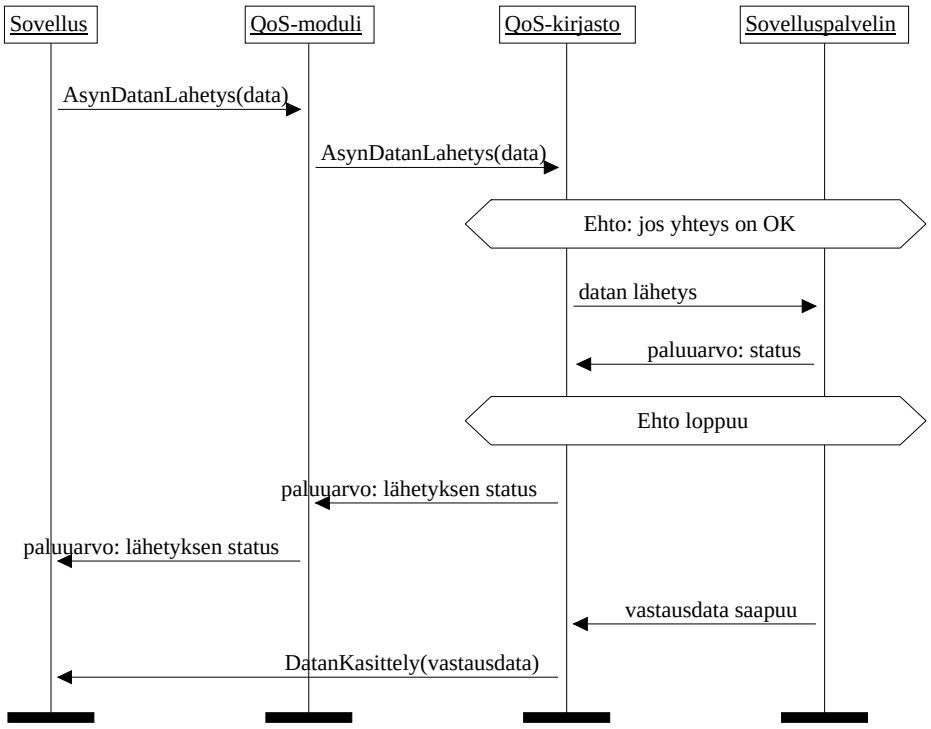
Asynkronisessa datan lähetyksessä on siis pari eroavaisuutta synkroniseen datan lähetykseen verrattuna, koska vastausta kohdepalvelimelta ei jäädä odottamaan ja myöhemmin saapuva vastaus välitetään sovellukselle takaisinkutsulla. Rajapinnassa eroavaisuudet näkyvät aikarajan poistumisena ja mahdollisena takaisinkutsuobjektin tunnisteena. Tämä tunniste ei ole tarpeellinen, jos sovellus kommunikoi vain yhden kohdepalvelimen kanssa, tai jos QoS-sovelluskehys toteutetaan siten, että jokaista kohdeosoitetta varten alustetaan oma objektinsa, joka sisältää vain tietyn kohdepalvelimen kanssa kommunikointiin vaadittavat tiedot. Tällöin QoS-sovelluskehyksen instanssi tietää mille takaisinkutsuobjektille data tulee lähettää. Käytännössä usean takaisinkutsuobjektin määrittely samalle yhteydelle on ongelmallista, koska vastausdatan saapuessa olisi jotenkin osattava valita oikea takaisinkutsuobjekti jolle data ohjataan.

Parametreina rajapinnassa on siis lähetettävä data, sekä tunniste takaisinkutsuobjektille. Operaation paluuarvo kertoo statuksen, jonka perusteella eri virhetilanteet kerrotaan sovellukselle. Kutsuttakoon asynkroniseen datan lähetykseen käytettävää rajapintaa nimellä *AsynDatanLahetys*.

4.6.2 Asynkroninen datan lähetys sovelluskehyksessä

QoS-moduli välittää kutsun ja saamansa parametrit edelleen QoS-kirjastolle, jonka ensimmäisenä tehtävänä on tarkistaa yhteyden tila. Yhteyden ollessa avoinna, seuraavaksi tarkistetaan onko sovellus välittänyt viitteen omaan takaisinkutsuobjektiinsa. Takaisinkutsuobjektin käyttö on välttämätöntä, koska takaisinkutsu on ainoa keino välittää kohdepalvelimen

myöhemmin mahdollisesti lähettämä data sovelluksen käsiteltäväksi. Oletuksena siis on, että QoS-kirjastolla on vain yksi viite sovelluksen takaisinkutsua varten. Tarkistusten jälkeen suoritetaan itse datan lähetys käytössä olevan QoS-menettelyn mukaisesti. Lähetyksen jälkeen operaation status palautetaan sovellukselle, joka tämän jälkeen käsittelee mahdolliset vastaukset takaisinkutsun avulla. Kuva 10 esittää sovelluksen ja QoS-sovelluskehyksen osien välillä tehtävät kutsut.



Kuva 10: Asynkroninen datan lähetys.

4.6.3 Poikkeustilanteet asynkronisessa datan lähetyksessä

Esitarkistusten aikana havaittavista ongelmista kerrotaan sovellukselle operaation paluuarvon kautta. Ongelmat voivat liittyä joko yhteyden tilaan tai puuttuvaan takaisinkutsuobjektiin. Näistä ongelmista ei voida toipua automaattisesti, vaan sovelluksen tulisi alustaa yhteys uudestaan ja antaa QoS-sovelluskehykselle toimiva viite takaisinkutsujen tekemistä varten.

Datan lähetyksen epäonnistuessa QoS-modulissa, yhteyden tila tarkistetaan ja lähetystä yritetään uudestaan, jos yhteyden tila sen mahdollistaa. Myös uudelleenlähetyksen epäonnistuessa tieto epäonnistuneesta lähetyksestä välitetään QoS-sovelluskehykselle ja edelleen sovellukselle.

4.7 Datan vastaanotto

Sovelluksen käyttäessä asynkronista datan lähetystä, voi vastaus tulla satunnaisena ajanhetkenä kyselyn lähetyksestä sen jälkeen, kun kohdepalvelin on saanut asiakassovelluksen kyselyn käsiteltyä. Lisäksi on mahdollista, että kohdepalvelin lähettää dataa ilman erityisiä kyselyjä asiakassovellukselta, tällöin on tarpeen toteuttaa mekanismi, jonka avulla data saadaan välitettyä sovelluksen käsiteltäväksi. Aiemmissa operaatiokuvauksissa mainittua takaisinkutsuobjektia käytetään tähän tarkoitukseen, sen on toteutettava QoS-sovelluskehyksessä määritelty operaatio, jolla vastaanotettu data voidaan välittää sovellukselle.

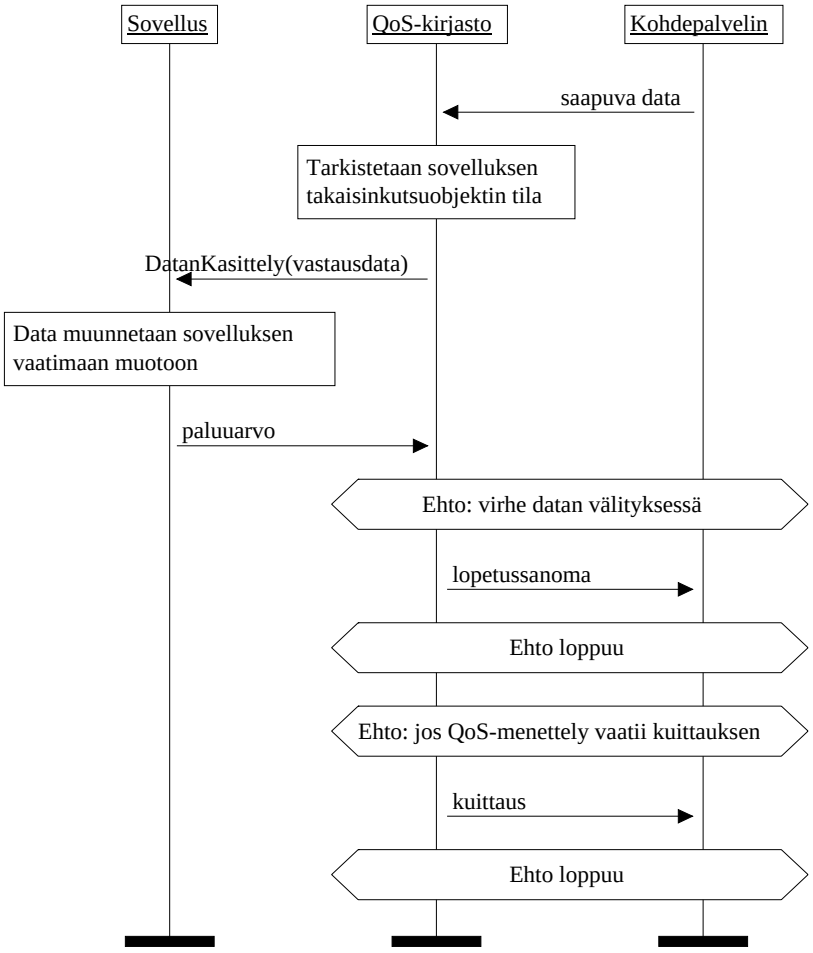
4.7.1 Takaisinkutsurajapinta datan vastaanottoon

Takaisinkutsurajapinnassa on sovellukselle välitettävä data, joka saatiin kohdepalvelimelta, sekä datan määrä tavuina. Sovelluksen datan käsittelyssä tapahtuvia virheitä varten tulee operaatiossa palauttaa statustieto, jolla QoS-kirjastoa informoidaan mahdollisista ongelmista takaisinkutsuobjektin kanssa. Kutsuttakoon tätä takaisinkutsurajapintaa nimellä *DatanKasittely*.

4.7.2 Saapuneen datan käsittely sovelluskehyksessä

Sovelluksen avattua yhteyden kohdepalvelimeen QoS-sovelluskehyksen avulla, kykenee myös kohdepalvelin lähettämään dataa asiakassovellukselle. Datalähetyksen saapuessa QoS-kirjastolle, tarkistetaan ensimmäisenä, että sovelluksen takaisinkutsuobjekti on saatavilla. Tarkistuksen jälkeen vastaanotettu data ja datan määrä tavuina välitetään sovelluksen takaisinkutsuobjektille, jonka velvollisuutena on tarkistaa vastaanotetun datan oikeellisuus. Käytännössä tämä tapahtuu muuntamalla datapuskuri sovelluksen tuntemaan muotoon. Muunnoksen onnistuessa datan käsittely on QoS-sovelluskehyksen kannalta onnistunut. Tässä tapauksessa sovelluksen takaisinkutsuobjektin toteutus ei ota kantaa itse datan sisältöön ja sen mahdollisesti sisältämiin virheisiin, näiden virheiden käsittely hoidetaan asiakas- ja palvelinsovellusten kommunikaation avulla. Siis vain datan siirtoon liittyvät tarkistukset otetaan huomioon, kun takaisinkutsuoperaation status palautetaan QoS-kirjastolle. Sovelluksen vastaanotettua QoS-kirjaston

välittämän datan, voidaan QoS-kirjaston puolella tehdä mahdolliset QoS-menettelyn vaatimat kuittaukset ja jäädä odottamaan seuraavaa datalähetystä, tai suorittaa seuraava sovelluksen pyytämä operaatio yhteyden hallintaan tai datan lähetykseen liittyen. Kuva 11 esittää datan vastaanottamisen käsittelyn QoS-sovelluskehyksen ja sovelluksen kesken.



Kuva 11: Datan vastaanotto.

4.7.3 Poikkeusten käsittely datan vastaanotossa

Poikkeustilanteet datan vastaanotossa liittyvät sovelluksen takaisinkutsuobjektin saatavuuteen ja sovelluksen puolella tapahtuvaan datan käsittelyyn. Kohdeosoitteeseen avattu yhteys on kunnossa, koska ehtona koko datan käsittelylle on, että ensin on tapahtunut datan lähetys kohdepalvelimen puolelta.

Sovelluksen takaisinkutsuobjektin puuttuessa ei dataa voida välittää sovellukselle, joten jollain keinolla täytyy pystyä kertomaan datan lähettäjälle, että sovellus ei voi vastaanottaa kyseisellä hetkellä dataa. Tällainen informointi ei kuitenkaan onnistu QoS-kirjaston toimesta automaattisesti muuten kuin sulkemalla koko yhteys dataa lähettäneeseen kohdepalvelimeen, koska asiakassovelluksen ja palvelimen välinen kommunikointiprotokolla on vain sovelluksen tiedossa. Yhtenä mahdollisuutena olisi yhteyden avauksessa tehtävässä takaisinkutsuobjektin välittämisessä QoS-kirjastolle välittää myös kommunikoinnin lopettava datapaketti, jolla kohdepalvelinta voitaisiin informoida datan vastaanotossa tapahtuvissa virhetilanteissa. Datapaketin sisällön merkitys on siis asiakassovelluksen ja palvelimen tuntema, ja tämän paketin perusteella palvelin osaa tunnistaa virhetilanteen ja lopettaa datan lähetyksen asiakassovellukselle.

Samaa lopetusdatapakettia voidaan käyttää, jos asiakassovelluksen vastaanottaman datan muunnos epäonnistuu takaisinkutsuobjektissa. Muunnoksen epäonnistumista merkitsevä virhe palautuu QoS-kirjastolle, joka edelleen joko sulkee yhteyden tai lähettää lopetussanoman palvelimelle.

4.8 Yhteyden parametrien muuttaminen

Tarpeen vaatiessa sovelluksien QoS-parametreja tulee voida muuttaa QoS-sovelluspalvelimella, tällainen tarve tulee esimerkiksi tilanteessa, jossa videoneuvottelusovellukselle varattu kaistanleveys ei riitä riittävän laadukkaaseen videokuvaan. Myös sovellusten määrä saattaa kasvaa siinä määrin, että kaistanleveysvaatimuksia on syytä alentaa, jotta verkon resurssit riittävät palvelemaan kaikkia neuvotteluun osallistuvia sovelluksia. Sovelluskohtaisten tietojen muuttaminen vaikuttaa jatkossa sekä seuraaviin avattaviin yhteyksiin että jo avattuihin yhteyksiin. Parametreja voidaan muuttaa yksittäisten sovellusten osalta tai esimerkiksi tietyn palvelun osalta, jolloin verkon resurssien hallinta onnistuu keskitetysti. Parametrien hallintaa varten tarvitaan erityinen QoS-hallintasovellus, jolla muutokset voidaan tehdä QoS-parametripalvelimelle. Tämän hallintasovelluksen tarkempi määrittely rajataan tämän tutkimuksen ulkopuolelle. Tässä luvussa tarkastellaan miten itse

parametrit välitetään jo käynnissä oleville sovelluksille ja miten uudet parametrit otetaan sovelluksissa ajoaikaisesti käyttöön.

4.8.1 Parametrien välittämiseen tarvittavat rajapinnat

QoS-parametrien hallintaan ja jakeluun tarvitaan useampaa eri rajapintaa, koska tässä operaatiossa on useita osapuolia, joiden välillä siirretään dataa.

QoS-parametrien hallittuun muuttamiseen tarvitaan oma rajapintansa. Tämän rajapinnan avulla on voitava määritellä kohdesovellus, tai palvelu, jonka parametreja ollaan muuttamassa, uudet parametrit, sekä tieto päivitetäänkö kyseisellä hetkellä käynnissä olevat sovellukset ja niiden käyttämät avoimet yhteydet käyttämään uusia parametreja. Ensimmäisenä parametrina tarvitaan kohdesovellukselle tai palvelulle tunniste, jonka avulla QoS-parametripalvelin osaa hakea muutettavan kohteen parametrit. Aiemmin yhteysparametrien hakua kuvaavassa aliluvussa (4.2) mainittiin yksilölliset tunnisteet (GUID), joiden avulla QoS-parametripalvelin liittää tallettamansa parametrit sovelluksiin tai palveluihin. Tässä tilanteessa voidaan olettaa, että QoS-hallintasovellus saa erillisellä kutsulla listan palveluista, joista hallintasovelluksen käyttäjä voi valita muutettavan sovelluksen tai palvelun. Toisena parametrina, tai parametreina, tarvitaan itse QoS-parametrit, joita voi olla useita riippuen käytettävästä QoS-menetelmästä. Tällaiset arvot voidaan välittää taulukkona nimi+arvo –pareja, joista nimi määrittelee parametrin tyypin ja arvo vastaavasti parametrin arvon. Tällaisella geneerisellä parametrilla voidaan välittää minkä tahansa parametrin tai parametrijoukon tiedot QoS-parametripalvelimelle. Kolmantena parametrina välitetään arvo, jolla kerrotaan QoS-parametripalvelimelle otetaanko uudet arvot käyttöön välittömästi, vai vasta kun parametreja tarvitseva sovellus käynnistyy seuraavan kerran. Tällä parametrilla voidaan välttää tilanteita, joissa täytyisi tehdä laajamittaista yhteyksien uudelleenneuvottelua, koska tämä kuormittaisi verkkoa hetkittäisesti. Käynnissä olevien sovellusten informointi tulisi siis käytännössä kyseeseen, jos sovelluksia on käytössä vähän. QoS-hallintasovelluksen ja parametripalvelimen välistä rajapintaa kutsuttakoon nimellä *LisaaParametrit*.

QoS-parametripalvelimen ja QoS-modulin välille tarvitaan rajapinta, jolla muuttuneet parametrit välitetään käynnissä oleville sovelluksille. Tiedot kierrätetään QoS-modulin kautta QoS-kirjastolle, joka hoitaa edelleen yhteyden uudelleenneuvottelun uusilla parametreilla. Tämä kierrätys tehdään sen vuoksi, että vain QoS-moduli kommunikoi QoS-parametripalvelimen kanssa. QoS-kirjaston tehtävänä on hoitaa QoS-menetelmäkohtainen yhteyden hallinta. Tässä rajapinnassa täytyy siirtää vain parametrit, joiden avulla yhteys tulee uudelleenneuvotella, mutta myös muuttumattomat parametrit tulee välittää QoS-modulille. Tätä rajapintaa kutsuttakoon nimellä *PaivitaParametrit*.

QoS-sovelluskehyksen yhteyden parametrien muutokseen liittyen tarvitaan vielä yksi rajapinta, jolla sovelluksen takaisinkutsuobjektia voidaan informoida yhteyden uudelleenneuvottelun epäonnistuessa. Tässä rajapinnassa ei varsinaisesti tarvita mitään parametreja, mutta yhteyden katkeamisen syy voidaan kuljettaa kutsussa, jos sovelluksen on tarpeellista esimerkiksi kertoa käyttäjälle yhteydessä tapahtuneesta virheestä. Tätä rajapintaa kutsuttakoon nimellä *YhteysSuljettu*.

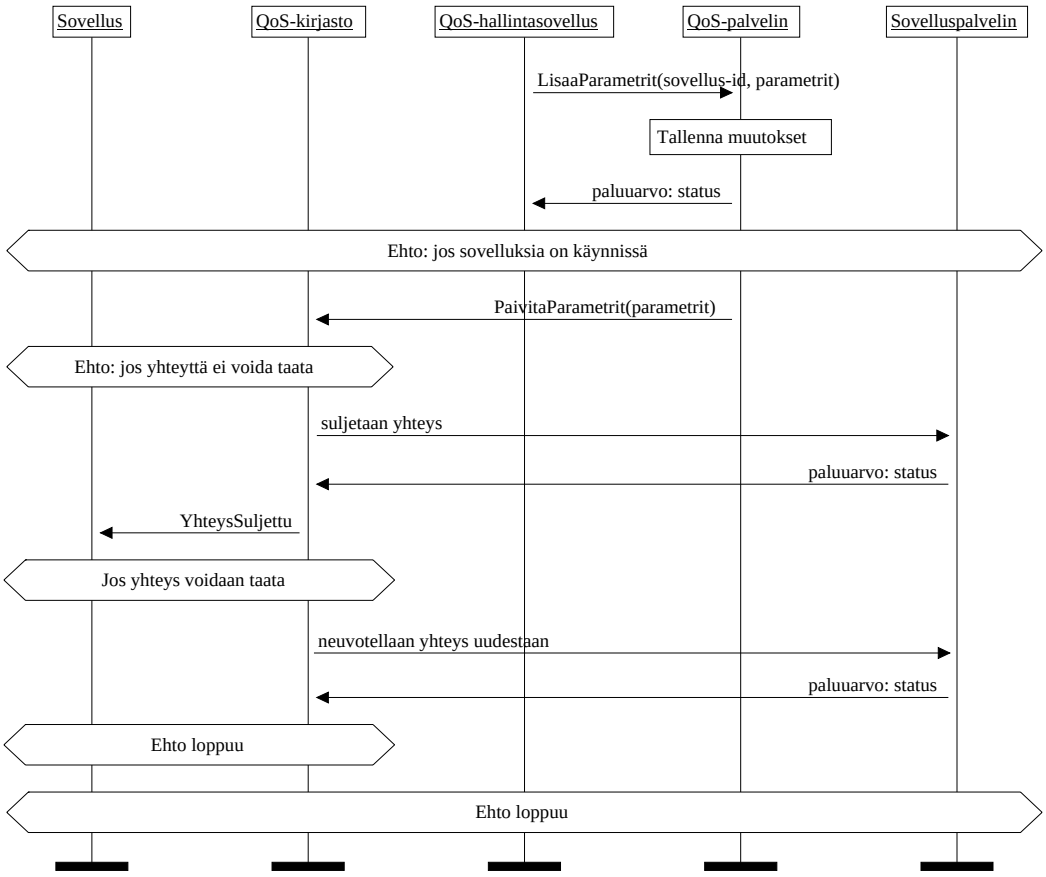
4.8.2 Parametrien muutosten käsittely sovelluskehyksessä

QoS-parametripalvelin vastaanottaa muuttuneet parametrit, jotka liittyvät joko yksittäiseen sovellukseen tai tiettyyn palveluun. Parametripalvelin hakee tietovarastostaan parametrina saatua tunnistetta vastaavat parametrit ja päivittää näiden arvot. QoS-parametrien hallintasovelluksen toiminnasta oletetaan, että se tarkistaa parametrien järkevyyden siten, että ristiriitaisia parametreja ei pystytä syöttämään. Ristiriitaisia parametreja ovat esimerkiksi viiveen ja luotettavuuden samanaikainen maksimointi. Vaatimukset lyhyelle viiveelle tarkoittavat, että tarpeen vaatiessa joitain datapaketteja joudutaan hylkäämään, ja tämä on suoraan ristiriidassa korkean luotettavuuden kanssa.

QoS-parametripalvelimen tallennettua parametrit tietovarastoonsa, se tarkistaa täytyykö muuttuneet parametrit jakaa jo käynnissä oleville sovelluksille. Jos ei, niin toiminta on valmis ja uudet parametrit tulevat jakoon, kun niitä tarvitsevat sovellukset pyytävät yhteysparametreja, muutoin parametrit on tiedotettava käynnissä oleville sovelluksille. Yksi tapa suorittaa parametrien jako on lähettää broadcast-sanoma kaikille verkossa oleville

koneille, jolloin käynnissä olevien sovellusten QoS-moduleissa huomataan, että parametreja ollaan muuttamassa. Toinen vaihtoehto olisi pitää yllä käynnissä olevien sovellusten listaa ja tiedottaa parametrien muutoksesta pienemmällä jakelulla, mutta tämä vaatisi selkeästi enemmän varmennuksia esimerkiksi parametripalvelimen uudelleenkäynnistysten ja verkkokatkosten varalta. Onkin siis selkeintä tehdä parametrien jakelu broadcast-sanomalla siten, että datan mukana kulkevat sovelluksen ja palvelun tunnisteet ja uudet parametrit. Käynnissä olevien sovellusten QoS-modulit osaavat poimia muuttuneet parametrit tunnisteiden perusteella.

QoS-modulin havaitessa sovelluksen käyttämään palveluun kohdistuvan broadcast-sanoman, se purkaa sanomasta uudet parametrit ja välittää ne QoS-kirjastolle. QoS-kirjasto vastaanottaa parametrit ja tekee käytetyn QoS-menetelmän mukaan yhteyden uudelleenneuvottelun uusilla parametreilla. Riippuen käytetystä QoS-menetelmästä, voidaan jo olemassa olevaa yhteyttä muuttaa, tai vaihtoehtoisesti avata kokonaan uusi yhteys kohdepalvelimelle ja purkaa vanha yhteys tämän jälkeen. Jos uuden yhteyden avaaminen ei onnistu, on myös vanha yhteys purettava siinä tapauksessa, että yhteyden QoS-vaatimuksia on juuri alennettu. Näin täytyy menetellä siksi, että vanhan yhteyden tulkitaan kuluttavan liikaa verkkoresursseja. Käytännössä yhteyden vaatimusten alentamisen pitäisi onnistua, jos parametreja voidaan muuttaa ilman uuden yhteyden avaamista vanhan rinnalle. Yhteyden uudelleenneuvottelun epäonnistuessa sovelluksen takaisinkutsuobjektille kerrotaan, että yhteys on sulkeutunut. Sovelluksen on näin ollen yritettävä avata yhteyttä uudestaan jossain myöhemmässä vaiheessa. Kuva 12 havainnollistaa yhteyden uudelleenneuvottelussa tapahtuvia kutsuja QoS-sovelluskehyksessä.



Kuva 12: Asetusten hallinta.

4.8.3 Poikkeustilanteet yhteyden uudelleenneuvottelussa

QoS-parametrien talletus parametripalvelimelle voi epäonnistua. Tämä ei saa kuitenkaan aiheuttaa talletettujen parametrien sotkeutumista siten, että QoS-sovelluskehystä käyttäville sovelluksille välitettäisiin käynnistysvaiheessa vääriä parametreja, vaan talletus on hoidettava siten, että kaikkien parametrien talletuksen pitää onnistua. Käytännössä tämä tarkoittaa esimerkiksi transaktioiden käyttöä, jos parametrit on talletettu tietokantaan.

Yhteyden uudelleenneuvottelussa voidaan törmätä pariinkin poikkeustilanteeseen. Ensimmäisenä täytyy tarkistaa voidaanko saatuja parametreja käyttäen neuvotella uusi yhteys ja toisena itse yhteyden uudelleenneuvottelun mahdollinen epäonnistuminen. Molemmissa tilanteissa päädytään siihen tulokseen, että sovelluksen käyttämää yhteyttä ei voida enää jatkaa, koska sille asetettuja QoS-vaatimuksia ei voida taata. Näissä tilanteissa sovelluksen takaisinkutsuobjektille ilmoitetaan yhteyden sulkemisesta ja

asiakassovelluksen täytyy yrittää yhteyden avausta myöhemmässä vaiheessa uudestaan, jos se haluaa jatkaa kommunikointia kohdepalvelimen kanssa.

4.9 Verkon palvelutason muuttuminen

Tietyn QoS-tason ylläpitämiseksi verkon resursseista on pidettävä jatkuvasti kirjaa ja verkon kapasiteettia on myös jatkuvasti mitattava, jotta mahdolliset kapasiteetin muutoksien aiheuttamat vaikutukset sovellusten palvelun tasoon saadaan minimoitua. Tietylle liikenneryhmälle varattu kaista verkossa voi sovellusten lisääntyessä ruuhkaantua siten, että sovelluksien QoS-vaatimuksia ei pystytä täyttämään, tai esimerkiksi jokin reititin verkossa vikaantuu ja liikenne joudutaan siirtämään kapasiteetiltaan pienempää linkkiä pitkin. Tällaisissa tilanteissa sovelluksia täytyy pystyä informoimaan ja tarvittaessa ohjata niitä pienentämään QoS-vaatimuksiaan, jotta tietty palvelun taso pystyttäisiin takaamaan myös poikkeustilanteissa. Tällainen dynaaminen QoS-vaatimusten säätäminen vaatii myös sovelluksilta skaalautuvuutta, eli niiden tulee valmistautua tilanteisiin, joissa optimilaatua ei aina ole saatavilla. Esimerkiksi videoneuvottelusovelluksissa tulisi voida laskea siirrettävän kuvan tarkkuutta tai kehysten määrää (frame rate) vähentää.

QoS-tason monitoroiminen on hieman kaksiteräinen miekka. Toisaalta sen tekeminen on välttämätöntä, jotta verkon käyttäytyminen olisi edes jotenkin ennakoitavissa, toisaalta taas monitorointiprosessi on osaltaan vähentämässä sovelluksien käytössä olevaa kapasiteettia. Ongelma on siis hieman sama kuin osassa QoS-menetelmiä, joissa resurssien varaukseen liittyy signalointia, joka osaltaan ruuhkauttaa verkkoa yhteyksien rakentamisvaiheessa. Kaikki QoS-tekniikat eivät tarvitse resurssien monitorointia, koska osa tekniikoista toimii täysin reititystasolla, eikä resursseja näin ollen varata yhteyskohtaisesti. QoS-sovelluskehyksessä monitorointi täytyy kuitenkin ottaa huomioon, koska sen poisjättäminen rajoittaisi sovelluskehyksen käyttömahdollisuuksia. Toisaalta monitoroinnin ollessa usean QoS-tekniikan ominaisuus, ei sovelluskehyksen tarvitse suuremmin asiaan ottaa kantaa, koska QoS-kirjasto, joka hoitaa QoS-menetelmäkohtaisen toiminnan, toteuttaa yleensä myös monitoroinnin seuraamisen osana käytettyä QoS-menetelmää.

4.9.1 Palvelutason muuttumiseen liittyvät rajapinnat

QoS-tekniikoiden hoitaessa verkon resurssien kirjaamisen ja monitoroinnin, ei sovelluskehyksen periaatteessa tarvitsisi ottaa huomioon tilanteita, joissa monitorointisovellukset informoivat palvelun tason muutoksista. Tämä siis siksi, että monitoroinnin tuloksiin reagoiva QoS-kirjasto toteuttaisi QoS-tekniikan mukaisen rajapinnan, jonka kautta muutokset tulisivat suoraan QoS-kirjastolle. Periaatteessa verkoissa voisi kuitenkin toteuttaa omankin mekanismin palvelun tason monitoroimiseksi, ja tähän tarkoitukseen QoS-sovelluskehys voisi tarjota rajapinnan, jonka kautta sovellusten QoS-vaatimuksia voitaisiin säätää dynaamisesti verkon kuormituksen mukaan. Geneerinen rajapinta mahdollistaisi myös kaupallisten verkon monitorointisovellusten integroinnin QoS-sovelluskehystä hyödyntäviin sovelluksiin. Tämä tietysti edellyttäisi, että käytössä olisi jokin erillinen sovellus, joka tulkkaisi kaupallisten sovellusten analysoimat tulokset QoS-sovelluskehyksen kielelle ja edelleen sitä hyödyntävien sovellusten käyttöön.

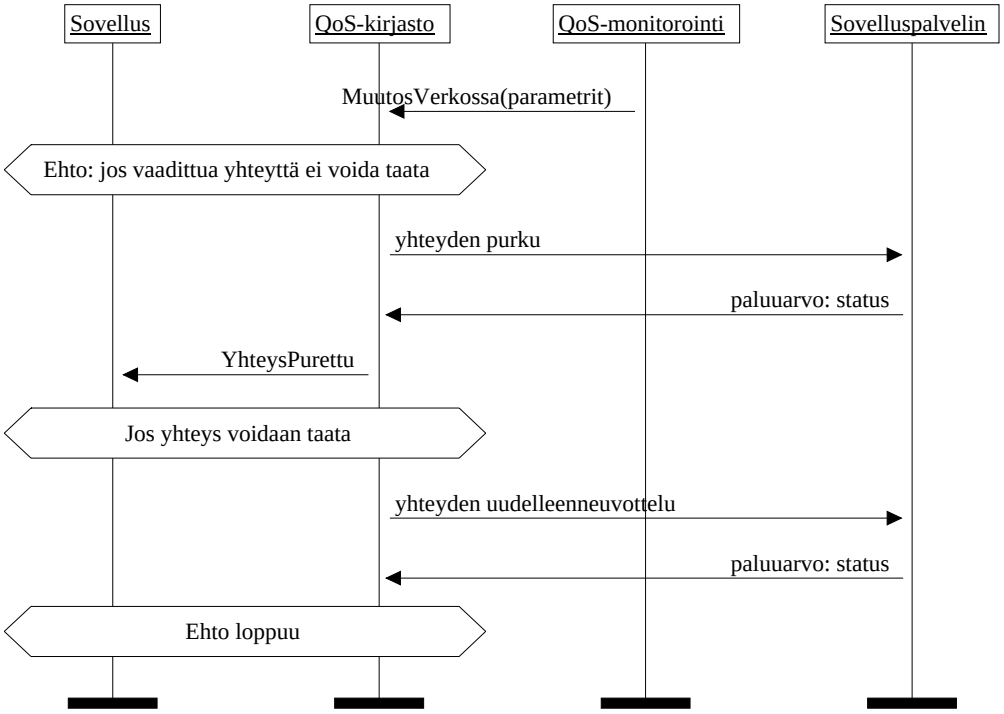
QoS-sovelluskehystä hyödyntävälle prosessille täytyy tarvittaessa pystyä ilmoittamaan, että vaadittua palvelun tasoa ei enää pystytä tarjoamaan. Tällainen rajapinta määriteltiin jo edellisessä luvussa (*YhteysSuljettu*), jossa käsiteltiin parametrien muuttamista, eli tuota samaa rajapintaa käytetään kertomaan sovellukselle yhteyden katkaisemisesta.

Geneerisempää verkon resurssien monitorointia varten täytyy määritellä QoS-sovelluskehykseen rajapinta, jonka avulla voidaan välittää erillisen verkon monitorointisovelluksen tuottamat tulokset QoS-sovelluskehystä hyödyntävien sovellusten käyttöön yhteysparametrien säätämistä varten. Luonteeltaan verkon analysoinnista tuotetut parametrit ovat uusia maksimiarvoja eri parametrien (viive, kaistanleveys, luotettavuus) osalta, QoS-kirjaston tehtäväksi jää näiden parametrien perusteella joko nostaa tai laskea käytössä olevan yhteyden QoS-vaatimuksia. Periaatteessa rajapinta sisältää hyvin paljon samaa, kuin parametrien muutokseen liittyvä *PaivitaParametrit*-rajapinta, mutta verkon monitoroinnista saatavien tulosten liittyessä yksittäisiin parametreihin, samaa rajapintaa ei voida käyttää. Yksittäisten QoS-parametrien muutokset tulee siis

käsitellä eri tavalla, kuin sovellus- tai palvelukohtaisten parametrijoukkojen muutokset. Rajapinnassa pitää siirtää muuttuneen parametrin tyyppi ja sen uusi maksimiarvo, näitä arvoja voi tulla useampia kerralla, joten parametrina täytyy siirtää lista <tyyppi, arvo> –pareja. Muita parametreja ei tarvita, QoS-sovelluskehyksen tehtävänä on sovelluskohtaisesti päättää, miten verkon resurssien saatavuuden muutokset otetaan huomioon avoinna olevissa yhteyksissä.

4.9.2 Verkon palvelutason muutosten käsittely sovelluskehyksessä

Sovelluskehyksen kannalta palvelutason muutokset näkyvät vain QoS-kirjastossa, jos käytössä on tietyn QoS-tekniikan tarjoama verkon kapasiteetin monitorointi. QoS-kirjasto saa tällöin tiedon käytetyn QoS-tekniikan tarjoamalla tavalla ja yrittää sovittaa avoinna olevan yhteyden verkon käytössä olevaa kapasiteettia vastaten. QoS-kirjastolla on oltava tiedossa mikä on eri parametrien arvoalue, jotta päätös resurssien uudelleenvarauksesta tai yhteyden sulkemisesta voidaan tehdä. Verkon resurssien muutoksen ollessa sallituissa rajoissa, yhteys avataan uudestaan tai avoinna olevan yhteyden asetuksia muutetaan vastaamaan verkon tilannetta. Mahdollisissa virhetilanteissa sovellukselle voidaan tiedottaa yhteyden katkeamisesta takaisinkutsun avulla. Kuva 13 havainnollistaa QoS-tekniikan avulla tehtävää verkon resurssien valvontaa.



Kuva 13: Yhteyden muutos.

Mahdollista kaupallisilla sovelluksilla tehtävää verkon resurssien monitorointia ja QoS-sovelluskehyksen integrointia siihen ei tässä tutkimuksessa tarkemmin käsitellä, seuraavaksi kuitenkin hieman hahmotellaan minkälainen QoS-sovelluskehyksen toiminta tuossa tilanteessa olisi.

Verkkoon tarvitaan ensinnäkin QoS-sovelluskehystä hyödyntävä sovellus, joka osaa pyytää erilliseltä verkon monitorointisovellukselta sen tarjoamia palveluja ja lisäksi tulkita monitoroinnin tulokset QoS-sovelluskehyksen ymmärtämään muotoon, jotta yhteysparametreihin liittyvät muutokset saadaan välitettyä QoS-sovelluskehystä käyttäville sovelluksille. Riippuen käytetystä QoS-menetelmästä, QoS-moduli voi tilata esimerkiksi vuokohtaisesti monitorointipalvelun ja tällöin saadaan valvottua tietyn yhteyden tilaa. Toinen vaihtoehto olisi tehdä yleisempää valvontaa esimerkiksi tekemällä testipakettien lähetystä johonkin tiettyyn kohdeosoitteeseen ja tämän kaiuttamisen perusteella saada tietoa verkon viiveistä ja luotettavuudesta. Pääpiirteissään siis QoS-sovelluskehyksen tulisi tarjota valvonnan tilauspalvelu, jolla tiettyjen resurssien tilanteesta saadaan tietoa, sekä

muunnospalvelu, jolla valvonnan tulokset välitetään takaisin QoS-sovelluskehykselle. Itse verkon resurssien valvonta tapahtuu siis QoS-sovelluskehyksen ulkopuolella. Tällainen erillinen verkon resurssien valvonta voisi olla tarpeellinen tilanteissa, joissa käytössä oleva QoS-tekniikka ei tarjoa verkon resurssien valvontaa.

4.9.3 Poikkeustilanteet verkon palvelutason muutoksien käsittelyssä

Verkon monitoroinnin perusteella saatujen parametrien avulla tehtävän uuden yhteyden avauksen epäonnistuessa sovellukselle tiedotetaan yhteyden katkeamisesta takaisinkutsun avulla. On lisäksi mahdollista, että sovelluksen informointi yhteyden katkeamisesta epäonnistuu takaisinkutsun avulla. Takaisinkutsun ollessa ainoa keino tiedottaa sovellusta yhteyden muutoksista, ei sovellus saa tietoa yhteyden katkeamisesta ennen kuin se yrittää kutsua joitain QoS-modulin operaatioita.

4.10 Yhteyden katkeaminen

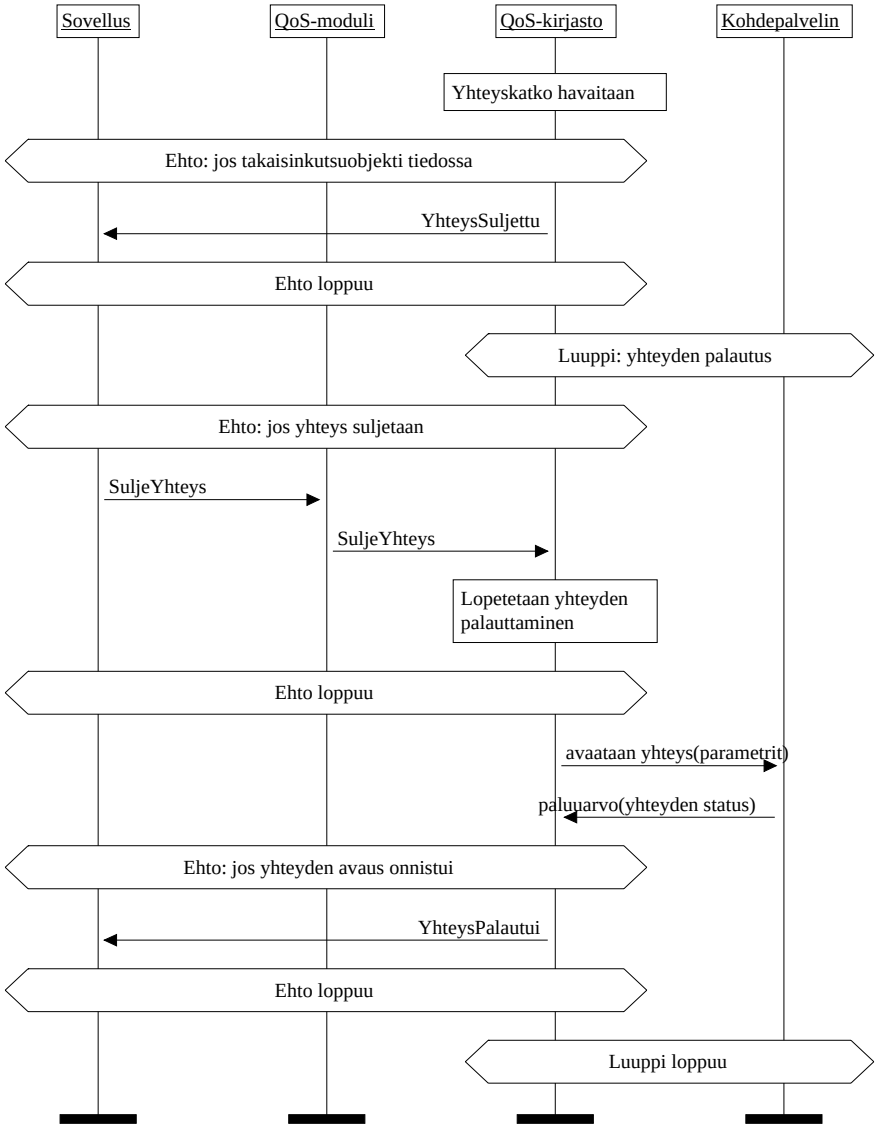
Epävakaassa tai ylikuormitetussa verkossa yhteyksien katkeilu saattaa olla yleistä ja sovellusten on varauduttava tähän. QoS-sovelluskehyksen tulee tarjota tällaisissa tapauksissa hallitut mekanismit yhteyksien sulkemiseen ja uudelleen avaamiseen. Sovelluskohtaisesti voidaan QoS-parametripalvelimelle tallettaa säännöt, joiden mukaan yhteyskatkotilanteissa toimitaan. Toiminta voi olla käytännössä joko vain sovellukselle ilmoittaminen yhteyden katkeamisesta, tai yhteyden uudelleen avaaminen tietyn ajan kuluttua. Toiminta riippuu tietysti täysin sovelluksen tarkoituksesta, yhteyden korkeaan luotettavuuteen nojaavien sovellusten voisi olettaa tyytyvän yhteyden hallittuun katkaisemiseen, koska luotettavuuden kannalta pahin on jo tapahtunut. Tilanteissa, joissa yhteyden katkeamista ei katsota äärimmäisen kriittiseksi, voitaisiin yrittää yhteyden uudelleen avaamista tietyn ajan kuluttua. Tällainen hypoteettinen tilanne voi toteutua esimerkiksi videoneuvottelussa, jossa videokuva katkeaa mutta ääniyhteys edelleen säilyy. Automaattisella yhteyden uudelleen avaamisella QoS-sovelluskehyksen puolella voidaan yksinkertaistaa itse sovelluksen toteutusta.

4.10.1 Rajapinnat yhteyden katkeamisen käsittelyyn sovelluskehyksessä

Sovelluskehyksen tulee tarjota sovellukselle kaksi rajapintaa, joiden avulla yhteyden katkeamisesta ja uudelleen avaamisesta voidaan ilmoittaa takaisinkutsuobjektin avulla. Yhteyden katkeamisesta tiedotetaan jo aiemmissa luvuissa mainitulla *YhteysSuljettu*-rajapinnalla. Yhteyden palautumiseen tarvitaan taas uusi rajapinta, tämän rajapinnan ei tarvitse välittää mitään erityistä tietoa. Lähinnä riittää, että sovelluksen takaisinkutsuobjekti saa tiedon yhteyden palautumisesta. Kutsuttakoon tätä rajapintaa nimellä *YhteysPalautui*. Parametreja ei tarvita, koska takaisinkutsuobjekti liittyy aina yhteen avoinna olevaan yhteyteen.

4.10.2 Yhteyden katkeamisen käsittely sovelluskehyksessä

QoS-sovelluskehyksen puolella yhteyden katkeaminen havaitaan QoS-kirjaston puolella, jossa itse yhteyden käsittely tehdään. Edellyttäen, että sovellus on välittänyt viitteen takaisinkutsuobjektiinsa, QoS-kirjasto ilmoittaa *YhteysSuljettu*-rajapinnan avulla yhteyden katkeamisesta. Tämän jälkeen riippuen sovellukselle asetetuista parametreista, QoS-kirjasto joko jää odottamaan seuraavia operaatiokutsuja sovellukselta, tai se yrittää avata yhteyttä tietyn ajan kuluttua uudestaan, kunnes yhteys palautuu tai sovellus pyytää yhteyden sulkemista. Yhteyden palauduttua QoS-kirjasto ilmoittaa sovellukselle *YhteysPalautui*-rajapinnan avulla yhteyden uudelleen avaamisesta. Kaikki yhteyteen liittyvät lähetystoiminnot ovat tänä aikana estettynä, sovelluksen tekemät datan lähetysyritykset estetään automaattisesti, eikä normaalisti datan lähetyksessä tehtävää yhteyden uudelleenavaamista yritetä, koska QoS-kirjasto on jo yrittämässä yhteyden uudelleen avaamista. Sovelluksen toteutuksessa tulisikin ottaa huomioon, että turhia datan lähetysyrityksiä ei tehdä, jos takaisinkutsun kautta on saatu tieto yhteyden katkeamisesta. Kuva 14 esittää sovelluksen ja QoS-sovelluskehyksen välistä toimintaa yhteyden katkeamisen tapauksessa.



Kuva 14: Yhteyden katkeaminen.

4.10.3 Poikkeustilanteet yhteyden katkeamisen käsittelyyn liittyen

QoS-kirjaston yrittäessä tiedottaa sovellukselle yhteyden katkeamisesta tai uuden yhteyden avaamisesta, voi sovelluksen takaisinkutsuobjektin toteuttaman rajapinnan suorituksessa tapahtua virheitä. Takaisinkutsun ollessa ainoa keino tiedottaa sovellukselle yhteyksissä tapahtuneista muutoksista, ei QoS-kirjastolla ole muuta mahdollisuutta, kuin jäädä odottamaan mahdollisia kutsuja sovellukselta ja informoida ongelmista näiden kutsujen paluuarvojen kautta. Takaisinkutsuobjektin vikaantumisen aiheuttamien ongelmien estämiseksi sovelluksen tulisi odottaa vain tiettyyn pisteeseen asti yhteyden

palautumista. Jos yhteys ei palaudu, niin järkevintä olisi sulkea yhteys hallitusti ja pysäyttää verkon resursseja kuluttava QoS-kirjaston yhteyden palauttaminen.

5. QoS-tarpeiden kuvaaminen geneerisesti rajapinnoissa

Tässä luvussa pyritään tunnistamaan yleiset QoS-teknologioiden käyttöön liittyvät osat, jotka QoS-sovelluskehyksen tulisi tarjota sovelluksille. Mitä yleisempiä ominaisuuksia tunnistetaan, sitä laajemmin sovelluskehys voisi palvella erilaisia QoS-tekniikoita käyttäviä sovellusalueita.

5.1 QoS-tekniikoille yhteiset parametrit

Eri QoS-tekniikoissa on tunnistettavissa useita samoja ominaisuuksia, joita pyritään vakioimaan tarpeellisen palvelun laadun takaamiseksi sovelluksille. Näitä ominaisuuksia ovat *kaistanleveys (bandwidth)*, jonka avulla määrätään sovelluskohtaisesti käytetyn verkkokapasiteetin tarve. Kaistanleveys on oleellinen vaatimus esimerkiksi videoneuvotteluissa ja muissa reaaliaikaista kuvaa ja/tai ääntä välittävissä sovelluksissa. Yleensä yhteyden rakentamisessa pyydetään tiettyä kaistanleveyttä, mutta myös minimi- ja maksimiarvot voidaan määritellä ja yhteyden nopeus voi tällöin vaihdella verkon ruuhkautumisen mukaan.

Toinen melko läheisesti kaistanleveyteen liittyvä ominaisuus on *viive (delay)*, jonka avulla määritellään kauanko datapakettien kulku verkossa saa enimmillään kestää. Kaistanleveys ja viive liittyvät toisiinsa siten, että yleensä suurta kaistanleveyttä vaativat sovellukset tarvitsevat myös lyhyen viiveen. Videoneuvotteluissa lyhyt viive vaikuttaa siten, että uutta dataa vaaditaan tietyn ajan kuluessa, jos dataa ei saada tarpeeksi nopeasti niin sillä ei enää tehdä mitään. Käytännössä tällainen viiveen ylittyminen datan välityksessä näkyy katkoksina videon tai äänen toistossa.

Viiveen lisäksi QoS-parametrina voi olla *viiveen vaihtelu (jitter)*, joka ilmaisee raja-arvot paljonko datapakettien viive saa vaihdella verkossa. Viiveen vaihtelu eri datapakettien välillä aiheuttaa ongelmia erityisesti reaaliaikaisissa

sovelluksissa, joilla on tiukat odotukset datan saapumisajan suhteen. Tällaisissa tapauksissa vaaditaan siis pienen viiveen lisäksi pientä viiveen vaihtelua.

Luotettavuus (*priority*) ilmaisee datapakettien tärkeyden ja tätä tietoa käytetään reitittimissä, kun datapaketteja täytyy hylätä reititysjonojen täyttyessä. Luotettavuus näkyy QoS-parametreissa pienenä datapakettien katoamismääränä. Yleensä korkea luotettavuus ja pieni viive ovat toistensa poissulkevia vaatimuksia. Näin siksi, että luotettavuuden takaaminen voi aiheuttaa datapakettien viipymistä verkon laitteiden jonoissa, kun taas pieni viive aiheuttaa helposti jonossa olevien pakettien hylkäämisen. Luotettavuuden kustannuksella siis joudutaan tinkimään viiveestä. Toisaalta, jos myös kaistanleveyttä on tarjolla tarpeeksi, niin nämäkin vaatimukset voivat esiintyä samanaikaisesti, koska suuremmalla kaistanleveydellä jonot verkon laitteissa ovat pienempiä. Jonojen vähentämisellä päästään pienempään viiveeseen ja myös korkeampaan luotettavuuteen, kun jonoista ei tarvitse hylätä datapaketteja.

QoS-menetelmäkohtaisesti tarvitaan usein erilaisia, samaan ominaisuuteen liittyviä useampia parametreja. Esimerkiksi ATM ja RSVP ilmaisevat kaistanleveystarpeen ja viiveet eri tavoin, tällöin ei voida päästä tilanteeseen, jossa myös parametrit olisivat täysin geneerisiä.

5.2 Parametrien määrittely rajapinnoissa

Geneerisyyden saavuttamiseksi rajapinnoissa käytettävät parametrit tulee määritellä siten, että eri QoS-tekniikoihin liittyvät yhteysparametrit saadaan välitettyä samoilla rajapinnoilla. Eri QoS-tekniikoilla on erilaisia parametreja yhteyden rakentamiseen liittyen, ja samaa tarkoittavat parametrit saattavat olla eri esitysmuodoissa eri tekniikoiden välillä. Kaikkien eri tekniikoiden parametrit voitaisiin tietysti sisällyttää johonkin tietotyyppiin, joka sitten kulkisi parametrina rajapinnoissa. Tämä ei kuitenkaan ole järkevää jo yksistään siitä syystä, että uusien QoS-tekniikoiden tullessa tietotyyppejä täytyisi jatkuvasti päivittää. Rajapintojen parametrit kannattaa määritellä siten, että käytössä olevista QoS-tekniikoista ei tehdä mitään ennakko-oletuksia. Tällaiseen tilanteeseen päästään, kun parametrit välitetään <tyypin nimi, arvo> -pareina, joissa tyypin nimi kertoo mistä parametrista on kyse. Esitystapa voi

olla vaikkapa merkkijono. Arvo taas kertoo, mikä on annetun tyypin nimen mukaisen parametrin arvo. Tällainen parametrien määrittelytapa ei rajoita välitettävien parametrien joukkoa, ehtona toimivuudelle tietysti on, että rajapinnan toteuttaja osaa tulkita ja käsitellä erityyppiset parametrit. Toinen vaihtoehto olisi käyttää esimerkiksi CORBA-standardin Any-tyypin tapaista määrittelyä, jossa rajapinnoissa välitettävä data muunnetaan ennen lähetystä Any-muotoon ja rajapinnan toteuttaja taas purkaa Any-tyypin sisältämän datan ymmärtämäänsä muotoon. Tämä vaatisi erillisen rajapintamäärittelykielen ja kielelle sopivan kääntäjän käyttöä, joka edelleen generoisi tyypinmuunnoksiin vaadittavan koodin. Tuloksena olisi kyllä avoin rajapinta parametrien välityksen suhteen, mutta tämä edellyttäisi tyyppien määrittelyä samalla rajapintamäärittelykielellä ja johtaisi edelleen rajapinnan versioitumiseen uusien QoS-tekniikoiden myötä.

QoS-sovelluskehyksen rajapinnoissa tarvitsee tällaisina geneerisinä parametreina kuljettaa tietoja QoS-mekanismikohtaisista yhteysparametreista ja menettelytavoista erilaisissa poikkeustilanteissa. QoS-yhteysparametreista esimerkkinä voisi käyttää vaikkapa kaistanleveyttä, joka esitettäisiin muodossa <”bandwidth”, ”512”>, jossa arvokenttä tarkoittaisi kaistanleveyttä kilobitteinä. Useita parametreja voisi välittää listamuodossa, jossa on useita em. esimerkin mukaisia parametrien arvoja. Menettelytavat poikkeustilanteissa taas voidaan kuvata esimerkiksi tyyppi-arvoparilla <”conn_break”, ”restore”>, jolla kerrotaan, että yhteyskatkotapauksissa QoS-kirjasto yrittää palauttaa yhteyden automaattisesti katkoksen havaitsemisen jälkeen. Taulukko 1 sisältää edellisen luvun operationaalisista kuvauksista kerätyt menettelytavat poikkeustilanteissa.

Poikkeustilanne	Tunniste	Arvot	Selostus
Yhteys katkeaa	”conn_break”	”restore”, ”wait”	Yhteyskatkotilanteessa toimitaan arvon mukaan, ”restore” yrittää palauttaa yhteyden, ”wait” jää odottamaan sovelluksen kutsuja.
Yhteyden uudelleenneuvottelu epäonnistuu	”renegotiation”	”retry”, ”ignore”	Jos uudelleenneuvottelu epäonnistuu, voi QoS-kirjasto yrittää operaatiota uudestaan tai jättää

			vanhan yhteyden avoimeksi.
Datan lähetys epäonnistuu	”data_send”	”retry”, “abort”	Datan lähetyksen epäonnistuessa voidaan lähetystä yrittää uudestaan (“retry”) tai tulkita tilanne epäonnistuneeksi heti virheen havaitsemisen jälkeen (”abort”).
Yhteyden avaus epäonnistuu	”open_fails”	”abort”, ”best_effort”, “degrade”	Yhteyden avauksen epäonnistuessa QoS-modulissa voidaan operaatio tulkita epäonnistuneeksi (“abort”) tai yhteys voidaan yrittää avata ilman QoS-takeita best effort –tyyppisenä. Lisäksi QoS-vaatimuksia voitaisiin alentaa (”degrade”) ja yrittää yhteyden uudelleenneuvottelua.

Taulukko 1: Menettelytavat poikkeustilanteissa.

Taulukko 2 luettelee QoS-yhteysparametrit, jotka on tallennettu parametripalvelimelle, ja joiden avulla QoS-kirjasto avaa yhteydet. Itse yhteysparametrien lisäksi määritellään myös valittu QoS-tekniikka. Näiden parametrien lisäksi voidaan tarvita myös joukko QoS-menetelmäkohtaisia parametreja.

Parametri	Tunniste	Arvo	Selostus
Vuonleveys	”bandwidth”	”<kbits/s>”	Yhteyden nopeus kilobitteinä
Viive	”delay”	”<µs>”	Datapakettien maksimiviive mikrosekunteina.
Viiveen vaihtelu	”jitter”	”<µs>”	Viiveen vaihtelu maksimissaan mikrosekunteina.
Luotettavuus	”priority”	”high”, ”normal”, ”low”	Yhteyden luotettavuustaso sallitun pakettien hukkumisen suhteen. Arvo ”high” tarkoittaa korkeinta luotettavuutta jossa pakettien hukkumista ei hyväksytä, ”normal” tarkoittaa normaalia luotettavuustasoa jossa paketteja saattaa hukkua jos verkko on ruuhkautunut ja

			”high”-luotettavuustasolla varustettuja paketteja saapuu jonoon. Arvolla ”low” ei vaadita mitään takeita luotettavuuden suhteen, esimerkiksi UDP-protokollan yli välitetyt paketit eivät välttämättä saavu perille koska protokolla ei varmista pakettien perillemenoa.
QoS-tekniikka	”qos_type”	”<QoS-menetelmän nimi>”	Käytettävän QoS-menetelmän nimi, esimerkiksi ”ATM”, ”RSVP” tai ”DiffServ”. Tämän tiedon avulla QoS-moduli tekee päätöksen oikean QoS-kirjaston lataamisesta yhteysparametrien haun yhteydessä.

Taulukko 2: QoS-yhteysparametrit.

6. Sovelluskehyksen malli

6.1 Toteutusvaihtoehtoja

Toteutusvaihtoehdot jakautuvat lähinnä kahteen eri kategoriaan. Toisessa QoS-sovelluskehys toteutetaan erillisinä versioina eri toteutusympäristöihin ohjelmointikieliriippuvaisesti. Siis esimerkiksi Java- ja C++-ohjelmointiympäristöihin toteutettaisiin kaikki rajapintojen määrittelyt vastaavalla kielellä. Tällaiseen vaihtoehtoon päätyminen lisää työmäärää, koska samat rajapinnat täytyy määritellä useaan kertaan. Mahdolliset sovelluskehyksen koodin päivitykset täytyisi myös tehdä jokaiseen rajapintamäärittelyyn erikseen.

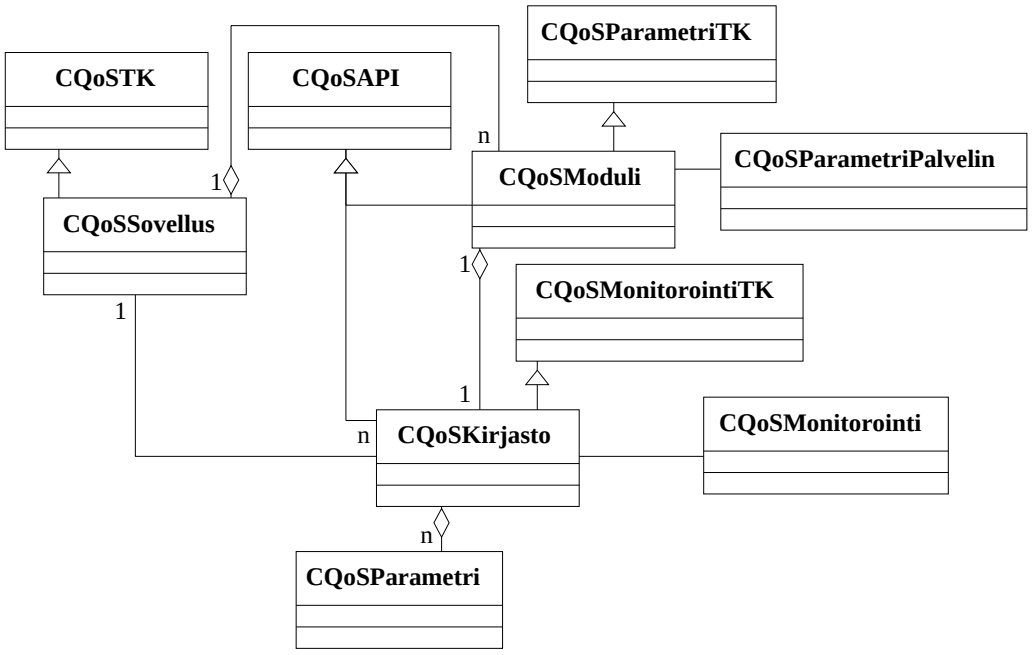
Toisena vaihtoehtona ovat erilaiset komponenttiteknologiat, kuten CORBA (Common Object Request Broker Architecture) tai COM (Component Object Model). CORBA on valmistaja- ja kohdeympäristöriippumaton teknologia, jonka avulla sovellukset voivat kommunikoida toistensa kanssa CORBA IDL-kielellä määriteltyjen rajapintojen avulla. COM taas on Microsoftin® kehittämä teknologia, jonka avulla voidaan rakentaa käyttäjän määrittelemiä rajapintoja toteuttavia komponentteja. Myös COM käyttää IDL-kieltä, COM IDL:ää rajapintojen määrittelyyn. CORBA on teknologioista

laajemmin tuettu, ja se soveltuu näin paremmin laajoihin järjestelmiin ja isoihin verkkoihin, COM taas sopii paremmin samalla koneella tai pienessä verkossa käytettävien ohjelmistokomponenttien rakentamiseen, koska sen skaalautuvuus ja hallinta on CORBA:aa hankalampaa. Molempien teknologioiden avulla voidaan rakentaa ohjelmointikieliriippumattomia rajapintojen määrittelyjä, koska määrittelykieli on geneerinen. Ohjelmointia varten rajapinnat käännetään erityisellä IDL-kääntäjällä, joka generoi rajapinnoista määrittelyt ja tarvittavat valmiit ohjelmamodulit halutulle ohjelmointikielelle. Yleisimmin molemmissa teknologioissa käytetään C++- ja Java-kieliä itse sovellusten toteutukseen.

6.2 Esimerkki toteutuksesta

Esimerkki QoS-sovelluskehyksen toteutuksesta asiakassovelluksen puolella tehdään tässä C++-kieltä käyttäen. Koodin avulla on lähinnä tarkoitus hahmottaa, miten rajapintojen määrittelyt tehtäisiin, ja miten järjestelmän eri osat liittyvät toisiinsa, mitään toimivaa prototyyppiä ei tuoteta. Rajapintojen määrittely tehdään virtuaalisia metodeja sisältävien luokkien avulla. Määrittelyissä käytetään apuna STL-kirjastoa (Standard Template Library) datajäsenien esittelyssä ja käsittelyssä.

6.2.1 Luokkakaavio



Kuva 15: QoS-sovelluskehyksen luokkakaavio

Yllä oleva luokkakaavio kuvaa QoS-sovelluskehyksen osat ja niiden suhteet toisiinsa. Sovellus koostuu seuraavista luokista: *CQoSSovellus*-luokka kuvaa itse sovellusta ja sen toimintalogiikkaa, *CQoSAPI* (API, Application Programming Interface) on luokka, joka sisältää datan lähetykseen ja vastaanottoon liittyvät rajapintamäärittelyt, *CQoSModuli* on aiemmissa luvuissa mainittu QoS-moduli, joka hoitaa kommunikoinnin mm. QoS-parametripalvelimen kanssa sekä ohjaa lähetettävän datan QoS-parametrien mukaan ladatulle QoS-toteutukselle, *CQoSKirjasto* on QoS-spesifisen koodin sisältämä luokka, joka hoitaa QoS-tekniikkakohtaiset yhteyksien avaukset, uudelleenneuvottelut, sekä datan lähetyksen ja vastaanoton, *CQoSTK* on takaisinkutsurajapinnan toteuttava luokka, jonka avulla informoidaan sovellusta yhteyden katkeamisista ja saapuneesta datasta, *CQoSParametriTK* ja *CQoSMonitorointiTK*-luokat toteuttavat takaisinkutsurajapinnan parametrien hallintaa ja QoS-resurssien monitorointia varten. Sovelluksen ulkopuolelle jäävät *CQoSParametriPalvelu*-luokka, joka varastoi QoS-parametrit, ja *CQoSMonitor*-luokka, joka hoitaa tarvittaessa QoS-resurssien monitoroinnin ja palvelutason muutoksista tiedottamisen sovellukselle. Nämä kaksi viimeistä luokkaa saavat vastaavien takaisinkutsutoteutusten viitteet CQoSModuli-luokan instanssilta.

CQoSSovellus-luokan datajäsenenä esitellään *CQoSModuli*-luokan instanssi ja tämän jäsenen alustuksessa kutsutaan alustusmetodia, jossa haetaan QoS-yhteysparametrit parametripalvelimelta. Parametrien hakemisen jälkeen *CQoSModuli* tietää minkä QoS-toteutuksen tarjoavan *CQoSKirjasto*-luokan instanssin sen tulee ottaa käyttöön, tässä esimerkissä *CQoSKirjasto*-luokasta on vain yksi instanssi, jonka tehtävänä on toteuttaa eri QoS-tekniikat. Tarkoitukseen sopivan *CQoSKirjasto*-luokan valinnan jälkeen *CQoSSovellus* välittää perimäänsä *CQoSTK*-luokkaan viittaavan osoittimen *CQoSModuli*-luokalle ja edelleen *CQoSKirjasto*-luokalle yhteyden avausvaiheessa. *CQoSKirjasto*-luokan instanssi tallettaa *CQoSTK*-tyyppisen osoittimen ja pystyy tämän osoittimen avulla välittämään tarvittavia tietoja sovellukselle yhteyden tilasta tai saapuneesta datasta.

6.2.2 Luokkien määrittelyt

CQoSTK-luokka toteuttaa takaisinkutsurajapinnan, jolla *CQoSKirjasto*-luokan instanssi pystyy välittämään sovellukselle tietoa saapuneesta datasta tai yhteyden tilasta. *CQoSSovellus*-luokka perii tämän luokan ja saa tätä kautta käyttöönsä tarvittavat takaisinkutsumetodit. *CQoSTK*-luokan metodit on määritelty puhtaina virtuaalisina, jolloin niiden toteutus tehdään perivässä luokassa.

```
class CQoSTK {
public:
  // yhteyden katkeaminen
  virtual int YhteysSuljettu(std::string szSulkemisenSyy) = 0;
  // yhteyden palautuminen
  virtual int YhteysPalautui() = 0;
  // datan saapuminen
  virtual int DatanKasittely(BYTE *pData, long lDatanMaara) = 0;
  // datan maksimilähetysnopeuden muutospyyntö
  virtual int MuutaMaksimiNopeutta(long lMaksimiNopeus) = 0;
};
```

CQoSSovellus-luokka sisältää sovelluksen toimintalogiikan ja datajäseninään tarvittavat QoS-sovelluskehyksen luokkien instanssit. *CQoSSovellus*-luokka perii *CQoSTK*-luokan ja toteuttaa näin sen määrittelemät takaisinkutsurajapinnat. Oheisessa luokan esittelyssä ei määritellä sen tarkemmin mahdollisia sovelluksen omia metodeja, vaan pelkästään yhteyksien käsittelyyn ja käyttöön liittyvät metodit.

```
class CQoSSovellus : public CQoSTK {
public:
  CQoSSovellus() {
    // datajäsenten luonti
    m_pQoSModuli = new CQoSModuli();
    m_pQoSTK = new CQoSTK();
    // sovelluskohtaiset alustukset
    ....
  }
  ~CQoSSovellus() {
    if ( m_pQoSModuli ) {
      delete m_pQoSModuli; m_pQosModuli = 0;
    }
  }

  // CQoSTK-luokan perittyjen metodien toteutus
  // yhteyden katkeaminen
  int YhteysSuljettu(std::string szSulkemisenSyy) {
    // tiedotetaan käyttäjälle yhteyden katkeamisesta
    cout << ”Yhteys suljettu, syy: ” << szSulkemisenSyy << endl;
  }
  // yhteyden palautuminen
  int YhteysPalautui() {
    // ilmoitetaan käyttäjälle yhteyden palautumisesta
    cout << ”Yhteys palautui” << endl;
```

```
  }
  // datan saapuminen
  int DatanKasittely(BYTE *pData, long lDatanMaara) {
    // sovelluskohtainen datan käsittely
  }
  // datan maksimilähetysnopeuden muutospyyntö
  int MuutaMaksimiNopeutta(long lMaksimiNopeus) {
    // sovelluskohtainen datan lähetysnopeuden muutos
  }
  // loput sovelluksen metodit...
private:
  CQoSModuli *m_pQoSModuli;
};
```

CQoSParametri-luokka sisältää yksittäisen QoS-parametrin tiedot <tyyppi, arvo> -pareina. Tämän luokan instansseja voi olla vaihteleva määrä *CQoSKirjasto*-luokan datajäseninä, määrä riippuu suoraan QoS-menetelmästä ja käytettävistä parametreista. Luokka määritellään seuraavasti.

```
class CQoSParametri {
public:
  // konstruktori ja destruktori
  CQoSParametri() {
    m_szParametriTyyppi = m_szParametri = NULL;
  }
  ~CQoSParametri() {
    if ( m_szParametriTyyppi ) {
      delete m_szParametriTyyppi; m_szParametriTyyppi = NULL;
    }
    if ( m_szParametri ) {
      delete m_szParametri; m_szParametri = NULL;
    }
  }
  // datan hakumetodit
  std::string HaeParametriTyyppi() {
    return m_pszParametriTyyppi;
  }
  std::string HaeParametri() { return m_szParametri; }
  // datan asetusmetodit
  void AsetaParametriTyyppi(std::string szParametriTyyppi) {
    m_szParametriTyyppi = szParametriTyyppi;
  }
  void AsetaParametri(std::string szParametri) {
    m_szParametri = szParametri;
  }
private:
  // datajäseninä parametrin tyyppi ja parametrin arvo
  std::string m_szParametriTyyppi;
  std::string m_szParametri;
};
```

Sovellukselle tarjottavat metodit, jotka toteuttavat yhteyksien hallinnan ja datan lähetyksen, esitellään *CQoSAPI*-rajapintaluokassa, joka peritään *CQoSModuli*- ja *CQoSKirjasto*-luokissa. *CQoSModuli*- ja *CQoSKirjasto*-luokkien periessä tämän luokan, mahdollistuu samalla operaatioiden kierrättäminen

CQoSModuli-luokan kautta *CQoSKirjasto*-luokalle. Tämän luokan tarjoaman rajapinnan avulla siis sovellus avaa ja sulkee yhteydet, ja lähettää dataa synkronisesti tai asynkronisesti.

```
class CQoSAPI {
public:
  // yhteyden avaus
  virtual int AvaaYhteys(CQoSTK *pTK) = 0;
  // yhteyden sulkeminen
  virtual int SuljeYhteys() = 0;
  // synkroninen datan lähetys
  virtual int SynDatanLahetys(BYTE *pData, long lDatanMaara,
                  long lMaksimiOdotusAika,
                  BYTE *pVastausData,
                  long lVastauksenPituus) = 0;
  // asynkroninen datan lähetys
  virtual int AsynDatanLahetys(BYTE *pData, long lDatanMaara) = 0;
  // palvelimen tiedottamiseen käytettävä sanoma jos datan
  // käsittely takaisinkutsuobjektissa epäonnistuu
  virtual int LopetusSanoma(BYTE *pSanoma, long lDatanMaara) = 0;
};
```

QoS-parametripalvelin ilmoittaa sovellukselle parametrien muuttumisesta *CQoSParametriTK*-takaisinkutsurajapinnan välityksellä. Rajapintaluokan määrittely pitää sisällään metodin, jonka avulla muuttuneet parametrit välitetään parametripalvelimelta sovelluskehykselle. Metodin parametreina on sovelluksen tunniste ja lista uusista yhteysparametreista.

```
class CQoSParametriTK {
public:
  virtual int PaivitaParametrit(std::string szSovellusTunnus,
               std::vector<CQoSParametri*> vQoSParametrit) = 0;
};
```

CQoSModuli-luokka on keskeisessä osassa QoS-sovelluskehystä, koska se hoitaa yhteydet QoS-parametripalvelimeen ja tekee lisäksi päätöksen tarvittavan *CQoSKirjasto*-luokan instanssin alustamisesta. Luokka perii kaksi rajapintaluokkaa, *CQoSAPI*- ja *CQoSParametriTK*-luokat. Ensimmäisen avulla sovellus avaa yhteyden ja lähettää dataa, toinen on takaisinkutsurajapinnan toteutusta varten, eli sillä saadaan tietoa muuttuneista parametreista sovelluksen ollessa käynnissä.

```
class CQoSModuli : public CQoSAPI, CQoSParametriTK {
public:
  CQoSModuli() { m_szSovellusTunnus = m_pQoSKirjasto = NULL; }
  ~CQoSModuli();
  int HaeYhteysParametrit(std::string szKohdeOsoite,
                          std::string szSovellusTunnus) {
    std::vector<CQoSParametri*> vQoSParametrit;
    // haetaan QoS-parametripalvelun osoite, oletetaan että
```

```
    // osoite on staattisessa muuttujassa
    if(s_pQoSParametriPalvelu->HaeParametrit(szSovellusTunnus,
                                szKohdeOsoite,
                                &vQoSParametrit)) {
      // parametrien haku onnistui, alustetaan QoS-kirjasto
      m_pQoSKirjasto = new CQoSKirjasto(vQoSParametrit);
      m_szSovellusTunnus = strdup(szSovellusTunnus);
    }
    else
      return -1;
    return 0;
  }

  bool HaeYhteydenTila() {
    bool bPaluuarvo = false;
    if ( m_pQoSLib )
      bPaluuarvo = m_pQoSLib->YhteydenTila();
    return bPaluuarvo;
  }
  // perittyjen metodien toteutukset
  // CQoSAPI - kutsut välitetään CQoSKirjasto-luokalle
  // yhteyden avaus
  int AvaaYhteys(CQoSTK *pTK) {
    return m_pQoSKirjasto->AvaaYhteys(pTK);
  }
  // yhteyden sulkeminen
  int SuljeYhteys() {
    return m_pQoSKirjasto->SuljeYhteys();
  }
  // synkroninen datan lähetys
  int SynDatanLahetys(BYTE *pData, long lDatanMaara,
                  long lMaksimiOdotusAika,
                  BYTE *pVastausData,
                  long lVastausDatanMaara) {
    return m_pQoSKirjasto->SynDatanLahetys(pData, lDatanMaara,
                         lMaksimiOdotusAika, pVastausData,
                         lVastausDatanMaara);
  }
  int AsynDatanLahetys(BYTE *pData, long lDatanMaara) {
    return m_pQoSKirjasto->AsynDatanLahetys(pData, lDatanMaara);
  }

  // CQoSParametriTK
  int PaivitaParametrit(std::string szSovellusTunnus,
              std::vector<CQoSParametri*> vQoSParametrit) {
    // tarkistetaan että parametrit ovat tälle sovellukselle
    if ( strcmp(szSovellusTunnus, m_szSovellusTunnus) == 0 ) {
      // lähetetään uudet parametrit CQoSKirjasto-luokalle
      m_pQoSKirjasto->PaivitaParametrit(vQoSParametrit);
    }
    else
      return -1;
    return 0;
  }

private:
  CQoSKirjasto *m_pQoSKirjasto;
  std::string m_szSovellusTunnus;
};
```

QoS-monitorointisovellus, joka ei sisälly suoraan käytettyyn QoS-tekniikkaan, välittää verkon muuttuneen resurssitilanteen *CQoSMonitorointiTK*-rajapinnan avulla *CQoSKirjasto*-luokan instanssille. Rajapinnan metodin parametreina välitetään muuttuneet parametrit, eli uudet maksimiarvot esimerkiksi kaistanleveydelle, jonka verkko voi maksimissaan tarjota.

```
class CQoSMonitorointiTK {
public:
  virtual int MuutosVerkossa(
                std::vector<CQoSParametri*> vParametrit) = 0;
};
```

CQoSKirjasto-luokka sisältää QoS-menetelmäkohtaiset toiminnot yhteyksien hallintaa ja datan käsittelyä varten, näihin toimintoihin liittyvät metodit tulevat *CQoSAPI*-luokan perinnän kautta. *CQoSKirjasto*-luokan datajäsenenä ovat QoS-parametrit, jotka on haettu QoS-parametripalvelimelta *CQoSModuli*-luokan instanssin alustuksessa. Luokka tarjoaa myös metodin, jonka avulla voidaan päivittää tarpeen vaatiessa uudet yhteysparametrit sovelluksen ollessa käynnissä. Luokka perii myös *CQoSMonitorointiTK*-luokan, jonka metodien avulla saadaan tietoa verkon resurssitilanteen muutoksista, jos käytössä oleva QoS-menetelmä ei sisällä tällaista toiminnallisuutta. Tässä luokan määrittelyssä *CQoSKirjasto*-luokan käyttämänä yhteysprotokollana on ATM. Esimerkkikoodi perustuu Linux ATM API:n [ATM API] määrittelyihin ja mallikoodeihin.

```
class CQoSKirjasto : public CQoSAPI, CQoSMonitorointiTK {
public:
  CQoSKirjasto();
  CQoSKirjasto(std::vector<CQoSParametri*> vQoSParams);
  ~CQoSKirjasto();
  // QoS-yhteyden avaus
  int AvaaYhteys(CQoSTK *pQoSTK)
  {
    // tarkistetaan ja talletetaan callback-viite
    if (pQoSTK)
      m_pQoSTK = pQoSTK;

    // yhteysprotokollana on ATM, alustetaan yhteys...
    int atm_protokolla = FindParam(“protocol”);
    if ((s = socket(PF_ATMPVC,SOCK_DGRAM,atm_protokolla)) < 0)
    {
      return –1;
    }
    atm_qos qos;
    memset(&qos,0,sizeof(qos));
    // asetetaan lähetys- ja vastaanottoparametrit
    qos.txtp.class = HaeParametri(“traffic_class”);
    qos.txtp.min_pcr = HaeParametri(“min_pcr”);
```

```
    qos.rxtp = qos.txtp;
    if(setsockopt(m_socket,SOL_ATM,SO_ATMQOS,&qos,sizeof(qos))<0)
    {
      return -1;
    }
    // avataan yhteys...
    if (connect(m_socket,(struct sockaddr *) &addr,
            sizeof(addr)) < 0)
    {
      return -1;
    }
    return 0;
  }
  // yhteyden sulkeminen
  int SuljeYhteys() {
    if ( m_socket ) {
      (void)close(m_socket);
      m_socket = 0;
    }
    else
      return –1;
    return 0;
  }
  // synkroninen datan lähetys
  int SynDatanLahetys(BYTE *pData, long lDatanMaara,
                  long lMaksimiOdotusAika,
                  BYTE *pVastausData,
                  long lVastauksenPituus) {
    // ...koodi on muokattu Linux ATM API:n esimerkistä...
    char *buffer,*start;
    struct atm_buffconst bc;
    ptrdiff_t pos;
    size_t length,buf_len;
    ssize_t size;

    length = sizeof(bc);
    if (getsockopt(m_socket,SOL_SOCKET,SO_BCTXOPT,(char *) &bc,
          &length) < 0)
      return -1;
    buf_len = lDatanMaara-bc.size_off+bc.size_fac-1;
    buf_len = buf_len-(buf_len % bc.size_fac)+bc.size_off;
    if (buf_len < bc.min_size) buf_len = bc.min_size;
    if (!(buffer = malloc(buf_len+bc.size_fac-1))) {
      return –1;
    }
    pos = (ptrdiff_t) (buffer-bc.buf_off+bc.buf_fac-1);
    start = (char *) (pos-(pos % bc.buf_fac)+bc.buf_off);
    if (lDatanMaara != buf_len)
      memset(start+lDatanMaara,0,buf_len-lDatanMaara);
    if ((size = write(m_socket,start,buf_len)) < 0)
      return –1;
    if (size != buf_len)
      return –1;
    // odotetaan vastausta... normaali ATM API:n read-operaatio
    // on blokkaava kunnes dataa saapuu, joten tässä käytetään
    // read_timed-operaatiota, jossa aika on määritelty
    // parametrina (toteutus voisi olla vaikkapa erillisen
    // säikeen avulla, joka tapetaan jos aikaraja täyttyy)
    return read_timed(m_socket, pVastausData, lVastauksenPituus,
                  lMaksimiOdotusAika);
  }
  // asynkroninen datan lähetys
  int AsynDatanLahetys(BYTE *pData, long lDatanMaara) {
     // toteutus sama kuin SynDatanLahetys-metodissa, mutta
```

```
     // viimeinen operaatiokutsu jätetään tekemättä...
     return 0;
  }
  // QoS-parametrien päivitys
  int PaivitaParametrit(std::vector<CQoSParametri*> vParams) {
    // päivitetään QoS-parametrit...
    atm_qos qos;
    memset(&qos,0,sizeof(qos));
    // asetetaan lähetys- ja vastaanottoparametrit
    qos.txtp.class = HaeParametri("traffic_class");
    qos.txtp.min_pcr = HaeParametri("min_pcr");
    qos.rxtp = qos.txtp;
    if(setsockopt(m_socket,SOL_ATM,SO_ATMQOS,&qos,sizeof(qos))<0)
    {
      return -1;
    }
    return 0;
  }

private:
  // hae parametri merkkijonomuodossa
  std::string HaeParametri(std::string szHakuAvain) {
    std::string szPaluuarvo = "";
    for ( int i = 0; i < m_vQoSParametrit.size(); i++ ) {
      if ( m_vQoSParametrit[i]->HaeParametriTyyppi() ==
            szHakuAvain )
        return m_vQoSParametrit[i]->HaeParametri();
    }
    return szPaluuarvo; // parametria ei löytynyt
  }
  // hae parametri lukumuodossa
  int HaeParametri(std::string szHakuAvain) {
    for ( int i = 0; i < m_vQoSParametrit.size(); i++ ) {
      if ( m_vQoSParametrit[i]->HaeParametriTyyppi() ==
            szHakuAvain )
        return atoi((char*)m_vQoSParametrit[i]->HaeParametri());
    }
    return –1; // parametria ei löytynyt
  }
  // yhteyteen liittyvät muuttujat
  int m_socket;
  CQoSTK *m_pQoSTK;
  std::vector<CQoSParametri*> m_vQoSParametrit;
};
```

CQoSParametriPalvelu-luokka kuvaa QoS-parametripalvelimen rakennetta, se tarjoaa metodit parametrien hakemiseksi, tietyn QoS-toteutuksen (*CQoSKirjasto*-luokan alustuksen ja palautuksen), sekä QoS-parametrien hallintaa varten parametrien lisäysmetodin ja QoS-applikaatiolistan hakemisen.

```
class CQoSParametriPalvelu {
public:
  // konstruktori ja destruktori
  CQoSParametriPalvelu();
  ~CQoSParametriPalvelu();

  // QoS-parametrien haku sovellukselle
  int HaeParametrit(std::string szSovellusTunnus,
          std::string szOsoite,
```

```
          std::vector<CQoSParametri*> vQoSParametrit);
  // QoS-kirjaston toteutus palauttaminen
  int HaeQoSKirjasto(std::string szQoSMetodi,
              CQoSKirjasto *pKirjasto);

  // lisää uudet QoS-parametrit sovellukselle
  int LisaaParametrit(std::string szSovellusTunnus,
          std::string szOsoite,
          std::vector<CQoSParametri*> vUudetParametrit) {
    bool bHakuTulos = false;
    for ( int i = 0; i < m_vQoSSovellusParametrit && !bHakuTulos;
           i++ ) {
      if ( m_vQoSSovellusParametrit[i].szSovellusTunnus
  // strukti sovellusten parametrien tallennusta varten
  struct SovellusData {
    std::string szSovellusTunnus;
    std::string szKohdeOsoite;
    std::vector<CQoSParametri*> vQosParametrit;
  };
  // datakysely QoS-hallintasovellukselle
  int HaeSovellustiedot(std::vector<SovellusData>*
                  pvSovellusData) {
    *pvSovellusData = m_vQoSSovellusParametrit;
    return m_vQoSSovellusParametrit.size();
  }
private:
  std::vector<SovellusData> m_vQoSSovellusParametrit;
};
```

CQoSMonitorointi-luokka tarjoaa *CQoSModuli*-luokalle tiettyjen QoS-resurssien valvonnan tilauspalvelun. Palvelun avulla *CQoSModuli* antaa osoittimen toteuttamaansa *CQoSMonitorointiTK*-rajapintaan, jonka avulla *CQoSMonitorointi*-luokka voi tiedottaa *CQoSKirjasto*-luokan instansseja resursseissa tapahtuneista muutoksista. Tilattavat muutokset ovat hyvin yleisluontoisia, koska QoS-metodeilla (esimerkiksi ATM tai RSVP) on yleensä omat mekanismit resurssien valvontaan. Tällä toiminnolla palvellaankin lähinnä ei-yhteydellisiä mekanismeja, jotka eivät perustu yksittäisten yhteyksien luomiseen, vaan lähinnä tarjoavat eri liikenneluokille erilaisia reitityspalveluja, joiden avulla datapaketit käsitellään eri kiireellisyydellä. Valvontatulosten perusteella sovelluksia voidaan *CQoSKirjasto*-luokan avulla opastaa vähentämään tai kasvattamaan lähettämänsä datan määrää.

```
class CQoSMonitorointi {
public:
  CQoSMonitorointi();
  ~CQoSMonitorointi();

  struct ValvontaKohde {
    std::string m_szMonitorointiKohde;
    CQoSMonitorointiTK *m_pMonitorointiTK;
    int nAvain;
  };
```

```
  // valvonnan tilaus halutulle kohteelle
  int TilaaValvonta(std::string szMonitorointiKohde,
                    CQoSMonitorointiTK *pMonitorointiTK,
                    int *pnAvain) {
    // talletetaan valvottava kohde ja palautetaan avain jota
    // käyttäen valvonta voidaan lopettaa
    ValvontaKohde *pVK = new ValvontaKohde;
    pVK->m_szMonitorointiKohde = szMonitorointiKohde;
    pVK->m_pMonitorointiTK = pMonitorointiTK;
    *pnAvain = pVK->nAvain = m_vValvontaKohteet.size()+1;
    m_vValvontaKohteet.push_back(pVK);
    return 0;
  }
  // valvonnan peruutus
  int PeruutaValvonta(int nAvain) {
    for ( int i = 0; i < m_vMonitorointiKohteet.size(); i++ )
      if ( m_vMonitorointiKohteet[i]->nAvain == nAvain ) {
        delete m_vMonitorointiKohteet[i];
        m_vMonitorointiKohteet.erase(
                          m_vMonitorointiKohteet.begin()+i);
        break;
      }
    return 0;
  }
private:
  std::vector<ValvontaKohde*> m_vValvontaKohteet;
};
```

6.3 Sovelluskehyksen sovellusalueet

Sovelluskehyksen koostuessa eri koneille ripotelluista toiminnoista, kuten QoS-parametripalvelimesta ja mahdollisesta QoS-monitoroinnista, kohdeympäristönä olisivat lähinnä isot verkot, joissa on paljon käyttäjiä ja joissa resurssien määrää täytyy hallita käyttäjämäärien ja verkon hetkittäisen muun kuormituksen mukaan. Tällaisissa ympäristöissä keskitetty parametrien hallinta ja jakelu toimii tehokkaasti, koska useiden asiakaskoneiden konfigurointi erikseen lisää vaivaa, virhemahdollisuuksia ja näiden kautta kustannuksia. Yhtenä esimerkkinä tällaisesta kohdeympäristöstä voisi olla jotakin QoS-mekanismia hyödyntävä yritysverkko, jossa työntekijöiden välinen kommunikointi tehdään videoneuvottelusovellusten avulla. Yritysten työntekijöiden ollessa eri paikkakunnilla, mahdollisesti jopa eri maissa, olisi tarpeen pystyä konfiguroimaan järjestelmää keskitetysti, jotta kaikilla neuvotteluun osallistuvilla työntekijöillä olisi tarpeelliset resurssit käytössään videoneuvottelusovelluksien käyttöä varten.

Erillisine palvelimineen varustettuna tällainen järjestelmä ei vaikuttaisi soveltuvan pieniin ympäristöihin. Pienissä ympäristöissä konfiguroinnin tarve

on myös pieni, jolloin tultaisiin toimeen staattisemmalla konfiguraatiolla, jossa sovelluksella on tarkoitukseen sopiva QoS-kirjasto käytössään ja tällä hoidetaan tarvittavat yhteyksien hallinnat. Turhan raskas järjestelmä ei ole tarkoituksenmukainen myöskään mobiileissa ympäristöissä, joissa järjestelmän käyttäjät eivät sijaitse aina samassa paikassa vaan saattavat tarvita yhteyksiä paikoista, joista esimerkiksi parametripalvelimelle ei ole pääsyä. Tällaisiin ympäristöihin voitaisiin rakentaa tästä QoS-sovelluskehyksestä riisuttu versio, jossa QoS-sovelluskehyksen asiakkaan koneella sijaitsevat osat osaisivat tunnistaa verkossa käytettävän QoS-menettelyn ja tämän perusteella valita sopivan QoS-kirjaston sovellusten kommunikointia varten. Sopivien QoS-yhteysparametrien valinnassa täytyisi noudattaa jotain yhteistä sopimusta, koska keskitettyä parametripalvelua ei olisi käytössä. Tämä sen vuoksi, että sovellukset saattaisivat varata liikaa resursseja ja näin toisten käyttäjien resurssien varaukset voisivat estyä.

7. Yhteenveto ja loppusanat

Tässä tutkielmassa on esitelty eri QoS-menetelmiä ja niiden ominaisuuksia. Näiden menetelmien ominaisuuksien perusteella on pohdittu geneeristä QoS-sovelluskehyksen mallia, jonka avulla QoS-menetelmät voitaisiin erottaa sovelluslogiikasta siten, että itse QoS-menetelmäkohtainen toiminta voitaisiin valita ajoaikaisesti kohdeympäristössä mahdollisesti olevan QoS-toteutuksen mukaan. Geneerisyydellä voitaisiin myös päästä tilanteeseen, jossa samassa toimintaympäristössä sovellukset voisivat tukea samanaikaisesti useita erilaisia QoS-tekniikoita. Tällaisella QoS-sovelluskehyksellä mahdollistetaan sovellusten siirrettävyys ympäristöstä toiseen, ja tätä kautta sovelluksesta tehtävien eri binääriversioiden määrää voitaisiin vähentää. Sovellusta hankkivan asiakkaan hankintakulut ehkä pienenevät, koska sovellus ei pakottaisi käyttämään mitään tiettyä QoS-menetelmää.

Sovelluskehyksen eri osien ja rajapintojen vaatimusten ja tarvittavien ominaisuuksien tunnistaminen tehtiin käyttötapaus–menetelmän pohjalta. Käyttötapauksissa pohdittiin mitä esiehtoja ja askelia kuhunkin toiminnallisuuteen liittyy, ja näiden askelien perusteella saatiin kasattua eri

toiminnallisuuksiin kohdistuvat vaatimukset ja eri QoS-sovelluskehyksen osien välillä tarvittavat rajapinnat. Rajapintojen tunnistamisen jälkeen hahmoteltiin geneerinen malli jonka avulla erityyppiset parametrit saadaan välitettyä QoS-sovelluskehyksen eri osien välillä kohdeympäristöstä tai parametrien muodosta riippumatta. Lopuksi esiteltiin luokkakaavio järjestelmän eri osista, ja hahmoteltiin sovelluskehyksen esimerkkikoodia C++ -kieltä käyttäen.

Normaalisti sovelluskehysten tekeminen perustuu aiemmin valmistuneihin projekteihin, joissa käytetyt tekniikat ja ohjeistus ovat käytännössä havaittu toimiviksi. Seuraavien ohjelmointiprojektien helpottamiseksi voidaan aiemmin tehdystä aineistosta tunnistaa yhteiset osat ja tehdä niiden perusteella yleiset rakennuspalikat uusien sovellusten rakentamista varten. Tässä tutkielmassa tuotettu sovelluskehyksen malli ei kuitenkaan perustu mihinkään aiempaan tehtyyn sovellukseen, vaan se perustuu eri QoS-tekniikoiden ominaisuuksiin ja sovellusten oletettuihin tarpeisiin QoS-järjestelmissä. Tuotetun sovelluskehyksen toimivuutta ei siis voida todentaa todellisessa ympäristössä, eikä sen mahdollisia saavutettuja hyötyjä tai ilmeisiä puutteita voida selkeästi osoittaa. Esimerkkikoodin perusteella voidaan kuitenkin päätellä, että QoS-menetelmiä hyödyntävien sovellusten rakentaminen yksinkertaistuisi, kun kaikki QoS-menetelmäkohtainen koodi voitaisiin kapseloida omaan moduliinsa. Lisäksi QoS-yhteyksien käsittelyyn liittyy useita erityistilanteita, jotka voidaan myös kapseloida sovelluskehyksen sisään. Esimerkkikoodissa käytettiin QoS-menetelmänä ATM-protokollaa ja QoS-kirjastoon kasatun ohjelmakoodin määrän perusteella nähdään, että sovelluskoodin tulisi sisältää hyvinkin monimutkaisia QoS-menetelmäkohtaisia parametrien käsittelyjä. Toki sovelluksessa nämä operaatiot voitaisiin funktioida ja tätä kautta yksinkertaistaa sovelluksen koodia, mutta samalla menetettäisiin modulaarisuus, jonka tämä sovelluskehys voi tarjota.

Ongelmana QoS-sovelluskehyksen rakentamisessa tuotantokäyttöön on saada sovellusten kehittäjien kesken konsensus sovelluskehyksen arkkitehtuurista ja eri sovelluskehyksen välisten osien kommunikoinnista. Maailmanlaajuiseen yhteisymmärrykseen on ehkä mahdotonta päästä, mutta

erilaisiin QoS-ympäristöihin sovelluksia tuottava yritys pystyisi käyttämään tällaista sovelluskehystä hyväkseen.

Katson saavuttaneeni johdantoluvussa asettamani tavoitteet tälle tutkielmalle. Hahmottelemaani QoS-sovelluskehystä ja sen rajapintoja voidaan käyttää sovellusten vaatimien QoS-menetelmien parametrien hallintaan ja jakeluun. Täytyy kuitenkin huomioida se, että esittelemäni sovelluskehys on vasta ensimmäinen hahmotelma. Seuraava askel QoS-sovelluskehyksen kehityksessä olisi tehdä sille varsinainen toteutus ja tätä käyttäen toteuttaa QoS-tekniikoita tarvitseva sovellus jossakin toteutusympäristössä. Esittelemääni sovelluskehystä ja sen toimivuutta ei siis sellaisenaan vielä voida kriittisesti arvioida pelkän rajapintojen ja järjestelmän osien luetteloinnin avulla. QoS-sovelluskehyksen käytännön toteutusta ja sen kautta tehtävää todellista toimivuuden arviointia ei sisällytetä tähän tutkielmaan, vaan se jätetään mahdolliseksi jatkotutkimusaiheeksi.

Viiteluettelo

[ATM API] Linux ATM API: Documents. http://lrcwww.epfl.ch/linux-atm/doc.html, 2000.

[ATM Forum] The ATM Forum, http://www.atmforum.org/, 2003.

[Becker ja Geihs, 2000] Christian Becker, Kurt Geihs, Generic QoS-support for CORBA. Fifth IEEE Symposium on Computers and Communications (ISCC 2000), 2000, 60-65.

[Black, 1999] Darryl P. Black, Building Switched Networks, 1999.

[Bragg, 1999] Arnold W. Bragg, Quality of Service: Old Idea, New Options. IT Professional, September/October 1999, 38-44.

[Fayed et al, 1999] Mohamed E. Fayad, Douglas E. Schmidt, Ralph E. Johnson, Building Application Frameworks. John Wiley & Sons, 1st edition, 1999.

[Kaario, 2002] Kimmo Kaario, TCP/IP-verkot. Docendo Finland Oy, 2002.

[Karr et al., 2001] David A. Karr, Craig Rodrigues, Yamuna Krishnamurthy, Irfan Pyarali, Joseph P. Loyall, Richard E. Schantz, Application of the QuO Quality-of-Service Framework to a Distributed Video Application. Proceedings of the International Symposium on Distributed Objects and Applications, 2001, 299-310.

[Kilkki, 1999] Kalevi Kilkki, Differentiated Services for the Internet. McMillan Technology Series, 1999.

[Krishnamurthy et al, 2001] Krishnamurthy Y, Kachroo V, Karr DA, Rodrigues C, Loyall JP, Schantz RE, Schmidt DC. Integration of QoS-Enabled Distributed Object Computing Middleware for Developing Next-Generation Distributed Applications. Proceedings of the ACM SIGPLAN Workshop on Optimization of Middleware and Distributed Systems (OM 2001), 2001, 230-237.

[Matsui et al., 1999] Yasunori Matsui, Seiji Kihara, Atsushi Mitsuzawa, Satoshi Moriai, Hideyuki Tokuda, An Extensible Object Model for QoS Specification in Adaptive QoS Systems. Second IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 1999, 129-133.

[Schmidt ja Kuhns, 2000] Douglas C. Schmidt, Fred Kuhns, An Overview of the Real-Time CORBA Specification. Computer, 2000, 56-63.

[Shepard, 2000] Susan J. Shepard, Policy-Based Networks: Hype and Hope. IT Professional, January / February 2000, 12-18.

[Wang et al., 2000] P.Wang, Y.Yemini, D.Florissi, John Zinky, A Distributed Resource Controller for QoS Applications. Proceedings of the 2000 IEEE / IFIP Network Operations and Management Symposium, 2000, 143-156.

Liite 1: Vaatimusten kartoitus, käyttötapaukset

1. Yhteysparametrien haku

Yleinen kuvaus: Yhteysparametrien hakeminen käsittää sovelluskohtaisten asetusten hakemisen keskitetyltä palvelimelta, joka sisältää kaikkien QoS-palveluja käyttävien sovellusten tiedot yhteysparametrien osalta.

Esiehdot: QoS-tekniikkaa käyttävän sovelluksen vaatimat yhteysparametrit on lisätty palvelimelle. Sovellus on kutsunut sovelluskehyksen tarjoamaa yhteyden avaamismetodia.

Osallistujat: QoS-moduli, parametripalvelin.

Toiminnan kuvaus:

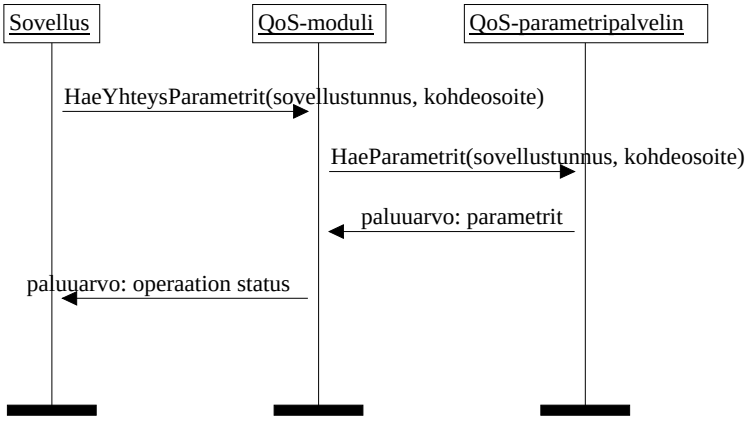
1. QoS-moduli ottaa yhteyden parametritiedot sisältämään palvelimeen, sovelluskohtaisena tunnisteena käytetään esimerkiksi sovelluksen nimeä ja versiotietoa, sekä liikennöinnin kohdeosoitetta.
2. Palvelin hakee tietovarastostaan sovelluskohtaiset yhteysparametrit. Parametrit määrittelevät mitä QoS-menetelmää käytetään, tarvittavat minimi- ja maksimikaistanleveydet sekä viivevaatimukset. Parametrilista voi vaihdella käytettävästä QoS-menettelystä riippuen.
3. QoS-moduli vastaanottaa yhteyden avaamisessa tarvittavat tiedot ja tallettaa yhteysparametrit paikallisesti käytön nopeuttamista varten seuraavien yhteyksien avaamisessa.

Jälkiehdot: Sovelluskohtaiset yhteysparametrit on siirretty parametripalvelimelta asiakaskoneelle QoS-sovelluskehyksen käyttöön.

Poikkeustilanteet:

1. Parametripalvelin ei sisällä tietoja sovellukselle. Parametrit palautetaan best effort –yhteyden mukaan ilman spesifisiä QoS-vaatimuksia.
2. Parametripalvelimeen ei saada yhteyttä. Sovelluskehys käyttää best effort –yhteyttä oletusarvoisesti, ellei sovelluksen tietoja löydy paikallisesti tallennettuna edellisistä yhteyden avauksista.

Operaatiokaavio:



2. Yhteyden avaus

Yleinen kuvaus: Yhteyden avauksessa sovellus käyttää QoS-modulin palvelua yhteyden avaamiseen haluttuun kohdeosoitteeseen. Yhteyden avauksessa käytetään sovellukselle määriteltyä QoS-menettelyä ja yhteysparametreja.

Esiehdot: Ennen yhteyden avausta on suoritettu *Yhteysparametrien haku* – käyttötapauksen kuvaamat toiminnot.

Osallistujat: Yhteyttä avaava sovellus, QoS-moduli, QoS-kirjasto.

Toiminnan kuvaus:

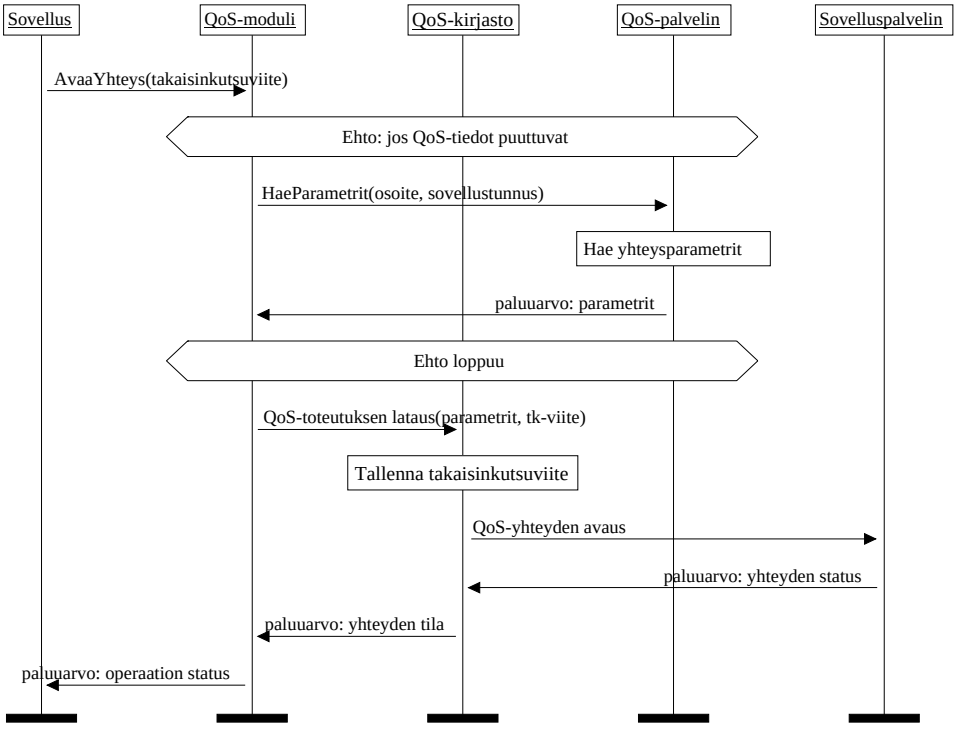
1. Yhteyttä avaava sovellus käyttää QoS-modulin palvelua yhteyden avaamiseen. Koska yhteyden avaamiseen tarvittavat QoS-parametrit ovat QoS-sovelluskehyksen tiedossa, ei palvelussa tarvita erillisiä parametreja yhteyteen liittyen.
2. QoS-moduli tutkii aiemmin haetut yhteysparametrit ja päättelee sen perusteella, mitä QoS-kirjastoa käytetään yhteyden rakentamisessa.
3. QoS-moduli lataa QoS-kirjaston ja käyttää kirjaston tukeman QoS-menettelyn yhteyden avauspalvelua välittämällä tarvittavat yhteysparametrit.
4. Tieto yhteyden avauksen onnistumisesta palautetaan sovellukselle.

Jälkiehdot: Yhteys on avattu sovellukselle määriteltyjä yhteysparametreja käyttäen. Yhteys on valmis käytettäväksi datan lähetykseen ja vastaanottoon.

Poikkeustilanteet:

1. QoS-kirjastoa ei löydy tai sen lataus epäonnistuu. Yhteyden avaus epäonnistuu ja virhestatus palautetaan sovellukselle.
2. Yhteysparametreja noudattavaa yhteyttä ei saada avattua riittämättömien verkkoresurssien vuoksi. Yhteyden avaus epäonnistuu ja virhestatus palautetaan sovellukselle.

Operaatiokaavio:



3. Yhteyden sulkeminen

Yleinen kuvaus: Sovellus suljetaan ja yhteyden varauksessa käytetyt resurssit vapautetaan.

Esiehdot: Yhteys on avattu *Yhteyden avaus* –käyttötapauksen mukaisesti.

Osallistujat: Yhteyden avannut sovellus, QoS-moduli, QoS-kirjasto.

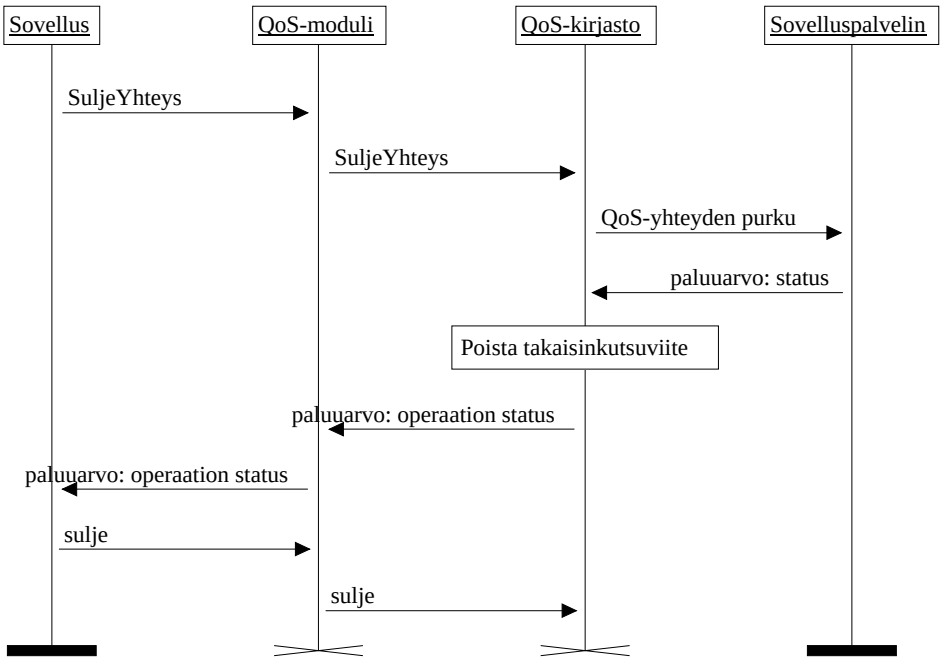
Toiminnan kuvaus:

1. Sovellus käyttää QoS-modulin palvelua yhteyden sulkemiseen.
2. QoS-moduli kutsuu yhteysparametrien mukaan valitun QoS-kirjaston yhteyden purkamistoimintoa.
3. QoS-kirjasto vapauttaa yhteyteen sidotut resurssit.
4. Tieto operaation onnistumisesta palautetaan QoS-modulille ja edelleen sovellukselle.

Jälkiehdot: Yhteyttä varten varatut resurssit on vapautettu.

Poikkeustilanteet: Ei poikkeuksia.

Operaatiokaavio:



4. Datan lähetys synkronisesti

Yleinen kuvaus: Sovellus lähettää dataa kohdeosoitteeseen varatun yhteyden avulla. Operaatio katsotaan päättyneeksi kun vastaus on saatu kohdeosoitteesta.

Esiehdot: Yhteys on avattu *Yhteyden avaus* –käyttötapauksen mukaisesti.

Osallistujat: Yhteyden avannut sovellus, QoS-moduli, QoS-kirjasto.

Toiminnan kuvaus:

1. Sovellus muodostaa lähetettävästä datasta puskurin, joka lähetetään verkon yli vastaanottajalle.
2. Sovellus käyttää QoS-modulin palvelua datan lähettämiseen, parametreina lähetettävä datapuskuri, puskurin koko, sekä aika, jonka verran vastausta maksimissaan odotetaan.
3. QoS-moduli välittää tiedot edelleen QoS-kirjastolle.
4. QoS-kirjasto käyttää varattua yhteyttä datan lähetykseen.
5. Tieto lähetyksen onnistumisesta palautuu QoS-kirjastolle.
6. Jos lähetys onnistui, QoS-sovelluskehys jää odottamaan vastausta kohdeosoitteesta.

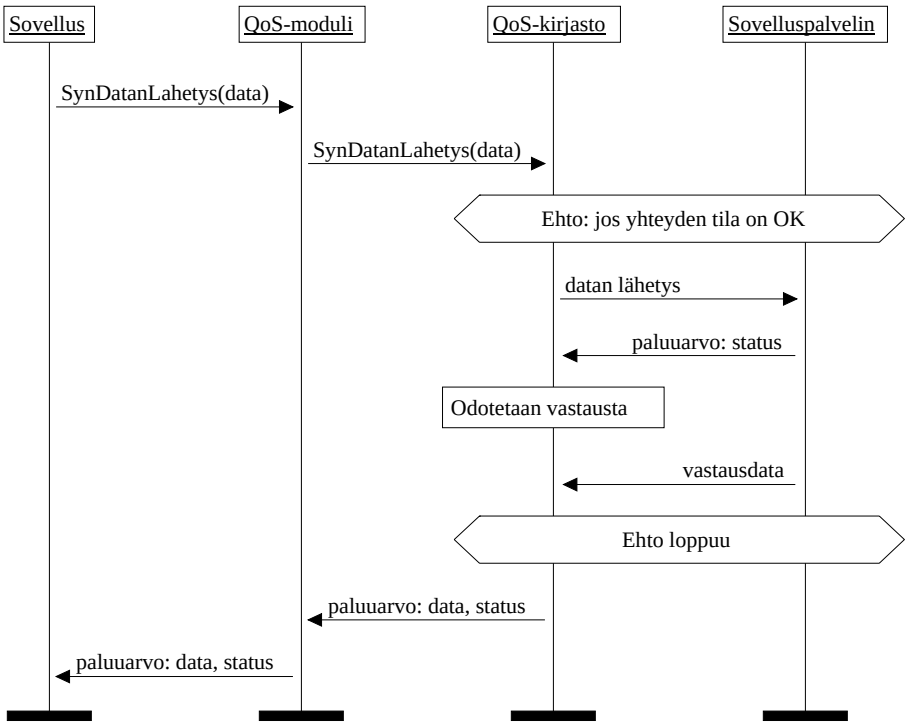
7. Vastauksen saavuttua operaation status ja vastausdata palautetaan sovellukselle.

Jälkiehdot: Sovelluksen muodostama datapuskuri on välitetty vastaanottajalle.

Poikkeustilanteet:

1. Datan lähetys QoS-kirjastossa epäonnistuu. Yhteyden tila tarkistetaan ja lähetystä yritetään uudestaan. Myös uudelleenlähetyksen epäonnistuessa tieto epäonnistuneesta lähetyksestä välitetään QoS-modulille ja edelleen sovellukselle.
2. Vastausta ei saada määritellyn odotusajan aikana. Virhestatus palautetaan sovellukselle jossa edelleen tehdään päätös seuraavista toiminnoista.

Operaatiokaavio:



5. Datan lähetys asynkronisesti

Yleinen kuvaus: Sovellus lähettää dataa kohdeosoitteeseen varatun yhteyden avulla. Operaatio päättyy kun data saadaan lähetettyä onnistuneesti kohdeosoitteeseen.

Esiehdot: Yhteys on avattu *Yhteyden avaus* –käyttötapauksen mukaisesti.

Osallistujat: Yhteyden avannut sovellus, QoS-moduli, QoS-kirjasto.

Toiminnan kuvaus:

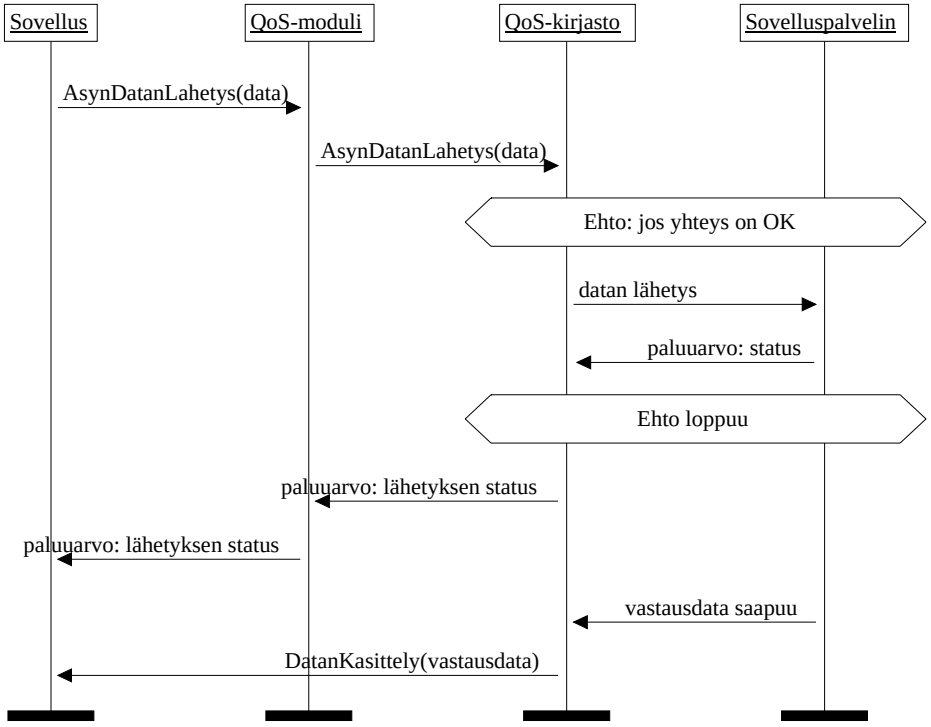
1. Sovellus muodostaa lähetettävästä datasta puskurin joka lähetetään verkon yli vastaanottajalle.
2. Sovellus käyttää QoS-modulin palvelua datan lähettämiseen, parametreina lähetettävä datapuskuri ja puskurin koko.
3. QoS-moduli välittää tiedot edelleen QoS-kirjastolle.
4. QoS-kirjasto käyttää varattua yhteyttä datan lähetykseen.
5. Tieto lähetyksen onnistumisesta palautuu QoS-kirjastolle.
6. QoS-kirjasto palauttaa datan lähetyksestä saadun statuksen QoS-modulille ja edelleen sovellukselle.

Jälkiehdot: Sovelluksen muodostama datapuskuri on välitetty vastaanottajalle. Mahdollinen vastaus saadaan takaisinkutsumekanismin avulla.

Poikkeustilanteet:

1. Datan lähetys QoS-kirjastossa epäonnistuu. Yhteyden tila tarkistetaan ja lähetystä yritetään uudestaan. Myös uudelleenlähetyksen epäonnistuessa tieto epäonnistuneesta lähetyksestä välitetään QoS-modulille ja edelleen sovellukselle.

Operaatiokaavio:



6. Datan vastaanotto

Yleinen kuvaus: Sovellus vastaanottaa dataa kohdeosoitteesta. Vastaanotettu data voi olla vastaus asynkroniseen datan lähetykseen tai erillinen datalähetys sovellukselle riippuen sovellusalueen toteutuksesta.

Esiehdot: Yhteys on avattu *Yhteyden avaus* –käyttötapauksen mukaisesti.

Osallistujat: Yhteyden avannut sovellus, QoS-moduli, QoS-kirjasto.

Toiminnan kuvaus:

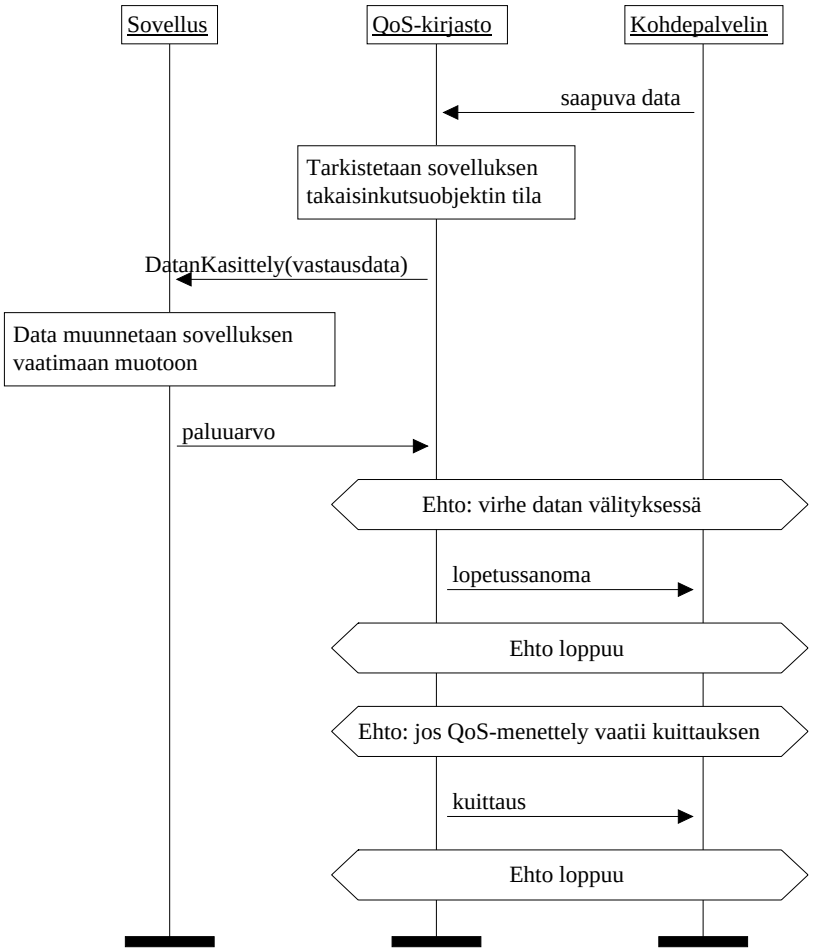
1. Kohdeosoitteessa tuotettu data lähetetään varattua yhteyttä käyttäen sovellukselle.
2. QoS-kirjasto vastaanottaa datan ja lähettää datan edelleen sovellukselle käyttäen takaisinkutsuviitettä.
3. Sovellus käsittelee datan ja takaisinkutsuviitettä käyttäen tehty kutsu palaa QoS-kirjastolle.
4. QoS-kirjasto tekee mahdolliset kuittaukset kohdeosoitteeseen käytetyn QoS-menettelyn mukaisesti.
5. QoS-kirjasto jää odottamaan mahdollista uutta datalähetystä.

Jälkiehdot: Kohdeosoitteesta lähetetty datapuskuri on välitetty sovellukselle. QoS-kirjasto on valmis vastaanottamaan lisää dataa tai käsittelemään sovelluksen tekemiä operaatioita.

Poikkeustilanteet:

1. Datan lähetys sovellukselle takaisinkutsuviitettä käyttäen epäonnistuu. Yhteys puretaan ja sovelluksen on avattava uusi yhteys jos kommunikaatiota kohdeosoitteeseen halutaan jatkaa. Takaisinkutsuviitteen ollessa ainoa keino informoida sovellusta yhteyden tilasta, ei sovellus saa tietoa yhteyden katkeamisesta ennen kuin se yrittää datan lähetystä QoS-kirjastoa käyttäen.

Operaatiokaavio:



7. Yhteyden parametrien muuttaminen (renegotiation)

Yleinen kuvaus: Sovellus voi tarpeen vaatiessa pyytää yhteyden uudelleenneuvottelua. Tällainen tilanne tulee kyseeseen esimerkiksi

tilanteessa, jossa yhteyden kaistanleveyttä halutaan kasvattaa tai vähentää tilanteessa jossa datan määrä ei mahdollista tarpeeksi tasokasta palvelua (esim. videokuvaa), tai verkon ei katsota enää pystyvän tarjoamaan aiempien yhteysparametrien mukaista palvelua.

Esiehdot: Yhteys on avattu *Yhteyden avaus* –käyttötapauksen mukaisesti.

Osallistujat: Yhteyden avannut sovellus, QoS-moduli, QoS-kirjasto.

Toiminnan kuvaus:

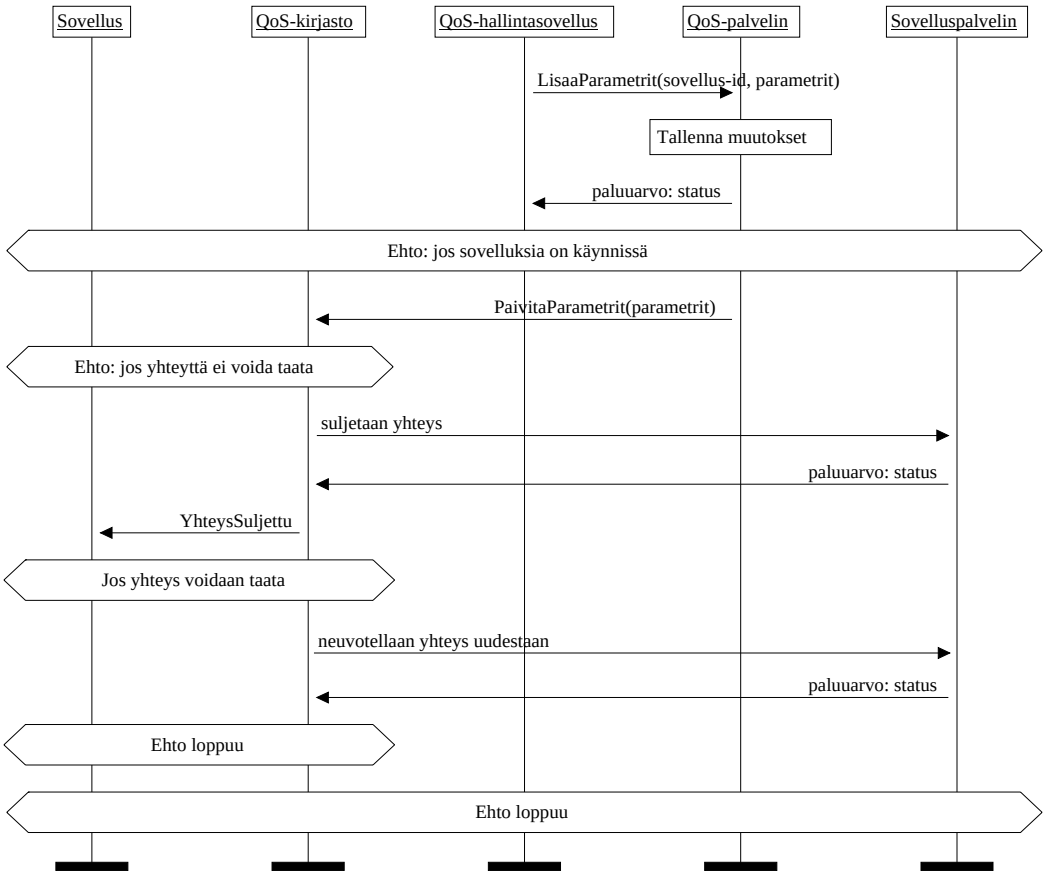
1. Sovelluksen konfiguraatio muuttuu keskitetyllä yhteysparametripalvelimella.
2. QoS-moduli vastaanottaa konfiguraatiopalvelimelta tiedon parametrien muuttumisesta.
3. QoS-kirjasto saa QoS-modulilta tiedon uusista parametreista.
4. QoS-kirjasto avaa uuden yhteyden kohdeosoitteeseen ja purkaa vanhan yhteyden jos avaus onnistui.
5. Jos yhteyden parametrien muutos epäonnistuu siksi, että verkko ei pysty takaamaan uusien parametrien mukaista palvelun tasoa, niin QoS-kirjasto sulkee yhteyden ja ilmoittaa takaisinkutsumekanismin avulla sovellukselle yhteyden sulkeutumisesta.

Jälkiehdot: Yhteysparametreihin tehdyt muutokset on otettu käyttöön QoS-kirjaston toteuttamassa yhteydessä kohdepalvelimelle.

Poikkeustilanteet:

1. Uuden yhteyden avaus epäonnistuu ja sovelluksen informointi asiasta epäonnistuu takaisinkutsuviitteen avulla. Takaisinkutsuviitteen ollessa ainoa keino informoida sovellusta yhteyden muutoksista, ei sovellus saa tietoa yhteyden katkeamisesta ennen kuin se yrittää kutsua QoS-modulin operaatioita.

Operaatiokaavio:



Kuva 1: asetusten hallinta

8. Verkon palvelutason muuttuminen

Yleinen kuvaus: Tietylle liikenneryhmälle varattu kaista verkossa voi liikenteen lisääntyessä ruuhkaantua siten, että sovelluksien QoS-vaatimuksia ei pystytä täyttämään, tai esimerkiksi jokin reititin verkossa vikaantuu ja liikenne joudutaan siirtämään kapasiteetiltaan pienempää linkkiä pitkin. Tällaisissa tilanteissa sovelluksia täytyy pystyä informoimaan ja tarvittaessa ohjata niitä pienentämään QoS-vaatimuksiaan, jotta tietty palvelun taso pystyttäisiin takaamaan myös poikkeustilanteissa..

Esiehdot: Yhteys on avattu *Yhteyden avaus* –käyttötapauksen mukaisesti.

Osallistujat: Yhteyden avannut sovellus, QoS-moduli, QoS-kirjasto.

Toiminnan kuvaus:

1. Sovelluksen konfiguraatio muuttuu keskitetyllä yhteysparametripalvelimella.

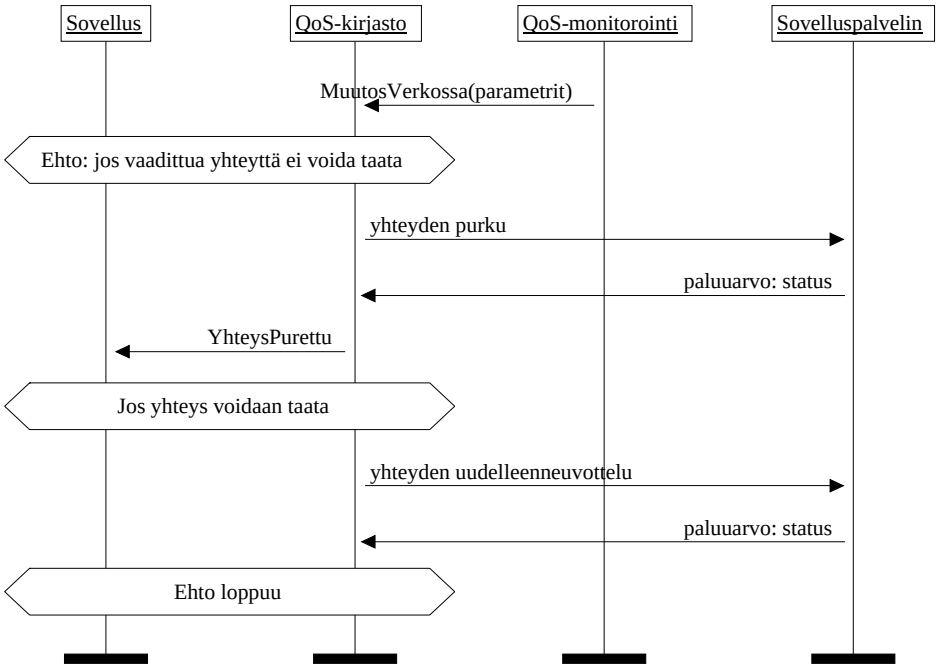
2. QoS-moduli vastaanottaa konfiguraatiopalvelimelta tiedon parametrien muuttumisesta.
3. QoS-kirjasto saa QoS-modulilta tiedon uusista parametreista.
4. QoS-kirjasto avaa uuden yhteyden kohdeosoitteeseen ja purkaa vanhan yhteyden jos avaus onnistui.
5. Jos yhteyden parametrien muutos epäonnistuu syystä että verkko ei pysty takaamaan uusien parametrien mukaista palvelun tasoa, niin QoS-kirjasto sulkee yhteyden ja ilmoittaa takaisinkutsumekanismin avulla sovellukselle yhteyden sulkeutumisesta.

Jälkiehdot: Yhteysparametreihin tehdyt muutokset on otettu käyttöön QoS-kirjaston toteuttamassa yhteydessä kohdepalvelimelle.

Poikkeustilanteet:

1. Uuden yhteyden avaus epäonnistuu ja sovelluksen informointi asiasta epäonnistuu takaisinkutsuviitteen avulla. Takaisinkutsuviitteen ollessa ainoa keino informoida sovellusta yhteyden muutoksista, ei sovellus saa tietoa yhteyden katkeamisesta ennen kuin se yrittää kutsua QoS-modulin operaatioita.

Operaatiokaavio:



9. Yhteyden katkeaminen

Yleinen kuvaus: Epävakaassa tai ylikuormitetussa verkossa yhteyksien katkeilu saattaa olla yleistä ja sovellusten on varauduttava tähän. QoS-sovelluskehyksen tulee tarjota tällaisissa tapauksissa hallitut mekanismit yhteyksien sulkemiseen ja uudelleen avaamiseen.

Esiehdot: Yhteys on avattu *Yhteyden avaus* –käyttötapauksen mukaisesti.

Osallistujat: Yhteyden avannut sovellus, QoS-moduli, QoS-kirjasto.

Toiminnan kuvaus:

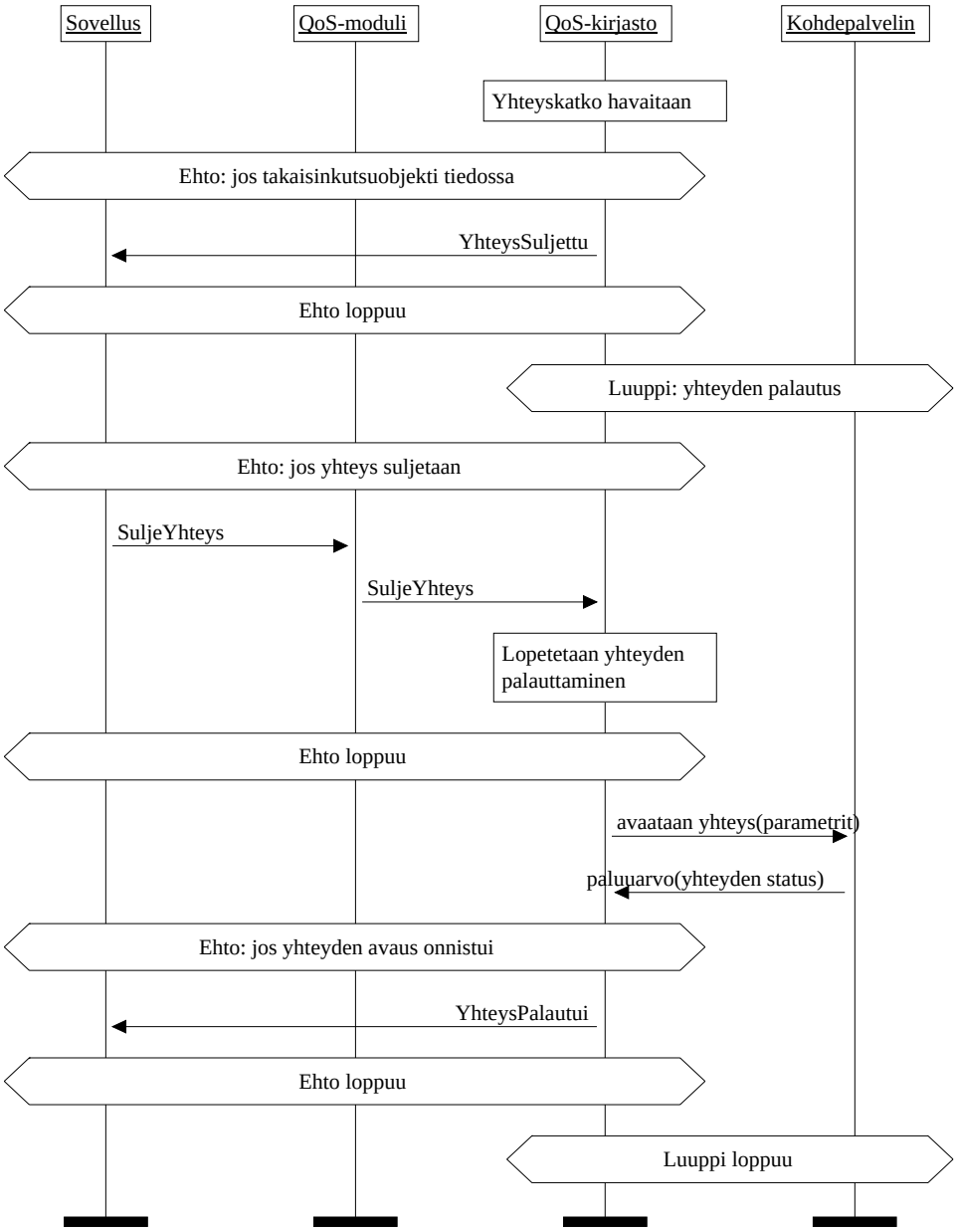
1. QoS-kirjasto saa tiedon yhteyden katkeamisesta.
2. QoS-kirjasto tiedottaa sovellusta takaisinkutsuobjektin kautta yhteyden katkeamisesta.
3. Jos sovelluksen QoS-parametrit sisältävät menettelyn yhteyden automaattiseksi palauttamiseksi, aloitetaan silmukka jossa yhteyden palauttamista yritetään tietyin väliajoin.
4. Yhteyden palautuessa sovellukselle ilmoitetaan asiasta takaisinkutsuviitteen avulla.
5. Sovellus voi keskeyttää yhteyden avaussilmukan sulkemalla yhteyden.

Jälkiehdot: Yhteys on avattu uudestaan odottamattoman sulkeutumisen jälkeen.

Poikkeustilanteet:

1. Yhteyden sulkemiseen liittyen voidaan törmätä tilanteeseen, jossa takaisinkutsuobjektin kutsuminen epäonnistuu. Takaisinkutsuobjektin ollessa ainoa keino tiedottaa sovellusta yhteydessä tapahtuvista muutoksista, on QoS-kirjaston jäätävä odottamaan, kunnes sovellus kutsuu QoS-modulin operaatioita, jotta virhetilanne saadaan ilmoitettua sovellukselle.

Operaatiokaavio:


Sovellus
QoS-moduli
QoS-kirjasto
Kohdepalvelin
Yhteyskatko havaitaan
Ehto: jos takaisinkutsuobjekti tiedossa
YhteysSuljettu
Ehto loppuu
Luuppi: yhteyden palautus
Ehto: jos yhteys suljetaan
SuljeYhteys
SuljeYhteys
Lopetetaan yhteyden palauttaminen
Ehto loppuu
avaataan yhteys(parametrit)
paluuarvo(yhteyden status)
Ehto: jos yhteyden avaus onnistui
YhteysPalautui
Ehto loppuu
Luuppi loppuu