PLANLEGGING AV A1

Kommunikasjon, nett og tjenester – Innlevering 1.1

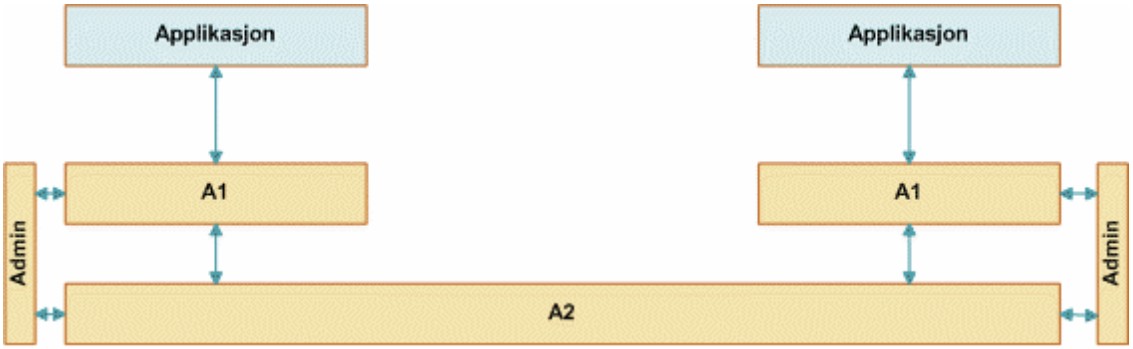
Gruppe 23

Innledning

Dette dokumentet tar for seg implementasjonsplanen for A1, det fobindelsesorienterte nettet som skal lages ifm. Fellesprosjektet.

Oppgavebeskrivelse[1]

Applikasjonen som utvikles i fellesprosjektet er avhengig av å kunne kommunisere over et nettverk. Målet med oppgaven i KTN er å kunne tilby denne applikasjonen en pålitelig overføring av informasjon. Et proprietært forbindelsesløst nett, kalt A2, eksisterer allerede. Det er mulig å kontrollere feilene som kan oppstå i A2 ved å bruke Admin. Vi skal realisere forbindelsesorientert nett basert på A2. Dette forbindelsesorienterte laget er kalt A1, og er kun delvis implementert. Deres oppgave blir derfor å gjøre de nødvendige endringer i A1, slik at vi kan tilby pålitelig overføring av informasjon til applikasjonen. Figuren under skisserer hvordan komponentene forholder seg til hverandre.



Både A2 og Admin er ferdig implementert og skal ikke endres. Dokumentasjon av A2 er gitt i [A2-Ud] og dokumentasjon for Admin er gitt i [Adm-Ud]. Disse blir levert i egne javapakker kalt henholdsvis no.ntnu.fp.net.cl (A2) og no.ntnu.fp.net.admin (Admin). Alle forandringer som er nødvendig for å tilby forbindelsesorientert overføring for applikasjonen gjøres i A1. Et koderammeverk [A1-Djc] med tilhørende dokumentasjon [A1-Doc] som kan benyttes som basis blir utlevert. Som et utgangspunkt for design og realisering av denne eksisterer en dokumentasjon [A1-Ud] som kan benyttes som et kravgrunnlag for realisering av A1.

Feil fra A2

Vi må ta hensyn til følgende feil[2]:

FEIL	ÅRSAK	KONSEKVENS
PAKKETAP	Pakken kom aldri frem til sin destinasjon.	Pakken kom aldri frem og informasjonen er tapt.
PAKKE FORSINKET	Pakken ble forsinket men dukket opp til slutt.	Pakken kommer kanskje to ganger til mottaker, siden den måtte sendes på nytt.

[1] Fra Fellesprosjektkompendiet
[2] A2-Ud.pdf

PAKKEN HAR FEIL	Pakken har blitt ødelagt underveis, og checksum er feil.	Pakken kan inneholde feil informasjon når den kommer til destinasjonen. Hvis headeren er endret, kan pakken ende opp på feil sted.
SPØKELSESPAKKE	En pakke fra hvor som helst (eller fra ingensteds) påstår den tilhører prosessen basert på headerinformasjon, og er derfor tatt imot.	En pakke som ikke skal komme ankommer. Kan ofte bli eliminert med en checksum-sjekk.

Segmentstruktur i utlevert kode

Segmentstrukturen er bestemt av klassen KtnDatagram. Her finnes følgende felter:

- **MyPort**. Portnummeret pakken kommer fra.
- **MyAddress**. Hostname til der pakken kommer fra.
- **RemotePort**. Portnummer til dit pakken skal.
- **RemoteAddress**. Hostname til dit pakken skal.
- **Ack**. Heltallsfelt for å sette hvilken pakke man bekrefter
- **Checksum**. Heltallsfelt for å sette checksum
- **Flag**. Enum med ulike flagg, se under.
- **Seq_nr**. Sekvensnummer
- **Payload**. Streng som inneholder data som skal sendes.

FLAGG	BESKRIVELSE
ACK	Flagg for å bekrefte en pakke
SYN	Flagg for å opprette en forbindelse
SYNACK	Flagg for å opprette en forbindelse
FIN	Flagg for å rive ned en forbindelse
NONE	Intet flagg

Overordnet beskrivelse

For at A1 skal være i orden, må disse aspektene være i orden:

- En instans A1 må kunne opprette en forbindelse til en annen instans av A1 kjørende på en annen prosess på en annen maskin.
- A1 må sikre at pakkene mottas i riktig rekkefølge.
- Feilene i A1 må være transparente for applikasjonen.

Denne planen er kraftig inspirert av implementasjon av TCP.

Sette opp forbindelse

Forbindelsen settes opp med en «three way handshake» på samme måte som TCP.

1. Serveren lager en Connection-instans med kjent portnummer, og kaller accept() på denne.
2. Klienten oppretter en Connection-instans uten definert portnummer. Det vil generes et tilfeldig portnummer i intervallet 10'000-20'000. Det vil også genereres et tilfeldig sekvensnummer for denne connection-instansen mellom 0 og 10'000.
3. Klienten sender en SYN til serveren der headeren er fylt opp med sine egne innstillinger (blant annet port og sekvensnummer)
4. Serveren mottar SYN. Serveren oppretter en ny Connection-instans med tilfeldig portnummer og sekvensnummer som beskrevet ovenfor. Fra denne sendes det en SYNACK.
5. Klienten mottar SYNACK, lagrer servers port- og sekvensummer i egen tilstand. Klienten sender ACK på at dette er mottatt.
6. Oppkoblingen er nå satt opp.

Rive ned forbindelse

Forbindelsen rives ned på samme måte som TCP. Nedenfor vil «klient» bli brukt til den som tok initiativ til nedrivingen, og «server» den som mottar melding om at oppkoblingen skal ned.

1. Applikasjonen på klienten kaller close() på Connection. Oppkoblingen sender FIN til serveren og setter seg selv i en tilstand hvor det ikke lenger er lov å sende/motta data.
2. Serveren mottar FIN fra klienten, og setter seg i en tilstand hvor det ikke er lov til å sende/motta data. Den sender en ACK på dette. Etter den har sendt en ACK, sender den selv en FIN til klienten.
3. Klienten mottar FIN, og sender ACK på dette. Fra klienten sin side er oppkoblingen nå tatt ned (etter en timer på 30 sekunder er løpt ut).
4. Serveren mottar ACK. Serveren satt til CLOSED etter 30 sekunder.

Sende/motta data

Her brukes ordet «klient» til den som sender data og «server» til den som mottar.

1. Connection på klienten mottar data (en streng) fra applikasjonen. Connection lager et datagram av dette. Dette sendes til serveren.
2. Serveren mottar pakken, kontrollerer om alt er riktig (checksum, sekvensnummer osv.). Sender ACK tilbake. Connection-klassen har holdt programmet ventende med et recieve()-kall. Dette returnerer nå strengen. Serveren vil normalt prosessere dataene og kalle receive() på nytt.

3. Klienten mottar ACK og vet at pakken er mottatt. Timeren skrus av og Connection-instansen blir klar for å sende flere pakker.

Hvordan skal applikasjonen bruke Connection?

Applikasjonen er nødt til å holde styr på to tråder, én for mottak og én for sending. Dersom mottak mottar EOFException, er oppkoblingen revet ned (eller i ferd med å rives ned), og send-data vil få en feilmelding om at oppkoblingen er i en ugyldig tilstand for sending av data. Applikasjonen må derfor selv sørge for at det ikke blir sendt data etter at EOFException har kommet.

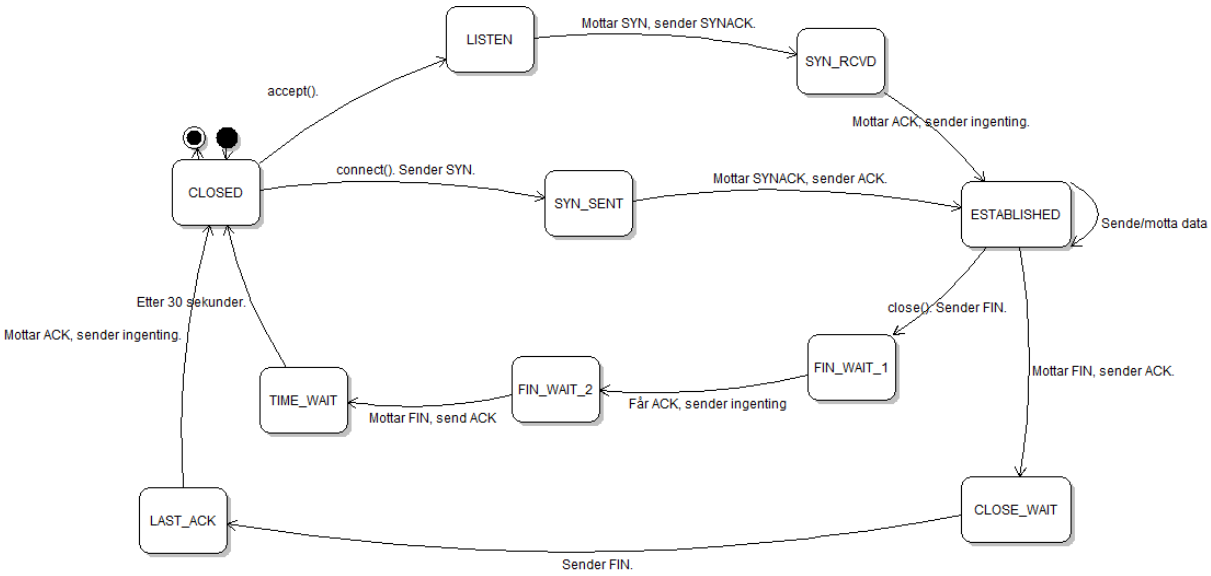
Pipelining og congestion control

Effektivitet og utnyttelse av båndbredde er ikke et krav i denne oppgaven. Det vil derfor kun være én pakke i omløp av gangen (per retning), altså en form for «**stop-and-wait»**-protokoll. Følgelig blir det heller ingen behov for å implementere metningskontroll (congestion control).

Tilstandsdiagrammer

Oppkobling og nedkobling

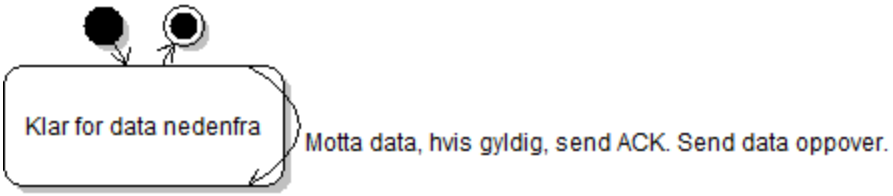
Bruker i denne delen tilstander som definert i AbstractConnection. Disse er de samme som er brukt i Computer Networking, A Top-Down Approach (Kurose & Ross). Vi har valgt å lage et felles tilstandsdiagram for oppkobling uavhengig om du er server eller klient.



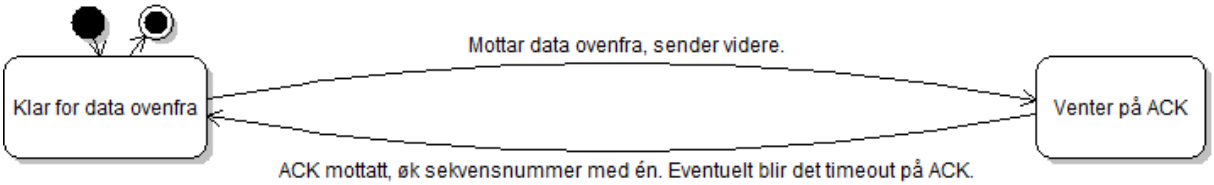
Sending og mottak

Hver connection vil ha to tråder, en for mottak og en for sending. Kode for sending og mottak av data fra A2 er allerede implementert i den utleverte koden (AbstractConnection).

Dette tilstandsdiagrammet gjelder mottak av data:



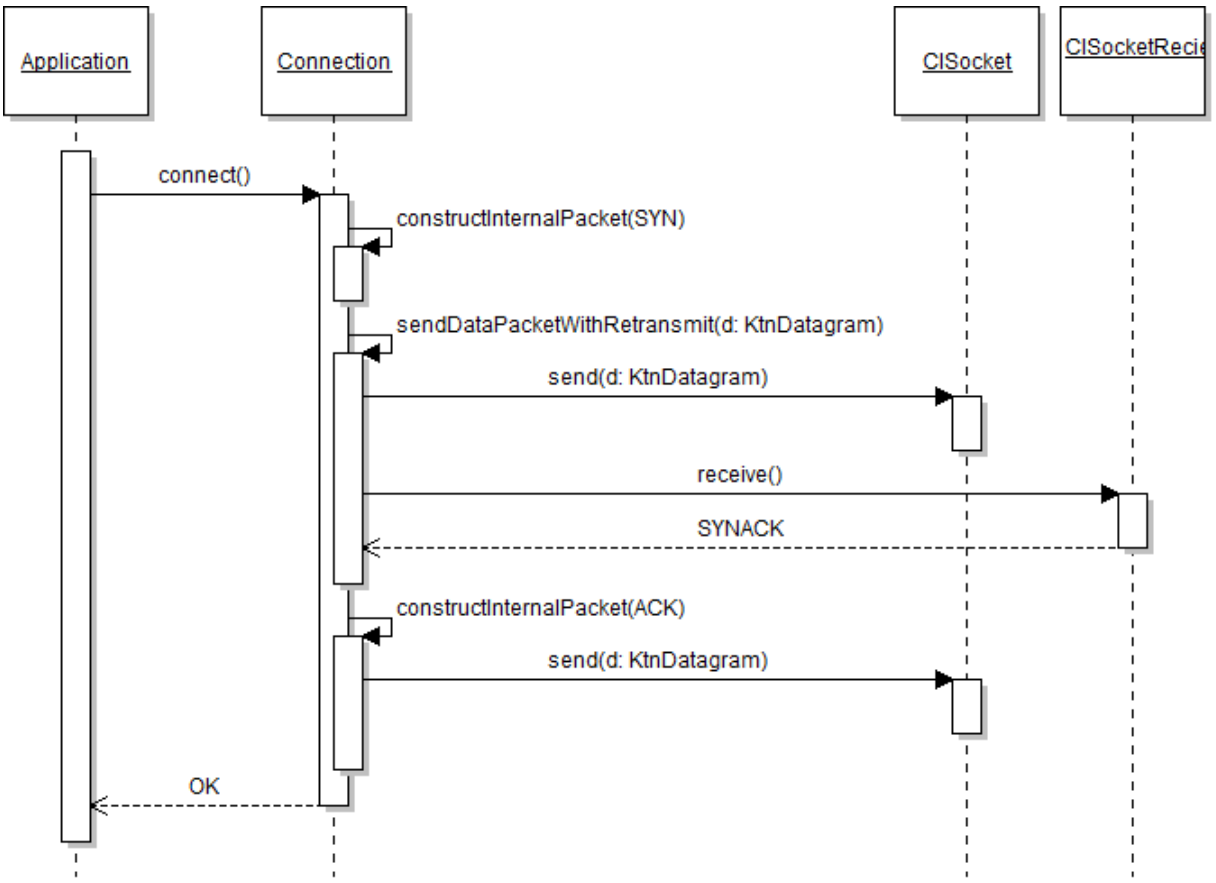
Dette gelder sending av data:



Sekvensdiagrammer

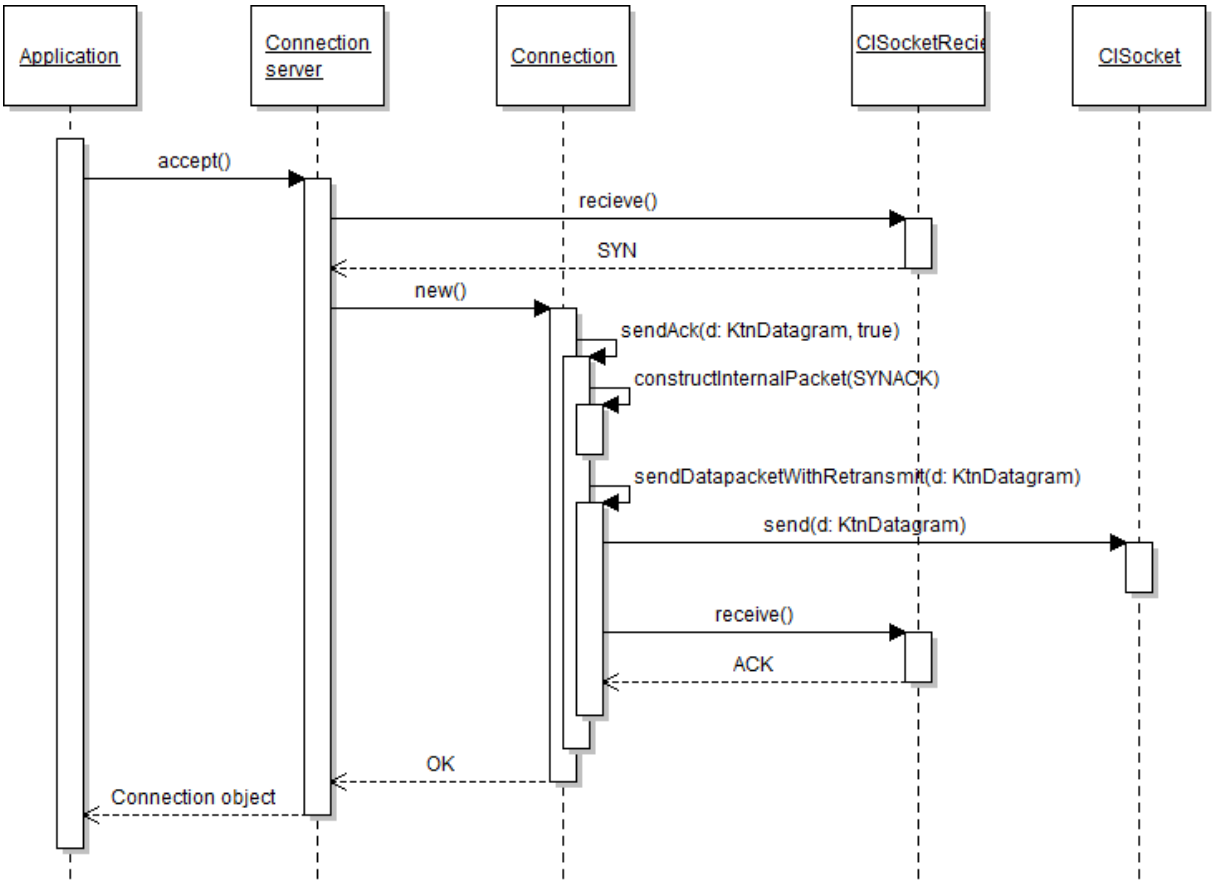
Oppkobling (klient)

Dette sekvensdiagrammet viser hva en klient ville gjort dersom den hadde koblet seg opp mot en server som stod og ventet på innkommende forespørsler med accept()

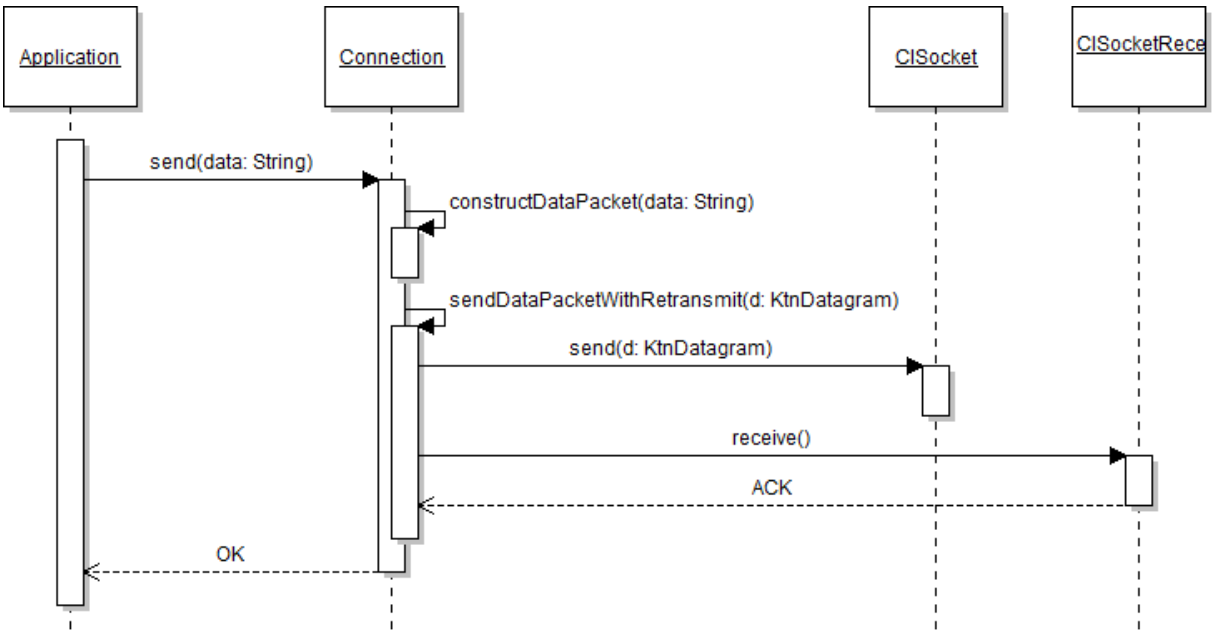


Oppkobling (server)

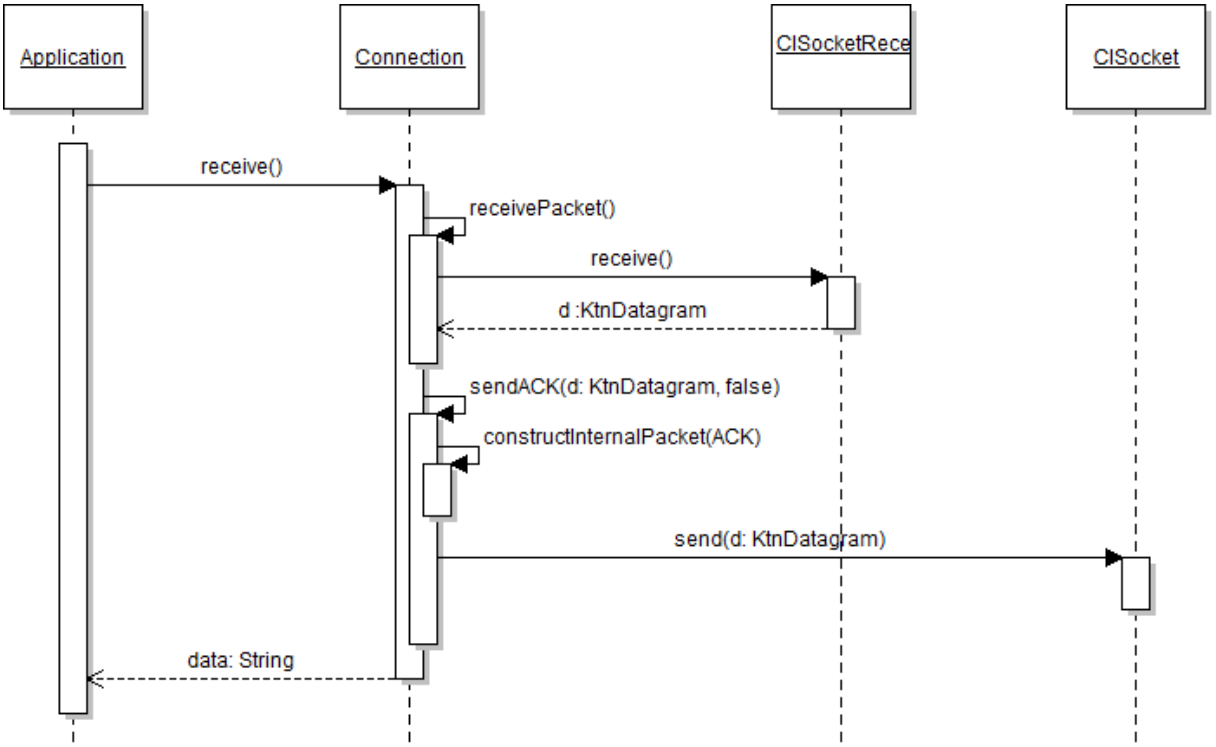
Serveren holder en egen instans av Connection oppe, her kalt ServerConnection. Ved mottak av SYN, oppretter den en ny Connection-instans og sender SYNACK derifra (med nye portnumre for oppkoblingen).



Sending av data

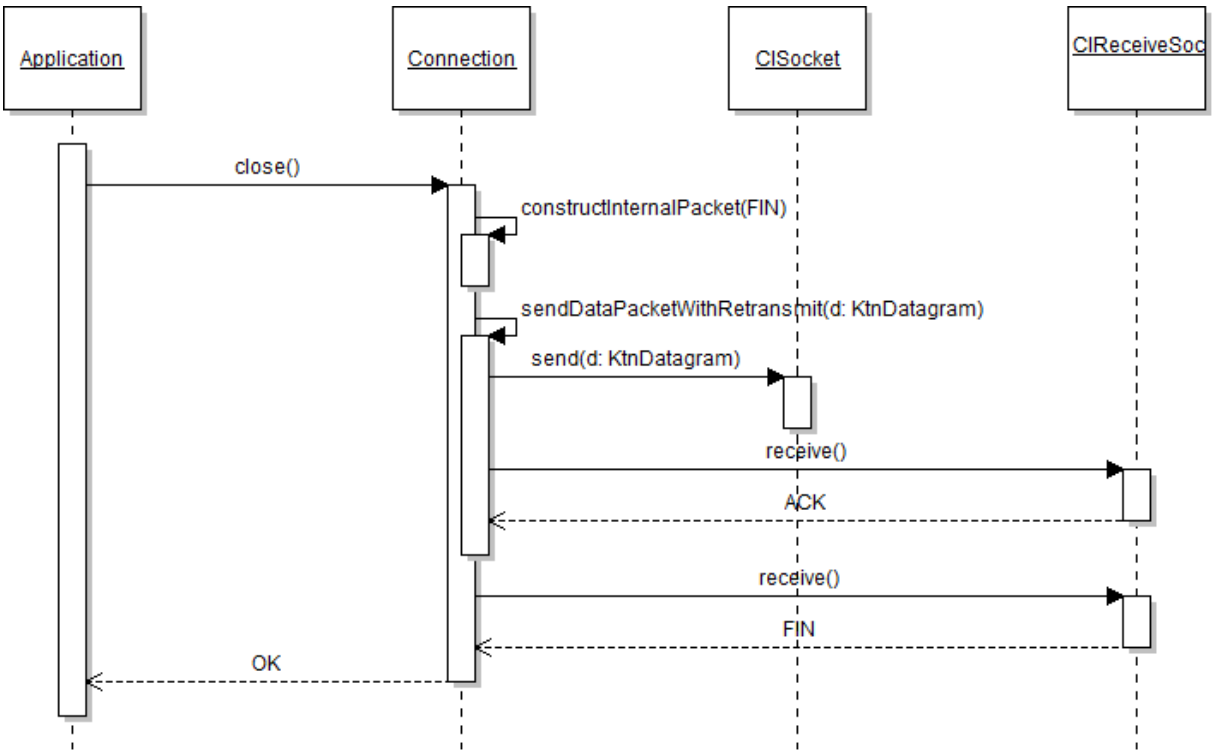


Mottak av data



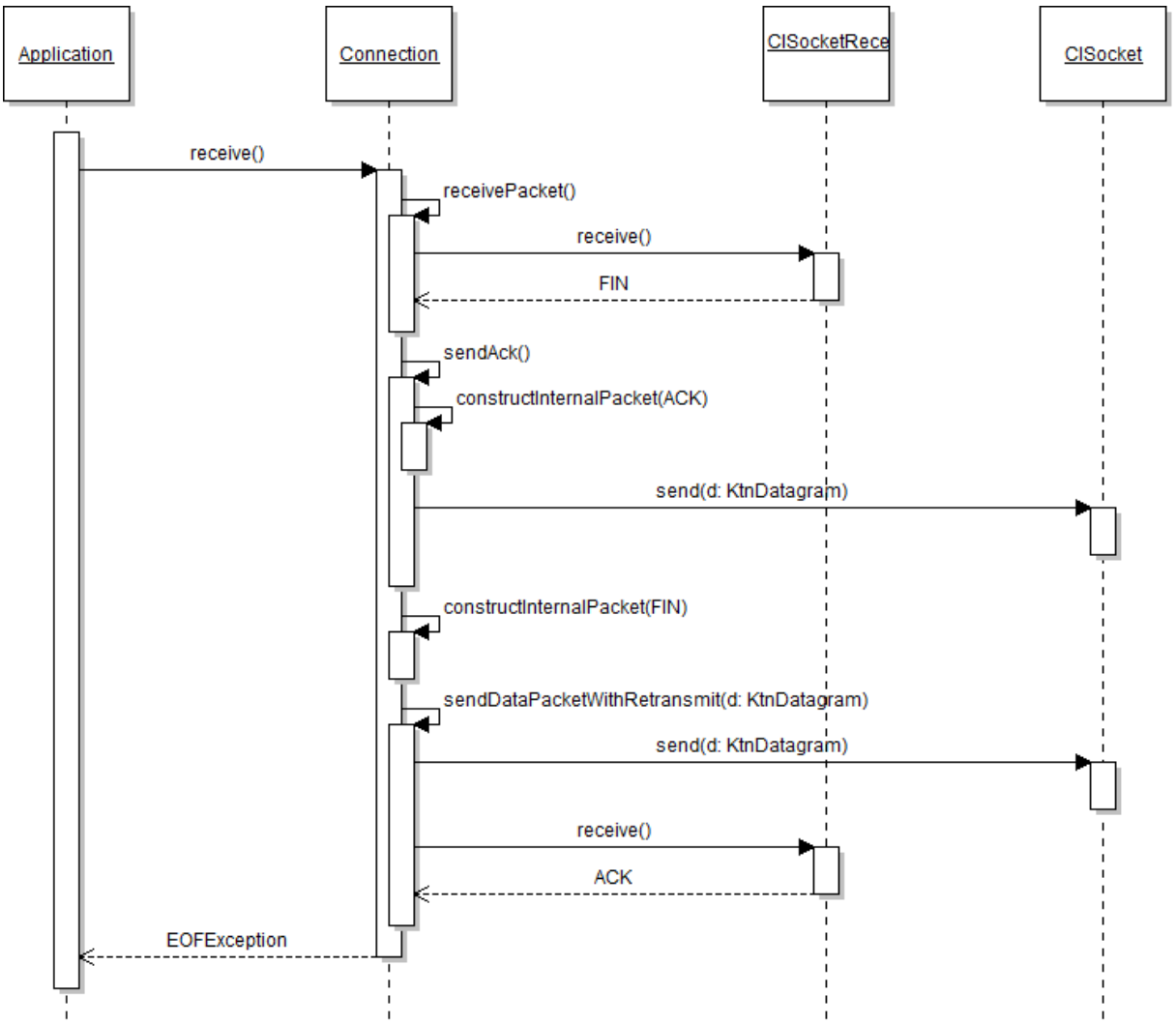
Nedkobling, klient

Viser sekvensdiagram for applikasjonen som tar initiativ til en nedkobling.



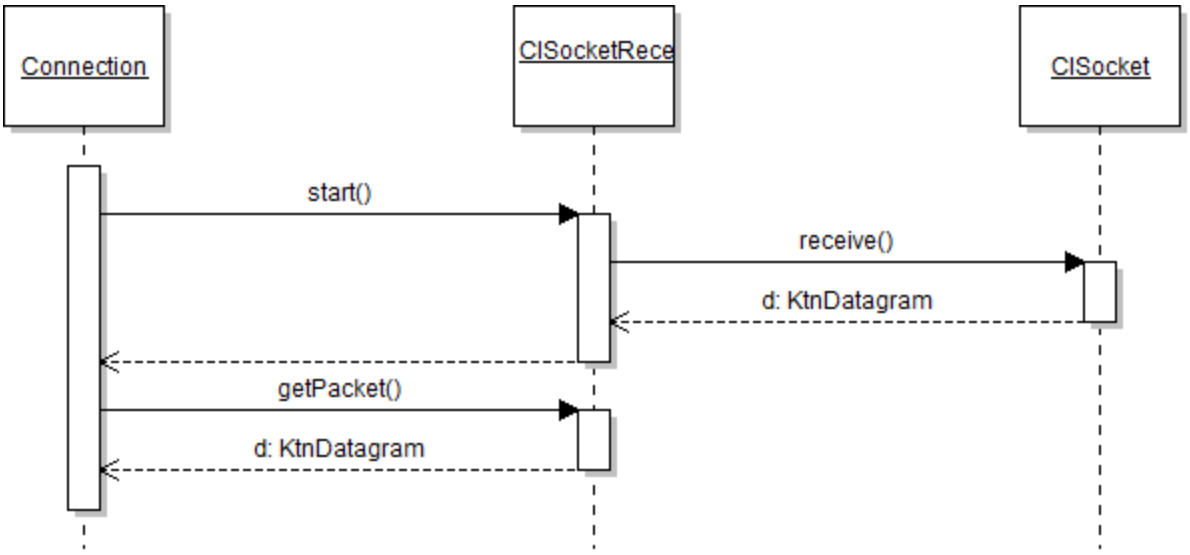
Nedkobling, server

Viser sekvensdiagram for applikasjonen som mottar FIN.



ClSocketReceiver

På sekvensdiagrammene har vi hoppet over noen metodekall på ClSocketReceiver for å gjøre sekvensdiagrammene mer oversiktlig. Når man kaller receive() på denne klassen, skjer dette:



Feilhåndtering

Her kommer en liste over de forventede feilene og hvordan disse skal håndteres i vår implementasjon av A1.

Pakketap

Hver pakke som sendes i A1 vil være tilknyttet en timer. Dersom ACK ikke er mottatt innen timeout, sendes pakken på nytt. Dersom ACK-pakken forsvinner, vil dette fortsatt fungere.

Forsinket pakke

Oppkoblingen har hele tiden styr på hvilken pakke den forventer av mottaker. Dersom den mottar en pakke som den ikke forventer, forkast pakken. Dette kan for eksempel være en pakke den allerede har mottatt.

Pakken har feil

Beregn checksum for segmentet, og sammenlign med checksum som står i headerfeltet til segmentet. Dersom dette ikke stemmer, forkast pakken. Siden det ikke sendes noe ACK, vil pakken komme på nytt etter at den har timet ut. Checksum er ikke garantert å oppdage bitfeil, men sannsynligheten for at den ikke finner feil på et korrupt segment er svært liten. Dersom den ikke oppdager feilen, godtar A1 pakken.

Spøkelsespakker

Flere spøkelsespakker vil kunne ha feil checksum, og vil derfor bli fjernet i checksum-beregningen. Dersom checksum stemmer, må man sjekke om sekvensnummeret matcher det forventede sekvensnummeret. Dersom dette også stemmer, sjekker man alle felt som angår porter og hostnames og sjekker om de matcher tilstanden til den aktuelle Connection. Dersom pakken består alle disse testene, er det ikke annet å gjøre enn å sende den videre til applikasjonen, selv om den ikke hører hjemme der.

Testplan

Systemet må først testes uten feil. Hensikten med dette er å avdekke eventuelle feil i oppkobling, datasending og nedkobling.

Når disse testene går igjennom må systemet testes med feil konfigurert i Admin. Ved å utføre tester uten feil først, er det lettere å isolere feilkildene.

Testing uten feil

Test-ID	Oppkobling1
Beskrivelse	Klient-Serverpar oppretter forbindelse
Startbetingelse	Server har opprettet en Connection hvor accept() er kalt.
Inndata	Server sitt hostname og port
Steg	1. Klient oppretter Connection-objekt. 2. Klient kaller connect() mot hostname og port til server.
Forventet resultat	Dersom hostname og port er riktig konfigurert, og brannmurer satt opp riktig, vil Connection-klassen havne i en gyldig tilstand og koden vil kjøre videre. Hvis ikke vil socketen løpe ut og connect() gir en SocketTimeoutException
Avhengigheter	Ingen

Test-ID	SendMotta1
Beskrivelse	En allerede opprettet forbindelse
Startbetingelse	Server og klient har hver sinn instans av Connection. Instansene er koblet sammen som beskrevet i Oppkobling1
Inndata	Streng med testdata
Steg	1. Server kaller receive() på sin Connection. 2. Klient kaller send() med testdataene på sin Connection. 3. Server skriver ut testdataene i konsollen.
Forventet resultat	Recieve()-metoden skal holde koden helt til den mottar noe. Dersom dataene ble sendt, skal serveren skrive ut testdataene i konsollen. Dersom dataene ikke kommer frem (på grunn av feil i det underliggende nettet) skal serveren fortsette å vente på en pakke, imens klienten gir en IOException.
Avhengigheter	Oppkobling1

Test-ID	SendMotta2
Beskrivelse	Som SendMotta1, men bytte rollene (seder/mottaker)
Avhengigheter	Oppkobling1, SendMotta1

Test-ID	KobleNed1
Beskrivelse	En allerede opprettet forbindelse
Startbetingelse	Server og klient har hver sin instans av Connection. Instansene er koblet sammen som beskrevet i Oppkobling1
Inndata	Ingen
Steg	1. Klient kaller close().

Forventet resultat	Begge instansene av Connection skal havne i CLOSED-tilstanden
Avhengigheter	Oppkobling1

Test-ID	KobleNed2
Beskrivelse	Som KobleNed1, men bytte rollene (seder/mottaker)
Avhengigheter	Oppkobling1, KobleNed1

Testing av sekvens med og uten feil

Det vil settes opp programvare som sender over tallene 1-100 gjennom A1 via A2. A2 konfigureres i henhold til test-spesifikasjonen. Testen er bestått dersom alle pakkene ankommer i rekkefølge, eventuelle andre resultat beskrives «begrunnelse».

ID	TAP	FORSINKET	FEIL	SPØKELSE	BEGRUNNELSE
SEKVENS	0%	0%	0%	0%	Teste om pakkene ankommer i riktig rekkefølge.
TAP1	10%	0%	0%	0%	Presisert i oppgavetekst.
TAP2	50%	0%	0%	0%	Presisert i oppgavetekst.
TAP3	99%	0%	0%	0%	Forbindelsen er så dårlig at IOException skal gis.
FORS1	0%	10%	0%	0%	Presisert i oppgavetekst.
FORS2	0%	50%	0%	0%	Presisert i oppgavetekst.
FEIL1	0%	0%	10%	0%	Presisert i oppgavetekst.
FEIL2	0%	0%	50%	0%	Presisert i oppgavetekst.
SPØK1	0%	0%	0%	10%	Presisert i oppgavetekst.
SPØK2	0%	0%	0%	50%	Presisert i oppgavetekst.
MIKS1	20%	10%	10%	20%	Reell situasjon der flere pakker kan ha de samme feilene.
MIKS2	50%	50%	0%	0%	Interessant å teste programmet på en upålitelig linje
MIKS3	0%	50%	50%	0%	Interessant å sjekke om forbindelsen timer ut på dette (sender ikke ACK tilbake på disse feilene).
MIKS4	75%	75%	75%	75%	Mye feil som kan skje her, morsomt å skje hva som oppstår og hvilke feil som gjør seg gjeldende.