BACHELOROPPGAVE:

BO24EB-23 Simulering – Trådløs Undervanns Sensornettverk

Jens Hoff Quirk

Mathias Mjelle Fiske

Steffen Nanthen Balthasar

21. mai. 2024

Dokumentkontroll

Rapportens tittel: BO24EB-23 Simulering – Trådløs Undervanns Sensornettverk	*Dato/Versjon* 21.05.24/1.0
	Rapportnummer: B024EB-23
Forfatter(e): Jens Hoff Quirk Mathias Mjelle Fiske Steffen Nanthen Balthasar	*Studieretning:* KOM21
	Antall sider m/vedlegg 54
Høgskolens veiledere: Anne-Lena Kampen Md al Shayokh	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver: HVL	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): Anne-Lena Kampen anne-lena.kampen@hvl.no Md al Shayokh muhammed.al.shayokh@hvl.no	

Revisjon	Dato	Status	Utført av
0.1	16.04.24	Opprettet en kopi av bachelor mal	Steffen
0.2	02.05.24	Første utkast for vurdering av veileder	Steffen, Mathias, Jens
0.3	07.05.24	Rettet på feil og fylt ut resultater	Steffen, Mathias, Jens
0.4	19.05.24	Fikset appendiks og kilder	Steffen
1.0	21.05.24	Ferdig rapport	Steffen, Mathias, Jens

Forord

Det føles naturlig å innlede det hele med å takke veilederne gruppen har hatt gjennom oppgaveprosessen: Anne-Lena Kampen og Md al Shayokh. Gjennom jevnlige fysiske møter og mailkorrespondanser har dere fungert som gode faglige støttespillere og gitt en klar vei for hvilken retning det var ønskelig at oppgaven skulle styres. Det er også satt stor pris på at dette samarbeidet har hatt en vennlig undertone der det har vært enkelt å senke skuldrene i de felles diskusjonene. Takk for at dere satt av den tiden dere gjorde og takk for et hyggelig samarbeid.

Vi vil også takke Ingvar Henne for gjesteforelesningen han ga i emnet ELE305 (Cyber-fysiske kommunikasjonsteknologier) høsten 2023. Denne forelesningen fungerte som en god overordnet innføring i temaet og den tilhørende PowerPoint filen har hjulpet gruppen med hvilken kontekst arbeidet skulle sees innenfor.

Sammendrag

Oppgaven er skrevet i sammenheng med SFI Smart Ocean. Det blir skrevet om problematikken rundt akustisk trådløs undervannskommunikasjon og kommunikasjonsprotokoller som er laget for å imøtekomme denne problematikken. Dette er gjort i kontekst av sensornettverk under vann. Disse protokollene blir i oppgaven testet ut gjennom simuleringer gjort med simuleringsrammeverket DESERT. En vesentlig del av oppgaven tar også for seg hvordan dette rammeverket fungerer og hvordan protokollene er implementert i rammeverket.

Det lå i utgangspunktet en forventning om at oppgaven skulle se på både MAC- og rutingprotokoller, men grunnet mangel på tid tar oppgaven kun for seg MAC-protokoller. DACAP, Aloha og T-Lohi blir i oppgaven diskutert som relevante protokoller og teorien bak disse blir fremlagt, men kun DACAP og Aloha blir brukt i simuleringene.

I kapittel 6 er det skrevet om DESERT rammeverket på en måte som forhåpentligvis fungerer som en pekepinn mot hvordan det skal brukes. Dette kapittelet leses best med vedlagt TCL-kode tilgjengelig.

Resultatene fra simuleringene gir et innblikk i hvor stor ytelse man forvente av DACAP og Aloha i forskjellige sammenhenger, men gir ikke noe nevneverdig innblikk i kraftforbruket til protokollene.

1 Innhold

Figurer

2 Innledning

2.1 Organisering av rapporten

«Innledning» kapitelet handler om oppdragsgiveren og problemstillingen. «Kravspesifikasjon» og «Analyse av problemet», presenteres oppgaven og analyse av problemstilling og eventuelle løsninger. Kapittelet «UW(Underwater) MAC-protokoller» viser teorien bak protokollene som blir brukt under simuleringene. «DESERT rammeverk» forklarer aspekter rundt DESERT rammeverket og hvordan det brukes. I «Simuleringer» diskuteres det rundt simuleringene som er gjort og lærdommene de har gitt angående både undervannsprotokollene og rammeverket. Det blir også vist hvordan systemet for større antall simuleringer er satt opp. «Analyse av simuleringene» tar for seg analyse av simuleringsresultatene. I «Diskusjon/Refleksjon» snakkes det om hvordan oppgaven har gått med hensyn på den opprinnelige planen. Under «Konklusjon» drøftes oppgaven som en helhet og hva som eventuelt trengs å gjøres videre.

Rapporten bruker mange utrykk og forkortelser på engelsk som blir forklart under appendiks. TCL, Bash og Python filer ligger som et vedlegg.

2.2 Oppdragsgiver

Oppdragsgiver er Høgskulen på Vestlandet (HVL). HVL er en statlig norsk høgskole med lokaler i Bergen, Førde, Haugesund, Sogndal og Stord. Høgskolen har rundt 16500 studenter og 2000 ansatte. [1] HVL har fagmiljøer innenfor helse- og sosialvitenskap, ingeniør- og maritime utdanninger, lærer-, kultur- og idrettsfag, natur- og samfunnsvitenskap, og økonomi og ledelse. [2] Høgskolen satser på forsking innenfor blant annet: ansvarlig innovasjon, bærekraftig energi og miljø, folkehelse, hav og datateknologi.

Bachelorprosjektet er assosiert til SFI Smart Ocean - senter for forskningsbasert innovasjon, opprettet 1. desember 2020, arrangert av institutt for fysikk og teknologi i (UIB) Universitetet i Bergen. SFI Smart Ocean ønsker å skape en plattform for fleksibel, robust, energieffektiv og kostnadseffektive sensornettverk for marin- og datatjenester. Målet er å kunne løse fremtidige sosiale og industrielle problemstillinger ved hjelp av undervannsensorene som samler data over tid. Senteret jobber med 20 industri og forsknings partnere på å innovere innen undervannssensorer langs Norgeskysten. De forsker på standardisert, lavenergi og lavpris for undervanns trådløs kommunikasjon, utvikle systemer for Internet of Underwater Things og

utvikle anti biobegroing nanobelleg på sensorer og teknologi for måling av større områder med høy stabilitet og målekvalitet. [3]

Figur 1 - SFI Smart Ocean Logo (Foto/ill.: SFI Smart Ocean)

2.3 Problemstilling

Under trådløs akustisk kommunikasjon under vann er det viktig at man i størst grad unngår kollisjoner samtidig som man utnytter båndbredden mest mulig. Dette er viktig grunnet lav båndbredde og begrenset batteritid på kommunikasjonsenhetene. Det er derfor nødvendig å velge kommunikasjonsprotokoller som løser disse utfordringene best mulig. Videre er det ønsket at vi gjennom simulering av undervanns sensornettverk skal finne de beste løsningene for denne typen undervannskommunikasjon.

2.4 Hovedidé for løsningsforslag

«Design, Simulate, Emulate and Realize Test-beds for Underwater network protocols» (DESERT), er et rammeverk utviklet basert på Network simulator 2, NS2. [4] Ved å teste ut MAC og nettverkslag protokoller som er laget for undervannskommunikasjon, kan man observere effekten av disse protokollene. Ved å bruke DESERT rammeverket for simulering er det mulig utføre dette uten fysiske tester. Dette er fordelaktig da det er veldig ressurskrevende å utføre disse testene fysisk.

3 Kravspesifikasjon

- Analyser av hvordan forskjellige lag 2 og 3 protokoller presterer i akustiske trådløse undervannsnettverk.
- Diskusjon rundt effektiviteten til protokollene med hensyn på pakkeleveringsratio, gjennomstrømming, antall pakker levert og energi.
- Dokumentasjon av hvordan vi har gjort de forskjellige simuleringene.
- Eventuelle forslag til ny metodikk i lag 2 og 3 kommunikasjonen eller endringer i de eksisterende protokollene.
- Lage brukermanual for bruk av DESERT rammeverket for lesere som skal jobbe med det.

4 Analyse av problemet

Gjennom simulering av akustiske trådløse undervannsnettverk vil det være viktig å få et grundig overblikk over hvordan diverse protokoller presterer. Det finnes allerede protokoller som er utformet spesifikt for undervanns sensornettverk. Vi skal ta utgangspunkt i noen av disse protokollene. Det vil være nødvendig å simulere scenarioer under ulike forhold for å få mest mulig informasjon om hver protokoll sin effektivitet. Protokollene vil trolig prestere forskjellig på utnyttelse av båndbredde og antall kollisjoner. Viktigheten av disse faktorene burde derfor drøftes opp mot hverandre når protokollene evalueres.

For at arbeidet vi har utført skal være lett å sette seg inn i og bygge videre på, er det behov for å forklare på en enkel og konsis måte hvordan simuleringene våre fungerer. Dette vil innebære å kommentere all kildekode godt, samt å skrive utfyllende om simuleringene i selve oppgaven.

I takt med simuleringen av eksisterende protokoller burde det vurderes om det finnes behov for nye kommunikasjonsalgoritmer. Ved eventuelle forslag til endring i protokoller eller forslag til nye algoritmer må det tas med begrunnelser for hvorfor vi mener det vil forbedre kommunikasjonen. Det burde også vises til data som viser de eventuelle fordelene forslaget har over eksisterende protokoller vi har analysert.

4.1 Utforming av løsning

DESERT rammeverket skal brukes for simulering av undervanns sensornettverk. Det er en vesentlig del av oppgaven å sette seg inn i dette rammeverket og inkludere informasjon rundt bruk av rammeverket i oppgaven.

Oppsettet for simuleringene vil være definert i TCL filer og det er derfor viktig å referere til disse når vi forklarer hvordan vi har gjort simuleringene.

For automatisering av simuleringer er planen å lage enkle BASH-scripter som prøver ut forskjellige protokoller under scenarier med like parameterverdier. BASH-scriptene må kunne mate TCL-koden informasjon om hvilke protokoller som blir brukt, hvordan nodene i nettverket er satt opp, pakkestørrelsen, trafikkmengde per node og diverse andre parametere.

Både TCL- og BASH-scriptene burde, som tidligere nevnt, være kommentert utfyllende og forklarende slik at det er klart hva de gjør. Dette skal etter planen gjøres på en uniform måte på tvers av all koden som blir skrevet.

Forslag til nye kommunikasjonsalgoritmer skal drøftes etter at vi har analysert de eksisterende protokollene. Avhengig av om vi kommer frem til mer effektiv metodikk, skal vi forklare hvordan de eventuelle nye metodene fungerer og, om mulig, få til simulering der de nye algoritmene er tatt i bruk.

Det viktigste er at vi forstår DESERT og kan bruke det til å simulere enkle sensornettverk.

4.1.1 Vurderinger i forhold til verktøy og HW/SW komponenter

Oppgaven spesifiserer at DESERT skal brukes for simulering av undervannsnettverk. Det er ingen tidligere erfaring i gruppen i forhold til bruk av DESERT eller den type nettverksimulatorer rammeverket er bygd på. Læring av DESERT kan også bli problematisk i forhold til at det er lite dokumentasjon for bruken av rammeverket. Utfordringen med å bruke og lære verktøyet er dermed regnet som en vesentlig del av oppgaven.

DESERT har kun støtte for Debian-baserte operativsystem og Ubuntu er derfor tiltenkt som operativsystemet rammeverket skal installeres og brukes fra. Det er blandet erfaring med Linux baserte operativsystemer i gruppen. Rammeverket har heller ingen GUI. Det forventes derfor noe tid til innføring i bruk av Ubuntu og BASH i denne sammenheng.

5 UW(Underwater) MAC-protokoller

Undervannskommunikasjon skiller seg fra kommunikasjon i luft. Elektromagnetiske bølger som brukes for mye av kommunikasjonen over land blir utsatt for stor demping under vann. Dette resulterer i at det er ikke er mulig å kommunisere ved hjelp av elektromagnetiske bølger på større avstander. Akustiske bølger blir utsatt for mye mindre demping under vann og er derfor foretrukket for undervannskommunikasjon mellom middels til store avstander. Det er utarbeidet flere protokoller spesifikt for å imøtekomme utfordringene som er unike ved undervannskommunikasjon. Det skal her ses på tre MAC-protokoller: Aloha, DACAP og T-Lohi.

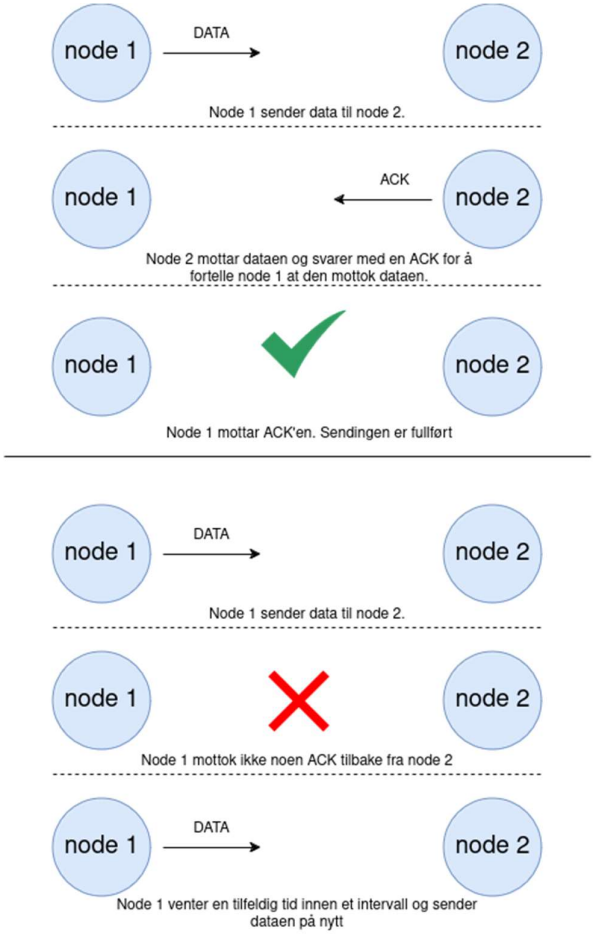
5.1 ALOHA

Aloha er opprinnelig en WAN-protokoll brukt til vanlig nettverkskommunikasjon over vann. Protokollen fungerer slik at avsender sender data rett ut til mottaker med en gang den har data klar for sending. Mottaker vil deretter send en ACK-melding tilbake til avsender om den mottok pakken. Hvis ikke avsender mottar en ACK-melding innen et spesifisert tidsrom, vil den vente en tilfeldig mengde tid (innen et spesifisert intervall) og så prøve sending av dataen på nytt. Dette er vist i figur 2.

Hvis avsender fortsatt ikke mottar en ACK-melding tilbake etter å ha prøvd et gitt antall ganger, vil den forkaste datapakken.

Aloha kan også implementeres med CSMA. Det eneste dette vil si er at avsender vil sjekke om det er signaler som blir sendt ved avsenders posisjon i øyeblikket avsender skal sende data. Noden har fortsatt ingen måte å vite om hele kanalen er ledig. Det er denne implementasjonen av Aloha som skal brukes til simuleringer videre i oppgaven.

Aloha er en relativt simpel protokoll, men kan være effektiv når det er lite trafikk på nettverket. Det skal, gjennom simuleringer senere i oppgaven, gjøres en vurdering om hvilke trafikkmengder og for hvilken mengde noder Aloha er en passende protokoll å bruke.



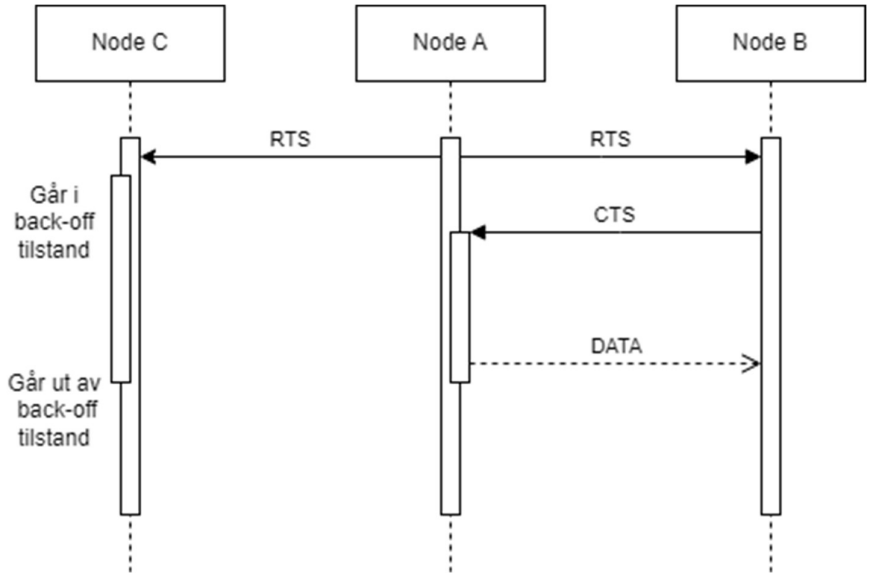
Figur 2 - ALOHA handshake

5.2 DACAP

DACAP er en usynkronisert protokoll som bruker forskjellige handshake lengder for forskjellige noder, basert på avstanden mellom dem. Protokollen fungerer slik:

- En node som vil overføre data til en mottaker node sender en «request-to-send» (RTS) til mottakeren.
- Mottakeren hører RTS signalet og svarer tilbake med en «clear-to-send» (CTS).
- Senderen beregner avstanden til mottaker basert på hvor lang tid RTS og CTS signalene brukte og bruker denne avstanden til å beregne ventetid. Ventetid = 2U/c, der U er avstanden mellom nodene og c er lydfarten under vann.

- Hvis mottakeren overhører en annen pakke som muligens kan kollidere med pakken den venter på å motta, vil den sende en kort advarsel til senderen og senderen vil avbryte prosessen med å sende pakken. Sender venter i så fall en tilfeldig valgt tid før den igjen prøver å sende pakken.
- Senderen mottar CTS og venter en stund, basert på noen forhåndsinnstilte parametere. Nodene må dele parametere om propagasjonsavstandene i nettverket for å vite hvor lenge handshaken skal vare, men trenger ikke samme interne klokke. Hvis mottakeren oppdager pakker etter den har sendt en CTS vil den videre sende en advarsel til senderen og senderen vil utsette sending av data. [5]



Figur 3 - DACAP protokollen.

Figur 3 forklarer hva som skjer når en node skal sende en pakke til en annen node. I dette eksempelet vil node A sende en pakke til node B. Node C lytter og hører at det skjer en transaksjon mellom node A og node B og går inn i en back-off tilstand slik at den ikke begynner å sende pakker som kan kollidere med kommunikasjonen deres. Node C vet ikke avstanden til node A, og dermed venter den en tid som samsvarer med den korteste konfigurerte handshakelengden. Dette kan føre til at effektiviteten til protokollen minker ved lengre avstander mellom nodene i forhold til hvor lang tid de bruker før de sender pakker. Når node C hører at pakken er sendt fra Node A til B så vet den at de er ferdige med å kommunisere med hverandre og går tilbake til normal tilstand.

Kanalen blir reservert til den første noden som får en CTS og det er ikke noe måte å sette prioritering for forskjellige noder og datapakketyper. Dette fører til at hvis noden har en pakke liggende klar rett etter den har sendt en pakke så vil den være den første til å ta over kommunikasjonen, siden de andre nodene må gå ut av back-off tilstanden før de kan reservere området for sending av pakker. Det er også satt inn en tilfeldig back-off forsinkelsestid for hver gang handshaken mislykkes.

DACAP er en protokoll som lytter etter signaler for å unngå kollisjoner. Dette fører til at nettverket er skalerbart og at nodene fritt kan bevege på seg.

5.3 T-Lohi

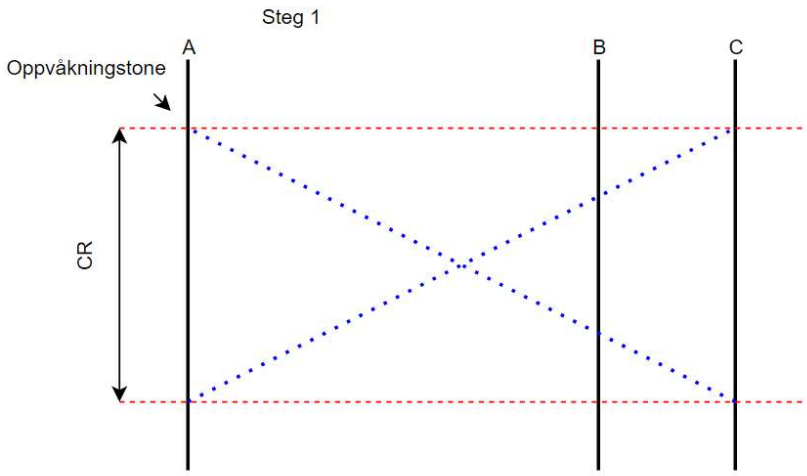
Tone-Lohi (T-Lohi) er en protokoll med hovedmål å oppnå god utnyttelse av kanalene, stabil gjennomstrømming og lavt energibruk. Protokollen har to hovedmåter den oppnår dette på. Den første er bruken av reservasjonsperioder som fører til mindre kollisjoner for sendte datapakker. Den andre er «oppvåkningstone» (OVT) som reduserer strømforbruket til modemet ved at mindre energi blir brukt til å høre etter data.

Protokollen fungerer på følgende måte:

Noder som ønsker å sende data sender en OVT til mottaker. Deretter lytter den til mediet i et gitt tidsrom kalt konkurranse/tvist-tid/contention round (CR) Hvis noden ikke hører andre OVT-er i løpet av en CR, vil den anta at mediet er ledig og begynne å sende data. Hvis den derimot hører andre OVT-er vil den trekke seg tilbake i et tilfeldig valgt tidsrom basert på hvor mange andre OVT den hørte. [6]

5.3.1 Synkronisert T-Lohi (ST-Lohi)

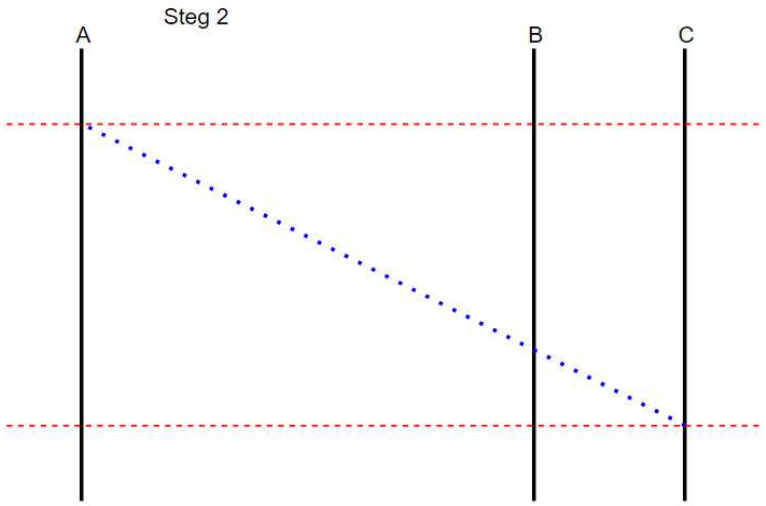
For synkronisert T-Lohi (ST-Lohi) er alle nodene synkronisert, og CR periodene er veldefinerte. Synkroniseringen har som fordel at den reduserer kollisjoner på tvers av CR. Tiden for CR er $CR_{ST} = \tau_{MAX} + T_{tone}$. Hvor τ_{MAX} er den maksimale propagasjonsforsinkelsen i nettverket og T_{tone} er tidslengden til OVT'en. Synkroniseringen har ulempen med at det kreves flere ressurser for å beholde synkroniseringen mellom nodene.



Figur 4 - T-Lohi steg 1.

For figur 4

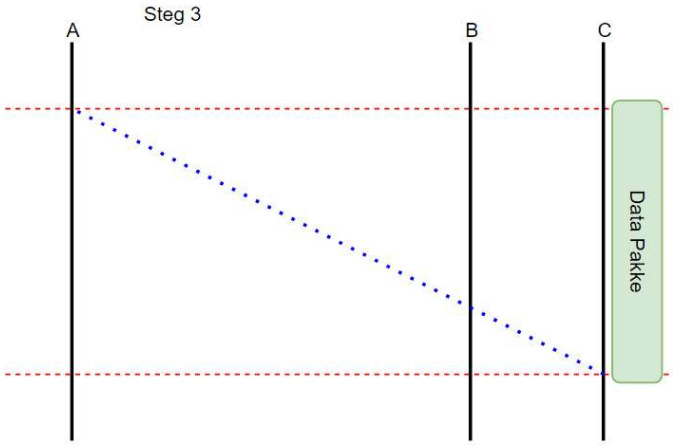
- Alle nodene starter CR på samme tidspunkt
- Nodene A og C sender oppvåkningstone
- Nodene A og C hører oppvåkningstonene til hverandre
- Nodene trekker tilbake og prøver igjen en tilfeldig valgt senere CR.



Figur 5 - T-Lohi steg 2.

For figur 5

- Node A sender oppvåkningstone
- A hører ingen andre oppvåkningstoner og reserverer dermed kanalen



Figur 6 - T-Lohi steg 3.

For figur 6

- Node A sender datapakken. [6]

5.3.2 Usynkronisert T-Lohi (UT-Lohi)

I usynkronisert T-Lohi vil nodene kunne starte reservering så fremst mediet er ledig. Hvis vi skal ha like stor garanti som ST-Lohi så må CR være dobbelt så stor, $CR_{cUT} = 2\tau_{MAX} + 2T_{tone}$. Dette tilfellet beskriver konservativt UT-Lohi (cUT-Lohi). Antall noder som ønsker å sende er ukjent, og for å redusere sannsynligheten for kollisjoner må man ta hensyn til det mest ekstreme tilfellet: at alle noder ønsker å sende samtidig og at de er lengst mulig vekke fra hverandre. Dette fører til at gjennomstrømmingen synker betraktelig. Et annet alternativ til konservativ UT-Lohi er aggressiv UT-Lohi. Den velger å se bort ifra de verste tilfellene og kutter dermed ut den ekstra tiden som cUT-Lohi legger på. Dette er som etterfølge av at de ekstreme tilfellene er såpass usannsynlige i forhold til den vanligste bruken. [6]

6 DESERT rammeverk

DESERT er et sett av C/C++ biblioteker for nettverksimuleringsprogrammet «NS-Miracle». «Network Simulator» (NS) er en diskré hendelsesimulator som kan simulere TCP-, ruting- og multicastingprotokoller over tråd og trådløs kommunikasjon. «NS-Miracle» bygger på NS ved å legge til moduler for krysslag kommunikasjon og er mer fleksibel og modulært oppbygd for å kunne legge til nye trådløse teknologier. [4] DESERT bibliotekene bygger på dette ved å legge til funksjonalitet for simulering av undervannsmiljøer og ved å implementere dedikerte undervanns protokoller.

6.1 Kjøring av simuleringer med DESERT

I DESERT installasjonskatalogen (DESERT_Underwater/DESERT_buildCopy_LOCAL/ om man fulgte installasjonsinstruksene i appendiksen) under 'bin' katalogen, ligger det en kjørbar ns (Network Simulator) fil. Denne filen kan kjøres med veien til en TCL fil som argument for å starte en simulering. TCL-koden styrer oppsettet til simuleringen og eventuell ønskelig utdata.

```
../bin ns ~/simulering.tcl
```

Figur 7 - Kjøring av simulering.

6.2 Oppsett av simuleringer i TCL fil

Her skal det sees litt nærmere på hva som kreves av innhold i TCL filen for å kjøre en simulering med DESERT. Mange detaljer rundt hva som ligger bak oppsettet i hvert steg er utelatt. Stegene er ikke ment som instruksjoner for oppsett av simuleringer, men som en pekepinn i hvordan oppsettet fungerer.

6.2.1 Innlasting av biblioteker

Ved å kjøre ns fra DESERT installasjonskatalogen har man tilgang på alle bibliotekene som er en del av DESERT. Disse lastes inn manuelt fra TCL-koden. Bibliotekene ligger lagret under 'lib' katalogen.

```
load libMiracle.so
load libMiracleBasicMovement.so
load libmphy.so
load libmmac.so
```

Figur 8 – Innlasting av DESERT biblioteker.

6.2.2 Spesifiser bruk av ns-miracle

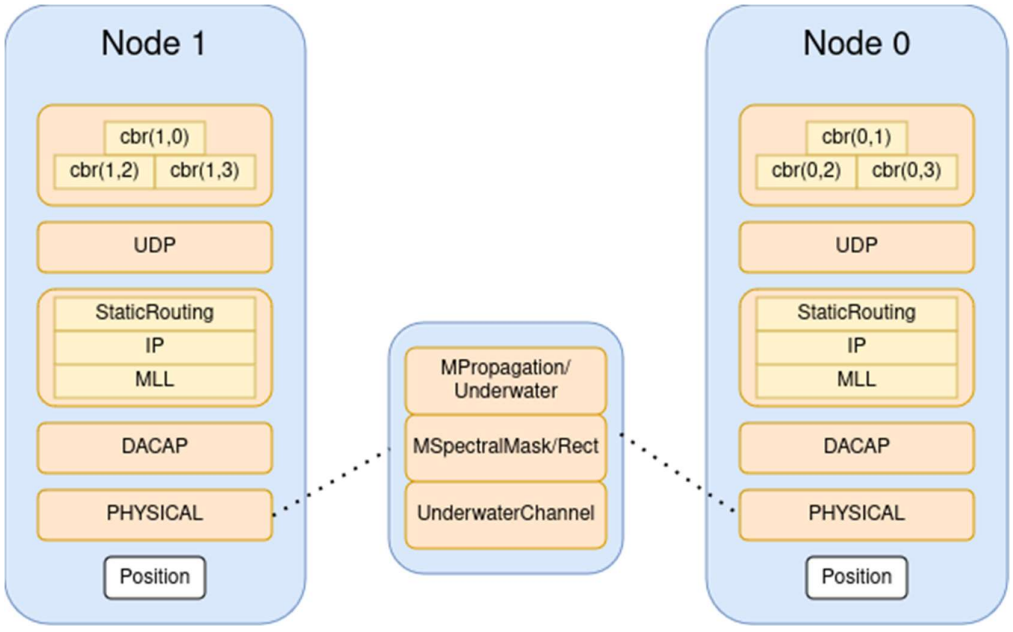
For å spesifisere at en ønsker å bruke ns-miracle må linjene i figur 9 være med. Simuleringer med DESERT rammeverket krever bruk av ns-miracle. Derfor er disse linjene et krav for å kjøre simulering av undervannsmiljøer.

```
set ns [new Simulator]
$ns use-Miracle
```

Figur 9 – Oppsett av ns-miracle simulator.

6.2.3 Oppsett av noder og kanal

For hver node i nettverket er det nødvendig å hente inn moduler fra DESERT. TCL-navnene til disse modulene er enklest å finne på dokumentasjonssiden til DESERT (https://signetlabdei.github.io/DESERT_Underwater_doc/html/index.html). Det kreves oppsett av moduler for applikasjon, transport, nettverk og data-link protokoller, samt en modul for emulering av det fysiske laget (statisk ruting for nettverkslaget trenger også en egen modul). Hver node sin modul for det fysiske laget må være assosiert med samme kanal.



Figur 10 – Nodeoppsett i DESERT.

```
set channel [new Module/UnderwaterChannel]
set propagation [new MPropagation/Underwater]
set data_mask [new MSpectralMask/Rect]
```

```
set cbr($id,$cnt)  [new Module/UW/CBR]
$node($id) addModule 7 $cbr($id,$cnt)   1  "CBR"
```

Figur 11 - TCL kode for oppsett av noder.

6.2.4 Modulkonfigurasjon

De fleste modulene har et sett med verdier som kan settes for konfigurering. Når et bibliotek lastes inn på starten av utførelsen av TCL-koden, vil en TCL-kode med standard oppsett av disse verdiene kjøres. Oversikt over disse TCL-kodene finnes i filer under kilde katalogen til DESERT.

```
Module/UW/DACAP set debug_              1
Module/UW/DACAP set seed                $opt(rngstream)
Module/UW/DACAP set max_prop_delay      $opt(maxProp)
```

Figur 12 - Modulkonfigurering

```
yonx  yonxPC  ../DESERT/data_link/uwdacap  pwd
/home/yonx/DESERT_Underwater/DESERT_Framework/DESERT/data_link/uwdacap

yonx  yonxPC  ../DESERT/data_link/uwdacap  ls
initlib.cpp  uw-mac-DACAP-alter.cpp     uw-mac-DACAP-tracer.cpp
Makefile.am  uw-mac-DACAP-alter.h
Makefile.in  uw-mac-DACAP-defaults.tcl
```

Figur 13 – Fil for standardverdioppsett i DACAP.

6.2.5 Sammenkobling av lag

Hver av lagmodulene må sammenkobles med den neste lagmodulen tilhørende samme node for at lagene kan kommunisere på tvers av hverandre. Eksempelvis kobles data-link laget opp mot det fysiske laget. Det fysiske laget til hver node kobles opp mot en felles kanal for hele nettverket.

```
$node($id) setConnection $cbr($id,$cnt)   $udp($id)   0
set portnum($id,$cnt) [$udp($id) assignPort $cbr($id,$cnt) ]
```

```
$node($id) setConnection $udp($id)   $ipr($id)   1
$node($id) setConnection $ipr($id)   $ipif($id)  1
$node($id) setConnection $ipif($id)  $mll($id)   1
$node($id) setConnection $mll($id)   $mac($id)   1
$node($id) setConnection $mac($id)   $phy($id)   1
$node($id) addToChannel  $channel    $phy($id)   1
```

Figur 14 – Sammenkobling av lag i TCL kode

6.2.6 Node posisjoner

Posisjonen til hver node i nettverket er gitt av x, y og z koordinater tilhørende et posisjonsobjekt som er assosiert med den aktuelle noden. Det er i oppgaven kun tatt for seg implementasjon av statisk posisjonsdata, men rammeverket har også støtte for posisjonsdata som forandrer seg over tid (noder som er i bevegelse).

```
set position($id) [new "Position/BM"]
$node($id) addPosition $position($id)
set posdb($id) [new "PlugIn/PositionDB"]
$node($id) addPlugin $posdb($id) 20 "PDB"
$posdb($id) addpos [$ipif($id) addr] $position($id)
```

Figur 15 - TCL kode for oppsett node posisjoner

6.2.7 Destinasjonsadresser, ARP- og rutingtabeller

Hver applikasjonslagmodul må konfigureres med destinasjonsadresser som det skal genereres trafikk mot. ARP-tabeller må fylles opp manuelt. Om det brukes statisk ruting i stedet for dynamiske rutingprotokoller, må også rutingtabellene fylles opp.

```
$cbr($i,0) set destAddr_ [$ipif(0) addr]
$cbr(0,$i) set destAddr_ [$ipif($i) addr]
$cbr($i,0) set destPort_ $portnum(0,$i)
$cbr(0,$i) set destPort_ $portnum($i,0)
$mll(0) addentry [$ipif($i) addr] [$mac($i) addr]
$mll($i) addentry [$ipif(0) addr] [$mac(0) addr]
$ipr($i) addRoute [$ipif(0) addr] [$ipif(0) addr]
$ipr(0) addRoute [$ipif($i) addr] [$ipif($i) addr]
```

Figur 16 - TCL kode for oppsett av ARP- og rutingtabeller.

6.2.8 Igangsetting av simulering og instruksjoner ved simuleringsslutt

Det er som regel ønskelig å utføre visse handlinger ved enden av simuleringen, som for eksempel utskrift av data. Disse må i så fall kjøres fra en dedikert prosedyre/funksjon. Linjene i figur 17 kreves for å sette i gang en simulering og burde være på enden av TCL filen. Merk at “finish” er navnet på prosedyren som skal kjøres ved simuleringsslutt.

```
$ns at [expr $opt(stoptime) + 250.0]  "finish; $ns halt"

$ns run
```

Figur 17 – TCL kode for igangsetting av simuleringen / Spesifisering av når simuleringen skal slutte og prosedyren som utføres ved slutten av simuleringen.

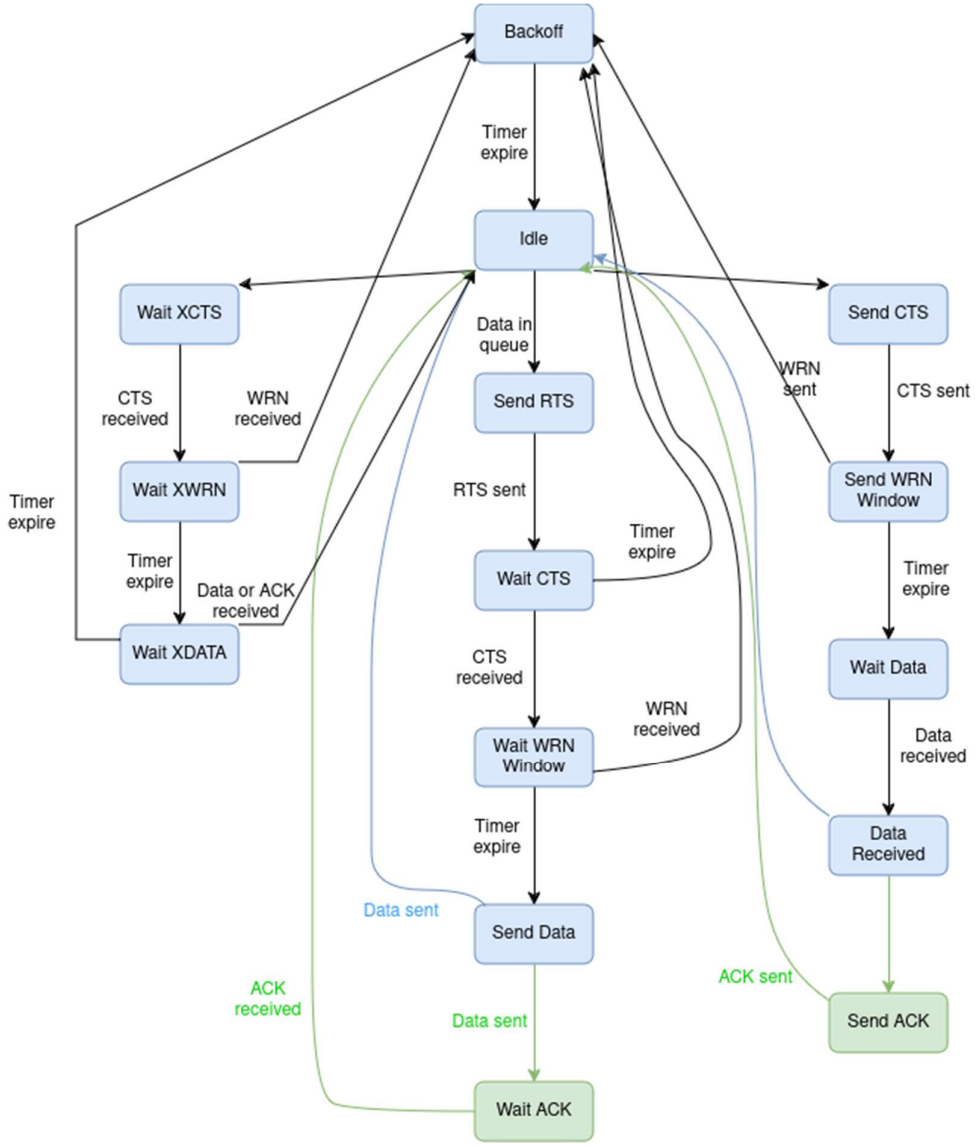
6.3 DESERT sin implementering av undervannsprotokoller

Hvordan en protokoll er implementert i DESERT kommer ikke nødvendigvis tydelig frem i noe av protokolldokumentasjonen. Gjennom studering av debug utdata fra simuleringer og fra å se på oppsettet til protokollene i TCL-koden, kan man allikevel danne seg et bilde av hvordan dette er gjort.

6.3.1 Implementering av DACAP

Når det kommer til implementasjonen av DACAP i rammeverket er det første som blir tydelig fra å studere debug utdataen at hver node kan befinne seg i forskjellige tilstander. Hvordan hver node responderer på visse hendelser er basert på hvilken tilstand den befinner seg i. Figur 18 viser et tilstandsdiagram som forklarer implementasjonen.

Fra samme utdata ser man at timere kan gå ut for å utløse hendelser. Lengden på disse timerne er basert på verdier som T_max (konfigurert maksimal propagasjonsforsinkelse) og T_min (minimums handshakelengde). Det er ikke gitt ut fra beskrivelsen av protokollen om T_max og andre verdier må settes manuelt eller om protokollen skal muliggjøre automatisk og/eller dynamisk tilegning. Det viser seg at DESERT kun har funksjonalitet for statisk manuelt konfigurert T_max. Viktigheten av å konfigurere denne riktig vil bli sett nærmere på i 7.2.



Figur 18 – Tilstandsdiagram for DACAP i DESERT.

6.3.2 Implementering av Aloha

DESERT har flere forskjellige implementasjoner av Aloha protokollen. Den implementasjon som skal brukes videre i oppgaven er Aloha med CSMA funksjonalitet. CSMA funksjonalitet i dette tilfellet vil kun si at protokollen lytter etter signaler i en gitt tid før den starter sending av data. Utenom dette er det ikke noe spesielt å nevne rundt implementasjonen av Aloha i DESERT.

7 Simuleringer

Denne delen vil ta for seg de forskjellige simuleringene som er gjort og resultatene de har gitt. Forskjellige simuleringer er gjort med forskjellige formål. Simuleringene i 7.2 er gjort for å i bedre grad forstå både rammeverket og implementasjonen av DACAP protokollen. Simuleringene i 7.3 er gjort for å kartlegge ytelsen til protokollene i varierende omstendigheter.

7.1 Utdatavariabler

For alle simuleringene som skal kjøres kommer det til å bli sett på følgende utdatavariabler:

Pakkeleveringsratio:

Pakkeleveringsratio vil, under simuleringene i denne oppgaven, bli kalkulert som totalt antall pakker levert delt på totalt antall pakker generert i applikasjonslaget. Pakkeleveringratioen vil med andre ord ikke gjenspeile antallet pakker som er forsøkt sendt over den fysiske kanalen. En pakkeleveringratio på 100% vil derfor ikke bety at det ikke har forekommet noen kollisjoner. En kan også ha null kollisjoner og allikevel ha en pakkeleveringsratio på under 100%. Denne måten pakkeleveringsratioen er kalkulert på er meget viktig å ta hensyn til når man analyserer resultatene fra simuleringene.

Antall pakker mottatt:

Antall pakker mottatt regnes som alle pakker registrert mottatt i applikasjonslaget hos alle noder i nettverket. I simuleringene som er gjort her vil dette si alle pakkene som er mottatt i applikasjonslaget til sinken.

Antall noder:

Antall noder er telt som det totale antallet noder i nettverket, ekskludert sinken, som ble brukt i simuleringen.

CBR Poisson periode:

CBR Poisson periode, målt i sekunder, er gjennomsnittet i tid mellom hver pakke som ble generert i applikasjonslaget hos hver node. Denne variabelen reflekterer, sammen med antall noder, trafikkmengden i nettverket. Mindre CBR Poisson periode gir større trafikkmengde.

Pakkestørrelse:

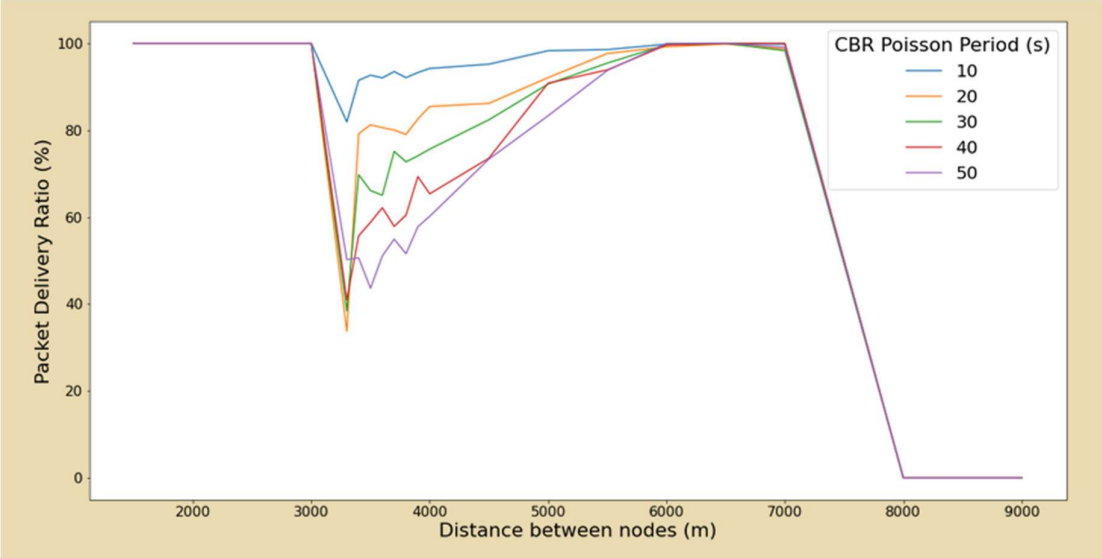
Pakkestørrelsen i byte som ble brukt i simuleringen. Dette er mengden data per pakke som ble generert i applikasjonslaget og inkluderer ikke headers som blir lagt på i lavere lag.

7.2 Innledende simuleringer

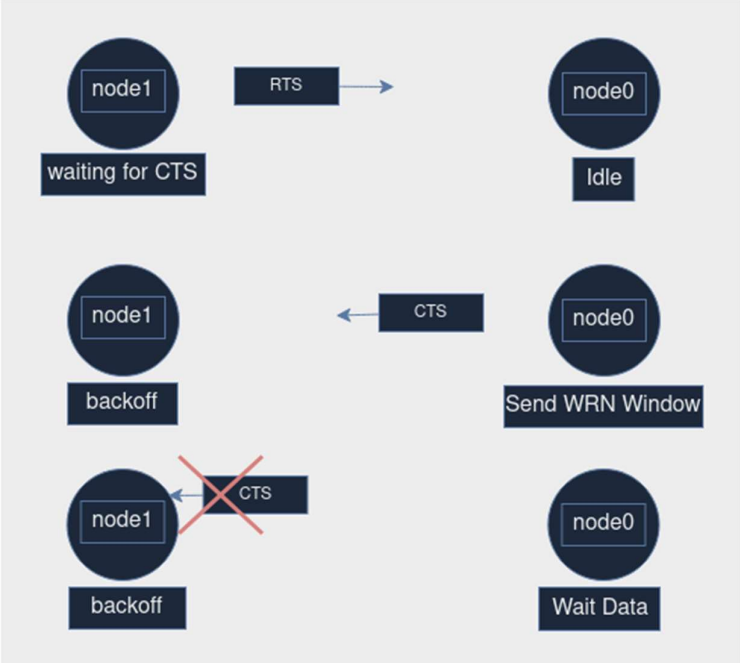
Innledningsvis i arbeidet var det klart at det fantes et stort behov for å bedre forstå hvordan rammeverket fungerte og hvordan undervannsprotokollene var implementert. De tidligste simuleringene var derfor dedikert til disse formålene. De viktigste funnene fra disse simuleringene vil bli formidlet her.

7.2.1 Viktigheten av riktig konfigurert maksimal propagasjonsforsinkelse i DACAP

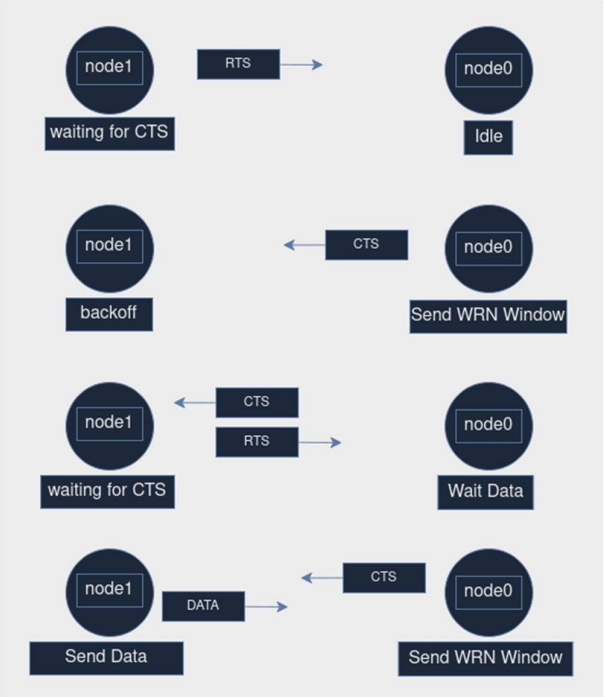
Ved å bruke standardkonfigurasjonen av DACAP-modulen i DESERT vil man merke at funksjonaliteten til protokollen svikter på større avstander. Simuleringer der kun én node sender trafikk mot sinken viser dette tydelig. Figur 19 viser en graf over pakkeleveringsratioen i et slik scenario over forskjellige avstander. Det er et plutselig fall i pakkeleveringsratioen ved rundt 3500 meters avstand, men denne går opp igjen ved enda litt større avstander. Debug-utdataen til simuleringen gir en idé av hvorfor dette skjer. Diagrammet i figur 20 viser kommunikasjonsforløpet ved en avstand mellom nodene på 3500 meter. Man observerer at avsendernoden for tidlig bytter tilstand fra “Wait CTS” til “Back-off”, som fører til at den forkaster CTS-pakken. Dette forklarer derimot ikke direkte oppgangen i pakkeleveringsratio når avstanden blir skrudd videre opp. Figur 21 viser kommunikasjonsforløpet ved en avstand mellom nodene på rundt 5000 meter. Her ser vi at oppgangen i pakkeleveringsratio er grunnet at avsendernoden sender ut en ny RTS før mottakerens CTS kommer frem. Dette gjør at avsender allikevel vil være i “Wait CTS” tilstanden ved mottak av CTS-pakken. Problemet her er at standard konfigurert maksimal propagasjonsforsinkelse (T_max) blir brukt. Man kan se fra filen uw-mac-DACAP-defaults.tcl at T_max har en standardverdi på 2 sekunder. Denne må konfigureres manuelt til en stor nok verdi for å unngå problematikken nevnt ovenfor.



Figur 19 - DACAP simulering med 2 noder med økende avstand fra hverandre



Figur 20 – Diagram over DACAP kommunikasjon ved avstand på 3500m.

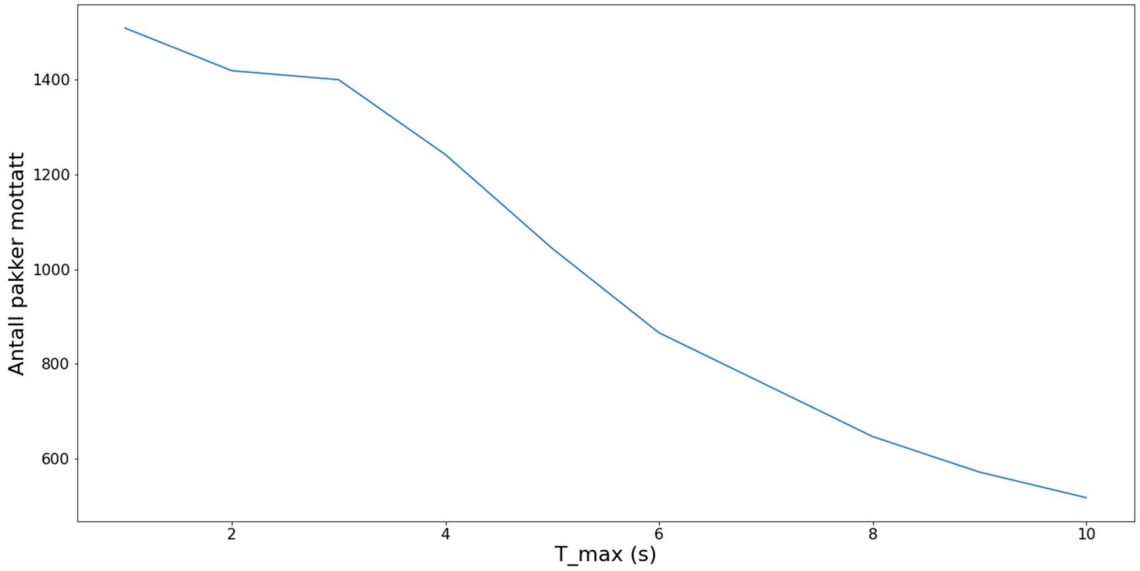


Figur 21 – Diagram over DACAP kommunikasjon ved avstander over 3500m.

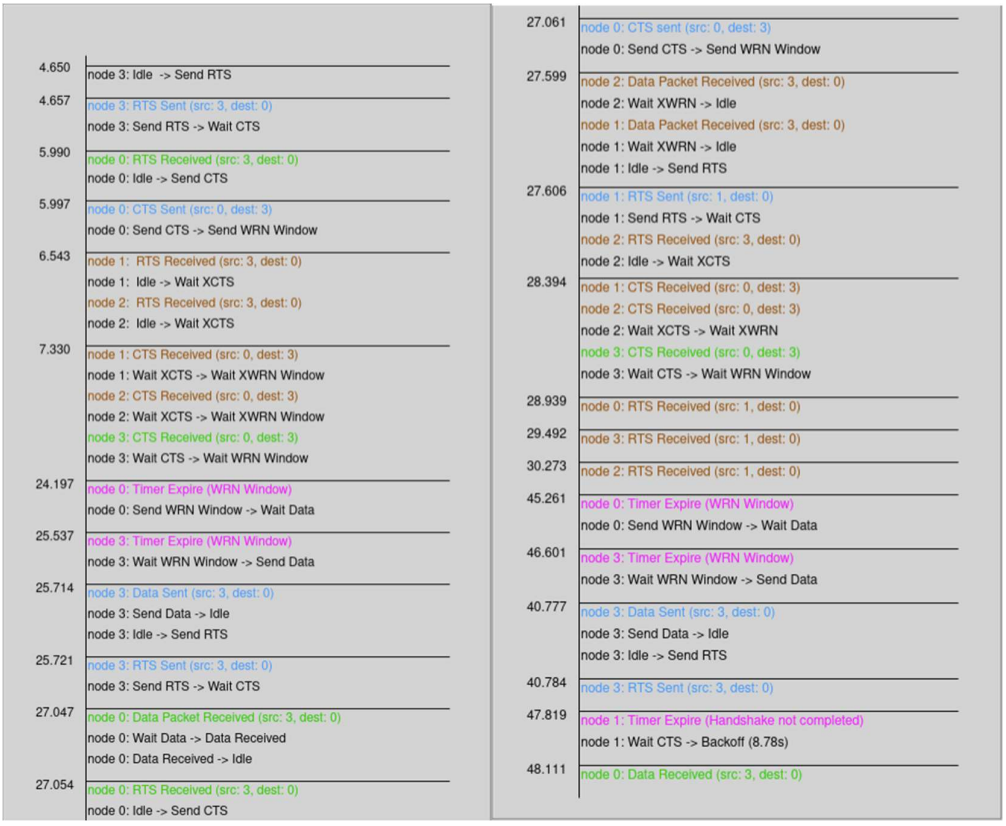
7.2.2 Effekten maksimal propagasjonsforsinkelse har på nettverksytelsen

Konfigurert maksimal propagasjonsforsinkelse (T_max) avgjør tiden en node i nettverket venter fra den sender en RTS-pakke til den går i 'Back-off' tilstand, og tiden fra den mottar en CTS-pakke til den starter sending av data (advarselsvinduet). Det er nødvendig med en T_max som er minst like stor som den faktiske maksimale propagasjonsforsinkelsen i nettverket, men man observerer at den maksimale mulige nettverksytelsen går ned jo høyere T_max er. Dette er grunnet at advarselsvinduet blir større og nettverkskommunikasjonen derfor tar lengre tid. Figur 22 viser nedgangen i nettverksytelse når T_max blir større. Figur 23 viser en tidslinje over kommunikasjonen i nettverket med T_max satt til 10 sekunder.

DACAP implementasjonen i DESERT har kun funksjonalitet for en statisk T_max og det er derfor nødvendig å manuelt konfigurere T_max slik at den reflekterer den faktiske maksimale propagasjonsforsinkelsen i forkant av nettverkets oppstart. Det er mulig at det kan være hensiktsmessig med en dynamisk T_max i nettverk der den maksimale propagasjonsforsinkelsen endres over tid (der nodene beveger seg).



Figur 22 - Graf over antall pakker mottatt når T_max stiger.



Figur 23 – Debug log fra simulering med høy konfigurert T_max.

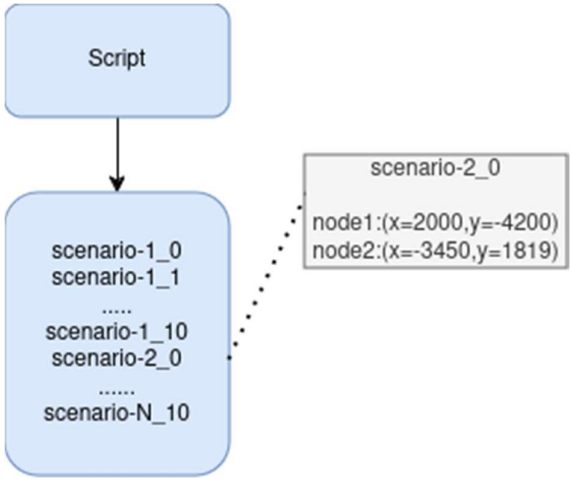
7.3 Simuleringer for analyse av protokollytelse

Hver av undervannsprotokollene har sine fordeler og ulemper. Det skal gjennomføres en stor mengde simuleringer med forskjellige parametere og scenarioer for å i best mulig grad kartlegge ytelsen til protokollene.

7.3.1 Generering av scenarioer

Det vil bli brukt en stor mengde scenarioer basert på én enkel modell. Figur 24 viser en skisse av denne modellen. Modellen baserer seg på én sink med n-antall noder tilfeldig plassert 500-5000 meter rundt seg. Alle nodene har lik dybde. Gjennom et enkelt Python-script genereres det 10 forskjellige scenarioer for hvert antall noder fra 1 opp til 15. Scenarioene er lagret som tekstfiler med nodenes posisjoner.

Figur 24 - Scenario med 1 til 15 noder plassert 500m til 5000m fra sink



Figur 25 – Generering av scenarioer

7.3.2 Parametere

Når det kom til valg av verdier på parameterne som skulle brukes i simuleringene var det først og fremst viktigst å bestemme hvilke parametere som endres på tvers av forskjellige simuleringer. Det ble bestemt at hvert scenarioene nevnt i 7.3.1 skulle simuleres med hver kombinasjon av følgende verdier:

Pakkestørrelse (i byte): 100, 300, 500, 1000, 2000
CBR Poisson Periode (i sekunder): 10, 20, 40, 80, 120, 200, 300

Med 5 forskjellige pakkestørrelser og 7 forskjellige trafikkmengder vil det blir gjort 35 ($7 \times 5 = 35$) simuleringer for hvert scenario. Det ble valgt å ikke endre på noen av de fysiske parameterne på tvers av simuleringene.

Andre parametere var like på tvers av alle simuleringene. Under finnes en liste med relevante verdier som er satt.

Fysisk lag (Module/UW/PHYSICAL):

Bitrate:	4800 bps
Båndbredde:	5000 Hz
Frekvens:	25000 Hz
Mottaksgrense:	4 dB
SNR Penalty:	0 dB
Overføringskraft:	180 dB re µPa

[Aloha] Datalink lag (Module/UW/CSMA_ALOHA):

Lyttetid:	0.5 s
Maks antall forsøk:	5

[DACAP] Datalink lag (Module/UW/DACAP):

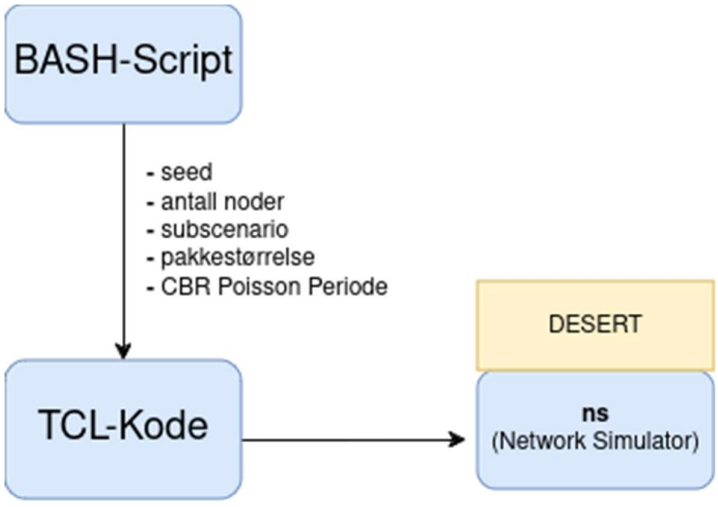
T_max:	3.5 s
T_min:	2.0 s
Lydfart under vann:	1500 m/s
Maks antall forsøk:	5
Seed:	1234
ACK:	på

Applikasjon lag(Module/UW/CBR):

Poisson trafikk:	på

7.3.3 Utførelse av større mengder simuleringer

Siden hver protokoll skal simuleres med mange forskjellige parametere i mange forskjellige scenarioer er det et behov for å automatisere disse simuleringene da det ville vært for ressurskrevende å utføre dem manuelt. Det ble av denne grunn utarbeidet et BASH-Script som kunne kjøre gjennom alle disse simuleringene og fortelle TCL-koden hvilke verdier og hvilket scenario den skal ta i bruk.



Figur 26 – Diagram som gir et overblikk av hvordan simuleringene blir utført.

7.3.4 Behandling av utdata

Det er viktig at utdataen fra simuleringene blir hentet ut og lagret på en strukturert måte. BASH-Scriptet henter derfor ut den nødvendige dataen fra simuleringene ved hjelp av programmet grep og enkle regulære uttrykk (RegEx). Dataen blir derifra lagret til en csv fil.

For hver simulering blir følgende relevant data lagret til csv filen:

-CBR Poisson Periode

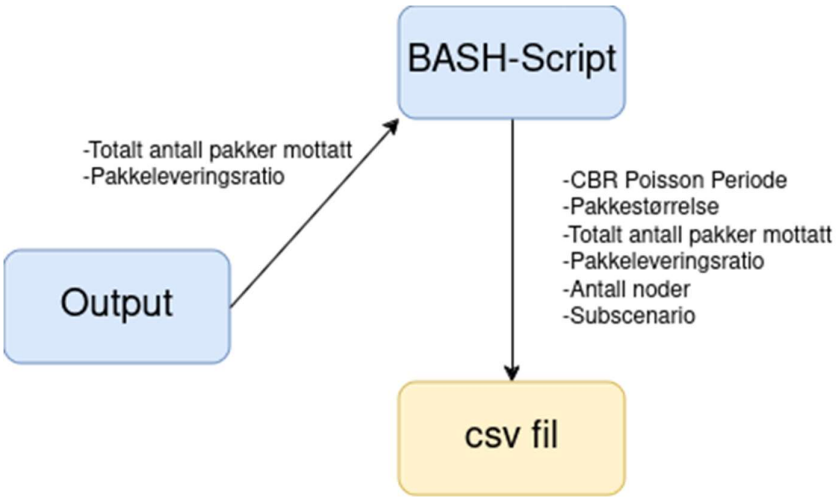
-Pakkestørrelse

-Totalt antall pakker mottatt

-Pakkeleveringratio

-Antall noder

-Subscenario



Figur 27 - Diagram over simulasjons prosessen

8 Analyse av simuleringene

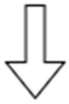
8.1 Behandling av simuleringsdata

Simuleringene ga innholdsrik utdata, men det finnes et behov for å putte det mer i system. Først og fremst vil vi ta snittet av alle subscenarioene som har identiske andre verdier. For hver kolonne med data er det også ønskelig å kalkulere gjennomsnittlig gjennomstrømmingsrate i b/s per node. Denne verdien finner vi ved følgende formel:

$$\left(\frac{Antall\ pakker\ mottatt \times Pakkestørrelse \times 8}{Antall\ noder \times Simulerings\ tid}\right)$$

Dataen er deretter klar til videre bruk for plotting og analyse.

CBR Poisson Periode	Pakestørrelse	Totalt antall pakker mottatt	Pakkeleverings-ratio	Antall noder	Subscenario
20	100	817	91.5	3	0
20	100	791	87.3	3	1
...	...	...	...	...	...
20	100	831	89.4	3	9



CBR Poisson Periode	Pakestørrelse	Totalt antall pakker mottatt	Pakkeleverings-ratio	Antall noder	Gjennom-strømming
20	100	813	89.4	3	21.68

Tabell 1 – Behandling av simuleringsdata.

8.2 Hva skal analyseres?

Gjennom analyse av simuleringsdataen er det ønskelig å kartlegge styrkene og svakhetene til hver protokoll, samt å se hvordan protokollene presterer opp imot hverandre. Hovedpunktene for analyse er som følger:

-Effektiviteten til DACAP med hensyn til pakkestørrelse

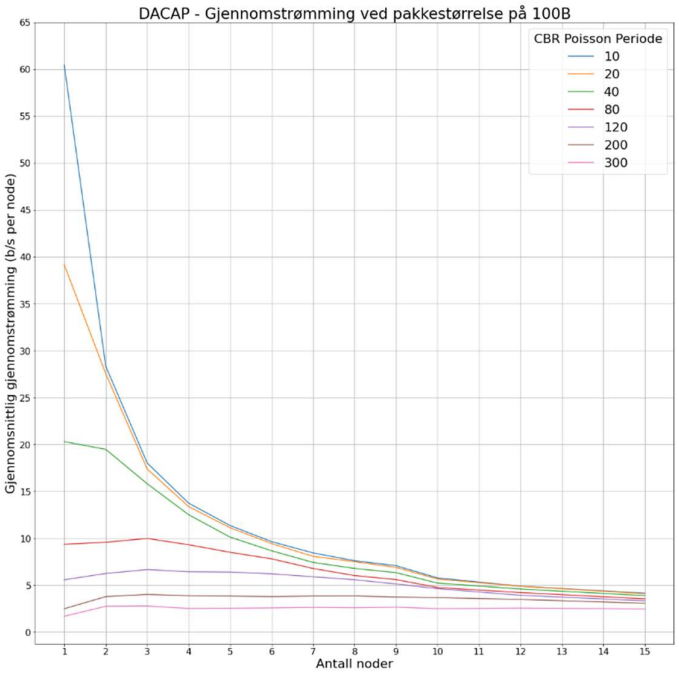
-Ytelsen til Aloha ved forskjellige trafikkmengder

-Ytelsen til DACAP versus ytelsen til Aloha

8.3 Analyser

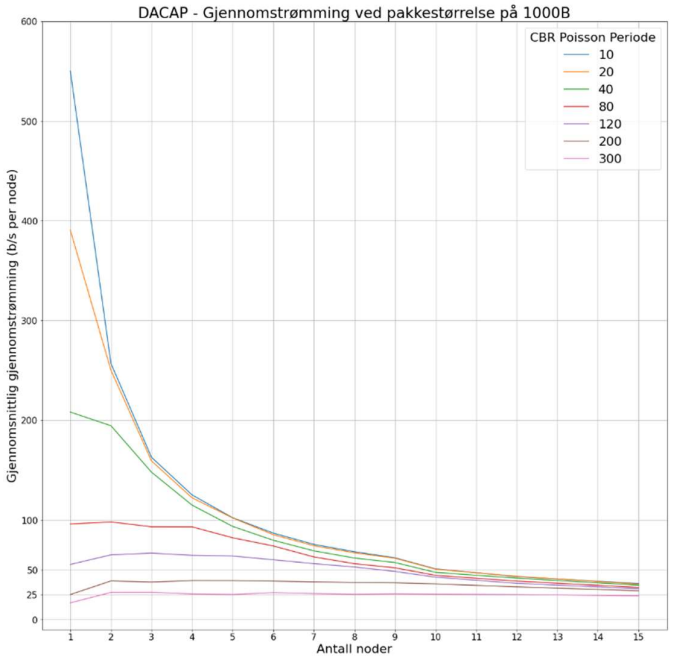
8.3.1 DACAP ytelse med forskjellige pakkestørrelser

Maksimal propagasjonsforsinkelse i nettverket er variabelen som i størst grad begrenser hvor stor trafikkmengde (målt i antall pakker) nettverket har kapasitet til når DACAP er i bruk. Det gir derfor god mening at det kan brukes en mye større pakkestørrelse uten at pakkeleveringsratioen blir påvirket i for stor grad. Fra figur 28 og 29 viser det seg at gjennomstrømmingen nesten tidobler seg når pakkestørrelsen endres fra 100B til 1000B. Dette virker å understøtte hypotesen om at pakkestørrelsen kan skrus opp uten å ha en stor påvirkning på pakkeleveringratioen. Videre viser figur 30 og figur 31 at pakkeleveringsratioen holder seg relativt stabil når pakkestørrelsen skrus opp.



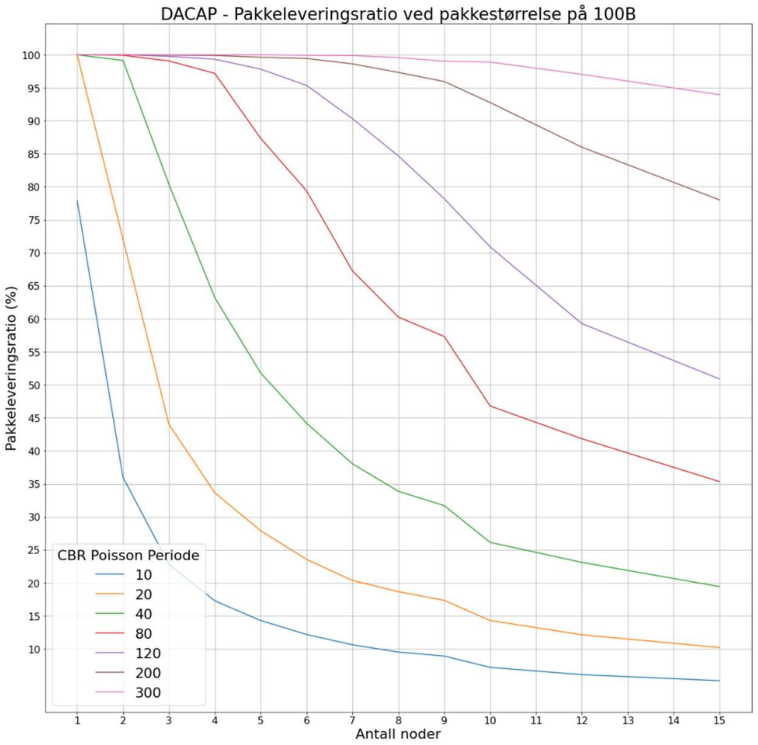
Figur 28 – Graf over gjennomstrømming per node for forskjellig antall noder

DACAP | Pakkestørrelse: 100B



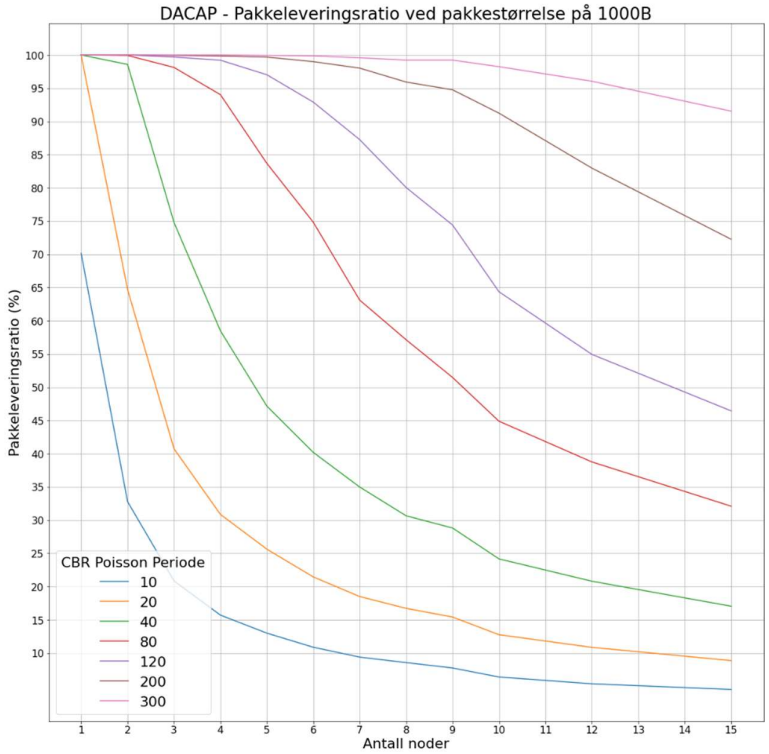
Figur 29 – Gjennomstrømming per node for forskjellig antall noder

DACAP | Pakkestørrelse 1000B



Figur 30 – Pakkeleveringsratio for forskjellig antall noder

DACAP | Pakkestørrelse 100B

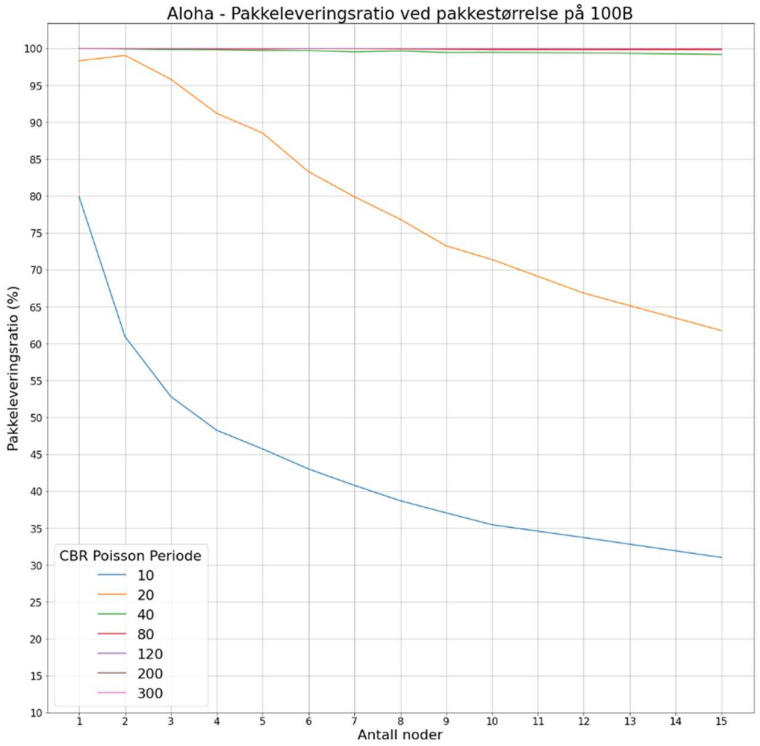


Figur 31 – Pakkeleveringsratio for forskjellig antall noder

DACAP | Pakkestørrelse 1000B

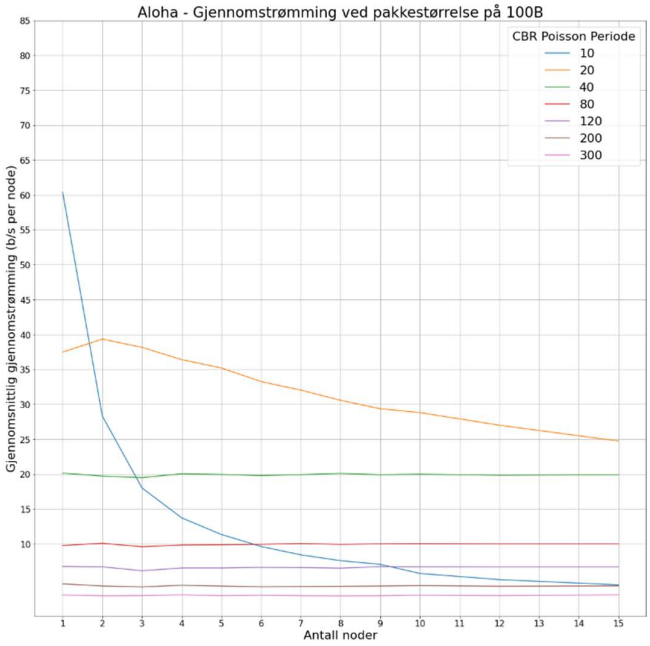
8.3.2 Aloha (med CSMA) sin ytelse over forskjellige trafikkmengder

Grunnet at Aloha sender data uten å vite sikkert om kanalen er ledig vil sannsynligheten for kollisjoner av datapakker være vesentlig større enn hos andre protokoller. Det er allikevel gode muligheter for at Aloha presterer godt i nettverk med mindre trafikk. Figur 32 figur 33 viser at Aloha presterer godt selv ved større trafikkmengder. Det er verdt å nevne at måten pakkeleveringsratioen kalkuleres på, som nevnt i 7.1, ikke forteller oss mye om hvor mange kollisjoner som har inntruffet. Det vil si at det finnes store muligheter for en høy mengde kollisjoner ved større trafikkmengder. Dette er problematisk med tanke på energibruken til sensorene.



Figur 32 – Pakkeleveringsratio for forskjellig antall noder

Aloha | Pakkestørrelse: 100B



Figur 33 – Gjennomstrømming for forskjellig antall noder

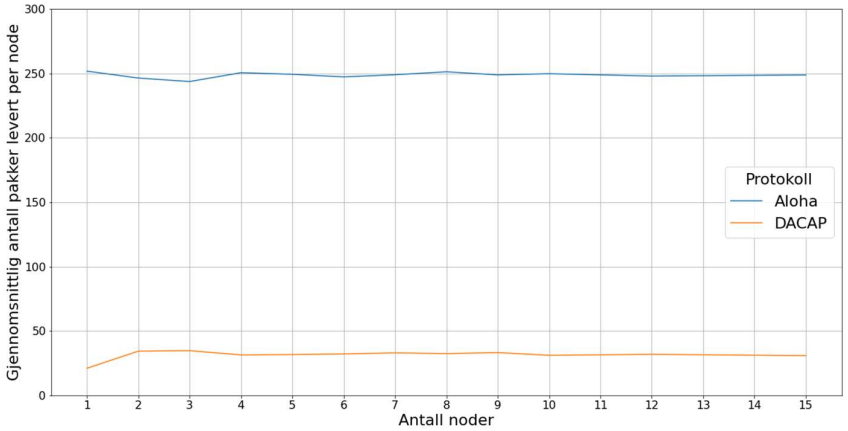
Aloha | Pakkestørrelse: 100B

8.3.3 Ytelsen til Aloha versus ytelsen til DACAP

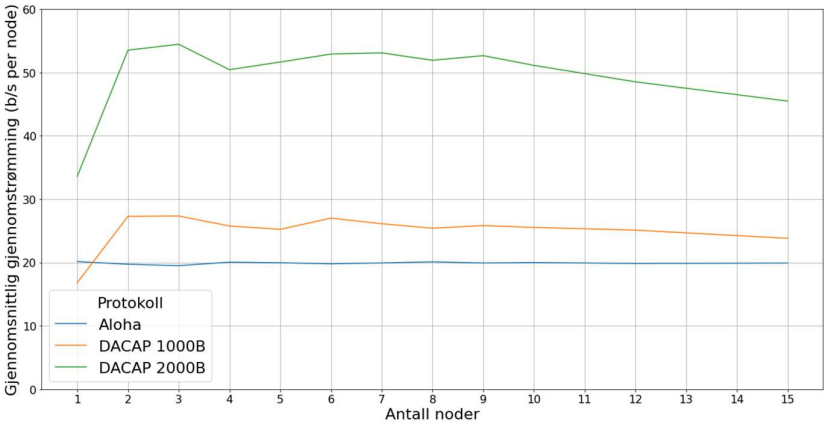
Selv om det er mange andre faktorer som spiller inn når det kommer til hvor godt en undervannsprotokoll presterer, er det fortsatt relevant å se på hvor mye data hver protokoll faktisk evner å distribuere. Protokollene skal her måles opp mot hverandre både når det kommer til gjennomstrømming i b/s per node og i totalt antall pakker levert per node. Det kommer til å velges ut en av de største trafikkmengdene fra hver protokoll der pakkeleveringsratioen er oppimot 100%.

I første omgang skal det sees på antall pakker levert per node. Det er valgt ut simuleringsdata fra DACAP med en pakkestørrelse på 100B og en CBR Poisson periode på 300 sekunder. Fra Aloha er det valgt ut data med en pakkestørrelse på 100B og en CBR Poisson periode på 40 sekunder. Figur 34 viser at Aloha er i stand til å levere nesten 10 ganger så mange pakker per node. Dette er forventet da Aloha trives best med små pakkestørrelser og DACAP trives best med store pakkestørrelser.

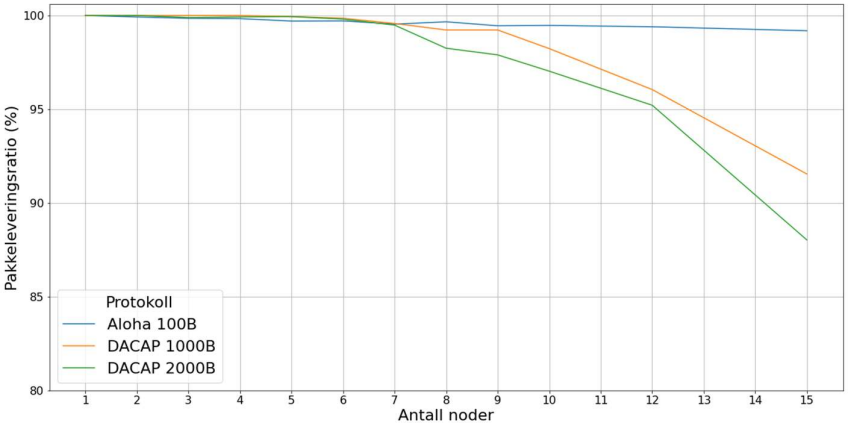
For å måle gjennomsnittlig gjennomstrømming per node brukes det den samme dataen fra forrige steg for Aloha. Dette gjøres fordi man ikke vil oppnå noen nevneverdig oppgang i gjennomstrømming ved større pakkestørrelser med Aloha, samt at det vil føre til flere kollisjoner. For DACAP skrus pakkestørrelsen opp til 1000B og 2000B, mens det blir beholdt samme CBR Poisson periode på 300 sekunder. Man ser fra figur 35 at DACAP med større pakkestørrelser er i stand til å levere større gjennomstrømming enn Aloha. Dette er spesielt tydelig med pakkestørrelser på 2000B. Man ser allikevel fra figur 36 at pakkeleveringsratioen for DACAP går ned jo flere noder som blir introdusert i nettverket.



Figur 34 – Gjennomsnittlig antall pakker levert
DACAP | CBR Poisson Periode: 300s
Aloha | CBR Poisson Periode: 40s



Figur 35 – Gjennomsnittlig gjennomstrømming
DACAP | CBR Poisson Periode: 300s
Aloha | CBR Poisson Periode: 40s



Figur 36 – Pakkeleveringsratio
DACAP | CBR Poisson Periode: 300s
Aloha | CBR Poisson Periode: 40s

9 Diskusjon/Refleksjon

Arbeidet startet med relativt optimistiske målsetninger der det ble siktet inn mot å få til alt det oppgavebeskrivelsen nevnte, pluss mer. Den opprinnelige fremdriftsplanen ble utarbeidet med hensyn til disse målsetningene. Det ble etter hvert tydelig at man trengte å forstå rammeverket og protokollene bedre for å bruke dataen på en pålitelig måte. Dette var utfordrende og tidskrevende da gruppen ikke klarte å finne tilfredsstillende dokumentasjon. På dette punktet ble det tydelig at fremdriftsplanen ville bli vanskelig å holde. Det ble ikke utarbeidet noen ny fremdriftsplan, men det ble bestemt at det skulle fokuseres på simulering av MAC-protokoller og at man mest sannsynligvis måtte droppe å se på rutingprotokoller. Til slutt endte det også opp med at det ikke ble foretatt noen omfattende simuleringer med T-Lohi, da det ikke var tilstrekkelig med tid til å sette seg inn i hvordan denne protokollen var implementert i DESERT. Det ble ikke puttet noe arbeid inn i å utarbeide idéer for ny protokollfunksjonalitet, igjen grunnet mangel på tid. Mye av arbeidet har også gått til å sette opp systemer for kjøring og analysering av simuleringer utenfor DESERT rammeverket, i form av Python kode og Bash scripter.

Det var tiltenkt at de vedlagte TCL, Bash og Python filene skulle være strukturert kommentert og enkle å sette seg inn i. Dette endte ikke opp med å være tilfellet og alle filene er lite kommentert, kaotisk satt opp og inneholder en del overflødig kode. Disse burde leses med hensyn til dette.

Sett tilbake på ting, med den lærdommen man nå har, er gruppen egentlig fornøyd med arbeidet som ble utrettet og tror og håper at arbeidet som er lagt inn kan brukes som utgangspunkt for videre arbeid på temaet.

10 Konklusjon

Oppgaven har tatt for seg gruppens erfaringer med hvordan DESERT rammeverket brukes, teorien bak utvalgte undervannsprotokoller og resultater fra simuleringer gjort med de utvalgte protokollene.

Det foreligger et håp om at det som er skrevet om DESERT, i tandem med TCL filen som ligger vedlagt denne oppgaven, skal fungere som et godt supplement for lesere som prøver å sette seg inn i bruk av rammeverket. Det som er skrevet om rammeverket er i stor grad skrevet som informasjon gruppen selv kunne ønske at fantes når arbeidet på oppgaven ble startet.

Ideelt sett hadde det vært diskutert i større detalj koblingene med hvordan DACAP protokollen er forklart i kapittel 5.2 og forklaringen om hvordan DESERT implementer den i 6.3.1. Resultatene fra 7.2 burde også vært trukket inn i. Ved videre arbeid på temaet anbefales det å bruke informasjonen fra de nevnte kapitlene til videre analyse på nøyaktig hvordan DACAP er implementert i rammeverket og eventuelle mangler eller forbedringspunkter som finnes.

Resultatene fra simuleringene gir en god pekepinn på ytelsen til protokollene når det kommer til antall pakker mottatt og gjennomstrømming i b/s. I Aloha simuleringene er det fortsatt et stort behov for å kartlegge antall kollisjoner som inntreffer for å vite hvilke trafikkmengder protokollen er passende for. Fra simuleringene gjort med DACAP kan man i utgangspunktet anta at det ikke har forekommet noen kollisjoner med datapakker ut ifra hvordan protokollen fungerer.

Oppgaven kan ikke vise til noen gode analyser på kraftforbruket til noen av protokollene som er testet ut. Siden dette er det kanskje viktigste aspektet når det kommer til vurderingen av protokollene, burde dette være et av hovedfokusene om oppgaven brukes som utgangspunkt for videre arbeid rundt temaet.

11 Referanser

[1] HVL, «Om Høgskulen på Vestlandet,» HVL, 29 12 2023. [Internett]. Available: https://www.hvl.no/om/. [Funnet 16 01 2024].

[2] HVL, «Forskning,» HVL, 29 12 2023. [Internett]. Available: https://www.hvl.no/forsking/. [Funnet 16 01 2024].

[3] SFI SmartOcean, «About the centre,» Norwegian centre for research-based innovation, [Internett]. Available: https://sfismartocean.no/about/. [Funnet 23 01 2024].

[4] R. Masiero, A. Saiful, F. Favaro, M. P. G. Toso, F. Guerra, P. Casari og M. Zorzi, «Publications: DESERT Underwater,» 05 2012. [Internett]. Available: http://telecom.dei.unipd.it/media/download/385/. [Funnet 23 01 2024].

[5] M. S. Borja Peleato, «Distance Aware Collision Avoidance Protocol for Ad-Hoc Underwater Acoustic Sensor Networks,» IEEE COMMUNICATIONS LETTERS, Vol. 11, NO 12, 2007.

[6] A. A. Syed, W. Ye og J. Heidemann, «T-Lohi: A new Class of MAC Protocols for Underwater Acoustic Sensor Network,» Proceedings - IEEE INFOCOM, Phoenix, AZ, USA, 13-18 April 2008.

[7] S. Jiang, «State-of-the-Art Medium Access Control (MAC) Protocols for Underwater Acoustic Networks: A Survey based on a MAC reference Model,» IEEE Communications Survey & Tutorials, vol. 20, NO. 1, 2018.

[8] TCL Developer Xchange, «About tcl: Features and Benefits,» [Internett]. Available: https://www.tcl.tk/about/features.html. [Funnet 23 April 2024].

[9] github, «nsmiracle github repository,» signetlabdei, Mars 2024. [Internett]. Available: https://github.com/signetlabdei/nsmiracle. [Funnet 2 April 2024].

[10] ISI USC, «The Network Simulator 2 - ns-2,» Information Science Institute Univeristy of Southern California, [Internett]. Available: https://www.isi.edu/nsnam/ns/. [Funnet 2 April 2024].

[11] Univeristy of Padova, «git DESERT_Undewater,» 29 Januar 2024. [Internett]. Available: https://github.com/signetlabdei/DESERT_Underwater?tab=BSD-3-Clause-1-ov-file. [Funnet 13 Februar 2024].

[12] University of Padova, «Installation Guidelines,» [Internett]. Available: https://signetlabdei.github.io/DESERT_Underwater_doc/html/INSTALL1.html. [Funnet 12 Mars 2024].

Appendiks A Forkortelser og ordforklaringer

ACK	ASCII-kode 0x06, brukes som bekreftelse
ARP	Address Resolution Protocol
BASH	Bourne Again Shell
bps	Bites per second
CBR	Constant Bit Rate
CR	Contention Round
CSMA	Carrier-Sense Multiple Access
csv	Comma Separated Values
CTS	Clear to Send
dB	Desibel
DESERT	Design, Simulate, Emulate and Realize Testbeds
GUI	Graphical User Interface
HVL	Høgskulen på Vestlandet
Hz	Hertz
MAC	Media Access Control
NS2	Network Simulator 2
OS	Operativsystem
OSI	Open systems Interconnections
OVT	Oppvåkningstone (wake up tone)
RTS	Request to Send
SFI	Senter for Forskningsbasert Innovasjon
TCL	Tool Command Language
UIB	Universitetet I Bergen
UW	Underwater
WAN	Wide Area Network

Appendiks B Prosjektledelse og styring

B.1 Prosjektorganisasjon

Arbeidsfordeling:

-Alle i gruppen skal bidra på hver del av prosjektet.

-Under studering av diverse kommunikasjonsprotokoller vil hvert gruppemedlem ta for seg hver sin protokoll og skrive utdypende om denne.

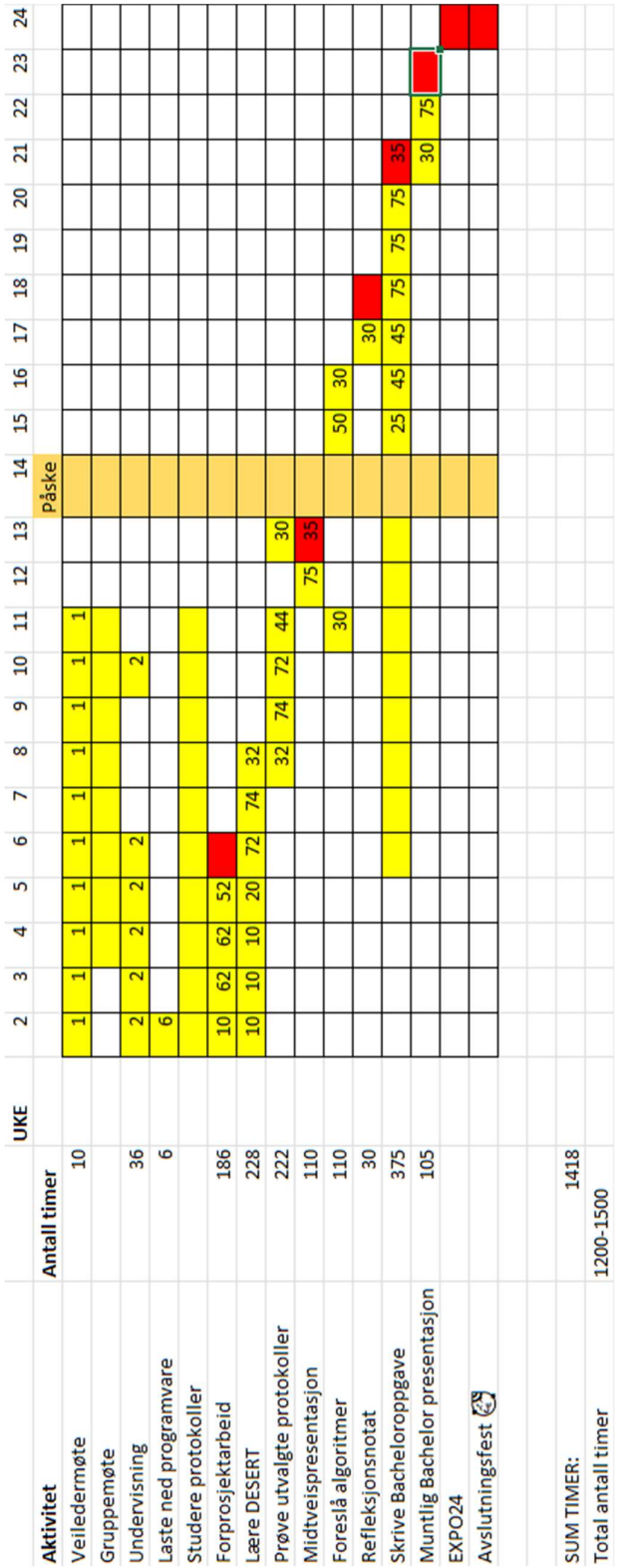
Prosjektleder:

Jens Hoff Quirk

B.2 Prosjektform

Vi planlegger å gå trinnvis gjennom punktene i prosjektet. Påfølgende deler av prosjektet vil som oftest avhenge av gjennomføring av tidligere deler og det vil derfor være naturlig å gjøre det på denne måten. Tempoet i prosjektet er planlagt å følge fremdriftsplanen ganske tett. Gruppen har som et mål å legge felles planer for utførelsen av de forskjellige delene av prosjektet på en slik måte at hvert medlem sitt individuelle arbeid godt kan integreres med de andre medlemmenes arbeid.

B.3 Fremdriftsplan



Aktivitet	Antall timer	UKE	2	3	4	5	6	7	8	9	10	11	12	13	Påske	14	15	16	17	18	19	20	21	22	23	24
Veiledermøte	10		1	1	1	1	1	1	1	1	1	1														
Gruppemøte																										
Undervisning	36		2	2	2	2	2				2															
Laste ned programvare	6		6																							
Studere protokoller																										
Forprosjektarbeid	186		10	62	62	52																				
Lære DESERT	228		10	10	10	20	72	74	32																	
Prøve utvalgte protokoller	222								32	74	72	44		30												
Midtveispresentasjon	110												75	35												
Foreslå algoritmer	110											30					50	30								
Refleksjonsnotat	30																		30							
Skrive Bachelorоppgave	375																25	45	45	75	75	75	35			
Muntlig Bachelor presentasjon	105																						30	75		
EXPO24																										
Avslutningsfest																										
SUM TIMER:	1418																									
Total antall timer	1200-1500																									

Appendiks C Brukerdokumentasjon

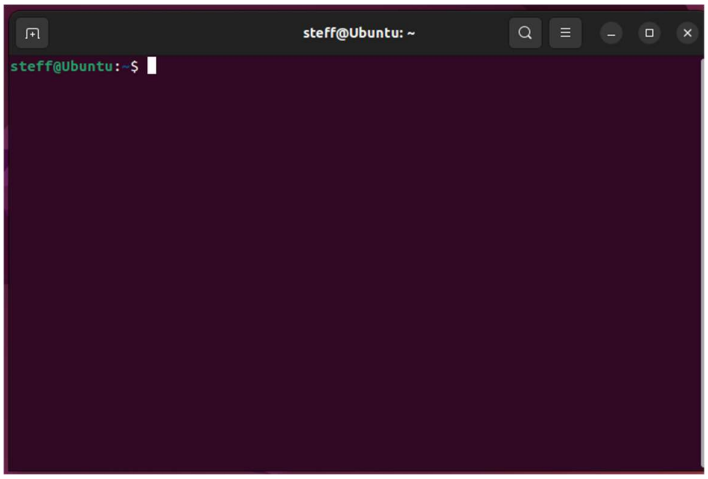
C.1 Brukerdokumentasjon

DESERT er et rammeverk som krever en debian basert linux OS for å kjøre. Vi har valgt å bruke Ubuntu med en virtual machine. Det oppsto noen problemer med tidligere versjoner av Ubuntu, det er anbefalt å bruke den nyeste versjonen.

C.2 Installering av DESERT

Installeringen er på engelsk for å gjøre det lettest for flest folk å kunne forstå installasjonsprosessen. Det gjør det også lettere å forstå siden kommandoer også er på engelsk. Denne installasjonen ble gjort i Ubuntu.

Assuming you have a new clean install of Ubuntu (as of writing, Version 22.04.3 LTS) you will be doing most of your installations on your terminal. Commands you need to write in the terminal are highlighted with yellow.



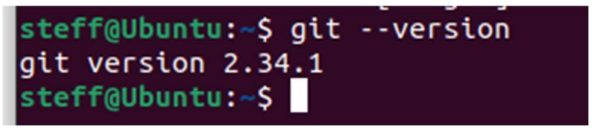
Installing git

To download files from git repositories you will first need to download git packages. Default packages are enough for downloading DESERT.

1. You can check if you currently have git downloaded by typing this command into the terminal:

   git --version

And you should receive an output like this:



2. If you don't have git, update your local package index:

   sudo apt update

   once you have updated, proceed to install Git:

   sudo apt install git

Installing the DESERT framework

With git you can now download the latest version of DESERT from their repository, you will need to install some more packages to use it.

You will need the following packages:

- build-essential
- autoconf
- automake
- libxmu-dev
- libx11-dev
- libxmu-headers
- libxt-dev
- libtool
- gfortran
- bison
- flex

Install them all with this command:

sudo apt-get install build-essential autoconf automake libxmu-dev libx11-dev libxmu-dev

libxmu-headers libxt-dev libtool gfortran bison flex

```
steff@Ubuntu:~$ sudo apt-get install build-essential autoconf automake libxmu-dev libx11-dev
libxmu-dev libxmu-headers libxt-dev libtool gfortran bison flex
```

Download DESERT files from git using this command:

git clone -b master https://github.com/signetlabdei/DESERT_Underwater.git

Locate the folder "DESERT_Underwater" using the terminal cd commands and cd into the framework folder:

cd DESERT_Underwater/DESERT_Framework

```
steff@Ubuntu:~$ cd DESERT_Underwater/DESERT_Framework
steff@Ubuntu:~/DESERT_Underwater/DESERT_Framework$ ls
at.tar.gz           install.log              ns-2.34.tar.gz          woss-1.11.0.tar.gz
commonFunctions.sh  install.sh               nsmiracle-1.1.2.tar.gz  woss-1.12.4.tar.gz
commonVariables.sh  ltmain.sh                nsmiracle-1.1.4.tar.gz  woss-1.12.5.tar.gz
DESERT              netcdf-4.2.1.1.tar.gz    otcl-1.14.tar.gz        woss-1.7.0.tar.gz
Docker              netcdf-4.7.3.tar.gz      patches.tar.gz          zlib-1.2.11.tar.gz
hdf5-1.14.2.tar.gz  netcdf-4.9.2.tar.gz      README.md               zlib-1.2.7.tar.gz
hdf5-1.8.13.tar.gz  netcdf-cxx-4.2.tar.gz    tcl-8.4.19.tar.gz
Installer           netcdf-cxx4-4.3.1.tar.gz tclcl-1.20.tar.gz
steff@Ubuntu:~/DESERT_Underwater/DESERT_Framework$
```

(You can check the contents of the folder you currently are in by typing in the command "ls")

set the executable flag of the installer and run it:

chmod +x ./install.sh

./install.sh –wizard

The wizard will guide trough the installation and choosing options such as your destination folder and optional addons.

To start using DESERT, set the environment variables:

source DESERT_Underwater/environment

Once the source is set you can run tcl files to start simulating.

Appendiks D Vedlagte filer

TCL fil for simuleringsoppsett:

dacapStaticScenario.tcl

BASH-scripts for kjøring av større mengder simuleringer:

bigSimDACAP.sh

bigSimAloha.sh

Python filer for plotting av data (Brukt med Jupyter Notebook):

bigSim.py

DACAP_ForskjelligeAvstander.py

propagasjonforsinkelseEffektPåYtelse.py [7]

Csv filer med simuleringsresultater:

dacapBigSimData.csv

alohaBigSimData.csv