




APRIL 17, 2023

# SIMPLEPERF RAPPORT

STUDENT ID = S351922  
Oslo Metropolitan University



1	INTRODUCTION	2
2	SIMPLEPERF	2
3	EXPERIMENTAL SETUP	6
4	RESULTS AND DISCUSSION	FEIL! BOKMERKE ER IKKE DEFINERT.
4.1	Network tools	7
4.2	Performance metrics	7
4.3	Test case 1: measuring bandwidth with iperf in UDP mode	7
4.3.1	Results	Feil! Bokmerke er ikke definert.
4.3.2	Discussion	8
4.4	Test case 2: link latency and throughput	8
4.4.1	Results	8
4.4.2	Discussion	9
4.5	Test case 3: path Latency and throughput	10
4.5.1	Results	10
4.5.2	Discussion	11
4.6	Test case 4: effects of multiplexing and latency	11
4.6.1	Results	11
4.6.2	Discussion	13
4.7	Test case 5: effects of parallel connections	13
4.7.1	Results	13
4.7.2	Discussion	13
5	CONCLUSIONS	13
6	REFERENCES	14

# 1 Introduction

I dette prosjektet skal jeg utvikle en simpleperf-verktøy som delvis får noe av funksjonalitet iperf. Iperf er nettverktøy som brukes til å måle båndbredde, hastighet, tap av pakker mellom to enheter i et IP-basert nettverk. Det kan også brukes til å teste både i TCP og UDP protokollene. Iperf kan brukes både intern i et lokalt nettverk og eksternt mellom to forskjellige nettverk, og dette gjør iperf til et nyttig verktøy for nettverksytelse testing og optimalisering.

Min Simpleperf-implementasjon vil tilby noe av de funksjonalitet som iperf, og brukerne vil kunne benytte seg av visse argumenter for å teste nettverksytelse. Disse argumentene vil inkludere mulighet til å bestemme mengden av data som skal sendes, eller å velge bestemme tid for overføringen. Brukerne vil også ha tilgang til å bestemme om tilkoblingen skal vær parallell eller enkeltstående.

Måten simpleperf vil fungere på er at det vil være en server som alltid er på, og hvis en eller flere klienter kobler seg til, vil server akseptere forespørsml og skrive ut at klient med IP og Port er tilkoblet. Fra klient side skal det sendes mye data som mulig, men disse dataene skal være i pakker på 1000 bytes. Etter at dataene er overført, vil det bli skrevet ut på både klient og server siden hvor mye data klient har sendt og hvor mye data server har fått.

Til slutt skal vi kjøre fem tester for å måle båndbredde og sjekke kobling forsinkelsen og hastighetsytelse på nettverket. Vi vil også undersøke effekten av multipleksing og parallell kjøring på våre båndbredder.

## 2 Simpleperf

Simpleperf er et verktøy som kan brukes for å utføre ytelsestesting, hastighet og kommunikasjonsprotokoll på nettverket mellom klient og server. Simpleperf kan kjøres på forskjellige data maskiner og fungerer ved at klient sender data til server over en TCP-protokoll og deretter måle hastighet og pålitelighet. Simpleperf består av tre hoveddeler: simpleperf.py, server.py og client.py. Jeg skal forklare hvordan de delene fungere.

### 2.1 simpleperf.py

Simpleperf.py er hoved delen i simpleperf. Denne delen integrerer funksjonalitet til klient og server. Simpleperf.py bruker argparse-modulen for å analysere argumenter og parameter som gis av brukerne, og starter enten server eller klient modus avhengig av hva argumentet er gitt. Ved hjelp av argparse kan brukerne velge ulike alternativer for å velge hvordan simpleperf skal fungere.

Simpleperf bruker en såkalte socket-modulen til å tillate kommunikasjon mellom server og klient over en TCP-protokoll. Klient sender data ved hjelp av send ()-funksjon og mottar data ved hjelp av recv ()-funksjon, etter all data er sendt, vil klient sende en 'bye' til server for å si at alt data er sendt. Server mottar data ved hjelp av recv ()-funksjon og sender en 'AKT' tilbake til klient ved hjelp av send ()-funksjon og avslutter tilkoblingen.

#### 2.1.1 checkport()

Simpleperf tar noen funksjoner og checkport () er en av dem. Checkport funksjonen tar inn en parameter port. Først konverterer parameteren port til heltall og sjekker om port nummeret mellom 1024 og 65535, og hvis ikke så kaster funksjon en ValueError med en feil melding. Hvis port tallet er gyldig vil funksjonen returnere tallet, hvis port taller er ugyldig vil da kastes en argparse.ArgumentTypeError med feil melding. I koden min bruker jeg ValueError når portnummeret er utenfor gyldig området og argparse.ArgumentTypeError når brukerne skriver en ugyldig port nummer.

### 2.1.2 check\_positive ()

Når dette funksjon kalles så sjekkes om parameteren er positivt heltall. Funksjonen tar en parameter (positive) og hvis tallet ikke er positivt heltall vil det kastes en ValueError ellers vil det returnere parameteren. For eksempel hvis jeg kaller funksjonen med 'abc' vil det kaste ValueError med feilmelding som er angitt, og hvis jeg kaller med '5' vil det returnere tallet 5.

### 2.1.3 check\_ip ()

Dette funksjon tar også inn parameteren (ip). For å kunne bruke dette må vi importere ipaddress, deretter vil funksjonen bruke ipaddress-modulen til å validere IP-adressen. Hvis IP-adress er gyldig vil funksjonen returnere IP-adress og hvis IP-adress er ugyldig vil det kaste en argparse.ArgumentTypeError med en feilmelding.

### 2.1.4 main()

Main () funksjonen i simpleperf.py er hoved funksjonen som styrer klient og server. Denne funksjon bruker argparse-modulen for å ta imot argumenter og analysere kommandolinje-argument på. Avhengig av hva arg-parse kommando er vil main() funksjonen parser og tolker argumentene og kaller disse funksjonene.

I main funksjonen har vi mange args-argumenter som lar brukerne å velge mellom, og kan velge mellom dem fritt. Verdien på disse argumenter kan velges fra kommando linje og hvis det blir ikke gitt noe verdi har de en standart verdi. Vi må huske at argparse er en modul i Python, for å kunne bruke dette må vi importere det først.

For at brukerne skal få mulighet til å kjøre simpleperf i server modus må man bruke,

```
python3 simpleperf.py -s ----- for server
```

For at brukerne skal få mulighet til å kjøre simpleperf i klient modus må man bruke,

```
python3 simpleperf.py -c -----for klient
```

i kommando linje. Hvis brukerne bruker -s sammen med -c vil sendes en feil melding at det er ikke lov å bruke -s sammen med -c. Må nevne det også at det er krav å bruke -s for å kjøre på server modus og -c for klient modus, hvis ingen av disse flaggene velges vil det sendes en feil melding og avsluttes programmet.

Hvis brukerne skriver i kommando linje den første kommandoen som har -s flagget vil vi kalle socket\_connection() med argumenter args.bind, args.port og args. På den måte lager vi en socket server som venter for en eller flere tilkoblinger fra klient.

For å indikere klient modusen må man bruke den kommandoen med flagget -c. Da kaller vi en rekke argumenter som args.server\_ip, args.port og args. Klienten tilkobles til simpleperf-serveren via en bestemt ip-adress og port nummer. Deretter vil klient sende mye data til serveren på 1000 biter, i en bestemt tid eller bestemt mengde data.

## 2.2 server modus

Server.py delen i simpleperf vil alltid være på og venter til klient og mottar data fra klient, server.py bruker argparse-modulen for å analysere kommandoen som angis fra brukerne. I server delen har jeg to funksjoner som jeg beskriver.

### 2.2.1 socket\_connection():

Denne funksjon operetter en socket i sever modusen, og tilkobler dette til bestemt IP-adresse og portnummeret som er gitt fra brukerne. Server modusen skal alltid lyte på alle inngående forbindelser ved hjelp av `s.listen()`, og etter tilkoblingene fra klient vil det opprettes en klientsocket ved hjelp av `loop`.

Siden serveren vår skal håndtere parallelt tilkobling, derfor lages det ny tråd for hver klient tilkoblingen ved hjelp av `thrading.Thread()`. Hver ny tråd som kommer, vil kjøre `handle_client`-funksjonen jeg har, og tar imot adressen til klient, hvilket format data kommer, hvilket argument har klienten, og data størrelsen.

### 2.2.2 `handle_client()`

Denne funksjon blir kalt fra `socket_connection` og har fem parametere. `Clientsocket` parameteren håndterer kommunikasjon. `Address` inneholder IP og port nummeret. `Data_size` er størrelsen på data som kommer til server, og `args` er argumentene som blir brukt i funksjon.

Dataene som kommer inn i server er det på `Byte` format derfor brukes dette parametere for å omgjøre data på andre enhet ved oppførsel.

Når klient tilkobles til server, vil det skrives ut en melding at klienten med bestemt IP og port nummer er tilkoblet. Deretter har vi en løkke som får data fra klient på 1000 biter så lenge det er data, denne løkke vil vente å motta data til den får en 'good bye' melding. Dette vil bety at klienten har sendt alt data, deretter vil server sender en bekreftelse og avslutter kommunikasjon.

Funksjon også bregner total tiden som ble brukt under tilkoblingen, totale mengden av data som server har mottatt og båndbredden. Til slutt skriver funksjonen resultatene av forbindelsen, som IP og port-nummeret til klient, intervalltiden, total data mottatt, og hastighet til tilkoblingen.

Jeg må nevne det også at det er dette funksjon som aksepterer flertrådighet (multithreading) tilkoblingen. Flertrådighet teknikk tillater flere tråd til å kjøre samtidig i en enkelt prosess, denne teknikken tillater bedre utnyttelse av ressurser slik at en oppgave kan deles i mindre deler og utføres parallell, men noen gang kan flertrådighet skaffe utfordring som å ha kontroll hvordan trådene er fordelt.

## 2.3 Klient modus

Klient modusen er den andre hoved delen i socket-programmet som oppretter TCP tilkobling til server. Deretter vil klient sende data til server i forhold til hva argumentene kommer fra kommando linje. For å kunne bruke klient modusen må man skrive i kommandoen.

```
python3 simpleperf.py -c
```

Via denne modusen har brukerne mulighet til å sende og motta data til serveren. Det er viktig å merke seg at serveren må være på og vi må koble oss til riktig IP og port nummeret, ellers vil vi få `Connection refused` feil melding. For å kunne koble seg til riktig server kan man bruke dette kommando

```
python3 simpleperf.py -c -I <server-ip> -p <server-port>
```

Klienten tar argumenter, og jeg har valg å lage funksjon for hvert argument som kommer fra kommando. Dette har jeg gjort for bedre forståelse og enkelt oppbygning for koden min.

### 2.3.1 `connect_client ()`

Etter at vi får tilgang til klient modusen, er det dette funksjon som har ansvar for å tilkoble til server ved å operette en TCP-socket. Denne funksjon tar to argumenter som er `server_ip` og port nummeret,

disse argumentene brukes for å koble seg til server. Dette funksjon bruker socket.socket() for å opprette ny socket og s.connect() til å tilkoble til server.

Hver av args-argumentet vil kalle dette funksjon for tilkobling til server. I tillegg har jeg en print som gir en beskjed at tilkoblingen er opprettet med hvilken port og IP address. Sist vil jeg nevne det også at hvis funksjonen skal ikke fungere som den skal vil vi få en exception melding.

### 2.3.2 run\_interval ()

Denne funksjon kobler seg til server ved å kalle connect\_client () – funksjon. For å kunne å tilgang til denne funksjon må vi bruke i kommandoen.

```
python3 simpleperf.py -c -I <server-ip> -p <server-port> -i <interval>
```

Hoved mål for denne funksjon er å sende data i et gitt tidsintervall. Funksjonen tar fem argumenter server\_ip, port, timer, format og args. Timer argument vil definere hvor lenge vil klient sende data. format gir data størrelsen, og args indikerer hvilket argument kommer fra kommando.

I funksjonen har jeg brukt en løkke som sjekker om det har gått et intervall ved 'args.interval' sekunder, hvis det har det bregner funksjonen hvor mye data som ble sendt i hvert intervallet, i tillegg printer jeg antall intervaller og båndbredde.

Til slutt skrives det ut en ny print der det står totale data mengde som ble sendt og totale tiden som ble brukt og båndbredde for hele tidsintervallet. Funksjonen avsluttes ved å kalle last () funksjonen.

### 2.3.3 run\_num ()

Denne funksjon bruker også connect\_client() funksjon til å koble til server og bruker args som argument. For å kunne få til dette funksjon bruker vi dette kommando

```
python3 simpleperf.py -c -I <server-ip> -p <server-port> -n XMB
```

Hoved målet med dette funksjon er å sende en bestemt data mengde til server. Funksjonen sjekker først om data typen som kommer fra kommandoen er riktig, hvis ikke vil den sende en feil melding ValueError.

Hvis det er riktig størrelse vil funksjonen først dele data i to deler og konvertere data til bytes og sende data til serveren igjen med s.send(), her brukte jeg en løkke som hjelper å være tilkoblet så lenge data sendes. Til slutt vil funksjonen skrive ut antall bytes som er sendt, på hvilket format og hvor lang tid ble det brukt. Funksjonen avslutter med å kalle last() funksjonen min.

### 2.3.4 run\_parallel()

Denne funksjon bruker også connect\_client() funksjon til å koble til server og bruker args som argument. For å kunne få til dette funksjon bruker vi dette kommando

```
python3 simpleperf.py -c -I <server-ip> -p <server-port> -P X
```

Denne funksjon starter en flere tråder tilkobling til å sende data til server parallelt. Denne funksjon lager en tom liste med tråder og bruker en løkke til å iterere over antall ønskende tilkoblinger som kommer fra kommando, og legger dem inn i listen vår. For hver tråd kjøres funksjonen paralel\_send\_data (). Denne funksjon øker hastighet og gir mulighet til å sende store data ved å bruke flere tråder.

### 2.3.5 paralel\_send\_data ().

Denne funksjon blir kalt for hver trådd av parallele funksjon, og tar fire argumenter, og kobles til server ved hjelp av connect\_client(). Funksjonen gir mulighet til å sende data fra flere tråder til server

samtidig ved å bruke flertråding. Funksjonen starter en løkke og sender data på 1000 bytes størrelsen så lenge vi er under tid argumentet. Etter løkken er ferdig vil funksjon skrive ut til skjermen overføringen med angitte typen, båndbredde, og til slutt kalle på last()-funksjon.

### 2.3.6 run\_default ()

Hvis det kommer intet spesifikt argument fra kommandoen vil det kjøres denne funksjon, kan på en måte si at denne funksjon lager en normal tilkobling. Denne funksjon bruker også args som argument og connect\_client() for å koble til server. Kommando ville se som ut

```
python3 simpleperf.py -c -----(kjøres på local host)
```

brukerne kan godt velge spesifikk IP og port nummer også om det ønskes. Funksjon bruker en løkke og sende mye data på 1000 bytes størrelse så lenge vi er under argument tiden. Etter at funksjonen er ferdig med løkken vil det skrives ut til skjermen tiden, mengden av data, data typen og båndbredde. Til slutt kaller på last()-funksjonen og avsluttes.

### 2.3.7 client\_connection()

Funksjonen tar parameteren server\_ip, port og args, som jeg har nevnt bruksområde for disse. Funksjonen kalles fra main() funksjonen vår som er i simpleperf. Denne funksjon utfører en rekke handlinger forhold til hva args argumentet kommer fra kommando.

### 2.3.8 Tillegg funksjon

conver(): denne funksjon tar to parametere total\_bytes som er størrelsen i bytes på data og format som er ønskede data enhet. Hoved mål for denne funksjon er å konvertere data fra en enhet til annen enhet. Hvilken enhet skal det være kan velges av brukerne. Kan brukes slik i kommando

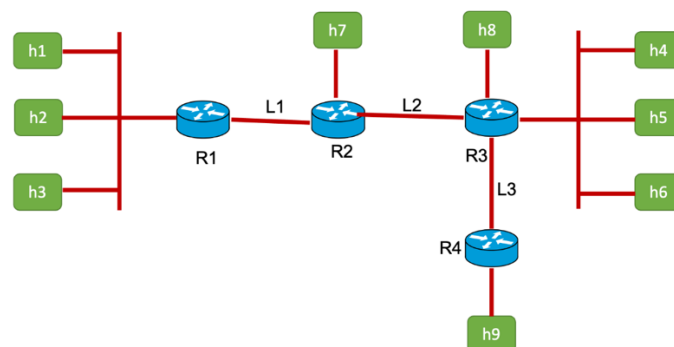
```
python3 simpleperf.py -c -I <server-ip> -p <server-port> -f X ----data enhet
```

unit er en ordbok so tar fire enheter, hver enhet har en størrelse som konverterer data på ønsket enhet. Deretter deler vi dataen vår på factor, factor er den ønskede enhet. Til slutt returnerer funksjon data avrundede på 2 desimaler, og ønske enheten.

last(): etter at klienten sendt alt data vil sende en 'good bye'-melding til server. Denne melding vil gi beskjed til server at klient er ferdig med data, og server vil sende tilbake en bekreftelse på dette. hvis klient ikke får respons og det oppstår noe feil vil det skrives en feil melding, og hvis respons tilsvarer forventning vil på klienten skrive en 'good bye!' og avslutter til koblingen ved s.close().

## 3 Experimental setup

På starten av prosjektet har vi fått en topologi som viser hvordan nettverket er bygd opp, som vi ser



De grønne fire kantene er vertene som er fra h1 --- h9, altså 9 verter. De blåene er rutene som er R1, R2, R3 og R4. L1, L2 og L3 viser tilkoblingene mellom rutene. Denne topologien vil jeg bruke til resten av reporten min. Vi har fått at L1 har båndbredde på 40Mbps og latency på 10ms, L2 har båndbredde på 30Mbps og latency på 20ms og L3 har båndbredde på 20Mbps og latency på 10ms,

## 4 Performance evaluations

I ytelse vurderingen skal jeg kjøre noen tester og ser på hvor godt oppfyller vår kravene. Her skal jeg bruke noe verktøy for å måle ytelsen.

### 4.1 Network tools

Jeg har brukt verktøyene iperf, simpleperf og ping for å utføre forskjellige tester. Siden vi vet hvordan iperf og simpleperf fungerer vil jeg også si litt om hva ping er. Ping er et verktøy som brukes til å teste tilgjengelighet og responstiden til en annen på et nettverk. For eksempel, når vi bruker ping-verktøy, sender vi pakke til en andre enhet og venter vi på respons. Hvis alt fungerte som det skal vil vi motta responsmelding som viser tiden for data ble sendt og returnert.

### 4.2 Performance metrics

I denne delen skal jeg måle hastighet, pålitelig, ytelsebregning og forsinkelsestid ved å bruke mitt verktøy.

### 4.3 Test case 1: measuring bandwidth with iperf in UDP mode

I denne testen skal vi sende data from Client to Server og samtidig måle båndbredden ved å bruke UDP (User Datagram protocol). UDP protokoll betyr at data levering er ikke pålitelig. Siden vi får ikke noe bekreftelse på mottak av data derfor kan det gå tapt, dupliser, eller ankom i annen rekkefølge av data enn det ble sendt. Derfor er det viktig å velge riktig størrelsen for å unngå tapt av data, når vi velger for -b. For å unngå tapt av vi kan sende lite data, men på denne måten har vi ikke brukt den totale kapasitet som nettverket har. Vi har en buffer-size og hvis vi velger -b tallet som er ikke langt unna fra buffer-size da mister vi enormt lite data på vei. Hvordan skal vi vite kapasitet til nettverket, så har vi fått allerede en topologi som viser båndbredden. På topologi står det hvor mye kapasitet finnes mellom rutene og her skal vi se om vi får det vi ønsker eller ikke. Hvis vi ikke får det vi ønsker, skal vi undersøke årsaken.

#### 4.3.1 H1---H4

I topologi har vi fått båndbredde og det jeg forventer her er å sende 30Mb data og ikke miste noe av det, la oss se hva vi får i resultat.

Antall test	Valgt verdi	båndbredde	mistet data
1	30Mb	29.2 Mbits/sec	7.1%
2	29Mb	29.2 Mbits/sec	3.9%
3	28Mb	29.2 Mbits/sec	0.57%
4	27Mb	28.3 Mbits/sec	0.0%

Jeg måtte kjøre flere tester for å finne ut den riktig mengde data jeg kan sende med 0% mist. På første testen bestemte jeg -b 30MB akkurat likt som topologi, men vi ser at 7.1% va data mistet på veien. Derfor prøvde jeg å minke -b til 29, 28 og 27. da jeg brukte -b å være 27 mistet jeg 0% av data. Dette viser at mengde data vi kan sende med minimal mist er noe mellom 27Mb og 28Mb. Det vil si at hvis vi prøver med 27.5, kan hende at vi ikke mister data.

#### 4.3.2 H1---H9

I denne testen vet vi fra topologi at båndbredde er 20Mb, la oss se hva vi får i resultat.

Antall test	Valgt verdi	båndbredde	mistet data
1	20Mb	19.4 Mbits/sec	7.2%
2	19Mb	19.4 Mbits/sec	2.4%



3	18Mb	18.9 Mbits/sec	0.0%
---	------	----------------	------

Her ser vi også at vi mister 0% av data når vi sender 18MB, dette vil fortelle oss at data vi kan sende gjennom dette nettverket uten å miste er mellom 18Mb og 19Mb.

#### 4.3.3 H1---H4

Topologi sier at båndbredden er 20Mb, la oss kjøre testene og se på resultat

Antall test	Valgt verdi	båndbredde	mistet data
1	20Mb	19.4 Mbits/sec	7.2%
2	19Mb	19.3 Mbits/sec	2.4%
3	18.5Mb	19.4 Mbits/sec	0.0%

Her jeg har prøvd desimal tall, vi ser at mister 0% av data når vi sender 18.5MB, dette vil fortelle oss at data vi kan sende gjennom dette nettverket uten å miste er mellom 18.5Mb og 19Mb. Siden når vi prøvde 19Mb har vi mistet 2.4%.

#### 4.3.4 Discussion

Siden jeg hadde informasjon om båndbredde fra topologien, forventet jeg ikke å miste data når jeg sendte like mye data som stod på topologi. Resultatene jeg har fått stemte ikke, grunnen til det kan være andre faktorer som påvirket nettverket som trafikk, overbelastning og støy. Derfor har jeg kjørt iperf-testen flere ganger for å finne ut hvor mye data kan jeg sendte over til server uten å miste noe av dem på vei.

Fra dette lærte jeg at hvis jeg ønsker å finne ut ytelsen av et nettverk, kan jeg kjøre flere tester med forskjellige mengde data og finne ut den riktige båndbredden. Siden nettverket er alltid preget av andre faktorer som trafikk, avstand og nettverkbetastning er det vanskelig å finne ut akkurat grense på å ikke miste noe data, men ved å bruke denne metoden at kjør flere tester kan vi tilnærme oss til det vi ønsker. En annen faktor som er viktig å huske at nå på dagen vi kjører testen, hvis vi kjøre på et tids punkt der det er flere brukere da får tregt nettverk.

### 4.4 Test case 2: link latency and throughput

RTT (Round Trip Time) brukes innenfor nettverk for å referere til tiden i (ms) en pakke bruker til å reise fra en kilde til en andre kilde og deretter får en bekreftelse som blir sendt tilbake til kilden som indikerer vellykket levering. RTT påvirkes av ulike faktorer som avstand mellom kildene, nettverkbetastning og etc. Høy RTT-verdi gir langsomme nettverksytelse. Båndbredde kan enkelt forklares er mengden av data som kan overføres i en gitt tid. Høy båndbredde betyr rask data overføring mens lav båndbredde peker for mindre dataoverføring. Jeg har nevnt disse fordi i denne saken skal vi sjekke båndbredde og Latency til r1 og r2, for å finne latency kan vi dele RTT i to.

#### 4.4.1 Results

##### L1 (r1 ----- r2)

Her skal jeg ping og kjøre simpleperf for å sjekke latency og båndbredde. Fra topologien forventer jeg at latency mellom r1 og r2 skal være 10ms og båndbredden skal være rund 40Mbps. La oss se hva vi får på test

```
10.0.1.2:8088  0.0-25      120.78 MB    38.65 Mbps ----- simpleperf
rtt min/avg/max/mdev = 20.329/20.602/20.870/0.153 ms ----- ping RTT
```

Fra simpleperf resultat kan jeg konkludere at nettverkforbindelse og destinasjon som finnes kan overføre opptil 38.65 megabit data per sekund som kalles båndbredden 38.65 Mbps. Jeg har ikke fått akkurat det jeg forventet, men vi er noe i nærheten, for å si hvorfor vi ikke fikk det vi forventet må vi kjøre flere tester.

RTT brukes til å teste tilkoblingskvaliteten mellom to enheter. Her har vi i dette tilfelle avg på 20.602 ms og mdev på 0.153 ms kan vi konkludere at RTT-verdien ligger tett på hverandre og tyder på at nettverket har lav latenstid og data overføres rask. Som jeg nevnt for å finne latency må vi dividere RTT på to, hvis vi gjorde det så får vi en latency på 10.3 som er ikke lang unna fra det vi forventet.

## L2 (r2 ---- r3)

Som sakt vi skal kjøre flere tester for å kunne beslutte en bra konklusjon. Her skal vi finne latency og båndbredde mellom r2 og r3. Det jeg forventer er å få en latency rundt 20ms og båndbredde rundt 30 Mbps som det står på topologi. La oss se resultatene.

```
10.0.3.2:8088  0.0-25      91.41 MB      29.25 Mbps ----- simpleperf
rtt min/avg/max/mdev = 40.402/40.679/41.273/0.168 ms ----- ping RTT
```

Fra simpleperf testen konkluderer vi at det overføres 29.25 megabit data per sekund, noe som indikerer at ytelseevaluering er innenfor forventet. Generelt sett kan vi si at ytelseevaluering fungerer bra. For være mer nøyte kan vi kjøre mer test.

Når det gjelder RTT (ping-tid) viser resultat at avg er 40.679, og hvis vi deler på to da får vi latency på 20.3 som er ikke lang unna fra det vi forventet. vi kjører flere tester for å være nøyte.

## L3 (r3 ---- r4)

Samme som de to andre testene, vi skal ping fra r4 til r3 og kjører vi simpleperf serveren i r4 og klienten i r3. Det jeg forventer på latency er 10ms og båndbredde på 20 Mbps, la oss se på resultat.

```
10.0.6.2:8088  0.0-25      60.13 MB      19.24 Mbps ----- simpleperf
rtt min/avg/max/mdev = 20.408/20.713/21.920/0.275 ms ----- ping RTT
```

Likt som tidligere testene ser vi her også at ytelseevaluering som er på 19.24 Mbps er lavere enn topologien 20 MB, men ganske nær. Derfor kan vi si at testen fungerte bra i forhold til forventet i topologien.

RTT viser at forsinkelse i nettverket også nær det vi forventet, altså latency blir 10.3 som er nær det vi forventet.

### 4.4.2 Discussion

Ut ifra alle de testene jeg har gjort så vi at det vi forventet av båndbredde og latency var ikke lang unna enn det vi har fått. For å si der konkret om det RTT og båndbredde vi har fått er det bra eller må forbedres må vi kjenne til kravene. For eksempel hvis vi skal bruke nettverket for videokonferanse ville det båndbredde og RTT som vi har fått i topologi fungere bra, men for å overføre store mengde data ville nettverket være under forventningen. Løsningen på det er forbedring av nettverket.

Grunnen til at vi ikke har fått akkurat det latency og båndbredde vi forventet kan skyldes mange årsaker. Jeg har nevnt tidligere også at et nettverk kan bli preget av trafikk som finnes i nettverket, overføringsforsinkelse, trafikkbelastning, avstand og kø forsinkelse. Disse årsakene har påvirket på at nettverket kunne ikke overføre alltid den maks grensen som ble satt for båndbredden i topologien, derfor har vi fått noe forskjell på båndbredde og latency.

## 4.5 Test case 3: path Latency and throughput

Dette testen er det likt som test 2 men her skal vi teste mellom verter, men ikke rutere. I topologi har vi fått forventet vente tid (latency) og båndbredde. Det som skjer i dette testen er at vi skal kjøre test mellom verter som kjøres gjennom ruter. Det vil si både verter og rutere vil være med i kommunikasjon og virke på latency og båndbredde, dette sier jeg fordi at rutere bruker litt tid til å kommunisere med hverandre også. Vi skal kjøre flere tester under for å sjekke hva får i resultat i forhold til forventning.

### 4.5.1 Results

#### h1 ---- h4

Det vi forventer av data overføring her er en latency på 10 ms mellom L1 og 20 ms i L2 som til sammen vi må forvente et vente tid på 30 ms som vil påpeke på en RTT på 60 ms. Når det gjelder gjennomstrømming forventer vi 30 MB. Dette er fordi hvis vi har flere båndbredde så gjelder det den laveste. Da kan vi starte testen vår med å ping og kjøre simpleperf.

```
10.0.5.2:8088 0.0-25 87.11 MB 27.87 Mbps ----- simpleperf
rtt min/avg/max/mdev = 61.214/62.854/72.553/2.252 ms ----- ping RTT
```

Simpleperf gir et resultat på bredbånd ligger på 27.87 Mbps noe som er mindre, men nær det vi forventet. Dette kan være fordi at det er noe andre faktorer som påvirker på nettverket som rute trafikk, trengsel og andre ting. Siden vi bruker to ruter, kan påvirket vårt resultat. Det som er viktig å merke seg her er at data vi sender går gjennom to forskjellige båndbredde en på 40 og en på 30, jeg har nevnt tidligere også at på slike situasjoner gjelder det alltid den laveste båndbredden.

RTT gir et gjennomsnitt på 62.854 noe som er nær av det vi forventer altså 60 ms. Hvorfor har vi lite forsinkelse er at nettverket ble på virket av andre faktorer. Det er sannsynlighet at vi bruker to ruter og dette kan øke kompleksiteten. For å få bedre oversikt kan vi kjøre flere tester.

#### h7 ---- h9

I dette testen forventer vi latency på 30 ms. Hvis vi dobler latency får vi RTT av nettverket som blir 60 ms. På same måte forventer vi en båndbredde på 20 Mb. Som testen over skal jeg ping h9 fra h7 og deretter skal kjøre simpleperf server på h9 og klient på h7 og ser vi på resultat som blir likt som vi forventer eller ikke.

```
10.0.7.2:8088 0.0-25 56.12 MB 17.96 Mbps ----- simpleperf
rtt min/avg/max/mdev = 60.947/61.859/62.628/0.409 ms ----- ping RTT
```

Båndbredde vi har fått er ikke lang unna fra det vi forventet, og dette lite forskjellen påpeker på at nettverket har blitt påvirket av trafikk, og andre forstyrrelsen. Forskjellige resultat enn det vi forventet kan også være at vi bruker to ruter her.

RTT forventet vi å få 60 ms, men i snitt har vi fått 61.859 noe som er ikke lang unna. Som nevnt i tidligere test kan vi konkludere dette forskjellen på at andre faktorer som påvirker nettverket, eller det at vi bruker to rutere som øker kompleksitet. Vi kjører mer tester for å få bedre oversikt.

#### h1 ---- h9

Dette er siste testen vår. Her forventer vi en båndbredde på 20 MB og en RTT på 80 ms (sum av latency). Vi kjører testen vår samme som de to andre testen og ser vi hva vi får i resultat. Her skal vi ping h9 fra h1 og deretter kjører vi simpleperf på de to slutt verter.

```
10.0.7.2:8088 0.0-25 56.55 MB 18.09 Mbps ----- simpleperf
rtt min/avg/max/mdev = 81.608/82.936/85.498/0.888 ms ----- ping RTT
```

Vi har fått båndbredde på 18.09 Mbps noe som ikke er langt unna fra 20 MB det vi forventet. Forskjellen medfører trafikk, avstand i nettverket og andre faktorer som påvirker på båndbredde. En andre faktor som mest sannsynlighet har påvirket på denne resultat kan være mange ruter vi har.

RTT som vi forventet å få og det vi har fått i resultat er ikke veldig forskjell. Ved å se på andre trafikk og flere rutere som finnes kunne ha påvirket på resultatet vårt.

#### 4.5.2 Discussion

Vi har kommet frem til at hvis vi har flere ruter mellom to end punkter vil det påvirke på resultat vi får i tillegg til andre faktorer. Fra dette testene har vi kjønt at hvis vi har flere ruter blir latency for disse rutene til en sum, og dobler vi det får vi til RTT av nettverket. Når det handler om båndbredde da skjønnte vi at hvis det finnes flere ruter som har forskjellige båndbredde da er det den minste båndbredde som telles. La oss se at vi har 10 ruter med forskjellige båndbredde og en båndbredde på 10Mbps. Da kan vi konkludere slik at uansett hvor stor er andre båndbreddene vil maksimum båndbredden vår være på 10Mbps.

#### 4.6 Test case 4: effects of multiplexing and latency

I disse testene skal vi se hvordan multipleksing og latency påvirker nettverks ytelse, ved å tillate flere kommunikasjon strømmer og bruke samme nettverksressurs. Enkelt kan jeg si at vi øker trafikk på nettverket og ser på hva vi forventer og hva får vi.

##### 4.6.1 Results

###### **h1 ----- h4 og h2 ----- h5**

Vi skal kjøre to part av vertene samtidig ved hjelp av ping og simpleperf og måle båndbredde og latency. I testen vi kjøre har vi vertene vår bindet sammen med tre rutere og vi har en forventning latency verdi mellom rutere på 30 ms og båndbredde på 30 MB. Hvordan er disse vertene koblet til rutere har vi fått fra topologien, men vil bare nevnet at vi kjører ping og simpleperf samtidig. Siden vi skal øke trafikk i nettverket ved kjøre flere verter samtidig, da forventer jeg høyere latency og lavere båndbredde. Vi skal kjøre tester og se på hvor nært vi kommer til forventningen.

10.0.5.2:8088 0.0-25 43.50 MB **13.92 Mbps** ---- h1-h4

10.0.5.3:8088 0.0-25 47.66 MB **15.25 Mbps** ---- h2-h5

rtt min/avg/max/mdev = 60.602/**74.212**/83.304/5.118 ms ---- h1-h4

rtt min/avg/max/mdev = 65.750/**74.660**/80.145/4.112 ms ---- h2-h5

Gjennomsnitt båndbredden jeg har fått i resultat er ca.14.6 Mbps som er mye lavere enn topologien som er 30 Mbps. Dette er sikkert fordi vertene deler rutere mellom seg. En viktig ting jeg merker er at det er forskjell verdi mellom de to verten jeg har en på 13.92 Mbps og den andre på 15.25 Mbps. Dette kan skyldes avstanden de to verten har i forhold til ruter. Det vil si at den verten som fikk verdi på 13.92 står lengre unna fra ruter i forhold til den som fikk verdi på 15.25 Mbps.

RTT som jeg har fått i resultat er mer enn det som er på topologi. Dette kan igjen skyldes at vi kjører simpleperf samtidig med ping. Økt trafikk vil øke RTT og forsinkelser. Det som var merkelig for meg her var at jeg fikk nesten samme avg RTT på mine verter. Dette kan være fordi at begge vertene deler nettverket mellom seg. For å få bedre oversikt kjører vi flere tester.

###### **h1 ----- h4, h2 ----- h5 og h3 ----- h6**

Vi skal legge til mer trafikk. Det tror jeg kommer til å få er enda lavere båndbredde og høyere latency på grunn av økt trafikken. Vi må nevne at latency kapasitet i nettverket er 30 ms og båndbredde er 30

MB La oss se hvordan testen foregår. Fra forrige test så jeg at vertene deler nettverket mellom seg, da tror jeg vertene får nesten likt båndbredde og latency.

10.0.5.2:8088	0.0-25	32.83 MB	<b>10.51 Mbps</b> ----h1-h4
10.0.5.3:8088	0.0-25	30.24 MB	<b>9.68 Mbps</b> -----h2-h5
10.0.5.4:8088	0.0-25	30.33 MB	<b>9.71 Mbps</b> -----h3-h6

rtt min/avg/max/mdev = 62.489/**73.937**/80.668/5.518 ms ----h1-h4  
rtt min/avg/max/mdev = 60.974/**73.394**/80.492/5.657 ms ----h2-h5  
rtt min/avg/max/mdev = 62.070/**74.523**/80.578/4.934 ms ----h3-h6

Slik som jeg forventet, fikk jeg lavere båndbredde. Siden vi har båndbredde på 30 MB, og jo mer trafikk er jo mindre hastighet får vi. Som jeg nevnt tidligere her har vi tre verter som deler rutere mellom seg, og vi ser at vertene deler nettverket og de får nesten 10 Mbps hver.

Som forventet jeg har fått høyere latency som er igjen på grunn av høyere trafikk.

#### **h1 ----- h4 og h7 ----- h9**

I denne testen skal jeg kjøre flere verter samtidig. Det som er spesielt å merke i denne testen er at disse vertene deler L2 med hverandre. Det jeg forventer er at h7 og h9 skal får mindre båndbredde, fordi at minimum båndbredde mellom h7 og h9 er 20 MB altså mindre enn h1 og h4, men begge har like mye på RTT. la oss sjekke hva får vi svar fra testene.

10.0.5.2:8088	0.0 - 25.0	63.98MB	<b>20.47 Mbps</b> ---- h1-h4
10.0.7.2:8088	0.0 - 25.0	27.66MB	<b>8.85 Mbps</b> -----h7-h9

rtt min/avg/max/mdev = 61.012/**70.028**/90.857/6.960 ms----h1-h4  
rtt min/avg/max/mdev = 60.552/**71.210**/80.332/6.102 ms-----h7-h9

Båndbredde i h7 og h9 er mindre og dett er ikke overraskende. Data som går gjennom h1 og h4 har et minimum grense på 30 MB som er større enn båndbredde for h7 og h9. Enkelt kan jeg si at grunnen til lave båndbredde på h7 og h9 er minimum båndbredde verdi på L3. Data vil overføres enklere i L2 men enn L3.

RTT mellom begge parene er nesten like. Dette er ikke noe overraskende siden h1 og h4 har 30 ms latency til sammen og h7 og h9 også har 30 ms til sammen. Det RTT vi har fått i resultat er høyere enn det vi har i topologi og grunn til det er trafikk som påvirket på nettverket. La oss kjøre flere tester for å få bedre oversikt.

#### **h1 ----- h4 og h8 ----- h9**

I denne testen skal jeg sjekke om mengden av rutere har noe påvirkning på latency og båndbredde. h1 og h4 skal bruke tre ruter mens h7 og h9 skal bruke bare to ruter for kommunikasjon. Bare ruter R3 deler de med hverandre la oss se hva effekten blir. Det jeg selv forventer er at siden det er ikke mye trafikk mellom disse tilkoblingene ville vi få et resultat som er nær topologien.

10.0.5.2:8088	0.0-25	83.82 MB	<b>26.82 Mbps</b> --- h1-h4
10.0.7.2:8088	0.0-25	59.98 MB	<b>19.19 Mbps</b> ----h8-h9

rtt min/avg/max/mdev = 60.218/**65.842**/80.426/6.508 ms ----h1-h4  
rtt min/avg/max/mdev = 21.185/**25.589**/29.778/2.799 ms ----h8-h9

Båndbredde vi har fått i resultat er det ikke så lang fra det vi forventet. vi kan si at i dette testen ville data som sendes fra de to tilkoblingen blir forsinket bare når de kommer på R3, derfor forskjellen er ikke mye. Det som vi kan merke her at antall ruter som kan påvirket båndbredden. Vi ser at båndbredde til h8 og h9 er nærmere til det vi forventet, fordi at denne tilkoblingen bruker mindre ruter.

Jeg har fått litt høyere RTT enn det som forventet. I topologi står det at L3 har et minimum latency på 10 ms som i RTT blir 20ms, denne lille RTT forskjell jeg har fått er det sikkert fordi det er trafikk rund ruter R3.

#### 4.6.2 Discussion

Etter å kjøre flere tester kan vi si generelt sett økt nettverkstrafikk vil føre til høyere forsinkelse altså latency og lavere båndbredde (bandwidth). I tillegg vil føre til nettverksbelastning (congestion) som kan forverre disse problemene. Enkelt sakt hvis vi har flere enheter vil de konkurrere om ressursene derfor vil det føre til redusert ytelse av nettverket. Da kan vi si at multipleksing og latens kan påvirke nettverksytelsen. Jeg har kjønt også at jo mer tilkoblinger er i nettverket vil det være mer trafikk, derfor sier vi at på hvilken tid måler vi båndbredde ville vi få forskjellige resultat. Jeg har kjønt det også at tilkoblingene i et nettverk vil kjempe for å utnytte mer ressurs fra nettverket, derfor blir ikke fordelingen likt mellom vertene. Hele disse gikk på at hvilken effekt har økt antall tilkoblingene i nettverket, det kan vi si også at måten vi fordeler vertene og rutene som finnes ville påvirket ytelse av nettverket. Tilkoblingen som bruker mindre ruter og har lavere trafikk og forsinkelse ville ha bedre båndbredde og latency.

#### 4.7 Test case 5: effects of parallel connections

I denne testen sjekker vi parallell data overføring. Når vi sier parallell tilkobling betyr det at fra h1 er det to forbindelse eller tråd til h4. Parallell betyr det at data kan reise fra flere veier mellom disse enhetene. Derfor forventer jeg bedre resultat mellom h1 og h4 enn resten. la oss kjøre test og se hva vi får i resultat, men får ville jeg si at topologi gir oss båndbredde på 30 Mbps og latency på 30 ms. Som forrige test vil vertene fordele nettverk ressurs mellom seg, men det vi skal sjekke her er at om h1 og h4 vil få bedre båndbredde enn resten eller ikke.

##### 4.7.1 Results

10.0.5.2:8088	0.0 - 25.0	21.36MB	<b>6.84 Mbps</b>	-----h1-h4
10.0.5.2:8088	0.0 - 25.0	20.99MB	<b>6.72 Mbps</b>	-----h1-h4
10.0.5.3:8088	0.0 - 25.0	21.17MB	<b>6.77 Mbps</b>	-----h2-h5
10.0.5.4:8088	0.0 - 25.0	24.31MB	<b>7.78 Mbps</b>	-----h3-h6

##### 4.7.2 Discussion

Vi ser her at jeg har fått det resultat jeg forventet. Mellom h1 og h4 har det overført mer data enn de resten, dette er effekten av parallell tilkobling. Siden h1 bruker parallell øker den totale data overføring hastighet. Det som skjer i en enkelt tilkobling er det at hvis denne tilkoblingen overbelastes eller mislykkes vil det føre til redusert hastighet, men i en parallell kan data bruker flere veier. Enkelt kan jeg si at hvis vi har to parallell tilkobling kan vi tolke som to verter som tilkobler, men i virkeligheten er det bare en.

## 5 Conclusions

Nå skal vi oppsummere hva har vi lært fra dette prosjektet. Prosjektet handlet generelt om hva er nettverk oppbygning, hvordan foregår kommunikasjon i flere situasjoner i TCP og UDP protokoll. På starten av prosjektet hadde vi en topologi som ved hjelp av dette skulle kjøre flere tester og lærte mye

om nettverk ytelse. Jeg møtte sette meg inn i fem forskjellige tester og lære meg på hvordan fungerer båndbredde og latency på nettverk.

I kode delen av prosjektet som krevet mye forståelse og tid, har jeg lært mye. Jeg har lært hvordan kan jeg lage et enkelt program som kan brukes for nettverk kommunikasjon mellom to pc-er. Programmet jeg har laget gir noe mulighet til brukerne for å velge mellom noen argumenter ved hjelp av argparser. Den kode delen var spennende for meg og det jeg likte var at hvordan bruker vi socket programmering til å kommunisere mellom pc-er.

Prosjektet var utfordrende, og det jeg mest slitt med var å bruke minne nett i Mac, og forstå hvordan jeg skal kjøre de forskjellige test-casene. Til slutt vil jeg si at jeg har prøvd mitt beste å forklare godt hva som foregår i prosjektet jeg har laget.

## 6 References (Optional)