BACHELOROPPGAVE:

BO19E-41 Application assurance for 802.11 medium

Hans Daniel Lambach
Sondre Skodvin Mjåtvedt
Carl Erik Koltveit

31. Mai. 2019

Dokumentkontroll

Rapportens tittel:	*Dato/Versjon*
BO19E-41 Application assurance for 802.11 medium	31. mai. 2019/0.16
	Rapportnummer: B019E-41
Forfatter(e): Hans Daniel Lambach Sondre Skodvin Mjåtvedt Carl Erik Koltveit	*Studieretning:* HKOM16 *Antall sider m/vedlegg* 37
Høgskolens veileder: Knut Øvsthus	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver: Atea	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): *Kim Systad email: kim.systad@atea.no tlf: 950 18 841* *Kristian Brown email: kristian.brown@atea.no tlf: 452 07 681*	

Revisjon	**Dato**	**Status**	**Utført av**
0.10	21.01.19	Forstudie punkt 1 & 2	Carl Erik, Hans Daniel, Sondre
v.0.11	23.01.19	Forstudie punkt 3	Hans Daniel
v.0.12	24.01.19	Prosjektledelse og styring	Carl Erik, Sondre
v0.13	09.04.19	Del 4 og oppdatert innhold etter forslag fra veileder	Hans Daniel
v0.14	29.04.19	Første utkast	Hans Daniel
v0.15	20.05.19	Revidert utkast	Carl Erik, Hans Daniel, Sondre
v0.16	31.05.19	Endelig utkast	Carl Erik, Hans Daniel, Sondre

Forord

Denne bacheloroppgaven ble gitt til oss av ATEA Bergen.

Oppgaven har vært spennende, utfordrende og ikke minst veldig lærerik. For denne oppgaven valgte vi å bruke løsninger som vi ikke har vært gjennom i pensum, som Python, SQL samt front-end utvikling. Vi fikk underveis i prosjektet tilegnet oss mye ny kunnskap om programmering, databaser, web applikasjoner og hvordan det er å jobbe i team for å skape et produkt.

Det har vært gøy å se hvordan vi har løst utfordringene vi har møtt på, med kunnskapen vi har tilegnet oss gjennom studieløpet. I løpet av disse tre årene på høgskolen har vi jobbet sammen i gruppe og utviklet et godt samarbeid som uten tvil har kommet godt med på denne oppgaven. Det har vært en utrolig reise og vi har brukt mye tid på å utvikle et felles produkt som vi er meget stolte av.

Vi har lyst til å utrette en stor takk til ATEA og Stian Eliassen for muligheten til å jobbe med et prosjekt som har vært såpass spennende og lærerikt. Våre veiledere og kontaktpersoner i ATEA, Kim Systad og Kristian Trebuchet Brown har alltid vært tilgjengelig, imøtekommende og svart på spørsmål som måtte dukke opp. Møtene vi har hatt med dem har gitt oss et stort utbytte i form av kunnskap og erfaring, som ikke bare har hjulpet oss med oppgaven, men som vi garantert vil ta med oss videre i arbeidslivet.

Vi vil også takke vår interne veileder på høgskolen, Knut Øvsthus, for å ha gitt oss verdifulle tips og retninger for prosjektet. Vi ville aldri fått en så god start på oppgaven hadde det ikke vært for Knut sin brede kompetanse innen fagfeltet samt gode tilbakemeldinger etter våre veiledningsmøter.

Sammendrag

Vi har fått i oppgave av ATEA å utvikle et overvåkningssystem for real-time applikasjons parametere i det trådløse nettverk. I dagens samfunn foregår mesteparten av all trafikk over internett, trådløst der IEEE 802.11 er den mest brukte trådløse standarden. Det trådløse nettverket har blitt et svært komplekst system og ofte er bedrifter avhengig av pålitelig trådløs nettverkstilkobling. Dermed er det viktig at det til enhver tid leverer god nok kvalitet. Prosjektet vårt går ut på å utføre målinger av forskjellige applikasjons parametere i et trådløst nettverk for å kunne oppdage og kartlegge eventuelle problemer.

Løsningen vi har kommet fram til i samarbeid med våre veiledere, er å bruke et utviklingsbrett med nødvendige funksjoner for å kunne programmere og realisere oppdragsgivers problemstilling etter kravspesifikasjonene. Vi valgte å bruke to Raspberry Pi, en som sensor og den andre som server. Et krav fra oppdragsgiver var at enhetene måtte ha Broadcom chipset som støtter 802.11ac standarden. Dermed valgte vi en passende Raspberry modell etter kravet. De to enhetene kommuniserer over det trådløse nettverket seg imellom og ved hjelp av et eksternt verktøy opprettes det datastrømmer mellom dem.

Testene som utføres skal emulere “Voice over IP” (VoIP) og “Hypertext transfer protocol” (Http) applikasjoner. Dette oppnår vi ved hjelp av verktøyet Iperf. Iperf lar oss opprette datastrømmer mellom to enheter ved å bruke enten “User datagram protocol” (UDP) eller “Transmission control protocol” (TCP). Iperf testene kjøres kontinuerlig med gitt tidsintervall for begge protokollene. Resultatene fra testene hentes ut og lagres i en lokal SQLite database. I en database browser kan SQLite databasen åpnes, og man kan gå gjennom tidligere tester som er utført om det skulle være nødvendig. De nyeste målingene er tilgjengelige gjennom en webside, her blir man også varslet dersom målingene ikke tilfredsstiller kravene vi har satt.

For å realisere prosjektet har vi tatt i bruk eksisterende verktøy og løsninger, for så å tilpasse det til vårt formål. Vi hadde flere ideer som vi ville implementere i den grafiske fremvisningen av målingene, men vi la heller fokus på å få systemet ferdigstilt. Vi har til slutt kommet frem til et system som vi er fornøyd med og som tilfredsstiller kravspesifikasjonene satt av oppdragsgiver.

Innhold

1 Innledning

1.1 Oppdragsgiver

Atea Norge er ca. 1.650 medarbeidere fordelt på 24 kontorer fra Hammerfest i nord til Kristiansand i sør. Atea-konsernet er Norden og Baltikums største, samt Europas nest største leverandør av IT-infrastruktur. Selskapet er notert på Oslo Børs og de omsatte for omtrent NOK 32,4 milliarder i 2017

Sammen med et rikt mangfold av kunder fra hele landet bygger Atea, Norge med IT. Atea er partnere med ledende internasjonale IT selskaper som Microsoft, Cisco, HP Inc, Hewlett Packard Enterprise, Apple, IBM, Dell EMC, Lenovo, Citrix og VMWare. De kan dermed tilby kunder en rekke hardware- og software løsninger fra verdens ledende teknologiselskaper. Konsulentene sitter på tekniske sertifikater og bred kompetanse innen systemintegrering. Dette gjør det mulig for Atea å designe og implementere løsninger til selv de mest komplekse problemstillingene innen IT.

Ateas overordnede mål er å være «The Place to Be» - et sted der medarbeidere, kunder, samarbeidspartnere og investorer møtes og samarbeider om å forme fremtiden ved hjelp av IT.

1.2 Problemstilling

IEEE 802.11 baserte WLANs representerer den mest brukte trådløse nettverkteknologien. Med bruk av kritiske applikasjoner i et datanettverk samt applikasjoner som filoverføring og video/audio overføring krever man at IEEE 802.11 skal kunne levere god nok tjenestekvalitet for “Quality of Service” (QoS).

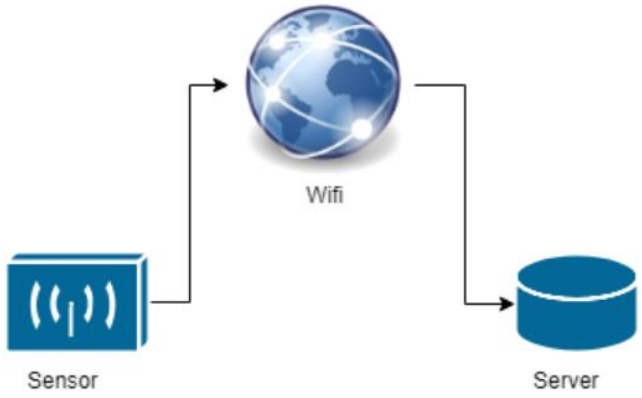
Store bedrifter med større lokaler har ofte komplisert wifi infrastruktur hvor opptil tusener av enheter kan være tilkoblet i større grad i dag enn tidligere. Det er ikke en selvfølge å kunne overvåke statusen til nettverket og man finner ofte ikke ut av problemer før brukerne selv rapporterer feil. Dette er ikke gunstig og kan i verste fall føre til økonomiske tap for en bedrift.

Atea vil bruke wifi baserte sensorer som skal måle ulike parametere for sann-tids applikasjoner, opp mot en server for å overvåke QoS. Disse målingene skal utføres kontinuerlig og gi en bruker jevne oppdateringer på nettverkets helse. Verdier for begge tester skal lagres i en lokal database på serveren. Dersom testene viser at kvaliteten på nettverket ikke er tilfredsstillende, skal det bli utløst en alarm med melding om at nettverket er under akseptabel standard.

1.3 Hovedidé for løsningsforslag

Oppdragsgiver har ikke noen konkrete løsningsforslag og vi står fritt fram til å "angripe" problemet. Oppdragsgiver hadde få ønsker om hvilken programvare, hvilket programmeringsspråk og type utstyr vi valgte å bruke.

Hovedideen vår er å bruke to Raspberry Pi 3 modell B+, en som sensor og en som server. Sensor skal emulere en bruker som hovedsakelig sender TCP/UDP-pakker over nettverket. Vi setter opp sensor til å sende datapakkene med et gitt tidsintervall til server. Serverens hensikt er å oppbevare informasjon om overføringskvaliteten og sende varselmeldinger til en bruker dersom kravet for QoS ikke tilfredsstilles. Bruker kan sitte fjerntliggende og overvåke nettverket når ønskelig.



Figur 1.1: Skisse av kommunikasjon mellom Sensor og Server via Wifi

2 Kravspesifikasjon

Vi skal programmere en sensor som sender en strøm av datapakker til en server og henter ut verdier for applikasjon parameterne. Ut ifra disse verdiene skal serveren gi bruker status på om QoS er god nok eller ikke.

- Sensor skal emulere en VoIP samtale og HTTP applikasjoner.
- Sensor skal kommunisere med en sentralisert server.
- Server skal vise status på applikasjoner som “OK” eller “IKKE OK” dersom målingene er under gitt terskel.
- Server skal lagre alle data lokalt i en tekstfil.
- Server skal sende nettverkets status til bruker og eventuelt ha kapasitet til et “on board” LED matrix display for å vise status og informasjon.
- WLAN Chipset må være fra produsenten Broadcom og støtte 802.11ac

Kravspesifikasjonene ble utformet i samarbeid med oppdragsgiver, men endret seg etterhvert som vi kom lengre ut i prosjektet. Disse endringene ble gjort etter veiledningsmøter med både oppdragsgiver og intern veileder.

Terskel på alarm er det heller ikke satt noe krav på men vi har kommet fram til en grense sammensatt av flere parametere i samråd med oppdragsgiver.

3 Analyse av problemet

Vi har behov for en sensor som kan emulere forskjellige protokoller og applikasjoner opp mot kravene. Enheten må derfor kunne programmeres for å implementere slik logikk. I tillegg skal sensor kommunisere trådløst med en sentralisert server og må derfor ha WIFI kapasitet.

Data skal lagres i en lokal database på server. En kan dermed gå inn og se hvor og når nettverket ble belastet og gjøre eventuelle tiltak. Databasen kan skrives ut hos brukeren og kan også slettes etter en gitt periode for å ikke bruke opp minnekapasiteten til server.

Informasjonen som skal vises til bruker må være informativ og enkel å forstå for selv en bruker med lite teknisk forståelse. For å gi best mulig brukeropplevelse vil det letteste være et "Graphical user interface" (GUI) som viser bruker status på nettverket og om kvaliteten er under ønsket grense.

3.1 Utforming av mulige løsninger

Etter møtet med våre veiledere fra Atea kom vi enige om å bruke et utviklingsbrett som sensor og server. Vi ble anbefalt å velge et brett med innebygget WIFI funksjonalitet og som lett kan programmeres til ønsket behov. Dermed falt valget på Raspberry Pi, det er en "single-board computer" som kan konfigureres til å gjøre en rekke oppgaver, som det også finnes mye støtte til.

Koden som skal skrives må være ryddig og oversiktlig for at andre skal kunne gå inn og se hva som har blitt gjort, i tilfelle de vil modifisere sensoren til å oppdage andre anomalier i et nettverk. Vi tar utgangspunkt i Iperf sin eksisterende kode. Server skal under prosjektet programmeres til å kun motta pakker fra én sensor, men vi tenker at et sluttprodukt skal kunne motta datapakker fra flere sensorer i ett og samme nettverk. For å få til pålitelig kommunikasjon mellom sensor og server bruker vi sockets, som setter opp forbindelse mellom enhetene. Når det kommer til serveren kunne vi bruke hva som helst, så lenge den lagrer og behandler informasjon fra sensoren. Vi vil lagre verdiene i en SQlite database lokalt på serveren.

For å prøve ut vår løsning tenker vi at Høgskolen på Vestlandet, campus Kronstad er perfekt. Vi vil teste sensor i et av de største auditoriene når det er tomt og når det er full forelesning for å sammenligne. Vi har lyst til å plassere sensoren i deler av bygget hvor mange samler seg, for å se hvordan dette påvirker trafikken, men også hvordan sensor takler overgangen til andre aksesspunkter i samme nettverk. Dermed kan vi danne oss et bilde over kritiske soner i nettverket i form av trafikk og belastning.

3.1.1 Vurderinger i forhold til verktøy og HW/SW komponenter

Hardware

Når det kommer til valg av hardware ble det ganske enkelt Raspberry Pi. Det finnes alternativer, men når det kommer til pris og funksjonalitet er Raspberry det beste valget. Det finnes mye dokumentasjon på ulike funksjoner og biblioteker som kommer godt med i programmeringsfasen.

- **Raspberry Pi 3 B+**

  Inneholder 32GB micro-SD minne, Broadcom 64-bits CPU, innebygget trådløs protokoll som støtter 802.11 standarder og Bluetooth 4.2LE og har kapasitet til å vise grafikk på LED display. Det følger også med et lite kabinett som vil beskytte mot skader.

Figur 3.1: Raspberry Pi 3 B+ utviklings brett

Software

På software delen er det noe å velge i når det kommer til forskjellige Software development kit (SDK) og Integrated developer environment (IDE). Vi har gjennom studieløpet vårt opparbeidet oss kunnskap om ulike programmeringsspråk, som C++, C#, Java og noe JavaScript som kommer godt med i oppgaven. I tillegg til programmeringsemnene har vi også hatt fag innen nettverksteknikk med deler av pensum fra Cisco sitt CCNA og CCNP kurs.

- **Raspbian**

  På SD-kortet installerer vi Raspbian. Raspbian er det offisielle Pi - operative systemet som er basert på Linux og har allerede pre-installert en rekke software for utdanning, programmering og generell bruk.

- **C # og Visual Studios**

  SDK og IDE som er mye brukt for programmering av firmware til mikrokontroller og som vi har kjennskap til fra ELE-124 Videregående Programmering. Visual Studios er en populær IDE og har mye dokumentasjon.

- **Python og PyCharm**

  Som C# og Visual studios er både Python og PyCharm populære SDK og IDE brukt til programmering. Vi har ikke vært borti Python tidligere, men med kjennskap til C++, C# og Java burde det ikke være store stor overgang. Python blir mye brukt i industri og da spesielt innen nettverksprogrammering. Det finnes en rekke moduler, biblioteker samt god dokumentasjon som passer vårt behov.

- **Iperf 2**

  Verktøy for aktive målinger av båndbredde i IP nettverk. Testene gir deg tilgjengelig båndbredde, pakketap og andre nettverksparametere. Versjon 2 har også mulighet for bidirectional testing som vi vil bruke for emulering av VoIP. Iperf er “open source”, som betyr at det er gratis og tilgjengelig for alle.

- **SQlite**

  Bibliotek skrevet i C-språk som implementerer et lite men raskt, pålitelig, effektivt og oversiktlig SQL database. Den er meget populær og er databasemotoren som er mest brukt. Kildekoden er gratis og tilgjengelig for alle.

- **Apache Web Server**

  En “open-source” hypertext transfer protocol (HTTP) server for moderne operativsystem. Det er et prosjekt som har som mål å levere en sikker, effektiv og skalerbar server som gir HTTP tjenester som oppdaterer seg med dagens HTTP standarder.

- **JQuery**

  JQuery er et JavaScript-bibliotek utviklet for å forenkle klient skripting av HTML og er et av de mest populære bibliotekene i bruk i dag. Det mulig gjør dynamiske nettsider og er et kraftig verktøy for front-end programmering.

- **Github**

  Gratis Git-server hvor vi kan dele programkoden med hverandre og være sikker på at det bare er medlemmer av prosjektet som har tilgang til filene som lastes opp. Dette er et samarbeidsverktøy som er testet i tidligere prosjekt vi har hatt sammen.

3.2 Konklusjon av problem

Den beste løsningen for oss var å bruke Raspberry Pi med Raspbian som operativsystem, Python som SDK og PyCharm som IDE. Bakgrunnen for dette er alle mulighetene vi får med Raspberry, samt enkel installasjon og konfigurering med Raspbian. Raspberry er på størrelse med et kredittkort noe som er ideelt for en sensor som skal ta liten plass. Den er i stand til å gjøre alt det man forventer av en vanlig PC noe som gjør den til et kraftig verktøy som passer vårt formål. Python som språk er noe vi må bruke litt tid på å sette oss inn i, men som vil være ubetydelig i forhold til fordelene. Python er oversiktlig, er brukervennlig og ble anbefalt av Atea. Vi vil bruke eksisterende verktøy som Iperf og SQlite som basis for vårt system, men modifisere og tilpasse koden til vårt formål. Til slutt vil vi vise all informasjon og data på en webside.

4 Realisering av valgt løsning

4.1 Forarbeid

For å realisere vår løsning var første steg å gå til innkjøp av Raspberry Pi. Vi ville starte så tidlig som mulig med å bli kjent med hardware og hvordan Raspbian samt Linux fungerer så det var viktig å få hardware på plass så raskt som mulig. Vi bestilte to enheter fra Komplett.no med alt av tilbehør som strømkabler, minnekort og et deksel. Vi møtte på noen små problemer når vi skulle installere og konfigurere Raspberry på grunn av feil med formatering av minnekort. Neste steg var å få applikasjonene for deling av dokumenter og kode, vi registrerte gruppen på Github og Google Drive. Så måtte vi planlegge hvordan vi skulle gå fram og hva som var nødvendig å få på plass først. Etter møter med veileder og oppdragsgiver kom vi fram til at vi skulle bruke den eksisterende koden til Iperf som er et verktøy for aktive målinger i et nettverk.

I forarbeid gikk det også en del tid til å lese oss opp på dokumentasjon, spesielt Iperf ettersom det var et fremmed verktøy for oss. Iperf har en klient og server funksjon og sender data strømmer for å måle throughput mellom de to enhetene. Strømmene er enten UDP eller TCP noe som lar oss å emulere web applikasjoner og VoIP samtaler.

Vi ble enige innad i gruppen om resten av funksjonene til systemet, og vi gjorde research for å finne fram til hva vi mente var beste løsning. Vi fant mange forskjellige kandidater til database og GUI løsninger, men valgte til slutt det vi mente ville være mest brukervennlig og gi oss det beste resultatet.

Selv om vi så for oss hvordan systemet ville se ut og planla mye av oppgaven i forarbeidet, ble flere funksjonaliteter til etter hvert som vi kom lenger ut i oppgaven og ble mer erfarne med både Python og Raspbian.

4.2 Sensor og Server

4.2.1 Konfigurasjon av enheter

Vi begynte oppgaven ved å konfigurere enhetene med riktig image. Operativsystemet vi endte opp med var Raspbian Stretch w/ Desktop. Dette skyldtes at vi ikke hadde mye erfaring med Raspbian eller Linux så da mente vi det ville være lettere å konfigurere enhetene med en desktop selv om det krever mer ressurser å kjøre dette operativsystemet, i stedet for å arbeide kun fra terminalen.

Den ene enheten skal brukes som sensor og kun sende informasjon, den andre skal fungere som server og lagre alt av informasjon. Et hinder vi møtte på i oppstartsfasen var at Eduroam ikke tillot Raspberryen å koble seg på nettverket. Linux og lignende operativsystem har trøbbel med å koble til Eduroam. Vi klarte å koble oss til nettverket ved å konfigurere wpa_supplicant filen men opplevde ofte at vi ble kastet ut etter noen minutter. Etter mye testing og feiling med konfigurasjon av filen ga vi opp. Dermed bestemte vi oss for å operere og utføre testene på HVLguest nettverket. Problemet med skolens gjestenettverk er at det er utdatert og med begrenset kapasitet. Siden de fleste studenter på skolen er tilkoblet til Eduroam, ville dette føre til at testene vi utførte ikke ville være særlig representative. Et annet problem var at Network Time Protocol (NTP) ikke ble synkronisert over nettet til tross for at vi hadde internettforbindelse, noe som ga oss feil klokkeslett på enhetene. Til tross for noen mindre hindringer, konkluderte vi med at logikken i koden fortsatt vil være den samme og konsentrerte oss om å jobbe videre med å emulere en ordinær VoIP samtale.

4.2.2 Forbindelse mellom sensor og server

For å lage en forbindelse må, per dags dato begge, enhetene være koblet til samme nettverk. Først opprettes en socket tilkobling mellom sensor og server, deretter må vi gi Iperf-klienten IP-adressen til server. Serveren vil lytte etter forespørselen og når det er blitt opprettet en forbindelse vil sensoren starte prosessen med å sende pakkene som skal emulere de forskjellige applikasjonene. Vi har to typer applikasjoner vi ønsker å emulere, VoIP og Web-trafikk. Vi vil da utføre to seperate tester og trenger dermed to forbindelser. VoIP testen bruker UDP og for å emulere web-trafikk bruker vi TCP i Iperf kommandoen. Hver enhet har en fil som inneholder IP-adressen og port nummeret sitt og iperf kommandoene til begge tester. Disse filene leses av hver gang programmet kjøres.

Dersom klienten sender en forespørsel om å starte en Iperf sesjon *før* server lytter etter tilkoblinger vil det oppstå en feilmelding fra Iperf-programmet. Derfor har vi implementert bruk av sockets for å sørge for at server setter opp Iperf-server *før* sensor starter Iperf-klienten. Sockets sender meldinger som “Start_VoIP” og “VoIP_Ready”. En annen nyttig funksjon av sockets er at metoden “.recv” fører til at koden venter til den mottar noe over socket og vi får dermed en pause på serversiden, helt til testene er ferdig og nye instrukser tildeles.

4.3 Iperf

4.3.1 Iperf2

IPerf er et verktøy for aktive målinger av *jitter*, pakketap, båndbredde og andre parametere i et nettverk. I oppgaven har vi tatt i bruk Iperf2, ettersom denne versjonen har mulighet for bidirectional testing av UDP-strømmen. I en bidirectional test vil server koble seg mot klienten når den får en tilkobling. Så det vil bli opprettet to datastrømmer. Dette er en nyttig funksjonalitet man kan bruke for en enda bedre emulering av VoIP. Programmeringsspråket brukt for å utvikle Iperf er C, og er cross-platform. Verktøyet er "open source" og tilgjengelig for alle. Iperf2 er nå i versjon 2.0.13 som ble lansert 22.01.19.

4.3.2 UDP

UDP er en såkalt forbindelsesløs nettverksprotokoll for overføring av datapakker. UDP gir ingen garantier for at en pakke blir levert, og ingen tilstand om UDP-meldinger lagres hos avsender etter at pakken er sendt ut på nettverket.

Siden UDP bare sender en vei, måler man bare *latency* fra avsender til destinasjon. Variasjonen i *latency*, kalles *jitter* og måles når avsenders pakker blir mottatt. Om variasjonen er høy, er ikke dette et godt tegn og man vil fort kunne oppleve en betydelig dårligere samtale. Iperf sammenligner tidsstempelet med systemets klokke for å kalkulere *jitter*.

En annen parameter vi ser på er pakketap. Vi kan forklare det som at du skal sette sammen et puslespill, hvor noen av brikkene mangler. Man kan sette sammen deler av settet men det vil ikke være komplett, og jo flere brikker som mangler jo mindre av det hele bildet ser man. Derfor er det viktig å ha så lite pakketap som mulig i et nettverk, siden UDP vil ikke rette opp i pakker som går tapt. Om for mange pakker går tapt under overførsel vil dette kunne oppleves av brukere ved at f.eks. bilde eller lyd i en videochat henger seg opp. For å kalkulere pakketap i målingen tar Iperf differansen mellom pakker sendt fra senderen og pakker mottatt hos mottakeren.

4.3.3 TCP

TCP er en forbindelsesorientert nettverksprotokoll. Kommunikasjon mellom endepunktene må være oppnådd før data kan utveksles. Derfor ønsker vi å se på andre parametere for å avgjøre om testen av nettverket er under ønsket nivå. TCP brukes av mange internettapplikasjoner, som WWW, email, file transfer og strømmetjenester. Denne protokollen er optimalisert for pålitelig og nøyaktig overførsel av data.

TCP er direkte påvirket av *latency*. Om tiden det tar før meldinger (pakker) når destinasjon blir for stor, kan man oppleve relativt lange "pauser" siden protokollen må rette opp i pakker som er ute av orden og/eller sender tapte pakker.

Retransmisjon av pakker er en parameter vi også ønsker å se på her. Om for mange pakker må sendes på nytt og rettes opp vil dette sinke overførselen og man kan oppleve tidsavbrudd. Antall pakker som sendes før avsender mottar et "ack-flag" avgjøres av TCP metningsvindu.

4.3.4 Terskel for alarm

Under vårt første møte med Atea kom vi frem til at en gunstig måte å utløse alarmen på, var å bruke mean opinion score (MOS) for den emulerte VoIP samtalen. MOS er et mål som brukes innen QoS og telekom. Ettersom vi hadde lite kjennskap til dette og det ikke var svært tidssensitivt tilbød våre eksterne veiledere å hjelpe oss med denne kalkulasjonen.

Etter litt frem og tilbake fant vi i samsvar med Atea ut at vi heller skulle sette opp en tallverdi for de relevante målingene. I UDP-testen setter vi grenser for *jitter* og pakketap. For TCP testen settes det grenser for retransmit og round trip time (RTT). Her ser vi også på båndbredden som er tilgjengelig mellom enhetene.

4.4 Database

Vi konfigurerte og satte opp en database med en tabell som inneholder kolonner med de verdiene vi vil ha fra målingene. SQlite kommer ikke med et brukergrensesnitt for å visualisere dataene, derfor lastet vi ned SQlite browser for Linux for å se over databasen og enkelt kunne modifisere den. Neste steg var å lage en klasse med en metode som oppdaterer tabellen med de nye verdiene fra målingene.

Metoden tar i bruk funksjoner som er importert fra SQlite biblioteket og som støtter Python. I og med at databasen er lagret lokalt på enheten finner metoden, databasefilen med fil lokasjon og oppretter en forbindelse med den. Neste steg er å hente ut variablene fra objektene og overføre det til databasen.

SQlite funksjonene som kommer med biblioteket og som vi bruker i vårt system er ".connect", ".cursor" og ".execute". Connect funksjonen oppretter et "connection-objekt" og er det som gjør at vi får en forbindelse med databasefilen. "Cursor" er en midlertidig arbeidsplass som blir opprettet i minnet hvor data er midlertidig lagret og manipulert før vi bruker "execute" funksjonen som binder resultatet til databasen. Siden vi henter ut verdier fra en tekstfil vil de bli konvertert til datatypen "text" når vi kjører en "query execution". Bruker kan gå inn i databasen på enheten og manipulere tabellen som ønskelig med SQlite sin browser.

Vi har i tillegg et ønske om en funksjon som skal lagre SQL databasen i en excel fil for å kunne visualisere tabellen på web-serveren.

Table: VoIP

New Record Delete Record

	Transfer	Bandwidth	Jitter	PLoss	PLossPerc	Currentdate	Currenttime
	Filter	Filter	Filter	Filter	Filter	25	Filter
1	246	67	0.457	0/838	0	25/05/2019	23:58:50
2	246	67	0.106	0/838	0	25/05/2019	23:55:50
3	246	67	0.462	0/838	0	25/05/2019	23:52:49
4	246	67	0.287	0/838	0	25/05/2019	23:49:50
5	246	67	0.174	0/838	0	25/05/2019	23:46:50
6	246	67	0.223	0/838	0	25/05/2019	23:43:49
7	246	67	0.531	0/838	0	25/05/2019	23:40:48
8	246	67	0.073	0/838	0	25/05/2019	23:37:48
9	246	67	0.577	0/838	0	25/05/2019	23:34:49
10	246	67	0.355	0/838	0	25/05/2019	23:31:49
11	246	67	0.237	0/838	0	25/05/2019	23:28:48
12	246	67	0.171	0/838	0	25/05/2019	23:25:49
13	246	67	0.561	0/838	0	25/05/2019	23:22:49
14	246	67	0.437	0/838	0	25/05/2019	23:19:48
15	245	66.9	1.227	1/838	0.12	25/05/2019	23:16:49
16	246	67	0.24	0/838	0	25/05/2019	23:13:47
17	246	67	0.469	0/838	0	25/05/2019	23:10:48
18	246	67	0.087	0/838	0	25/05/2019	23:07:48
19	246	67	0.927	0/838	0	25/05/2019	23:04:48
20	246	67	0.047	0/838	0	25/05/2019	23:01:47
21	246	67	1.068	0/838	0	25/05/2019	22:58:48

|< < 1 - 21 of 250 > >| Go to: 1

Figur 4.1: VoIP tabell i databasen med resultater fra UDP tester, som er vist i SQLite database browser.

Table: Http

New Record | Delete Record

	Transfer	Bandwidth	Retransmission	RTT	Currentdate	Currenttime
	Filter	Filter	Filter	Filter	25	Filter
1	38.4	32.1	164	33.242	25/05/2019	23:55:36
2	45	37.7	1	44.073	25/05/2019	23:50:36
3	50	41.8	0	79.753	25/05/2019	23:45:36
4	44.6	37.4	31	34.055	25/05/2019	23:40:36
5	48.7	40.8	65	31.972	25/05/2019	23:35:36
6	48	40.2	64	37.445	25/05/2019	23:30:35
7	41.3	34.6	303	30.006	25/05/2019	23:25:35
8	34.9	29.2	43	32.353	25/05/2019	23:20:35
9	48.9	40.9	88	29.946	25/05/2019	23:15:35
10	51.2	42.8	64	72.277	25/05/2019	23:10:35
11	49	41.1	64	44.675	25/05/2019	23:05:35
12	50.5	42.2	64	48.632	25/05/2019	23:00:34
13	51	42.6	32	134.324	25/05/2019	22:55:34
14	47.8	40	111	34.269	25/05/2019	22:50:34
15	49.4	41.3	64	38.746	25/05/2019	22:45:34
16	51.2	42.8	32	43.644	25/05/2019	22:40:34
17	49.7	41.7	64	60.104	25/05/2019	22:35:34
18	49.6	41.5	64	43.252	25/05/2019	22:30:34
19	43.4	36.3	65	24.636	25/05/2019	22:25:33
20	50.8	42.5	32	59.785	25/05/2019	22:20:33
21	52.3	43.7	96	49.346	25/05/2019	22:15:33

|< < 1 - 21 of 150 > >| Go to: 1

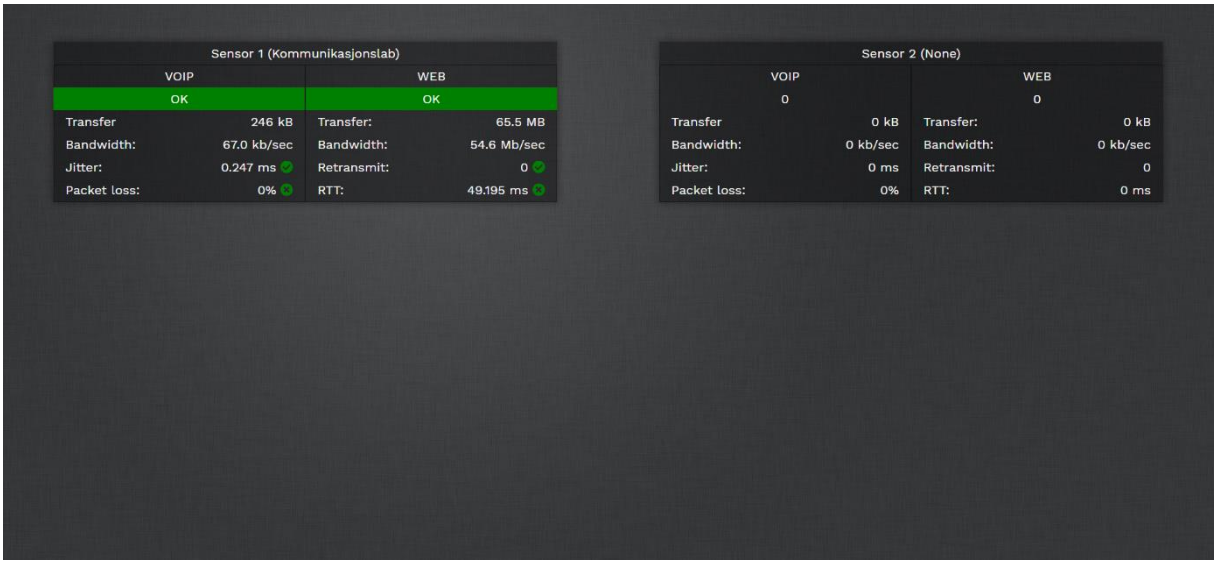
Figur 4.2: Http tabell i databasen med resultater fra TCP tester, som er vist i SQLite database browser.

4.5 Graphical user interface (GUI)

Målet med det grafiske grensesnittet var å kun vise essensiell informasjon hvor det ble vist på en oversiktlig og ryddig måte. Kravene til oppgaven var å vise status på både Voip og Http web trafikk i form av enten "OK" eller "IKKE OK" med tilsvarende farger, grønn eller rød. Vi valgte å gå for en nettleser-basert GUI da denne vil være lett tilgjengelig for bruker uavhengig av hvor personen befinner seg og hvilken enhet som benyttes.

Videre valgte vi å bruke Apache web server for å drifte nettsiden vår da Apache er "open-source" og det finnes mye dokumentasjon vi kan benytte oss av. Med en web server oppe kunne vi begynne å designe selve grensesnittet i HTML og CSS i tillegg til CSS-biblioteket Bootstrap. For at grensesnittet skulle være optimalt ønsket vi at kun informasjonen oppdaterte seg automatisk uten å oppdatere hele nettsiden. Dette fikk vi til ved å bruke JavaScript biblioteket jQuery, som er modulbasert og muliggjør dynamiske nettsider og funksjoner. Ved å bruke jQuery vil kun informasjonen vi ønsker på nettsiden oppdatere seg ved et gitt intervall vi selv velger.

Vi ønsker å ha en graf som viser historisk data i grensesnittet, som gir bruker et bilde på hvordan nettverket har oppført seg over tid. Dette var ikke et krav, men det hadde vært praktisk.



Figur 4.3: Grafisk grensesnitt (sensor 2 er ikke i drift, men viser muligheten for å benytte seg av flere sensorer.)

4.6 Fullstendig system

I det fullstendige systemet har vi lagt fokus på brukervennlighet. Det skal være lett å forstå seg på resultatene som presenteres. Her er det enda gjenstående arbeid for å gjøre den grafiske fremvisning enda litt mer intuitiv. Variablene som kan endres fra forskjellige nettverk henter vi ut fra den lokale tekstfilen. Om Iperf testene skal endres ligger disse også i samme tekstfil, dette gjør det enklere for brukere å konfigurere innstillinger som IP-adresse og parameternes størrelser på testene uten å endre på selve koden. Dersom størrelsene på testene endres må også uthenting av disse verdiene med "Regular expression" (RegEx) funksjonene endres.

Ved første gjennomgang av programmet vil begge tester threades, slik at de kan utføres seperat fra hverandre. Deretter utføres det nye målinger hvert tredje minutt for UDP og hvert femte minutt for TCP testen.

I sensoren sin kode åpnes filen med nødvendig informasjon for å sette opp sockets og Iperf strømmene og et objekt opprettes og tildeles informasjonen fra filen. Videre oppretter man kommunikasjon ved hjelp av sockets som vil prøve å opprette to forbindelser med serveren. Det utføres to ulike tester og for at det ikke skal oppstå problemer ved at begge testene sender ulike meldinger over samme socket, oppretter vi heller to sockets som er designert til hver test. Dersom kommunikasjon er opprettet mellom de to enhetene vil Iperf starte VoIP-sesjonen og HTTP-sesjon. For å oppnå kontinuerlig testing lager vi to while-løkker som ved fullført gjennomgang vil sende en ny melding om å starte testene og avvente svar fra server. Hvis sensor (klient) ikke klarer å opprette socket forbindelser med server vil det trigge en exception som gir bruker en feilmelding, for så å prøve å opprette en ny forbindelse etter et gitt tidsintervall helt til den lykkes.

Vi bruker sockets for å oppnå en "handshake" funksjonalitet mellom sensor og server. Forskjellige meldinger blir sendt mellom dem for å gi beskjed om at enhetene er klare til å starte en VoIP og HTTP sesjon. Programmet vil også settes på pause når den ene siden lytter over socketen. Dette bruker vi til vår fordel slik at programmet alltid vil kjøre med rett rekkefølge.

På server siden gjør vi det samme som for sensor. Tekstfilen med informasjon åpnes og et objekt lagres med IP-adresse, port og iperf kommandoer fra fil. Vi oppretter to server sockets med informasjon fra fil. Server lytter til den har to socket tilkoblinger, deretter klargjøres Iperf server når den får beskjed fra klient. Etter Iperf server er satt opp, svarer server med at den er klar til å utføre en ny test. På samme måte som sensor, er begge testene også threadet slik at de kan kjøres samtidig om intervallene skulle inntreffe på samme tidspunkt. For kontinuerlige tester har vi også brukt while løkker her. Ved fullført gjennomgang av løkken kaller vi "time.sleep()" funksjon, og oppgir ønsket antall sekunder koden skal sove. Siden sensor sender en melding *før* den lytter over socketen, er det instrukser klar når server skal gjenoppta koden. Etter en test er fullført vil dataene fra sesjonen lagres i en tekstfil lokalt på server. For å hente ut verdiene importerer vi RegEx biblioteket, og leter etter ønsket mønster, deretter hente ut verdiene og lagre dem i den lokale databasen.

De samme verdiene blir lagret i to forskjellige tekstfiler, en for Voip og en for Http. For å vise parameterne i sanntid på nettsiden blir tekstfilene lest i JavaScript og verdiene blir midlertidig oppbevart i scriptet. I samråd med oppdragsgiver har vi satt tallverdier for *jitter* og pakketap i VoIP testen og *retransmits* og *round trip time* i http testen.

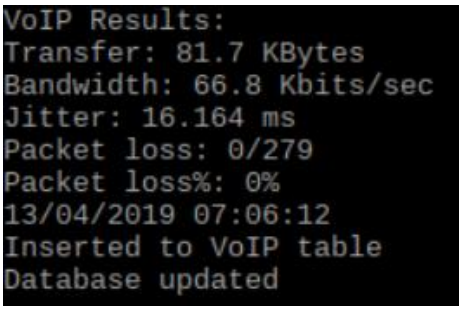
VoIP
Jitter: < 3ms
Pakketap: < 1,5%

For en typisk VoIP samtale vil man oppleve nedsatt samtalekvalitet med kun 1% pakketap. Dersom pakketapet overstiger 1,5% vil det dermed ikke lenger være akseptabel kvalitet på samtalen og status for VoIP settes til "IKKE OK" uavhengig av *jitter*. Dersom *Jitter* overstiger 3ms vil det også sette status på VoIP til "IKKE OK", uavhengig av pakketapet. Status på VoIP vil kun være "OK" dersom både *jitter* og pakketap tilfredsstilles mot tersklene.

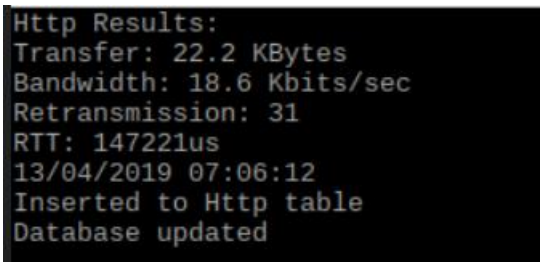
Http
Retransmittals: < 100
Round trip time: < 200ms

For http webtrafikk vil en bruker oppleve problemer dersom antall *retransmits* overgår 100 forsøk. Det samme gjelder dersom *round trip time* overstiger 200ms. Dersom én av parameterne ikke tilfredsstiller grensene vil status for webtrafikk settes til "IKKE OK".

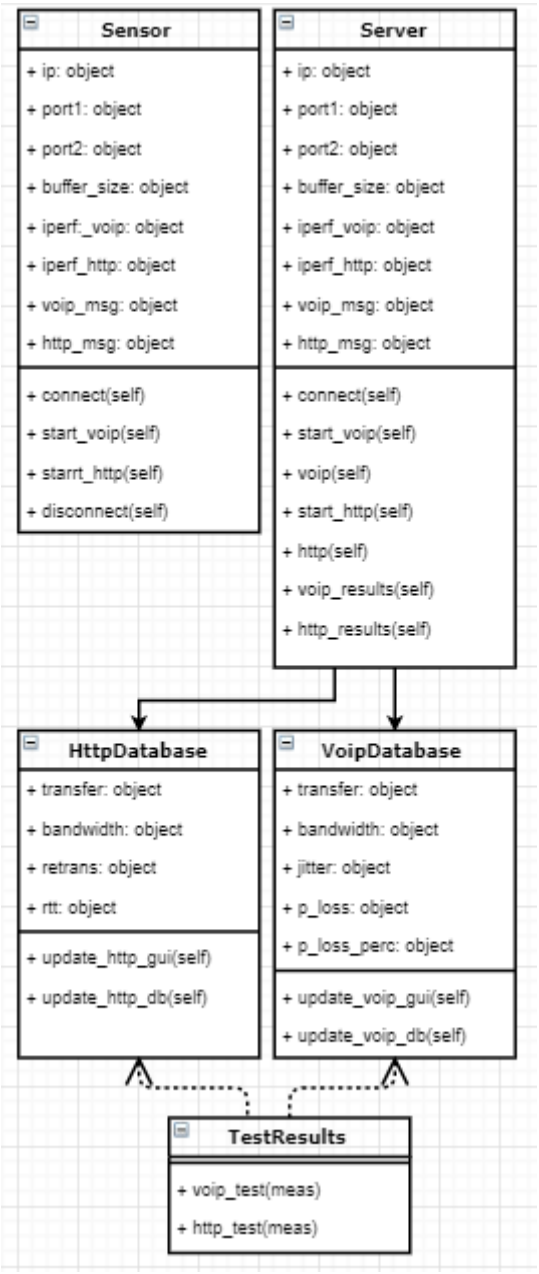
Ved å bruke jQuery vil kun ønskede elementer oppdateres i et intervall vi bruker. Dette intervallet har vi satt til ett sekund. Elementene vil da oppdatere seg hvert sekund ut fra tekstfilen med resultatene. Ingenting vil endres i grensesnittet før nye verdier fra den siste målingen overskriver tekstfilen. De nye verdiene vil bli oppdatert i grensesnittet senest ett sekund etter siste måling utføres. Verdiene blir dyttet inn og overkjører tidligere elementer i index.html filen ved å benytte JavaScript metoden "document.getElementById()".



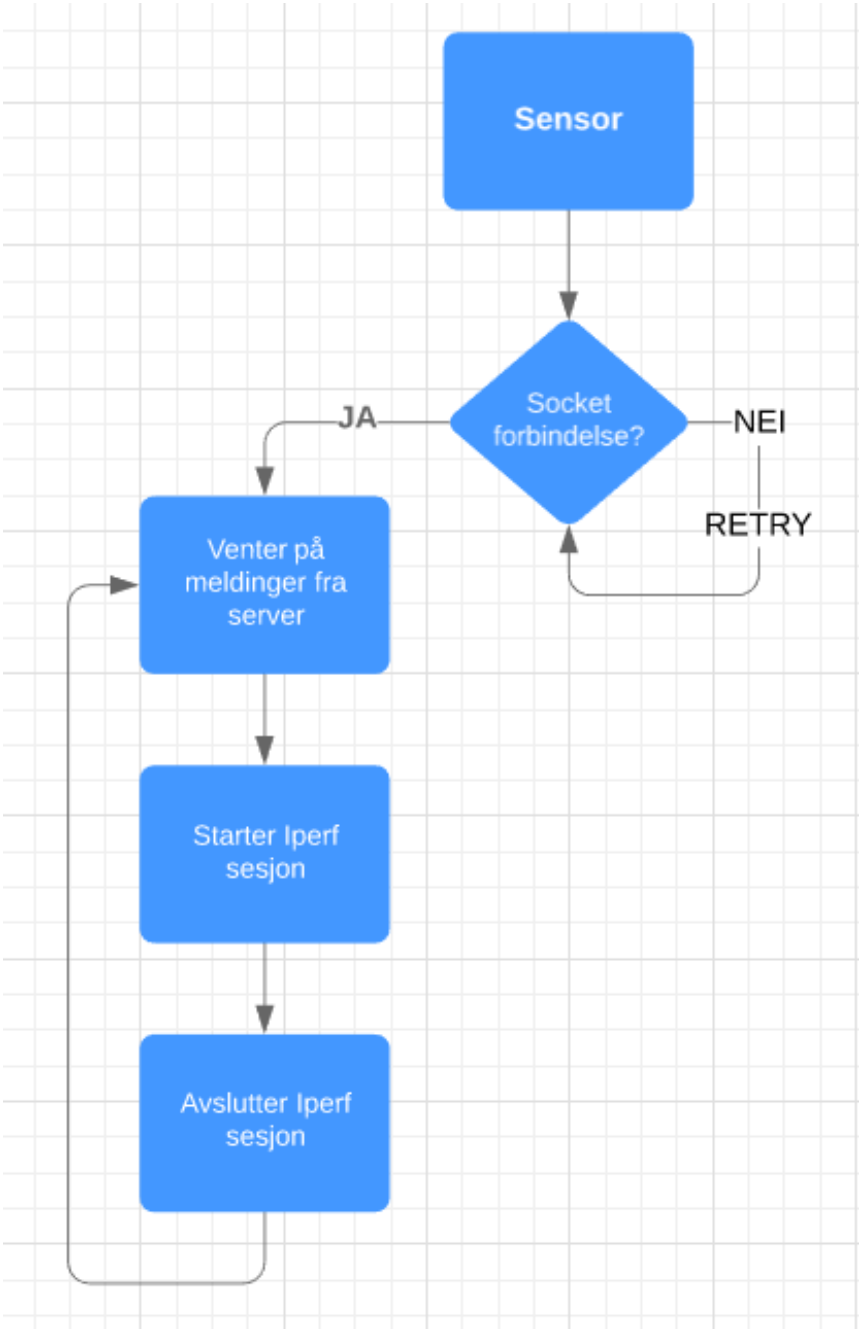
Figur 4.4: Resultater etter en UDP-test, sett fra konsoll vinduet.



Figur 4.5: Resultater etter en TCP-test, sett fra konsoll vinduet.



Figur 4.6: UML, Klassene og metodene i systemet



Figur 4.7: Flowchart for sensor

Figur 4.8: Flowchart for server

5 Testing

5.1 Testing på skolens gjestenettverk

Vi begynte med å utføre enkle Iperf tester på HVL sitt gjestenettverk (HVLguest), ganske tidlig. Før vi visste hvilke grenseverdier vi ville ha på målingene og før vi hadde et komplett system. Siden nettverket ikke var særlig belastet ga disse testene stabile resultater. Testene ble utført kun for vår egen del. Vi ønsket å se hvordan resultatene ville se ut før arbeidet med selve koden startet. For å kunne ta målingene ut av testene ble resultatene skrevet ut til en tekstfil.

En enkelt måling sier oss svært lite om helsen til nettverket så vi opprettet derfor en database for å lagre resultatene av alle utførte tester. Å kunne gå tilbake og se målinger fra tester som er utført, kan være svært nyttig i et slikt system. Da databasen var implementert var det lettere å se når nettverket var under belastning siden vi kunne bruke de andre testene som referanse. Målingene vi fikk fra testene var jevne. Vi opplevde sjeldent pakketap og når pakketap oppstod var det aldri mer enn noen få prosent. Der vi opplevde størst variasjon var på *jitter*. Men også denne var relativt jevn. Som regel var den lav og lå på rundt 1 ms, men kunne også stige periodevis til svært høy (30 ms).

5.1.1 UDP test

UDP testen kjøres hvert tredje minutt i 30 sekunder og skriver ut til en tekstfil med et intervall på 1 sekund. Vi prøvde å emulere en standard VOIP samtale som bruker G.711 codec. G.711 er en audio codec som gir lyd av høy kvalitet i 64 kbit/s, dermed satte vi også vår båndbredde til 64 kbit/s. For å oppnå enda mer realisme ønsket vi å legge til kommandoen for bidirectional testing. Da ville vi fått enda en linje med sammendrag av resultater når UDP testen var utført. Dermed måtte vi ha omstrukturert uthentingen av resultatene, i tillegg ønsket vi å hente ut alle målingene for alle intervallene i stedet for sammendrag av hele testen. I resultat filen for test kan man se at *jitter* kan variere kraftig over hele testen. Dette var da enda en grunn til at vi ønsket å hente ut verdier fra hvert enkelt intervall. Men ettersom testen var på 30 sekunder som er en relativt kort "samtale" så endte vi opp med å kun se på sammendraget.

5.1.2 TCP test

TCP testen kjøres hvert femte minutt i 10 sekunder. Vi vil emulere web-trafikk men her var det litt vanskeligere å finne ut av verdiene for transfer. Vi endte opp med å la Iperf kjøre en default TCP test. Dette er i grunn en speedtest, hvor client prøver å sende flest mulig bits per sekund. Etter at testen er over, sitter vi da igjen med båndbredden tilgjengelig mellom enhetene. For å få RTT fra resultatene på server siden, satte vi opp server som Iperf client og sensor som Iperf server. Årsaken er at Iperf server ikke har kjennskap til den totale tiden det tar for pakken å bli sendt fra client og tilbake. Round trip tiden som Iperf testen oppgir er svært høye og virket ikke helt korrekte. Dokumentasjon omtaler ikke akkurat hvordan denne parameteren måles. En løsning her kunne vært å implementere en vanlig "Internet control message protocol" (ICMP) ping test mellom enhetene og lagre denne i stedet for.

5.2 Testing på hjemmenettverket

De første testene som ble utført på hjemmenettverket ble gjort da ingen andre var hjemme, slik at belastningen var minimal. Da kunne vi se hvilke verdier man kunne forvente av nettverket. Som antatt ga testene tilbake gode resultater. I løpet av 3 timer ga UDP testen *jitter* verdier som aldri havnet under 0.5 ms og som rapporterte 0 pakker tapt. I samme tidsvindu rapporterte TCP testen en stabil båndbredde med gjennomsnitt på 45 Mbits/s. Det som var interessant var at selv om nettverket var stabilt, hadde vi også stor variasjon av retransmisjoner av pakker under TCP test, fra 0 til 128. Dette var noe vi ikke fikk da vi testet på HVLguest.

Etter videre testing ved lite belastning fikk vi de samme verdiene, som forventet. UPD testen ga like resultater som dagen før. *Jitter* parameteren var noe høyere men overskred ikke 2 ms. TCP testen rapporterte en gjennomsnittlig båndbredde på 55 Mbit/s, og nå med 0 retransmisjoner over et tidsvindu på 4 timer. Til slutt lot vi systemet utføre testene i 12 timer. Båndbredden til TCP testen varierte en del utover dagen. *Jitter* for UDP test var ganske stabil men man kunne se perioder hvor den rapporterte svært høye verdier for *jitter* før den gikk ned igjen.

Siden testene ble utført i et relativt stabilt nettverk, møtte vi ikke på noen nye *exceptions* før vi selv fremprovoserte feil. Vi slo av ruteren for å se hvordan programmet ville håndtere å miste nettverksforbindelsen. Uten nettverksforbindelse sendes det ikke flere meldinger over socketen og ingen videre instrukser vil bli gitt til enhetene. Når de har tilkobling til nettverket igjen fortsetter testene som før. Det samme vil skje om kun server mister internett-tilkobling. Slik koden er satt opp vil det skape problemer om kun sensor mister tilkobling og man vil måtte terminere programmet og starte det opp igjen.

6 Diskusjon

I forstudien ble det først antatt at vi skulle utvikle et system fra bunn av, med lite kunnskap om hva som fantes av eksisterende verktøy og løsninger som vi kunne ta i bruk. Gode tips fra veileder pekte oss i riktig retning, og valget av de løsningene var også noe Atea var kjent med og kunne anbefale oss å ta i bruk. Vi fikk dermed en god start på prosjektet.

Vi lærte mye underveis og etterhvert som vi fikk mer erfaring med programmering i Python og SQL gikk vi over til "object oriented programming" (OOP). Vi begynte veldig lett med "procedural programming" og fikk det til å fungere for en test, dette ble gjort for å få en idé om hvordan systemet kunne se ut, og for å lett kunne feilsøke ettersom vi ikke var så kjent med hverken Python eller Iperf. Dersom variabler skulle endres, måtte man gå gjennom hele koden og forandre på de manuelt. For et system som er tenkt å være et produkt ute på markedet er det rett og slett ikke godt nok.

Når vi skulle skalere programmet for implementasjon av en annen test og innføring til databaser fant vi ut at vi måtte gjøre hele systemet om til mer "object oriented" kode, vi brukte tid på å gjøre om koden og i prosessen om hvordan den kunne optimaliseres. Men tiden det tok for å lære seg OOP og overgangen fra static til OOP viste seg å være verdt den ekstra tiden da vi fikk en mye mer brukervennlig, oversiktlig og robust kode.

Når det kom til kalkulering av "MoS" fant vi ut at det ble vanskelig å finne riktig algoritme for vårt formål. "MoS" er vanligvis basert på tilbakemeldinger fra brukere som har brukt tjenesten, for eksempel etter en videosamtale så blir de bedt om å vurdere kvaliteten fra 1 – 5, men den kan også kalkuleres algoritmisk. Systemet vårt er ikke bygget på tilbakemeldinger fra brukere så vi måtte bestemme noen verdier for hva vi ville bruke i systemet. Etter et veiledningsmøte med Atea ble vi enige om å finne terskler basert på packet loss og *jitter* for UDP-testen samt *retransmittals* og *round trip time* for TCP-testen som Atea mener er akseptable.

7 Konklusjon

7.1 Videre arbeid og forbedringer

Systemet vi har utviklet er ment som et produkt som skal brukes ute hos bedrifter og virksomheter, som ikke har råd til dyre overvåkningssystemer. Det er per dags dato noen mangler som ikke gjør det til et fullstendig produkt. Systemet vårt fungerer som det skal, men vi må implementere noen "fail safe" funksjoner som skal sørge for at det kjører uten noen problemer dersom det oppstår strømbrudd for eksempel. Vi ønsker også å skrive en brukermanual for at brukere med minimalt med kunnskap skal kunne bruke dette produktet.

Vi har heller ikke opprettet noen ytterligere funksjoner som sørger for at bruker kan logge inn med brukernavn og passord, for å sørge for at det bare er brukere med rettigheter som har tilgang. Ved SSH tilkobling trengs bare IP-adresse, enhetens navn og passord til enheten. Vi ønsket også å ha en graf for databasen tilgjengelig på HTTP websiden, for å vise bruker historisk data.

En av kravspesifikasjonene vi fikk på starten var å ha et LED matrix display for å vise status på nettverket, dette ble valgt bort lengre uti prosjektet da grensesnittet viser god nok informasjon om sensoren.

7.1.1 Kode

Per dags dato fungerer koden men den er ikke særlig robust. Vi har jobbet en del for å sikre koden for feil og for å håndtere disse feilene, men gjennom testing oppdaget vi stadig nye feil som måtte håndteres. Når socket tilkoblingene er oppnådd vil koden kjøre i en while-løkke og testene vil bli utført kontinuerlig til programmet termineres eller en socket exception oppstår. Dersom enhetene skulle oppleve nettbrudd vil ingen meldinger sendes over socketen før de har internett-tilkobling igjen. Om nettverket skulle være nede en stund og enhetene får utlevert en ny IP-adresse ved tilkobling, vil dette gi en socket exception og programmet vil termineres. Man må så starte programmet på nytt manuelt. En mulig løsning på dette er å gi enhetene statiske IP-adresser på nettverket.

Vi skulle også programmere enhetene slik at de fant og lagret IP-adressen sin automatisk, for så å kjøre Python koden ved oppstart. Dette kom vi aldri til. Det er ikke en nødvendighet men en nyttig funksjon å ha. Om en ukjent feil inntreffer eller enhetene skulle oppleve strømbrudd kan man gjennom en terminal emulator som Putty, koble til enhetene via "Secure shell" (SSH) og restarte begge raspberries for å starte tester på nytt.

Koden skulle også blitt endret for uthenting av verdier. Vi henter kun sammendrag av test, når det kunne vært interessant å se på hele testen med alle intervaller. Ikke bare måtte koden endres, men også lagringen av disse verdiene i databasen. Siden vi kun lagrer sammendraget får vi en entry i databasen sin tabell. Om vi skulle lagret alle intervallene i den 30 sekunders lange testen ville det blitt ganske uoversiktlig. Vi har heller ikke implementert en mulighet for bidirectional UDP test. Ønsket om bidirectional kom fra oppdragsgiver litt ut i prosjektet og selv om det å aktivere selve funksjonen ikke var et problem, skapte det problemer for blant annet database metoden.

7.1.2 Grafisk fremvisning

Den grafiske fremvisningen holder mål opp mot kravspesifikasjonene vi har fått. Utenom kravene skulle vi gjerne lagt til flere funksjoner og laget et mer responsivt grensesnitt. Vi ville at en bruker skulle kunne konfigurere deler av systemet ved hjelp av en meny for innstillinger, som hadde forbedret den totale brukeropplevelsen.

Slik web-serveren er nå, må man koble seg til serveren via IP-adressen dens på samme nettverk. Dette er ikke en optimal løsning da vi ønsker muligheten for fjerntilkobling. Dette kan løses ved å åpne for portviderekobling mot server enheten. Når en tilkoblings forespørsel kommer via en spesifikk port som vi bruker skal forbindelsen viderekobles til en spesifikk enhet som vi selv angir. I vårt tilfelle Raspberry Pi som kjører Apache Http serveren. I praksis betyr dette at en bruker kan være tilkoblet et vilkårlig nettverk og likevel ha tilgang til web-serveren.

7.1.3 Autorisering

Sikring av database og web-server i form av autentisering er også noe vi ønsket å implementere. Databasen er en standard SQLite database, det vil si at ingen ytterligere justeringer er gjort for å sikre informasjonen den inneholder. For å få tilgang til websiden må man være koblet til det samme nettverket som serveren og ha IP-adressen. Et sikkerhetstiltak vi så for oss var et pop-up vindu med brukernavn og passord. Selve enhetene er heller ikke sikret med noe annet enn brukernavn og passord som vi selv satt under konfigurasjonen av enhetene.

7.1.4 Til slutt

Ellers har vi fullført oppgaven etter systembeskrivelsen og med resten av kravspesifikasjonene vi fikk av oppdragsgiver. Våre målinger emulerer TCP applikasjoner og en typisk VoIP UDP sesjon, vi har en sentralisert server som lagrer data lokalt og som viser status på nettverket.

Vi har utviklet et system som forhåpentligvis kan gi mer innsikt eller løse nettverksproblemer for mindre bedrifter, men det er absolutt rom for forbedringer. Vi ønsket å levere et system som hadde flere funksjoner og et bedre brukergrensesnitt.

Referanser

[1] Atea, *om Atea*. 2019. URL: https://www.atea.no/om-atea/

[2] Github, *samarbeidsverktøy og versjon kontroll*. 2019. URL: https://github.com/

[3] Google drive, *lagring og sikkerhetskopiering i sky*. 2019. URL: https://drive.google.com/drive/

[4] Iperf, *verktøy for aktive målinger i et nettverk*. 2019. URL: https://iperf.fr/

[5] SQlite, *bibliotek for å opprette database i C språk*. 2019. URL: https://www.sqlite.org/index.html

[6] Raspberry Pi, *Operativsystem og dokumentasjon.* 2019. URL*: https://www.raspberrypi.org/*

[7] Reddit, *Forum og diskusjon om nettverk*. 2019. URL: https://www.reddit.com/r/networking/

[8] IEEE Xplore, *Digital Library*. 2019 URL: https://ieeexplore-ieee-org.galanga.hvl.no/Xplore/home.jsp

[9] Regular expression operations. 2019 URL: https://docs.python.org/3/library/re.html

[10] Socket, *networking interface*. 2019 URL: https://docs.python.org/3/library/socket.html

[11] Stackoverflow, *online ressurs for koding*. 2019 URL: https://stackoverflow.com/

[12] Pandas, *data analysis*. 2019 URL: https://pandas.pydata.org/

[13] Kildekode til Iperf UDP. 2019 URL: https://github.com/esnet/iperf/blob/master/src/iperf_udp.c

[14] Apache, *Http server project.* 2019 URL: https://httpd.apache.org/

[15] Linode, *Network Throughput Testing with Iperf.* 2018 URL: https://www.linode.com/docs/networking/diagnostics/install-iperf-to-diagnose-network-speed-in-linux/

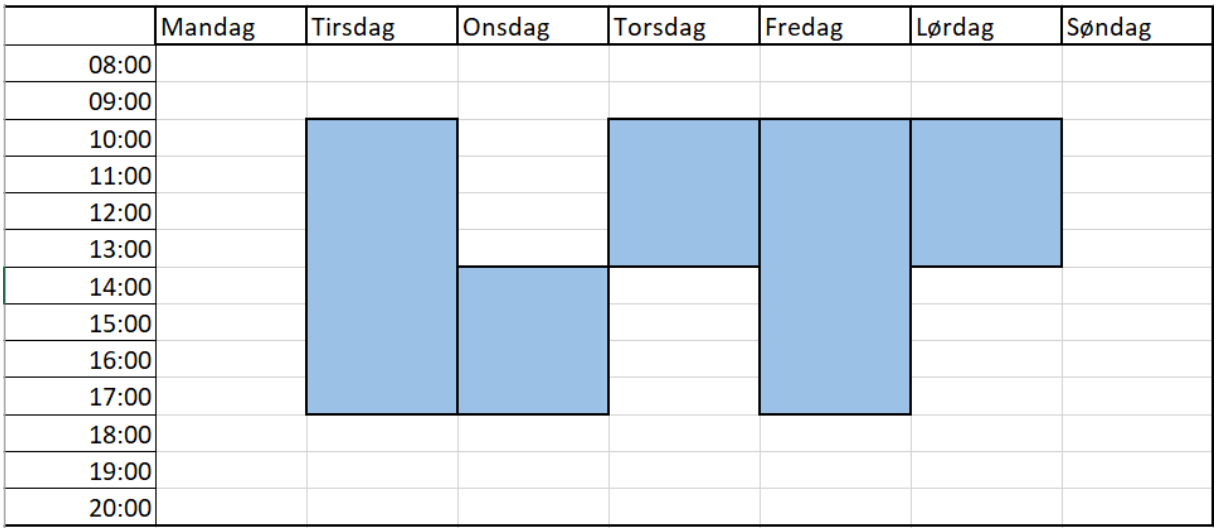
Appendiks

Appendiks A Forkortelser og ordforklaringer

ICMP	Internet Control Message Protocol
IDE	Integrated Developer Environment
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
MoS	Mean Opinion Score
NTP	Network Time Protocol
OOP	Object Oriented Programming
QoS	Quality of Service
RegEx	Regular Expression
RTT	Round Trip Time
SDK	Software Development Kit
SSH	Secure Shell
TCP	Transmission Control Protocol
VoIP	Voice over IP
UI	User Interface
UDP	User Datagram Protocol

Appendiks B Prosjektledelse og styring

Noe av det første vi gjorde da vi kom i gang var å bestemme arbeidssted og gå over timeplanene våre. Siden vi tar forskjellige valgfag viste det seg litt vanskelig å finne et klokkeslett som passet for alle. Vi fikk satt opp en timeplan til slutt som vi alle var fornøyd med. Selv om denne oppgaven tillater oss å jobbe selvstendig ville vi helst unngå dette.



Figur B.1: Timeliste

B.1 Prosjektorganisasjon

- Prosjektleder - Sondre Skodvin Mjåtvedt
- Teknisk ansvarlig - Carl Erik Koltveit
- Dokumentansvarlig - Hans Daniel Lambach

Siden vi har jobbet mye sammen på prosjekter i løpet av studiet vil vi prøve å fordele arbeidsmengde noenlunde likt. Vi har litt forskjellig kompetanse på ulike områder som gjør at noen må ta litt mer arbeid på f.eks. programmeringsbiten men alle skal få komme med sitt. Vi har gitt hverandre forskjellige roller og ansvar men vet av erfaring fra tidligere prosjekt, at vi alle bidrar med vår egen erfaring og at alle tre fremstår som ledere.

B.2 Prosjektform

Vi har jobbet mye sammen tidligere og ser ikke behov for ulike prosjektformer. Vi har god kommunikasjon innad i gruppen og er strukturerte, dermed kommer det ikke noen plutselige overraskelser når det kommer til deadlines. Vi jobber sammen på daglig basis, og har et godt forhold utenfor skole noe som gjør det mulig for oss å ikke jobbe så mye etter faste metoder.

B.3 Fremdriftsplan

					Uke	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14	w15	w16	w17	w18	w19	w20	w21	w22	w23	w24	w25	
					Man	7/1	14/1	21/1	28/1	4/2	11/2	18/2	25/2	4/3	11/3	18/3	25/3	1/4	8/4	15/4	22/4	29/4	6/5	13/5	20/5	27/5	3/6	10/6	17/6	
					Fre	11/1	18/1	25/1	1/2	8/2	15/2	22/2	1/3	8/3	15/3	22/3	29/3	5/4	12/4	19/4	26/4	3/5	10/5	17/5	24/5	31/5	7/6	14/6	21/6	
#	Aktivitet	Start Dato	Slutt Dato	Fram drift	Ansvarlig																									
1	Innhenting av oppgaver	1/9	1/11		Alle																									
2	Møte med oppdragsgiver	17/1			Gruppen																									
3	Forstudie arbeid	17/1	31/1		Gruppen	175 timer																								
4	Bestilling av Raspberry + tilbehør	19/1			Lambach																									
5	Formatering av rasberry	24/1			Koltveit																									
6	Forstudie innlevering	31/1			Mjåtvedt																									
7	Bachelor oppgave arbeid	1/2			Gruppen					1200 timer																				
8	Programmere sensor	1/2			Koltveit					200 timer																				
9	Programmere server	1/2			Mjåtvedt					200 timer																				
10	Iperf "VoIP" sesjon	4/2			Koltveit							20 timer																		
11	Sette opp database og script	1/2			Lambach					100 timer																				
12	GUI/Apache web server	18/2			Mjåtvedt							150 timer																		
13	Teste/kjøre applikasjon	4/3			Gruppen									200 timer																
14	Finpusse applikasjon	8/4			Gruppen														60 timer											
15	Midtveis presentasjon	1/4			Gruppen																									avtales med veileder
16	Demonstrasjon for oppdragsgiver	13/5			Gruppen																									avtales med Atea
17	Ferdigstille applikasjon	13/5			Gruppen																			50 timer						
18	Bachelor oppgave innlevering	31/5			Gruppen																									absolutt frist
19	Bachelor oppgave presentasjon	5/6	12/6		Gruppen																									avtales med veileder
20	EXPO / Avslutningsfest	13/6			Alle																									alle må stille

Figur B.2: Fremdriftsplan

B.4 Risikoliste

Problem	Sannsynlighet	Konsekvens	Tiltak
Fravær/Sykdom	Middels	Lav	God kommunikasjon og være fleksible
Mangel på utstyr	Lav	Høy	Vi har bestilt utstyr i god tid
Kode ikke ferdig i tide	Lav	Høy	Vi har planlagt og avtalt deadlines
Problemer med testing	Lav	Høy	Skrevet robust og god kode

Figur B.3: Risikoliste

Appendiks C Vedlegg

Se vedlagt zip-fil «BO19E-41 Vedlegg» for kildekode med timeliste

C.1 Kildekode

Vedlagt er kildekoden vår,

- *IperfServer.py* setter opp server sockets og gjør til rede for iperf test når den blir bedt om det.
- *IperfSensor.py* kobler seg til server sockets og starter iperf tester.
- *data_handler.py* håndterer alt fra uthenting av verdier, lagring av verdier i databasen. Verdiene sendes også til en tekstfil for resultater slik at JavaScriptet *UpdateRealTime.js* kan lese og vise målingene på websiden.
- *UpdateRealTime.js* leser de nyeste verdiene fra tekstfil og oppdaterer websiden med nye verdier. Inneholder også logikk for statusbaren.
- *Index.html* angir oppsettet av websiden. Biblioteket *Bootstrap* har blitt brukt.
- *Main.css* definerer utseende til *index.html*

Resultat tekstfiler og initialiserings filer er ikke vedlagt. Slik vi skrev programmet må man ha stien til filene for å kunne lese av, og skrive til disse.

C.2 Timeliste

Timelisten vår er vedlagt som Excel fil.