CHALMERS

Wireless Sniffer

Trådlös avlyssnings-enhet för analysering av trådlösa nätverk

Examensarbete inom Data- och informationsteknik

Per Nordeman
Jacob Aleniusson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2019

KANDIDATARBETE 2019

Analysering och test av avlyssnings-enheten Raspberry Pi 3B+ prestanda

Per Nordeman
Jacob Aleniusson

Institutionen för Data- och Informationsteknik

CHALMERS TEKNISKA HÖGSKOLA

GÖTEBORGS UNIVERSITET

Göteborg 2019

Analysering och test av avlyssnings-enheten Raspberry Pi 3B+ prestanda

PER NORDEMAN
JACOB ALENIUSSON

Handledare: Jörgen Kjellberg, Viktor Sandgren, Volvo Technology AB. Sakib Sistek, Chalmers Tekniska Högskola.
Examinator: Peter Lundin, Institutionen för Data- och informationsteknik

Kandidatarbete 2019
Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag: Trådlös avlyssnings-enhet, Raspberry Pi 3B+ samt externt nätverkskort.

Institutionen för Data- och Informationsteknik
Göteborg 2019

FÖRORD

Vi som har skrivit rapporten vill framförallt rikta ett tack till våra handledare Jörgen Kjellberg och Victor Sandgren på Volvo Trucks AB. Både för möjligheten att utföra arbetet men också för tankar och diskussion under arbetets gång.

Vi vill även tack vår handledare Sakib Sistek på Chalmers tekniska högskola, som assisterat med oss vid frågor och upplägg.

Per Nordeman & Jacob Aleniusson, Göteborg, 2019

SAMMANFATTNING

Dagens samhälle är kontinuerligt under en väldigt snabb utveckling sett till de flesta områden. Med ny teknik så börjar nya möjligheter att ta form, men ibland lämnar den snabba teknikutvecklingen brister efter sig. En av dessa brister är verifiering av den nya tekniken. Detta sker ofta då teknikutvecklingen sker väldigt hastigt. Ett område som ständigt står inför teknisk tillväxt och utveckling är fordonsindustrin. Elektriska fordon utvecklas världen över och präglar industrins framtid. Med det elektriska fordonets framkomst så tillkommer även digitalisering inuti och omkring fordonet. Samla statistik och få uppdatering av fordonets tillstånd är en väsentlig del i den moderna fordonsindustrin. Insamling av data sker ständigt av ett fordon, när den är i rullning, står stilla eller laddas. En stor del av denna insamling sker trådlöst och kan därför ibland vara osäker eller ha bristande kvalité. Denna data kan inte alltid en nå sin ämnade destination på grund av oförutsägbara faktorer, detta kan dock åtgärdas med hjälp av en tredje enhet, en avlyssnare. Systemen som idag diagnostiserar och verifierar data i fordonsmarknaden är väldigt dyra. Därför undersöks förutsättningar för att kunna konstruera en väldigt precis avlyssnare genom olika konstruktioner med hjälp av enkortsdatorn *Raspberry Pi 3B+*. Avlyssnaren analyserar datapaket som skickas mellan fordon och laddningsstation, där det primära målet är att utvinna resultat som kan användas för att bestämma vilken konstruktion och uppsättning som är optimal för avlyssnaren. Efter arbetet så konstaterades det att *Raspberry:s* interna nätverkskort har för låg prestanda för att fungera som avlyssnarenhet. Dock med externt nätverkskort och antenn presterar *Raspberry:n* godtyckligt som avlyssnings-enhet vid låga dataöverföringshastigheter. Men vid högre dataöverföringshastigheter blir dock hårdvaran bristande och begränsar således även möjligheterna att fungera som en avlyssnarenhet trots anslutet externt nätverkskort och antenn.

ABSTRACT

Today's society is continuously under very rapid development regarding most areas. With new technology, new possibilities start to take shape, but sometimes the fast technology development leaves gaps within itself. One of the gaps consists of lack of verification. This often occurs when the technology's development rate is very high. One of those areas were the technology growth is increasing at a high rate is the vehicle industry. Electrical vehicles is constructed all over the world and characterize the industries future. Together with its appearance comes the digitalisation both within and around the vehicle. While the vehicle is moving, standing still or charging a lot of data is collected from it. Mostly this occurs wireless and can sometimes be unsecure or lack in quality. Some of this data don't always reach its destination, this can be detected with the help of a third device, a sniffer. The systems which today diagnosis and verify data in the vehicle market are very expensive. Therefore investigation on how to construct a highly precise sniffer through different constructions on a *Raspberry pi 3B+*. The sniffer is supposed to analyze packets transmitted between client and server, the goal is to determine which setup the sniffer performs best with. The conclusion of the thesis is that the performance of *Raspberry Pi* internal network interface card(NIC) lack to function as a sniffer. With external NIC and antenna the ability to function as a sniffer are good at the lower data transfer rates, but at higher rates the ability decreases due to lack of the *Rasberry:s* internal hardware.

INNEHÅLLSFÖRTECKNING

Figurer

Terminologi

- Data - Teknisk representation av information, är oftast en sträng av 8 bitar
- User diagram protocol (UDP) - är ett förbindelselöst protokoll i transportskiktet för att skicka data över ett IP-nätverk.
- Transmission Control protocol/internet protocol (TCP) - är ett förbindelseorienterat protokoll som används främst inom kommunikation inom internet.
- Sniffer(avlyssnare/avlyssningsenhet) - Ett program/funktion som övervakar och analyserar nätverkstrafik.
- enkortsdator - En dator byggt på ett enda kretskort, med väsentlig hårdvara för att skapa en dator såsom minne, processor, in/ut funktioner (I/O).
- Wi-Fi - Trådlös nätverksteknik som gör att datorer och andra enheter kan kommunicera trådlöst.
- checksumma - Värde av ett datapaket för att kontrollera ifall de inte är korrupt.
- struktur-flaggor - Intern TCP-funktion som presenterar vad för slags packet det är (ACK, SYN, FIN).
- Internet protocol adress (IP-adress) - är en adress på internet som tillhandahålls när en ny enhet ansluts till ett nätverk, adressen identifierar anslutna enheter.
- bit - datavärde som endast kan anta värde 1 eller 0
- byte - 1 byte = 8 bit
- datapaket - Ett paket som innehåller information som skall överföras via nätet med en mottagar- och avsändaradress.
- Synkroniserings Paket (SYN-paket) - Synkroniseringsfunktion för TCP, samt när en kommunikations session skall påbörjas.
- Acknowledgment paket (ACK-paket) - Acknowledgment paket är ett bekräftelsepaket för att korrekt paket tagits emot.
- SYN/ACK-paket - Talar om vid påbörjad kommunikations session att anslutningen finns tillgänglig.
- Finish-paket (FIN-paket) - paket som skickas vid normal nedkoppling av en kommunikations session.
- Operativsystem - Är program som utgöra länken mellan datorn hårdvara och mjukvara (Windows, ISO, Linux).

- Comma separated values (CSV-fil) - En form av databas format, där varje värde är separerade med ett kommatecken.
- capture-fil - filformat för att spara ner internetpaket.
- Aircrack-ng - Terminalbaserat datorprogram med funktioner för att uppskatta säkerheten på ett nätverk, funktioner som att analysera och spara undan data, internetattacker, undersöka Wi-Fi korts duglighet.
- Python - Programmeringsspråk som är kraftfullt, smidigt och anpassningsbart.
- Monitor mode - Ett tillstånd som låter nätverkskortet att ta emot all datatrafik som sker på en viss trådlös nätverks kanal.
- Access Point (AP) - En enhet som skapar ett lokalt trådlöst nätverk för anslutning av till exempel en dator, mobiltelefon, surfplatta etc.
- Media Access Control adress (MAC-adress) - Är en unik adress och identifierare för varje nätverkskort.
- Controller Area Network (CAN) - CAN även kallad CAN-buss är ett säkert och snabbt kommunikationssätt som sker främst inom fordon. Kommunikationen sker mellan noder och styrenheter och kan ske antingen genom kabel eller trådlös.

1 INLEDNING

Det moderna samhället är allt mer på väg mot en värld av digitalisering och trådlös kommunikation. Nästan varje individ i världen har tillgång till någon form av teknik som arbetar trådlöst. Vägen mot digitalisering och trådlös kommunikation sker även inom fordonsindustrin och tillämpningen av den trådlösa utvecklingen blir allt mer lättillgänglig, då industrin präglas av elektriska fordon. Elektriska fordon lämnar utrymme till att implementera mer kraftfull hårdvara med interaktiva skärmar som visar flertaliga funktioner kring fordonet. Möjligheten och intresset att samla data kring fordon ökar även med denna utveckling, där den större delen av dataöverföring sker trådlöst. Vid applicering av den trådlösa dataöverföringen så kan kommunikationen vara bristande i oförutsägbara situationer. En alternativ åtgärd är att konstruera en tredje part som kan lyssna av datan som överförs mellan laddningsstation (server) och fordon (klient), detta tillvägagångssätt skulle kunna bidra till att öka säkerheten vid bristande kommunikation. Denna tredje part är en avlyssnare, som precist och trådlöst ska kunna verifiera data som skickas mellan en server och en klient. Avlyssnar-systemen som idag finns att köpa är extremt dyra och dessutom få, enligt Viktor Sandgren (Civilingenjör vid Volvo Technology Powertrain). Således finns en väldigt begränsad marknad som dessutom styrs av ett fåtal produkter. En alternativ, billigare men fortfarande precis lösning är således efterfrågade. Med hjälp av enkortsdatorn Raspberry pi existerar en hypotes att kunna ersätta dessa verifieringssystem. Det krävs då att den med hög säkerhet på mätningarna kan avgöra när data går förlorad.

1.1 Bakgrund

Det trådlösa nätverket är en av världens mest snabbväxande tekniker och sätter högre krav på hårdvara såväl som mjukvara. Samhället kräver högre hastigheter och hög kvalitet på kommunikationen. Som nämnt i avsnittet ovan sker även denna förändring inom fordonsindustrin. Projektet handlar just om att optimera och säkra den trådlösa kommunikationen. Arbetet har utförts på Volvo Technology AB (Volvo Group) på deras utgrening vid namn *Powertrain* som är specificerat på drivlinor. Problemet som skall hanteras är att säkerställa datapaketen som skickas trådlöst mellan fordon och laddningsstation, då denna kommunikation stundtals varit bristande. Arbetet är innefattat i en idé kring att göra en billig, effektiv och robust plattform för att logga data som skickas mellan fordon och laddningsstation. Plattformen skall agera som tredje part för att säkerställa att all data som utbyts är korrekt. Brister i en säker trådlös kommunikation är ett allmänt problem för flera fordonskategorier och källan för problemet är inte alltid identifierbart. Tidigare prototyper och test-enheter som är mindre kostsamma har haft problem med att bekräfta data som skickas mellan laddningsstation och fordon och därmed haft en begränsad överföringssäkerhet.

1.2 Syfte

Syftet är att undersöka möjligheterna att skapa en säker mobil enhet för avlyssning av trådlös dataöverföring mellan fordon och laddningsstation. Enheten skall preliminärt fungera som en "backup" för data när kommunikationen mellan laddningsstation och fordon blir bristfällig. Enheten skall designas utifrån aspekten att optimera prestandan i förhållande till en låg kostnad. Undersökning av avlyssnarens olika prestanda vid olika uppsättningar är således primär.

1.3 Mål

Målet är att ta fram förslag på plattform för ett robust prisbilligt system för att avlyssna kommunikation mellan fordon och laddningsstationer. Arbetet innebär i första hand att hitta en uppsättning av mjukvara och hårdvara som optimerar avlyssnaren prestanda i förhållande till ett relativt lågt pris. Arbetet skall även undersöka för- och nackdelar med respektive lösningsförslag. Utvärdering av mjukvara är hur olika program och programmeringsbibliotek analyserar nätverkstrafik, medan utvärdering av hårdvara innebär test av prestanda för internt nätverkskort och olika externa nätverkskort. Sammanställt är det som ska åstadkommas en sorts studie för att visa på vilket sätt en avlyssnare får bästa förutsättning att fungera så bra som möjligt, samtidigt som det klargörs vad som inte fungerar och eventuella lösningar.

1.4 Precisering av frågeställningar eller hypoteser

De flertaliga och komplexa faktorerna till varför kommunikation mellan fordon och laddningsstation blir bristfällig är något som kan analyseras av en tredjeparts avlyssnings-enhet. Frågeställningar och hypoteser som skall besvaras är; ifall *Raspberry Pi:s* egna nätverkskort klarar av att avlyssna dataöverföringen mellan klient(fordon) och server(laddningsstation). Ifall frågeställningen kring *Raspberry Pi:s* nätverkskort skulle visas vara bristande, leder frågan vidare till vilken uppsättning av externa nätverkskort som skulle kunna ersätta *Raspberry Pi:ens* nätverkskort. Frågeställningar som besvaras för de olika uppsättningarna är:

- Vid vilken signalstyrka blir avlyssnings-enheten bristande?
- Hur känslig är enheten mot utomstående störningar?
- Vilken dataöverföringshastighet mellan server och klient klarar avlyssnings-enheten av?
- Klarar avlyssnings-enheten av 5 GHz nätet?
- I vilket intervall presterar avlyssnings-enheten bäst, i 2,4 GHz nätet eller 5 GHz nätet?

1.5 Avgränsningar

Projektet kommer ej innefatta konstruktion av en gemensam plattform, utan kommer endast testa *Raspberry Pi:s* egna hårdvara och olika uppsättningar extern hårdvara. Under projektet är det inte en prioritet att kunna spara data i gemensamt filformat. Data som skall avlyssnas skall vara generell och inte specificerad för någon viss typ av datapaket. Däremot kommer data som undersöks och hanteras enbart vara TCP och UDP protokoll. Ty den ena enheten använder sig av CAN-data så ligger även projektet till grund för att med både samlad "WiFi-data" och CAN-data optimera felsökningen ytterligare. Under arbetstiden kommer dock avlyssnaren enbart att hantera "WiFi-data". Avlyssnings-enheten kommer under hela projektet bestå av en *Raspberry Pi* 3B+ men med eventuella uppsättningar av externa nätverkskort med tillhörande antenner.

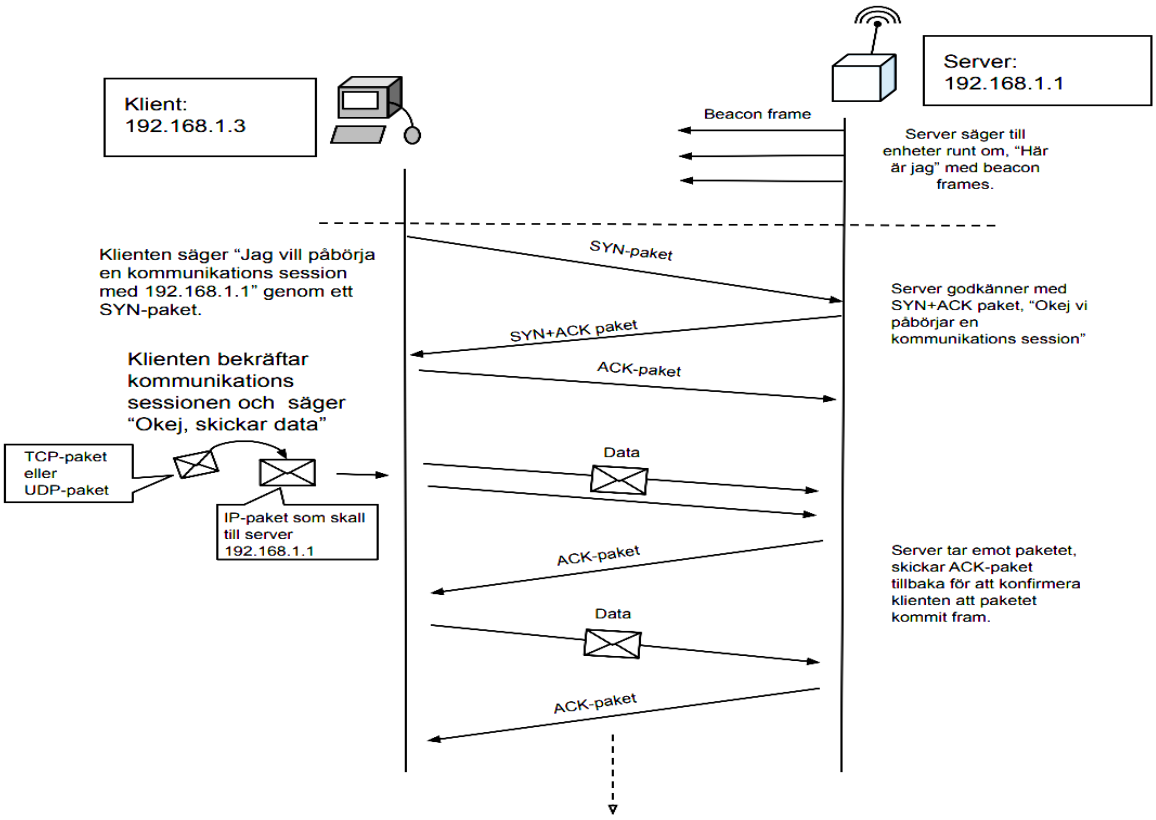
2 TEKNISK BAKGRUND

En stor del av den moderna trådlösa kommunikationen kan framstå som abstrakt och komplex. För att undvika missförstånd och misstag har man i samband med den trådlösa kommunikationens uppkomst tagit fram protokoll och riktlinjer för hur kommunikationen skall ske. I dagens samhälle sker mycket av kommunikationen trådlöst och främst genom trådlösa nätverk i form av mobil kommunikation och Wi-Fi. Trådlösa nätverken består också av dessa protokoll och riktlinjer. Ett exempel på riktlinje är vilken frekvens och bandbredd det trådlösa nätverket får använda sig av. Medan protokoll är hur paket av data skall hanteras och hur kommunikation upprätthålls. Dessa protokoll och riktlinjer är innefattade i olika modeller som används vid trådlös kommunikation. Den vanligaste modellen är TCP/IP-modellen som består av fem skikt varpå de vanligaste typerna av protokoll som sker i internettrafiken, TCP och UDP, existerar i det tredje skiktet.

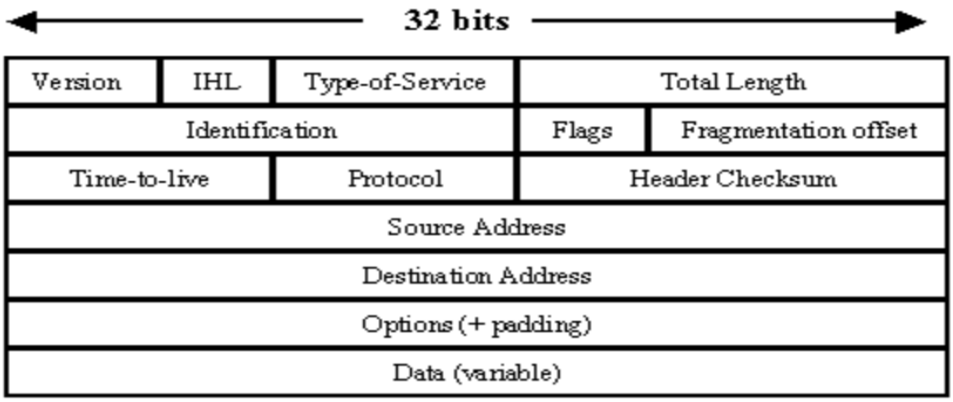
2.1 TCP/IP-modellen

TCP/IP-modellen är det som används mest inom trådlösa nätverk. Modellen består av fem skikt där skikten är uppdelade utifrån skiktens uppgift. De fem skikten är applikation, transport, nätverk, datalänksskikt och det fysiska skiktet. Varpå TCP och UDP protokollen existerar i transportskiktet.

I figur 2.1 visas händelseförloppen för kommunikation på ett trådlöst nätverk mellan en klient och en server. Händelseförloppet förklara först anslutningsfasen för klienten till servern och sedan hur kommunikationen sker. I förloppet så ses även hur TCP eller UDP-paket omsluts av ett IP-paket för att skickas mellan server och klient. I kapitel 2.4 så redovisas detta händelseförlopp mer ingående. Utseendet på IP-och TCP-paketets modell kan ses i figur 2.2, där längden av paketet består av 32 bitars sekvenser. Figur 2.2 beskriver utseendet av TCP/IP-paket. De sex första sekvenserna är funktionsbitar för TCP/IP-kommunikation. Varpå resterande sekvenser varierar i bitar beroende på datan som skickas.



Figur 2.1 Kommunikationsprocess Server och Klient. De tre första pilarna uppifrån är Serverns funktion att visa att den finns tillgänglig för anslutning. De tre pilarna under den streckade linjen beskriver anslutningsfasen. Resterande pilar visar en vanlig kommunikation.



Figur 2.2 IP-pakets arkitektur.

2.2 Kommunikationsprocess

Vid kommunikation via internet så skickas stora mängder information i form av ettor och nollor. Dessa ettor och nollor representeras i binär talform och kallas ofta för bitar som vid kombination kan bilda oändligt antal siffervärden.

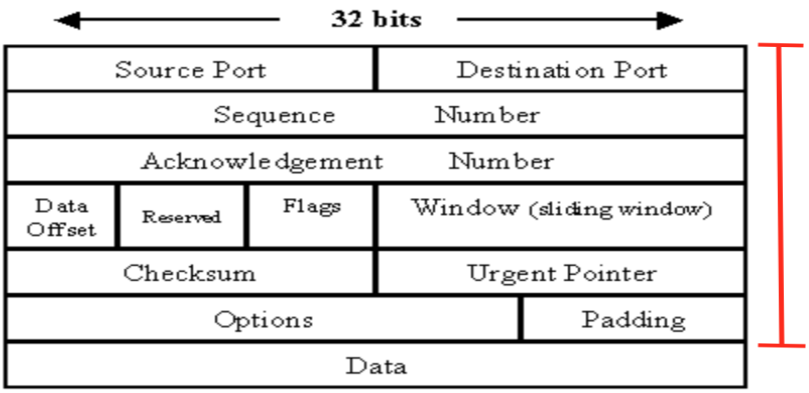
Vid kommunikation så bryts till exempel ett meddelande ner i bitar för att sedan skickas i en viss ordning. Ordningen av bitar bestäms utifrån meddelandets status. Olika status kan vara ifall meddelandet skall vara krypterat, vad för protokoll som skall användas eller vad det är för storlek på meddelandet. Mottagaren tar sedan emot dessa sekvenser av ettor och nollor för att sedan bygga ihop till ett paket och presentera det för mottagaren.

2.3 Internet protokoll

Ett länge känt protokoll inom trådlösa nätverk är IEEE 802.11 standarden. IEEE 802.11 är en samling nätverksstandarder som har utvecklats sedan internets uppkomst. Standarderna är specificerade för WLAN (Wireless ethernet) där de sätter riktlinjer för tillåten frekvens och dataöverföringshastighet för det trådlösa nätverket [4]. Datatrafiken som sker via trådlösa nätverk använder sig av protokollen TCP och UDP.

2.3.1 TCP-paket arkitektur

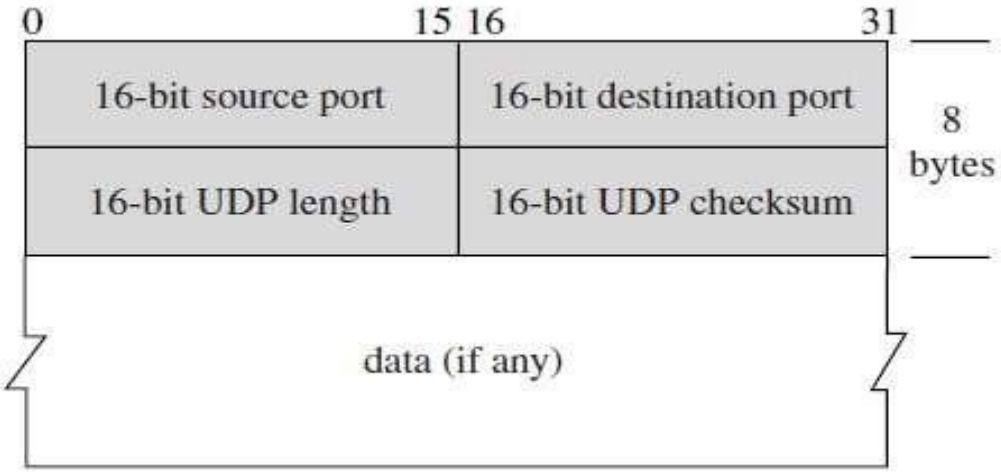
TCPs uppbyggnad består av 32 bitars sekvenser, där de sex första sekvenserna innehåller väsentlig information kring paketet. TCP-paketets uppbyggnad kan ses i figur 2.3. Ur TCP paketet kan destination (Destination port) och källadress (Source Port) utvinnas, sekvensnummer (Sequence number) som är ett nummer för TCP-protokollets ordningsfunktion, acknowledgement numret (Acknowledgement number) anger det sekvensnummer som nästa paket från mottagaren skall ha (ordningsfunktion), Data offset bestämmer de tre första sekvensensernas storlek, "Reserved" bitarna används sällan och brukar vara satta till 0, Flaggorna (Flags) anger vilken typ av TCP-paket det är, till exempel SYN, ACK och FIN. Fönster storlek (Window) anger kontinuerligt mottagarens buffertkapacitet för framtida paket. Checksumman kontrollerar så att paketet inte blivit korrupt. Brådskande data eller pekare på brådskande data (Urgent Pointer) är till för att TCP skall upprätthålla sin ordning av paketflödet.



Figur 2.3: Ett TCP-pakets arkitektur, det röda strecket innehåller paketets information varpå den sjunde och sista raden innehåller den anmäla informationen som skall transporteras.

2.3.2 UDP-paket arkitektur

Ett UDP-pakets arkitektur efterliknar ett TCP-paket till stor del, men innehåller mycket mindre funktioner se figur 2.4. Paketets längd består av 32 bitars sekvenser där de två första sekvenserna innehåller källadress, destinationsadress, funktionssekvensernas längd och checksumma som kontrollerar att paketet inte är korrupt. De resterande sekvenserna innehåller den allmänna datan som presenteras för användaren[5].



Figur 2.4 UDP-pakets arkitektur.

2.4 Paketflöde för TCP och UDP

Paketflödet beskriver hur datapaket utbytes mellan server och klient. Nedan redovisas paketflödet för TCP- och UDP-protokoll.

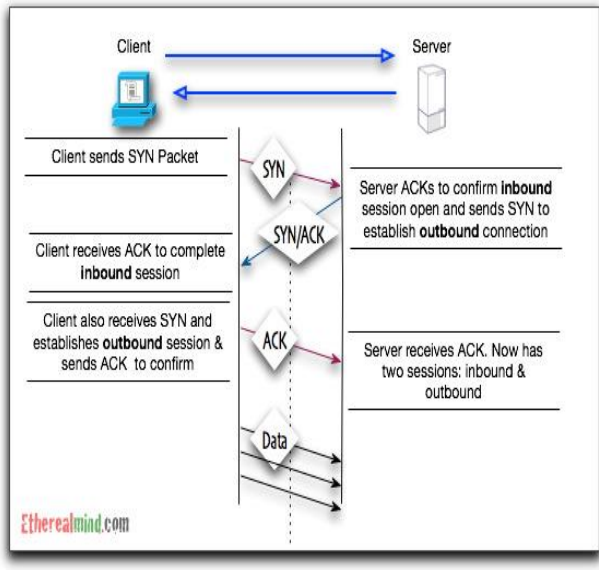
2.4.1 Paketflöde TCP-protokoll

Flödet av ett TCP-paket/nät kan ses i figur 2.5 och 2.6. Vid anslutningfasen så skickar klienten ett SYN-paket för att upprätthålla kommunikation. Server svarar klienten med ett SYN/ACK-meddelande för att bekräfta för klienten att server tagit emot paketet och att data kan skickas till server. Efter mottaget SYN/ACK-meddelande hos klient så skickar därefter klienten ett ACK-meddelande tillbaka, för att meddela server att data är på väg och att klienten har

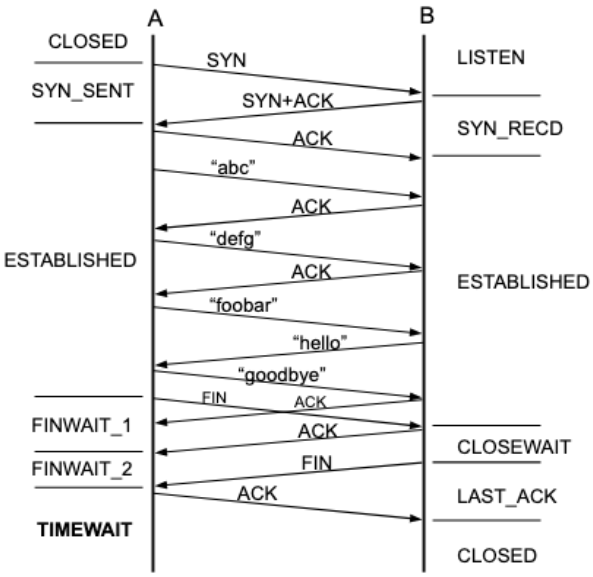
möjlighet att ta emot data. Efter upprättad inåtgående och utåtgående kommunikation mellan klient och server så skickats data.

Processen av SYN-, SYN/ACK, ACK-meddelanden kallas för “3-way-handshake” för att etablera en säker TCP-kommunikation. Fördelen med TCP-kommunikation är att TCP håller koll på borttappade paket och därmed ser till att paket som gått förlorade skickas igen. TCP-kommunikationen kontrollerar att paketen kommer fram i rätt ordning och ordnar även om ordningen på paketen om de kommit fram fel. Detta är med kostnaden av att dataöverföringshastigheten minskas då TCP-kommunikation innehåller extra funktioner.

TCP-kommunikationen kontrollerar vilken ordning paketen skickas med hjälp av sekvensnummer och ACK-nummer. ACK-värdet ges utav sekvensnumret adderat med paketets storlek adderat med ett. Sedan blir detta ACK-nummer det nya sekvens numret i datatrafiken. Vid en avslutad anslutning eller avslutat paketflöde så presenteras en FIN-meddelande för att berätta för klienten eller server att det inte finns inget mer att skicka eller att ta emot.



Figur 2.5 Visar anslutningsfasen mellan server och klient.



Figur 2.6: Vanligt paketflöde av datapaket.

2.4.2 Paketflöde UDP-protokoll

Paketflödet hos UDP skiljer sig till stor del från TCP. Server och klienter utbyter SYN och SYN/ACK-meddelanden för att upprätthålla kommunikation liknande TCP (figur 2.5). När servern tar emot UDP-paket från klienten så skickas ett ACK-meddelande tillbaka för att berätta att paket tagits emot. Ifall serverns ACK-meddelande skulle överskrida ett tidsintervall,

beroende på till exempel en dålig länk mellan server och klient, så kommer klienten göra ett försök att skicka paketet som överskridit sändningstiden. Denna återsändning sker normalt bara en gång, men kan justeras ut efter behov. Klienten kan därför inte garantera att server tar emot alla datapaket.

Till skillnad från TCP så hjälper inte UDP-protokollet till att hålla koll på paketen eller paketens ordning. Ett färre innehåll av funktioner resulterar i att datatrafiken sker betydligt snabbare än vid användning av TCP, men med konsekvens att datatrafiken inte blir lika säker[8].

2.5 Säkerhet över internet

Ju mer vi går mot ett digitaliserat samhälle så sparas även vår privata information digitalt. Vi kan enkelt via en dator eller smartphone logga in på banken eller skicka privata mail. Mycket av den privata information som sker digitalt kan enkelt ses utan något slags skydd. För att skydda dataöverföring som sker trådlöst har man skapat protokoll för att kryptera datan som skickas.

Inom lokala trådlösa nätverk så används säkerhets protokollen WPA eller WPA2 som står för Wi-Fi Personal Access. Varpå WPA2 är ny utveckling av WPA för att korrigera dess säkerhetsproblem. För att en krypterad dataöverföring skall fungera enligt WPA protokollet så krävs det att systemen som kommunicerar har samma krypteringsnyckel i början och slutet av en dataöverföring session. Utan samma krypteringsnyckel så kommer inte data som överförs kunna paketeras upp och presenteras[9][10].

2.6 Mjukvara för kommunikation

Inom programmering så förekommer användningen av programmeringsbibliotek. Biblioteken innehåller redan konstruerade funktioner som underlättar skrivning av programkod samt reducerar antalet rader kod som skrivs. Ett känt internt bibliotek i operativsystemet *Linux* som förekommer vid konstruktion av programkod i trådlösa nätverk är *libcap.* Biblioteket *libcap* erbjuder en funktion kallad *socket* eller *network socket* vilket tillåter programmeraren skapa en ändpunkt för att skicka ellet ta emot data på specificerad port eller IP-adress[11] .

2.7 Program och programmeringsbibliotek

För att analysera datatrafik finns det flertaliga verktyg och funktioner att använda. Det mest lättillgängliga verktyg som används vid analys av datatrafik är *Wireshark*[12]. Programvaran är gratis och används inom företag som väl i utbildande syfte. Programvaran har ett flertaliga funktioner för att analysera datatrafikens alla aspekter. *Wireshark* tillåter användaren att se

datatrafiken i realtid och kan enkelt skriva in filter för att presentera önskade pakettyper, adresser, protokoll. Datatrafiksgrafer och grafer över paketflöden är några andra funktioner som används väl vid analys av datatrafik. Wireshark finns även som textbaserad funktion men kallas då Tshark. Användning av Tshark är väldigt likt *Wireshark* med filterfunktioner. Dock får användaren själv välja vilka parametrar som skall presenteras[13]. Innehållet som Tshark presenter kan även sparas ner i valfritt filformat.

2.7.1 Python

Python är programmeringsspråk som baseras till stor del av användning av öppna bibliotek med tillhörande funktioner. Python tillåter användaren att arbeta snabbt och integrera programkod mer effektivt. Pythons utbud av bibliotek gör att språket blir mångsidig[14]. Vid sortering av CSV-filer så förekommer ett användbart bibliotek vid namn Pandas. Pandas är ett python baserat bibliotek som används för dataanalys. Pandas innehåller funktioner som låter användaren att orientera sig i olika textbaserade filer. Pandas kan enkelt stega genom en CSV-fil för att till exempel leta efter specifika nummer, ersätta rader eller kolumner[15].

3 METOD

3.1 Teoretisk metod

Vid arbete rörande trådlös kommunikation krävs det förståelse för data som överförs. Således behövs undersökning av internetprotokoll och olika typer av internetpaket utföras för att underlätta förståelsen under projektets gång. Undersökning av liknande projekt och liknande program har också förekommit för att garantera en bra start på projektet. Dessutom har ingående studier av hur TCP- och UDP-protokollet beter sig genomförts. Vilka är en del av 802.11 protokollet som beskrivits ovan i kapitel 2. Sammanfattningsvis har detta genomförts för att komma fram till vilken typ som är bäst att använda vid tester och som ger tydligast resultat. Protokoll som kommer undersökas är TCP och UDP. TCP används initialt i projektet, vid senare tester har UDP använts.

3.2 Plattform och operativsystem

Arbetsmiljö etableras på *Raspberry Pi* 3B+. I projektets fall så skall operativsystemet *Kali Linux* implementeras på *Raspberry Pi* som agerar avlyssnare. *Kali Linux* är det operativsystem som har flest gynnsamma funktioner och förutsättningar för det som efterlyses hos avlyssnar-enheten. För att analysera de avlyssnade paketen används gratisprogrammet *Wireshark* [WS]. På den andra *Raspberry Pi* 3B+ används standardsystemet *Raspbian*. Programkoden kommer främst bestå av programmeringsspråket Python för att underlätta arbetsgången. För att eventuellt optimera prestandan så kan programkoden behövas brytas ner i ett snabbare programmeringsspråk som till exempel C.

3.3 Mjukvara

3.3.1 Mjukvara för avlyssning

För att lyssna av kommunikationen mellan klient och server skall en “script” skrivas för avlyssning av överförda data, som sedan testas i en simuleringsmiljö. Scriptet är skrivet i bash shell vilket är radbaserad interpretator som gör att programmet körs direkt i terminalfönstret och inte behöver kompileras. Strukturen av scriptet är till för att förenkla och automatisera testprocessen. Scriptet skannar först efter önskat nätverksnamn med hjälp av interna funktioner som finns i operativsystemet. Scanningen presenterar vilken kanal och MAC-adress nätverket har. Ifall det önskade nätverket hittas så skall avlyssnings-enheten gå in i monitor mode för att se all data som sker på den kanalen nätverket ligger på. Med hjälp av datainsamlingsverktyget *Aircrack-ng* så kan specifik kanal och nätverk avlyssnas varpå all dataöver ing som sker på nätverket sparas i en capture-fil.

Aircrack-ng samlar även in information som är väsentlig för efterfrågade tester. Med *Aircrack-ng* så kan signalstyrka mellan nätverk och klient fås, samt signalstyrka mellan nätverk och

avlyssnings-enhet. Information som datahastighet och datamängd registreras också av *Aircrack-ng*. För att kunna dekryptera informationen som skickas mellan klient och server så krävs krypteringsnyckel. Med hjälp av *Aircrack-ng* funktion *Aireplay-ng* så kan denna nyckel genereras genom att tvinga enheter som är ansluta till nätverket att återansluta. När enheterna återansluter till nätverket så fångar *Aireplay-ng* krypteringsnyckeln och dataöverföringen kan dekrypteras. Att använda *Aircrack-ng* funktion *Aireplay-ng* är bara nödvändig ifall nätverket som skall avlyssnas har en säkerhetsnyckel.

3.3.2 Mjukvara för Server

Servern blir en punkt som tar emot data som klienten skickar. Således är koden uppbyggd för att ta emot data via serverns IP-adress. För att överflödig data inte ska skickas mellan klienten och servern så är även mottagningen av data port-specificerad. I serverns kod finns en angiven port som data tas emot via, denna port är öppnad i server-enheten och tillåter bara trafik av vald data typ (TCP/UDP). Koderna för TCP- respektive UDP-server är nästan identiska. Dom skiljer sig enbart genom vilka funktioner i olika bibliotek de olika datatyperna ska tas emot. TCP-servern behöver dessutom en återsändningsfunktion för att bekräfta mottagning klienten, detta är typ-specifikt och behövs inte för UDP.

3.3.3 Mjukvara för Klient

Klienten är konstruerad för att skicka en specifik datamängd av specifik typ. Koden är således, likt serverns, beroende av olika bibliotek för vilken datatyp som önskas skickas. Klienten skickar då data till en specificerad IP-adress (serverns) i koden. Klienten skickar denna data via en port som måste matcha serverns. Data som skickas utvidgas med hjälp av multiplikation till efterfrågad storlek (150 bytes). En räknare anger hur många paket som ska skickas och en implementerad delay på 0.1 sekunder gör så att sändningshastigheten matchas den som används i fältet. TCP-klienten har även en funktion för att ta emot bekräftelser från TCP-servern.

3.3.4 Verfieringsmjukvara

Då avlyssning sker på specifik kanal och nätverksadress gör att mängden data som avlyssnats minimeras. Dock är det mycket paket som fångats upp som inte hör till testerna, dessa behöver filtreras bort. För att verifiera att alla hundra paket som skickats mellan klient och server kommit fram så har verifieringsmetoden bestått av två delar. Den första delen är *Wireshark* som är ett specifikt program för att analysera internetpaket. Med *Wireshark* kan man filtrera bort trafiken som inte hör till önskat protokoll men också mellan vilka IP-adresser som datan skickats. Den andra metoden har varit att bryta ner capture-filerna som skapats av *Aircrack-ng* till att endast sända paket som skickas mellan server och klient. För att bryta ner capture-filerna så har ett script skrivits i programmeringsspråket python och *Wireshark:s* Bash shell baserade

funktion *Tshark*. Tillvägagångssättet har varit att först filtrera bort oönskade paket och spara över önskade parametrar till en CSV-fil med hjälp av *Tshark*. Sedan läser python skriptet in den sparade CSV-filen för att processas. För att processa CSV-filen som sparats undan har python biblioteket Pandas används. Pandas biblioteket innehåller många funktioner för att orientera och behandla CSV-filer och underlättar arbetet.

Python skriptet letar efter datan som skickas mellan server och klient som innehåller en räknare från 1-100 i hexadecimaltal (0-63). Räknaren ger varje paket ett unik innehåll som kan urskiljas med hjälp av python. Python scriptet stegar genom CSV-filen och letar efter det unika innehållet. Sedan spara python scriptet ner de de paket som är mellan start(0) och stop(63) värdet i CSV-filen. När processering av CSV-filen skett räknar scriptet sedan olika parameterar som genomsnittlig sändningstid, maximal sändningstid och minimal sändningstid. Den räknar även genom antalet matchningar som finns i CSV-filen. Antalet matchningar motsvara antalet paket som kommit fram medan de som inte kommit fram har ansetts som ett förlorat paket. Scriptet räknar även ut den genomsnittliga signalstyrkan mellan server och avlyssningsenhet.

3.4 Simuleringsmiljö och tester

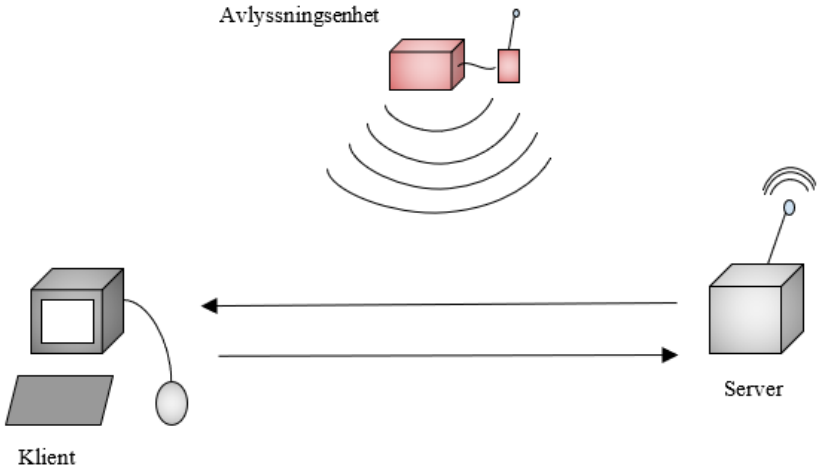
Acces point (AP) kommer representeras av en router som agerar server genom internetdelning, klienten är ansluten till serverns delade nätverk. Avlyssningen utförs av *Raspberry:n* som lyssnar på kommunikationen som sker mellan APn och klienten. Klienten kommer att vara ytterligare en *Raspberry pi 3B+*. Denna har dock bara ansvar för att skicka data till servern som avlyssnar-enheten undersöker. För kontrollera att enheterna kommunicerar med varandra samt överfört rätt typer av data så görs snabba tester i kontorsmiljön. Dessa tester kommer dock påverkas avsevärt av bakgrundsstörningar, således behövs en simuleringsmiljö för att testa själva frågeställningen. Genom att sätta upp en störningsfri simuleringsmiljö så kan tydliga resultat åstadkommas för att nå en så optimal slutsats som möjligt. Optimalt sett så är simuleringsmiljön helt störningsfri där enbart klienten, servern och sniffer är aktiva enheter. Dessvärre är det väldigt svårt att hitta dessa miljöer, därför kommer testerna ske i ett relativt isolerat rum i anslutning till Volvo. Störningar kommer således vara omöjliga att isolera helt vilket som kommer att tas i akt vid analysen av resultaten. Rummet som testerna utförs i är även begränsat i storlek, således kommer signalreglering ske genom att klä antenner/avlyssnare i aluminiumfolie. Detta dämpar styrkan som avlyssnaren har gentemot nätet mellan server och klient.

Tester som skall utföras i simuleringsmiljön kommer bestå av 10 tester för varje parameter förändring, dvs förändring av avstånd, signalstyrka, frekvensband, antenn, datatyp och vid scenariot då t.ex. en vägg blockerar vägen mellan avlyssnare och nätet. För varje test så kommer enbart datamängden vara konstant för att matcha den mängd som projektet efterlyser. Ur testerna kan sedan statistik utvinnas för att verifiera vilka parametrar som ger bäst resultat. Grafer skapades för att visualisera hur de olika parametrarna påverkar varandra.

4 KONSTRUKTION

Vid konstruktion av avlyssnings-systemet så användes en *Raspberry Pi 3B+* som representeras av det röda blocket i figur 4.1. *Raspberry:n* har ingen anslutning till varken server eller klient, utan ligger som en utomstående enheten. Det enda förhållandet *Raspberry:n* har till klienten och servern är att den avlyssnar datatrafiken på de frekvenskanaler klienten och server kommunicerar på. Det här var även ett av kraven för avlyssnings-enheten, som inte skulle ha någon anslutning till varken server eller klient. Då den större delen av hårdvaran utgjordes av *Raspberry:n* så skedde majoriteten av konstruktionen på mjukvaran. Den del av hårdvaran som varierade var externa nätverkskort med tillhörande antenner som anslöts till *Raspberry:n* via USB. Mjukvarodelen som utvecklats gjordes för att automatisera samt analysera testerna.

Klienten representeras av ytterligare en *Raspberry pi 3B+* vars mjukvara är konstruerad för att skicka ett önskat antal datapaket av önskad storlek och protokoll. Server består av en dator som är kopplad till en AP som gör det möjligt för klienten skicka datapaketen till den specifika IP-adressen(serverns IP-adress). Mjukvaran för servern presenterar datapakten som skickas av klienten i realtid för att kontrollera att alla paket kommit fram.



Figur 4.1: Illustration av fysisk uppsättning

4.1 Avlyssnarenhet Hårdvara & mjukvara

Avlyssnar-enheten består av *en Raspberry Pi 3B+* med operativsystemet *Kali Linux*. Till skillnad från operativsystem *Rasbian* som används vanligtvis av *Raspberry Pi* så är *Kali linux* ett mer lämpat operativsystem för nätverks analysering. Med *Kali Linux* medföljer det redan installerade programvaror och programmeringsbibliotek som är avsedda för nätverkanalysering. *Kali linux* låter även användaren att ständigt befinna sig som “superuser”

vilket ger privilegier till göra avancerade systemförändringar som underlättar arbetsprocessen[16].

Vid val av avlyssningsprogram eller verktyg så faställdes *Aircrack-ng* att vara det mest lämpliga verktyget vid nätverks analysering. *Aircrack-ng* är ett mångsidigt verktyg som tillhandahåller många parametrar för nätverket som analyseras. Verktyget tillåter användare att enkelt gå in och ut ur monitor mode med funktionen *Airmon-ng*. Mäta hastighet och signalstyrka för nätverken samt lyssna på nätverk med specifik IP-adress och frekvenskanal med funktionen *Airodump-ng*. *Airmon-ng* och *Airodump-ng* är subfunktioner till *Aircrack-ng*. *Aircrack-ng* medföljer även i *Kali Linux* vilket underlättar arbetsprocessen då allt redan är installerat [17].

Verifiering och analysering av fångade datapaket utfördes på programmet *Wireshark*, *Tshark* och ett eget konstruerat script i programmeringsspråket Python. *Wireshark* och *Tshark* medföljer *Kali linux* och är verktyg som är väldigt frekvent inom nätverk analysering som tidigare nämnt i kapitel 2.8. Vidare så analyserades de fångade datapaketen med hjälp av *Tshark* och python script. *Tshark* sparar ner önskade parametrar från fångade datapaket med hjälp olika filter till en CSV-fil. CSV-filen behandlas sedan av ett python script som presenterar genomsnittlig signalstyrka, antalet önskade paket som kommit fram (se kapitel 3.3.4) samt antalet fel flaggor som skett under tiden de 100 önskade paketen skickas[Appendix B2-B5].

4.2 Avlyssnare Hårdvara

Hårdvaran för avlyssnaren är en *Raspberry Pi 3B+* med tillhörande externa nätverkskort. Då en hypotes fanns kring *Raspberry:s* nätverkskort att den eventuellt inte skulle klara av olika dataöverföringshastigheter eller för brusiga kanaler så inkluderades två externa USB nätverkskort. De två externa korten har krav att klara av 5GHz bandet samt att kunna arbeta i monitor mode.
De två korten är *ALFA AWUS051NH* och *ALFA AWUS036ACH* där de båda uppfyller kraven som ställts[18].

5 TESTNING

Under arbetet så utfördes tester vid två olika tillfällen, vid det första så gjordes tester med transmission av TCP-paket. Vid det andra och slutgiltiga testet så gjordes tester med transmission av UDP-paket.

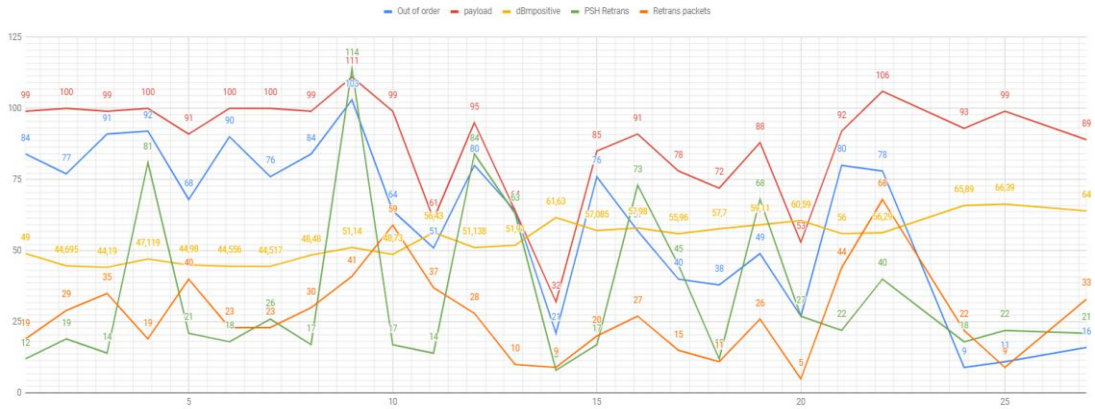
5.1 TCP-tester

TCP-protokollet fungerar på så sätt att paketen skickas från klienten till servern, när paketet kommit fram till servern skickar servern tillbaka en bekräftelse att paketet kommit fram. Om paketet däremot inte anländer hos servern utan går förlorat under sändningen så ber servern klienten att skicka om paketet. Det innebär att i princip alla TCP-paketen alltid kommer att komma fram till servern under testerna. Detta gör det svårt att räkna de paket som går förlorade. Eftersom vid varje test skickas det iväg 100 paket mellan klient och server. Av dessa så kommer alltid alla 100 paket komma fram, men alla kommer inte komma fram första gången dem skickas. Det är alltså bara de paket som kommit fram direkt som kan anses ha sänts på rätt sätt. De andra räknas som bidrag till paketförluster. En annan nackdel med analysen av TCP är att Wireshark ibland har svårt att skilja på vad som är en återsändning och vad som inte är det. Detta innebär att när data skickas väldigt fort vilket den gjorde vid de första TCP-testerna så kommer avlyssnaren via Wireshark ibland uppfatta att fler paket har kommit än vad som har skickats, alltså fler än 100. Detta inträffar alltså när Wireshark inte kan skilja mellan en vanlig sändning av paketet och en återsändning, vilket uppstår på grund av snabba hastigheten som är resultatet av två möjliga bakomliggande faktorer. Första faktorn kan vara att servern inte är tillräckligt snabb att hantera paketen, till slut “dumpar” servern paketen på grund av brist i hårdvarans prestanda vilket missuppfattas av *Wireshark* som klassar paketen som det vore att de inte kommit fram. Den andra faktorn kan vara att paketen anländer så tätt att att *Wireshark* klassar dem som återsändningar eller “ur turordning”. *Wireshark* har en känd fallgrop som innebär att paket med för kort mellanrum mellan ankomsterna har svårt att klassificeras av *Wireshark*. För att motverka detta så kan en delay läggas in hos klienten så att servern hinner behandla alla paket, detta gjordes dock inte under under de dessa första TCP-tester och är således en möjlig felkälla.

5.1.1 Initiala TCP-tester

Eftersom första TCP-tester utfördes i en relativt störningsfri miljö kan det antas att enbart enheternas prestanda kunnat påverka resultaten. Testerna gjordes även enbart med Raspberry Pi:s interna nätverkskort och inte med antenn. Utformningen av testerna bestod av olika avstånd mellan de tre enheterna. Vid det första testet så var avståndet 5 meter mellan respektive enhet. Vid det andra testet 10 meter och slutligen 20 meter. Figur 5.1 nedan visar en sammanställning av testerna. Värt att nämna är att just dessa tester utfördes utan någon hastighetsbegränsning. Hastighet mellan servern och klienten är således så hög som 2.4GHz-

nätet, enheterna och signalstyrkan tillåter. Detta innebär att det blir svårt för avlyssnaren att hinna med vissa sändningar.



Figur 5.1: Graf där X-axeln representeras av de olika tester (0-9, 5 meter) (10-19, 10 meter) (20-29, 20 meter). Y-axeln representer data för de olika sändningarna av TCP-paket. Blå = paket anlände i fel ordning, Röd = antal paket som kommit fram, Gul = signalstyrka i dBm, Grön = antal tvingade återsändningar, Orange = antal återsändningar.

I grafen framgår det att alla sändningar av paket kräver ett viss antal återsändningar. Vid de tio första testerna kan också en relativt stabil mottagning av ungefär 100 paket utläsas (punkter 0-9 på X-axel i figur 5.1). Två väldigt höga peakar av tvingade återsändningar gestaltar sig även i grafen, med största sannolikhet som resultat av att servern har varit långsam att ta emot paketen. Alternativt att paketen från klienten har skickats för hastigt efter det framtvingade “3-way-handshaket”. I båda fallen så lägger sig paketen som ska skickas i en “kö”, när kön tillslut blir full så tvingar klienten servern att ta emot dem ovillkorligen. Paketen som anländer i fel ordning (blå linje) är även de förhållandevis många under de första tio testerna. Detta beror med största sannolikhet på att klienten och servern har en bra signalstyrka relativt varandra vilket gör att klienten sänder paketen med en mycket högre hastighet, en hastighet som servern inte klarar av att behandla helt korrekt. Signalstyrkan (gul linje) som avlyssnaren har till nätverket mellan klienten och server är hög under dessa första tio tester, eftersom samma avstånd hålls mellan alla enheter så antas att när denna minskar/ökar så sker detsamma för signalen mellan server och klient. Observera att signalstyrkan i grafen är i absolutbelopp, det vill säga att ett mindre värde i grafen betyder ett större värde i verkligheten, och vice versa.

5.2 UDP-tester

Paket som skickas via UDP-protokollet beter sig, som beskrivet tidigare, annorlunda gentemot de som skickas med TCP-protokollet. UDP skickar iväg data även om klienten inte får någon bekräftelse från servern att den har mottagit paket. Således elimineras problemet med att ha

paket som kommer i fel ordning helt. Återsändningar av paket minimeras också avsevärt. Problemen för Wireshark att klassificera paket elimineras i samma process då UDP skickar iväg alla paket utan mottagningsbekräftelse, så skickas de i princip alltid med samma mellanrum. Detta innebär att det inte skapas någon "kö" som klienten tvingar servern att till slut ta emot. Således blir kraven på *Wireshark* mindre och paket kan uteslutande sorteras i rätt kategori. Vissa UDP paket kan återsändas, varje paket kan dock återsändas högst en gång och det kan bara återsändas om klienten själv noterar det. Beteendet är beroende av koden, i klientens fall så uppstår dessa fall via socketen. Med hjälp av avlyssnaren klassar *Wireshark* återsändningarna som "*Retries*" och dem kan då filtreras bort via flaggan *fc.retry* som sätts positiv vid en återsändning. Retry paket räknas in som förlorade paket i UDP-testerna eftersom de egentligen inte nått sin destination som tänkt och utan socketen så hade dem inte kommit fram.

I UDP-testerna så har uppsättningen av enheterna ändrats. Istället för att ha ett varierande avstånd mellan servern och klienten så har dessa nu hela tiden en god signal mellan varandra. Istället modifieras enbart avlyssnaren och får således en varierande signalstyrka relativt nätet mellan server och klient. Till skillnad från tidigare tester, TCP, så ger detta en tydligare bild av själva avlyssnaren prestandan.

5.2.1 Sluttester UDP

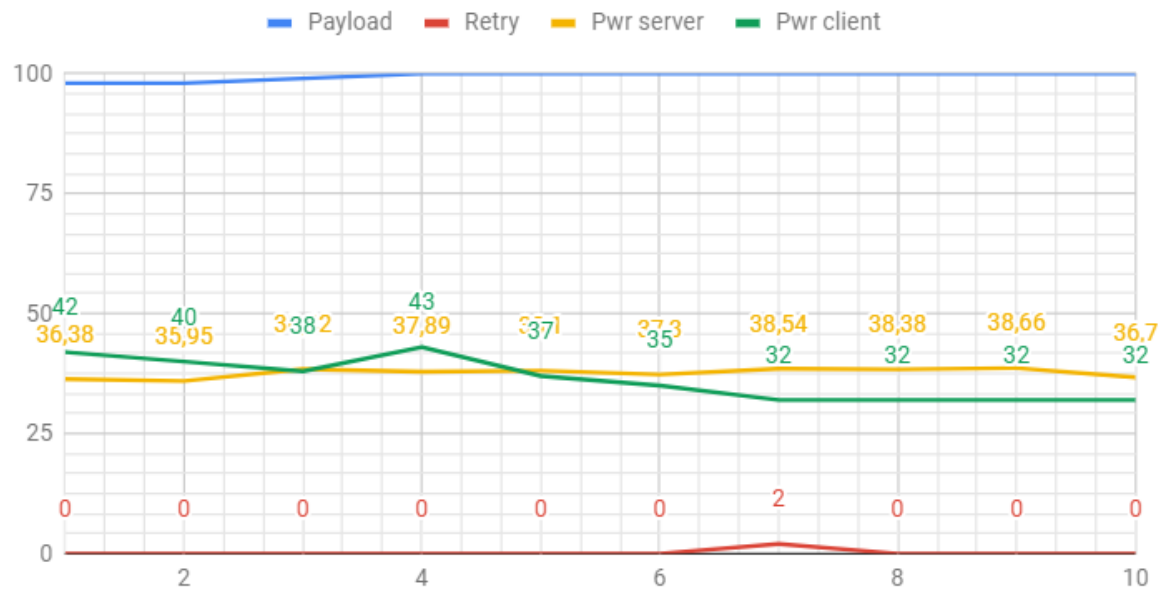
Sluttesterna för UDP har utförts i flera steg. Först så utfördes tester där *Raspberry pi 3B+* interna nätverkskort användes för att avlyssna sändning på olika signalstyrkor relativt sändningen, dessa gjordes enbart på 2.4GHz-nätet eftersom det inbyggda kortet inte klarar 5GHz. I sluttesterna begränsades dataöverföringshastigheten mellan klienten och server till att varje paket skickas med 0,1 sekunders mellanrum. Vid varje mätpunkt (signalstyrka) görs mätningar i serie om fem eller tio stycken. Senare tester utfördes genom att ansluta en antenn till avlyssnaren. Testerna för antennen utförs på liknande signalstyrkor som för det interna kortet. För att dämpa signalstyrkan på antennen så kläddes den i olika mängd folie, vilket dämpar mottagar-förmågan. För antennen utförde först tester på 2.4GHz- och sedan 5GHz-nätet. Två olika antenner användes för de olika näten. För 2.4 GHz användes en *ALFA awus051nh[20]* och för 5 GHz en *ALFA-awus036ach[21]*. Vid användning av 2.4GHz eller 5 GHz näten så kan 802.11 protokollet variera. Variationen är beroende av hastigheten på nätet vilket gör att data paketeras upp olika för varje 802.11 protokoll [22]. Generellt så gäller att ju högre hastighet som paketen skickas med ju mer komplex upp-packningsmetod krävs och ställer i sin tur högre krav på hårdvaran som behandlar den. Den mest avancerade metoden som påträffas under testerna är 802.11ac vilken bara tillämpas vid 5GHz. Följande gäller och är relevant för testerna, vid 2,4Ghz så kan 802.11-b, -n, -g för 5GHz används 802.11-a, -n, -ac. För fullständig översikt se tabell 5.1.

IEEE Standard	802.11a	802.11b	802.11g	802.11n	802.11ac
Frekvens	5GHz	2,4GHz	2,4GHz	2,4/5GHz	5GHz
Max hastighet	54 Mbit/s	11 Mbit/s	54 Mbit/s	600 Mbit/s	1 Gbit/s

Tabell 5.1: Visar använda 802.11-standarder under tester.

5.2.1.1 Raspberry Pi interna kort

Dessa tester utfördes med hjälp av *Raspberries* interna nätverkskort. Främst för att få en relativ utgångspunkt för att kunna bedöma antennprestanda men också för att se i vilken utsträckning *Raspberry pi* kan användas som avlyssnarenhet utan extern hårdvara. Räckvidden på det interna kortet är väldigt liten jämfört en (extern) antens, *Raspberry:n* hade då problem med att hitta nätet när signalstyrkan mot nätet blev mindre än -45 dBm. Således gjordes enbart tester på det interna kortet tills dess att signalen mellan nät och avlyssnare inte kunde erhållas längre. En visualisering av dessa resultat ges i figur 5.2.

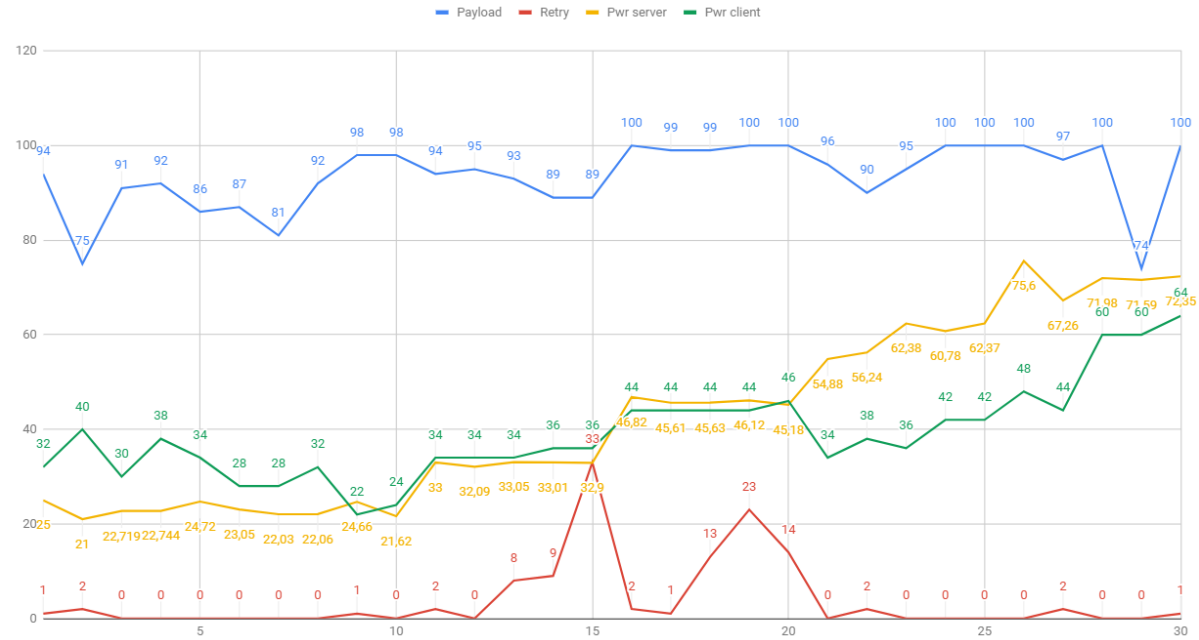


Figur 5.2: Resultat från tester med Raspberry pi 3B+ interna nätverkskort på 2,4GHz-nätet. X-axeln representerar 10 tester med en varierande signalstyrka mellan -32 till - 43 dBm sett till avlyssnaren. Y-axeln representerar olika data vid UDP-testerna. Blå = antalet paket som anlänt. Grön = signalstyrkan från klienten till servern. Gul = signalstyrkan från servern till avlyssnaren. Röd = antalet återsändningar som gjorts under testet.

Under ovanstående tester i figur 5.2 så har servern och klienten haft väldigt god signalstyrka relativt varandra. Detta medför att felkällor mellan server och klient minimeras avsevärt och avlyssnarens prestanda blir i fokus. Genomgående för alla tester är att avlyssnaren kan urskilja i princip alla paket som kommer fram till servern från klienten. I de två initiala testerna så missas ett antal paket av avlyssnaren. Troligt är att den relativt låga signalstyrkan till servern gör att avlyssnaren inte ser dessa två paket. Att lyckas hålla en helt jämn signalstyrka mellan sändning av paket är väldigt svårt att åstadkomma och kontrollera. Således missas antagligen paketen på grund av en låg signalstyrka i skedet då de ska fångas upp av avlyssnaren. Signalstyrkan från avlyssnaren till servern relativt signalstyrkan från servern till klienten växlar under testerna. Anledningen till detta beteende är svårt att definiera då inga ändringar har tillämpats mellan testerna. Stor sannolikhet är att någon utomstående störning har påverkat anslutningen. Störningar har försökts att isoleras i så stor mån som möjligt men är svåra att eliminera helt, därför är det möjligt att det i detta fall varit en störning som påverkat resultatet. Antalet återsändningar som klienten försöker göra är som förväntat väldigt få eller inga alls hela test-serien (figur 5.2) igenom. Detta beror framförallt på den goda anslutningen mellan klienten och servern. Avlyssnaren identifierar två stycken återsändningar vid det sjunde testet. Möjligt är att det finns återsändningar i de två första testerna som avlyssnaren inte har lyckats identifiera.

5.2.1.2 ALFA AWUS051NH 2,4 GHz-nät

Testerna när den första antennen anslöts genomfördes genom att en bärbar antenn av modell *ALFA AWUS051NH* kopplades till avlyssnarenheten. Antennen stödjer enligt datablad 2,4GHz- och 5GHz-nät [DB1]. Testerna med denna antenn utförs dock enbart på 2,4GHz-nätet då antennen visade sig för svag för att lyckas analysera 5GHz-nätet. Testerna som görs med denna antennen utförs vid den första mätpunkten i en serie bestående av tio tester, resten av mätpunkter utförs i serier om fem tester per mätpunkt. Anledningen till detta är framförallt för att undersöka inom vilket intervall på signalstyrkan som antennen presterar bäst. Under de första testerna har den mycket god signal till servern/nätet och därför är det lätt att tro att det är där antennen presterar som bäst. För att kunna verifiera det så väl som möjligt så gjordes därför en utökad serie tester vid de styrkorna. Totalt utfördes trettio tester med antennen på 2,4GHz nätet. Resultatet kan ses i figur 5.3 nedan.



Figur 5.3: Graf som visar resultaten för sluttesterna av UDP med ALFA XX på 2,4GHz-nätet. X-axeln representerar 30 tester med varierande signalstyrka. Alla mätpunkter är i serie om 5 tester förutom första som består av 10 tester. Y-axeln representerar olika data för testerna. Blå = antalet paket som anlänt. Grön = signalstyrkan från klienten till servern. Gul = signalstyrkan från servern till avlyssnaren. Röd = antalet återsändningar som gjorts under testet.

Under de tio första testerna (1-10) så erhålls en väldigt god signal mellan klienten och servern. Däremot visar grafen att signalstyrkan mellan avlyssnaren och klienten är sämre, den är fortfarande bra men betydligt lägre, än den från avlyssnaren till servern. Under dessa tester så är alltså signalstyrkan från avlyssnaren till de andra enheterna så hög som antennen tillåter. Samtidigt visar den blå linjen att avlyssnaren ser relativt väldigt få paket under denna tiden. Skillnaden i signalstyrka från klienten till servern och avlyssnaren till servern är relativt stor och genomgående under den första test-serien. Detta är med stor sannolikhet en bakomliggande anledning till att avlyssnaren har svårt att identifiera alla 100 paket som skickas från klienten. Differensen innebär att avlyssnaren ser när paket tas emot men inte när de skickas. Således har den antagligen också svårt att avgöra när det är en återsändning som sker, detta förklarar de tester med noll upptäckta återsändningarna under de första tio testerna. Vidare kan även den låga delen av identifierade paket (blå linje figur 5.3) vara en konsekvens av att antennen har en för kraftig signalstyrka sett till servern och att den på så sätt ligger utanför sitt föredragna arbetsområde.

Nästa serie tester, 11-15 i figur 5.3, så har signalstyrkan från antennen sänkts med hjälp av folie. Folien viras runt antennen i lager tills det att önskad signalstyrka till nätet är uppmätt. Till skillnad från de första tio testerna så är nu signalstyrkan från avlyssnaren till servern väldigt jämn den mellan server och klient. Signalen från klienten till servern är i stort sett oförändrad medan signalen mellan servern och avlyssnaren har sjunkit några dBm och lagt sig på samma nivå som nätet. Fortfarande så lyckas inte avlyssnaren identifiera alla 100 skickade paket. Däremot så lyckas den se ett mycket högre antal återsändningar mellan servern och klienten.

Med andra ord så kan avlyssnaren verifiera att alla paket inte kommit fram första gången då den "kompletterar" identifierar flertalet återsändningar. Avlyssnaren kan alltså i detta fall stärka att den sett alla paket som kommit fram till servern men också att några inte kommit fram direkt och därmed behövts skickas om. Två bakomliggande faktorer kan tänkas vara anledning till det mycket förbättrade resultatet. Först så kan det vara så att avlyssnaren har mycket lättare att "följa" och identifiera paketen när signalstyrkan till servern är väldigt lik den mellan klienten och servern. Men det kan också bero på att antennen fungerar mycket bättre inom det signalområdet och att den på så sätt mycket lättare kan identifiera de transmitterade paketen. Självklart är det möjligt att båda anledningarna ligger till grund för förbättringen.

I serien efter det, dvs 16-20 i grafen 5.3, viras antennen in i ytterligare lager av folie. Därmed sjunker signalstyrkan från avlyssnaren till servern med ca 10 dBm. Differensen är dock fortsatt mycket liten i signalstyrka till de två enheterna ty även signalstyrkan mellan servern och klienten sjunker i detta fall. Resultatet är likt den föregående seriens (16-20) och avlyssnaren lyckas fortfarande registrera flertalet återsändningar under sändningarna. Däremot lyckas den även se fler antal paket som kommit fram direkt utan att bli återsända. Serien präglas av att avlyssnaren vid nästan varje tillfälle lyckas identifiera 100 skickade paket mellan servern och klienten. Anledningen till det höga antalet återsändningar som sker i vissa fall under serien, beror nästan uteslutande på att servern inte har hunnit bekräfta mottagelsen av ett paket innan klienten har skickat ett nytt. Även om resultat i förra serien var en klar förbättring från tidigare, så är resultatet i denna serien betydligt mer pålitligt i och med det höga antalet paket som registrerat vid första sändningen. Följaktligen så kan antennen i detta fall ligga i ett ännu mer förmånligt arbetsområde gällande signalstyrka för dess prestanda.

I den näst sista serien, 21-25 i figur 5.3, så har signalstyrkan sänkts ytterligare med hjälp av fler lager folie. Nämnvärt under denna serien och även den kommande är att det är väldigt svårt att etablera en jämn signalstyrka genom hela testserien då signalen i fråga är väldigt låg. Således är den svår att balansera och resultaten blir därför mer sprida under dessa tester. Signalstyrkan till servern från avlyssnaren fortsätter försvagas i denna serien. Samtidigt ökar differensen mellan signalerna igen då styrkan mellan klient och server förstärks till tidigare erhållna värden (test 1-15). Varför signalen mellan klient och server förstärks är svårt att förklara men en möjlighet är att en konstant störning såsom en mobil eller annan kommunicerande enhet har stört signalen vid den tidigare serien. För denna serien innebär det dock som nämnt att differensen har blivit större. I samband med det så finns fortfarande en god förmåga hos avlyssnaren att identifiera de skickade paketen. Däremot så märks en tydlig försämring gällande möjligheten att upptäcka eventuella återsändningar. Samtidigt så är antalet som klassas att ha kommit fram direkt väldigt hög, det finns alltså en god möjlighet att sändningen av paket mellan server och klient har skett helt utan förlust av datapaket. Däremot så visar de tre först testerna i serien (21, 22, 23) att avlyssnaren inte upptäcker 100 paket mellan servern och klienten. Det kan således tänkas att signalstyrkan börjar röra sig utanför antennens föredragna arbetsområde.

För de sista testerna, 26-30 i figur 5.3, så sänks signalstyrka återigen. När nu avlyssnaren har så en så svag signal till servern och nätet får den svårt att framförallt analysera signalstyrkan

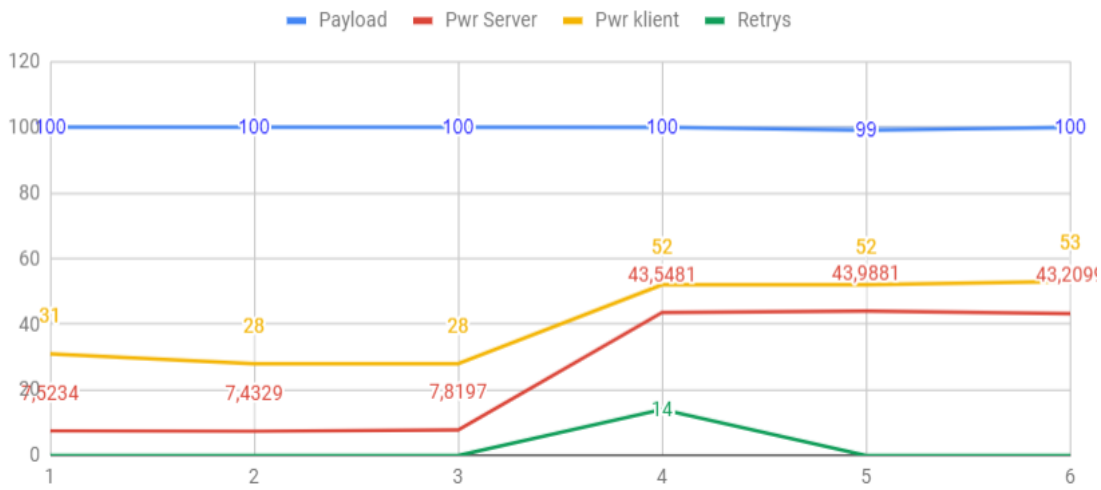
mellan klienten och servern. Således är den signalstyrkan (grön linje i figur 5.3) väldigt växlande och även opålitlig. Eftersom inga ändringar gjorts i förhållandet mellan server och klient utgår därför i analysen från signalstyrkan i test 26-27. Värdet i figur 5.3 för signalstyrka mellan server och klient försummas alltså. Trots en väldigt låg signalstyrka till servern kan avlyssnaren fortfarande identifiera stora antal sända paket. Den förlorar dock i princip helt förmågan att upptäcka återsändningar vilket märks i test nummer 29 då även antalet identifierade paket är väldigt låg. Således måste avlyssnaren klassas som väldigt opålitlig vid denna låga signalen. Dels eftersom den inte identifierar några återsändningar under hela serien och när den förväntas göra så (test 29). Men också för att den ger ett väldigt utstickande resultat under serien, vilket man i fält inte kan bortse ifrån.

5.3 Alfa AWUS036ACH

För att utföra 5GHz tester så användes en antenn av model Alfa *AWUS036ACH*[21]. Antennen användes för att göra både UDP och TCP tester. Testerna ämnade framförallt att undersöka förhållandet för resultat mellan 2,4GHz-testerna och 5GHz-testerna därför gjordes ett begränsat antal tester för vardera protokoll. Paketen skickas med 0,1 sekunders mellanrum. Tester med denna antenn utfördes enbart på 5G-nätet.

5.3.1 UDP-tester

Antennen testades först helt utan signal-begränsning. Den avlyssnade alltså först nätverket med så hög signalstyrka som möjligt. Vid senare tester sänktes signalstyrkan genom att öka avståndet mellan avlyssnaren och klienten samt avskärma för att komma ner i de signaler som ansågs intressanta. Testerna utfördes i två serier innehållandes vardera tre tester. Resultaten syns i figur 5.4.



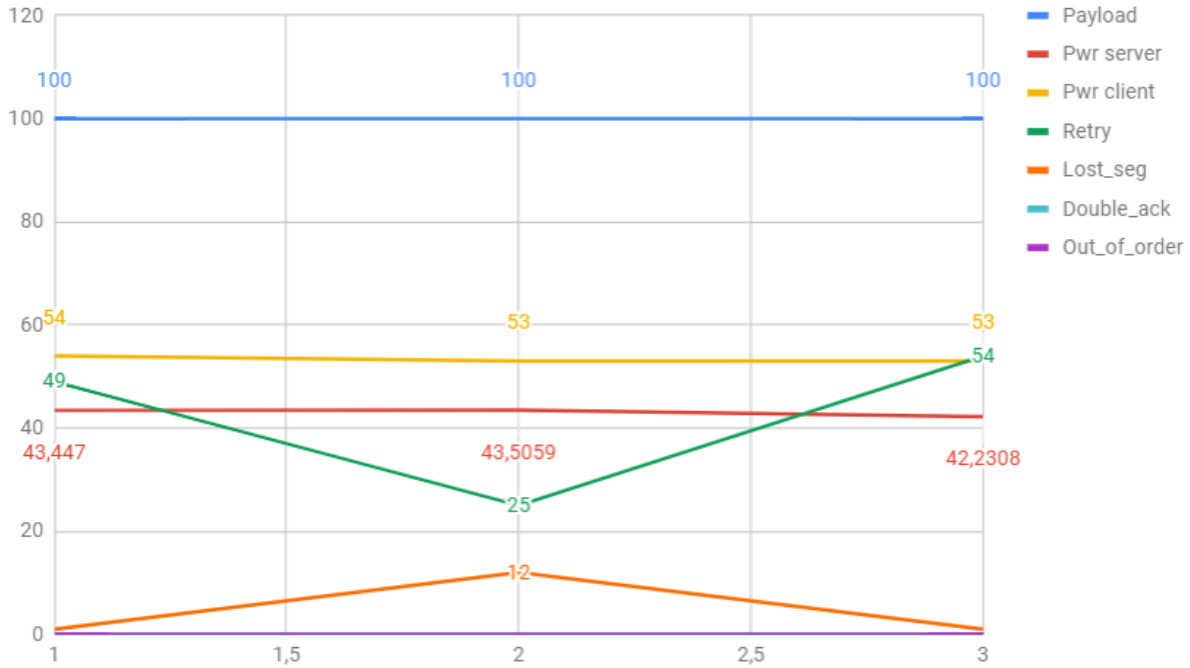
Figur 5.4: Graf som visar resultat för tester vid 5GHz med awus036ach. X-axeln representerar antalet gjorda tester där 1-3 är första serien och 4-6 är andra serien. Y-axeln representera olika data för testerna. blå = antalet paket som kommit fram direkt. Röd = signalstyrkan mellan server och avlyssnaren.
Gul = signalstyrka mellan servern och klienten. Grön = antalet återsändningar.

För den första serien av tester, 1-3 i figur 5.4 så har avlyssnaren en väldigt stark signal till servern. Notera att signalstyrka i grafen är i absolutbelopp. Avlyssnaren lyckas under hela serien identifiera alla 100 paket som skickas mellan servern och klienten. Dock identifieras inte en enda återsändning. Signalstyrkan mellan server och klient är väldigt liten jämfört den som är mellan avlyssnaren och servern. Anledning till att återsändningarna inte identifieras kan vara på grund av att avlyssnaren inte är kapabel till det. Samtidigt så är det möjligt att sändningen av paketen har skett utan att någon återsändning gjorts. 5GHz-nätet använder generellt mer komplexa modulationsmetoder, se tabell 5.1. Det innebär att dekrypteringen av paketen är mer krävande för avlyssnaren, men med en ny antenn så ändras förutsättningarna för vilka signalområden avlyssnaren är optimerad för. För den andra serien sänks signalstyrkan mellan avlyssnaren och servern markant, närmare -36 dBm. Även signalstyrkan mellan servern och klienten sjunker med flertalet dBm. Avlyssnaren lyckas fortfarande identifiera paketen till i princip full utsträckning. Samtidigt lyckas den vid test nummer 4 i figur 5.4 upptäcka ett för seriernas fall stort antal återsändningar. Troligen beror dessa på en störning mellan server och klient vilket tvingar ett försök till återsändning.

5.3.2 TCP-tester

Framförallt för att verifiera att inget oväntat beteende uppstår när det använda protokollet skiftas från UDP till TCP så gjordes en serie bestående av tre tester med TCP-paket på 5GHz-nätet. Signalstyrkan som valdes för kommunikationen mellan avlyssnare och server var den som visat sig vara optimal vid tidigare tester för UDP (ungefär -43 dBm). Testerna utfördes

under samma förutsättningar som för den sista serien av 5GHz UDP (figur 5.4). Resultat av dessa TCP-tester ses i figur 5.5.



Figur 5.5: Figur 5.4: Graf som visar resultat för tester vid 5GHz med awus036ach. X-axeln representerar antalet gjorda tester (3 st). Y-axeln representerar olika data för testerna. Blå = antal paket som kommit fram till servern från klienten. Röd = signalstyrka mellan server och avlyssnaren.
Gul = signalstyrka mellan klient och server. Grön = antalet gjorda återsändningar. Orange = Antalet paket som inte alls nått destinationen. Turquoise = Antalet paket som blivit avlyssnade två gånger. Lila = paket som anlänt i fel ordning mot vad de skickades.

I denna serie tester så lyckas avlyssnaren vid alla tillfällen identifiera alla de 100 paket som skickas mellan server och klienten. Det ska dock tilläggas att eftersom TCP tillåter i förhållande väldigt många återsändningar så ges avlyssnaren större chans att identifiera just 100 paket eftersom klienten skickar paket tills det att 100 kommit fram och blivit bekräftade av mottagaren Signalstyrkorna hålls relativt varandra under alla testerna. Antalet återsändningar som detekteras är relativt, de antal som upptäckts i UDP-testerna, väldigt höga. Det ska dock finnas i åtanke att TCP generellt genererar mycket högre antal återsändnings-klassade paket. Vid det andra testet så förekommer även ett antal paket som avlyssnaren ser men som inte når servern efter att dem blivit skickade. Antalet paket som blivit avlyssnade två gånger samt paket som anlänt i fel ordning är noll.

6 RESULTAT & ANALYSER

Analyser och slutsatser beträffande resultaten i kapitel 5.

6.1 Analys TCP-tester

Från de första testerna som gjordes med TCP-protokollet så kan det dras en slutsats att *Raspberry pi 3B+* interna nätverkskort är i hög grad alldeles för svagt för att fungera som avlyssnaren när obegränsad hastighet används. Anledning till detta är att hårdvaran hos *Raspberry:n* inte är tillräckligt bra och att enhet då inte hinner med de komplexa modulationsmetoderna som tillämpas vid dessa hastigheter. Det ska dock sägas att *Raspberry:n* klarar av att upptäcka alla paketen relativt väl. Däremot har den genomgående under dessa tester svårt att klassificera paketen. Detta är väldigt viktigt under felsökning och analys av paket. Därför kan det fastslås att *Raspberry:n* med enbart dess nätverkskort är allt för opålitlig som avlyssnarenhet vid höga hastigheter. Är syfte enbart att fånga data så klarar den det i dessa fall då TCP alltid skickar om paketen om servern inte hinner med. Men ligger en analys till grund för identifieringen av paket så är *Raspberry:ns* nätverkskort för svagt. Antagandet att resultatet skulle bli samma vid tester med obegränsad hastighet med UDP-protokollet görs då protokollen inte belastar/ställer olika krav på avlyssnaren när det gäller att identifiera paket.

Skillnaden mellan prestandan på nätverkskort som används, vid de först TCP-testerna (figur 5.1) som görs via det interna *Raspberry* kortet och de tre sista (figur 5.5) som görs via antenn, är väldigt stor. Även om de sista gör via 5GHz-nätet och således använder mer komplicerade modulationsmetoder så är det ganska tydligt att dessa är mer pålitliga då avlyssnaren klarar av att skilja mellan bland annat återsändningar och paket som kommer i ordning. Vilket den inte klarade med det interna kortet. Samtidigt så skickas TCP-paketen som analyseras av antennen betydligt långsammare och ger således en ökad möjlighet för avlyssnaren att hinna med att undersöka alla paket. Sen är *AWUS036ACH* en antenn som ska ha och bevisligen har en kraftfullare hårdvara än vad *Raspberry:ns* nätverkskort. Därför är resultaten från testerna väntade. Med antennen ökar nämligen förmågan för avlyssnarenheten att identifiera alla paket. Dels eftersom den har en mycket större räckvidd men också för att den klarar av att behandla mer avancerade modulationsmetoder och samtidigt har större förmåga att identifiera paketen på rätt sätt.

6.2 Analys UDP-tester

UDP-testerna inleddes såsom TCP genom att testa *Raspberry pi 3b+* interna nätverkskort prestanda. Under UDP-testerna integrerades dock som påpekat tidigare en delay mellan paketen och tvingar således klienten att skicka paketen med ett mellanrum på 0,1 sekunder. Detta innebär dock inte att hastigheten för nätet sänks, det vill säga att samma modulationsmetoder används, men att avlyssnaren ges en betydligt större chans att hinna

fånga/se data. Möjligheten att utföra tester för ett flertal olika signalstyrkor på det interna kortet är dock väldigt begränsad. Detta eftersom räckvidden på kortet är väldigt liten, vilket innebär att avlyssnaren då inte kan se någon eller extremt lite data. Således togs beslutet att analysera det område där en god signal var etablerad och kortets kapacitet fortfarande testades relativt mycket, testerna gjordes även enbart på 2,4GHz-nätet då det inte klarar av 5GHz. I testerna så tycks det interna kortet prestera överraskande bra. Det lyckas fånga i princip alla paket som skickas och kommer fram mellan servern och klienten. Dock verkar kortet ha svårt att identifiera eventuella återsändningar. En teori rörande återsändningarna är att de blir ivägtvingade av klienten i väldigt hög hastighet och att de även förbi ser/struntar i den inlagda delayen. Således skulle en återsändning kunna ställa mycket högre krav på avlyssnaren än en vanlig sändning gör. Det skulle förklara kortets problem med att identifiera återsändningar. Det finns samtidigt ingen dokumentation som stödjer det ena eller det andra. Ett sätt att motverka denna eventuella felkälla vore att se till att även återsändningarna skickas med en delay, detta tillämpas i så fall i koden hos klienten. Det är samtidigt att det är svårt att verifiera antalet återsändningar om man inte avlyssnar på två ställen. Om till exempel servern både skulle avlyssna och agera server samtidigt skulle man kunna jämföra resultaten för att fastställa avlyssnarens kapacitet att upptäcka återsändningarna. Det ställer dock betydligt mer omständiga krav på servern som inte tillgodosågs under dessa sluttester. *Raspberry pi 3B+* egna nätverkskort är således delvis kapabelt att fungera som avlyssnar-enhet. Däremot blir enhet väldigt begränsad då den behöver vara väldigt nära nätverket mellan servern och klienten för att dels få anslutning och kunna se nätet. Men den behöver också ligga inom föredraget område för signalstyrka för att kunna analysera paketen som skickas. Skulle alltså kunna funka vid mindre uppsättningar men är mycket begränsad i större sammanhang.

Uppsättningen där en antenn av model Alfa *AWUS051NH* anslöts för att substituera det interna kortet var tänkt att testas på både 2,4GHz och 5GHz näten. Det visade sig bara fungera på 2,4 GHz-nätet. Däremot hade antennen större problem att analysera data som skickades via 5GHz-nätet. Antennen fångar data, det vill säga att den ser att det är aktivitet på nätverk och ser att paket skickas. Vad avlyssnaren däremot inte kan se är vad det är för data som skickas och vad. Den är helt enkelt krypterad. Anledningen till detta är att Alfa *AWUS051NH* enbart stödjer modulationsmetoderna 802.11-a, -b, -n, -g, men 5GHz använder 802.11-a, -n men framförallt -ac. Antennen fick alltså uteslutas från att användas vid 5GHz-nätet. Vid 2,4 GHz-nätet är antenn dock fullt kapabel till att både se och dekryptera datan som skickas. Vid de första testerna så tilläts antennen ha en så god anslutning och hög signalstyrka som möjligt till servern. Delay vid paket-sändningen från klienten är fortfarande tillämpad i dessa tester. Innan testerna utfördes så fanns en egen teori om att antennen vid denna höga signalstyrka skulle ha större problem att analysera datan än vid lägre signaler. För att verkligen kunna verifiera ett sådant utfall, så gjordes därför för dessa, en utökad serie bestående av 10 tester. För hela serien gäller även att signalstyrkan till servern är starkare än den mellan servern och klienten. Vid denna första serie så stärktes teorin gällande Alfa awus051nh:s prefererade signalområde. Detta eftersom resultat var relativt dåligt, kommande tester som utfördes vid lägre signalstyrkor. Avlyssnaren hade vid detta tillfälle svårt att både se direkt skickade paket och eventuella återsändningar. Dock finns det också en möjlighet att differensen i signalstyrka från

avlyssnaren till server och klient mot server påverkar avlyssnarens förmåga att identifiera paketen.

När signalstyrkan mellan avlyssnaren och servern sänktes förändrades resultatet. Annorlunda för detta test var dels att signalstyrkan sänktes men också som resultat av detta att differensen mellan de båda anslutningar minskade markant. Man kan för denna serie tester se att man direkt får en betydligt mer pålitlig och robust avlyssnar-enhet som till stor del klarar av att upptäcka återsändningarna samt alla skickade paket. Fortsatt så är det svårt att bekräfta en teori vad det gäller om antennen kan anses ha ett signalområde där den verkar som mest effektiv. Även den markanta differensen i signalstyrka, vilket skulle kunna vara den andra faktorn till ett bättre resultat, minskat. Det är först vid denna signalstyrka som antennen skulle kunna klassas som klart bättre än *Raspberry:ns* interna kort. Det är även runt denna signalstyrkan (-33 till -39 dBm) som testerna för det interna kortet gjordes. Därför kan man dels med ett bättre resultat men också en garanterat längre räckvidd konstatera det faktum att antennen överpresterar det interna kortet tydligt vid samma utomstående förutsättningar. När sedan signalstyrkan sänks ytterligare i nästa serie blir avlyssnaren ännu mer precis. Ofördelaktigt för att stödja teorin är att i detta fall så sjunker även signalen mellan server och klienten, således bibehålls den väldigt låga differensen mellan signalstyrkorna. I denna serien blir det framförallt tydligt att antennen vid dessa förhållanden i hög grad klarar av att analysera i princip all data som skickas mellan server och klient. Att den lyckas identifiera återsändningarna i den höga grad som den gör vid dessa fall tyder på att det är vid denna denna signal som antennen kan anses vara optimerad, med viss försiktighet ty differensen är fortsatt väldigt låg.

Vid tester i serier som följer så minskas signalstyrkan mellan avlyssnaren och servern markant. Att genom hela serien få en jämn signalstyrka vid dess tester på låga signaler är väldigt svårt, avlyssnaren har även svårare att mäta signalstyrkan mellan klienten och servern. Således uppstår en stor differens mellan signalstyrkorna. Detta leder till att avlyssnaren inte längre klarar av att identifiera återsändningarna i samma utsträckning som tidigare. Av detta att döma så kan det konstateras att i för dessa tester så har återsändningarna varit betydligt mer krävande för avlyssnaren att behandla. Troligt är som nämnt tidigare att återsändningarna förbi ser den implementerade delayen och på så sätt ställer högre krav på avlyssnaren. Kraven kan då bara bemötas under vissa förutsättningar. Viktigt att ta i akt är dock att det enda sättet att fullt ut verifiera detta är att avlyssna på två ställen samtidigt. Detta för med ytterligare komplikationer och därför har det inte tillämpats vid testerna. Därför ska slutsatsen värderas med försiktighet men att bedöma av testerna så ser man ett väldigt tydligt beteende och ett tydligt mönster som stödjer analysen.

När antennen *AWUS036ACH* ansluts till avlyssnaren så möjliggör det avlyssning på 5GHz-nätet. *Raspberry:ns* nätverkskort klarar inte av 5GHz och den tidigare antennen *AWUS051NH* är några år äldre än *AWUS036ACH*. *AWUS051NH* klarar av att identifiera och se aktiviteten på nätverk men kan som tidigare förklarat inte dekryptera datan vid de komplicerade modulationsmetoder som används vid 5G kommunikation. För 5GHz testerna så kan man tydligt se att *AWUS036ACH* är en betydligt kraftfullare antenn. Den lyckas analysera all data som skickas redan vid en förhållandevis väldigt hög signalstyrka. Den bibehåller sedan samma

prestanda när signalstyrkan mellan avlyssnare och server sänks avsevärt. Troligt är att skillnaden i resultat som uppstår mellan UDP-testerna mellan 2,4-5GHz näten snarare beror på skillnaden i antenn prestanda snarare än att typ av nät skiftas. Med ett stabilt stöd från testerna och resultaten kan slutsatsen att en *Raspberry pi 3B+* är kapabel till att avlyssna både 2,4GHz och 5GHz nätverk. Den är dock, för att klara av detta, väldigt beroende av en bra antenn. Det interna nätverkskortet är i synnerhet alldeles för svagt för ett pålitligt resultat och en för utdaterad eller okapabel antenn begränsar användningsområdet avsevärt i synnerhet till vilken grad man kan lita på resultatet den genererar.

6.3 Sammanfattning Slutsats

Efter alla tester så kan det tydligt ses att en *Raspberry pi:s* interna nätverkskort är för svagt för att på egen hand fungera som avlyssnarenhet. Detta främst av två anledningar, den har alldeles för kort räckvidd och den klarar bara av att analysera data någorlunda vid förhållandevis låga hastigheter. Vidare så noterades det att vid användning av antenner så är det viktigt att antennen är kompatibel med de modulationsmetoder som kan tänkas användas vid sändningarna av data. Detta märktes framförallt i tester med AWUS051NH. Denna antennen klarade inte av 5GHz-nätet men förbättrade resultaten i 2,4GHz-nätet betydligt. Den mer modern antennen AWUS036ACH som är kapabel till att analysera 5GHz-nätet användes således istället för att avlyssna 5GHz-nätet. Ett tydligt resultat kunde då tas fram med hjälp av AWUS036ACH. Vilken antenn som används är således av stor betydelse och vid en snabb teknikutveckling är det således viktigt att ha modern hårdvara. Till slut så tros *Raspberry pi 3B+* ha bristande hårdvara för att analysera data vid höga hastigheter som används vid 5GHz. Detta eftersom resultaten när antennerna används i vissa fall är bristfälliga när det kommer till identifierad data. Att testa alternativa enkortsdatorer är enda sättet att veta säkert. Vid lyckad konstruktion av en avlyssnings-enhet ger goda förutsättningar för Volvo Technology AB att verifiera och säkerställa data som skickas mellan fordon och laddningsstation, detta ger en påföljd av att snabbt upptäcka och åtgärda problem. Således kan en ökad kundnöjdhet och efterfrågan skapas för elfordon vilket gynnar samhället och miljön positivt. En ökning av elfordon skulle ge mindre avgaser och ljud i städer och därmed ge en ökad trivsel för stadsbor.

7 VIDAREUTVECKLING

Då konstruktionen av avlyssnaren har skett på en *Raspberry pi 3B+* kan fortsättning av tester på dess hårdvara vara avgörande. Skulle processorkraften vara för låg eller dess interna datapakethantering inte tillräckligt snabb så skulle alternativa enkortsdatorer vara en möjlig ersättning.

Eftersom möjligheterna och tiden har varit relativt begränsade rörande att genomföra tester så finns därför en stor möjlighet att ta vid där det saknas eller är otillräcklig samlad data. Det finns synnerligen många fler olika sorters tester som skulle kunna utföras och utvecklas. Inte minst finns det även flertalet obeprövade antenner och annan hårdvara som kan påverka resultaten på olika sätt.

Lyckas en helt säker avlyssnare konstrueras, med en *Raspberry pi* som grund, så ger det möjlighet till många agila lösningar för avlyssning. Bland annat skulle det ge en väldigt god möjlighet till att väldigt precist kunna logga olika fel, vart dem kommer ifrån, hur dem inträffar och vad som orsakat dem. På så sätt skulle en väldigt bred marknad öppna sig och avlyssnaren skulle kunna tillämpas på många sätt. Det skulle dock medföra externt jobb i form av utveckling av program som kan sammanställa och diagnostisera datan som samlas in. Nödvändigt skulle även bli att tillämpa gränssnitt för varje användningsområde så att loggning blir så optimerad som möjligt för det områden den är tänkt att verka inom.

BILAGOR

Bilaga A

A1. Kod för UDP-server

```
import socket
import sys
c=0     //Nollställer intern räknare

// Skapar UDP sockets
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) // Sockets för att
kommunikation via UDP

// Binder socketen till port och IP-address
server_address = ('192.168.1.3', 15121)         // Specificerar IP-adress och port för server
print('starting up on {} port {}'.format(*server_address))
sock.bind(server_address)

while True:     // Håller servern uppe tills den stängs ner
   c+=1         // Adderar internt när paket tas emot från klient
   print('\nwaiting to receive message')
   data, address = sock.recvfrom(1024)                  // Tar emot data

   print('received {} bytes from {}'.format(            // Skriver ut mottagen data
      len(data), address))
   print(data)
   print (c, data)

   if not data:
      break
```

A2. Kod för UDP-klient

```
import socket
import sys
import time
server_address='192.168.1.3' // Definierar vilken adress klienten ska skicka till

stuff = (b'000000')* 25 // Paketen blir tilldelade storlek
```

```
// Skapar UDP-socket
sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

server_address = ('192.168.137.1', 15121)
sock.connect(('192.168.137.1', 15121)) // Begär kontakt med server på port och IP


for i in range(100): // Skickar 100 paket
   payload = i.to_bytes(2, byteorder='big') + stuff // Varierar data som skickas
   sock.sendall(payload) // Skickar data
   time.sleep(0.1)     // Delayar data som skickas med 0.1s

   print('sending {!r}', repr(payload)) // Printar skickad data
```

A3. Kod för TCP-server

```
import socket
c=0 // Nollställer intern räknare
HOST = '192.168.1.3'            // Definierar serverns IP
PORT = 15120                    // Definierer serverns port

// Skapar TCP-Socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
   s.bind((HOST, PORT))         // Binder IP och port
   s.listen()                   // Lyssnar efter klient som ansluter
   conn, addr = s.accept()      // Accpeterar anslutningen
   with conn:
      print('Connected by', addr)
      while True:
         c+=1                   // Adderar till intern räknare
         data = conn.recv(1024) // Tar emot data
         print (c, data)
         if not data:
            break
         conn.sendall(data)     // Skickar data
```

A4. Kod för TCP-klient (utan delay)

```
import socket
c=0                             // Nollställer intern räknare
```

```
HOST = '192.168.1.3' //Serverns IP
PORT = 15120                    // Port för kommunikation

stuff = (b'000000')* 25         // Paketen blir tilldelade storlek

// Skapar TCP-socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
   s.connect((HOST, PORT)) // Ansluter till servers IP och port
   for i in range(100):         // Skickar 100 paket
      c+=1                              // Adderar till interna räknaren
      payload = i.to_bytes(2, byteorder='big') + stuff
      s.sendall(payload)                // Skickar data
      data = s.recv(1024)               // Får tillbaka bekräftelse från server
      print(c, 'Received' , repr(data))
```

Bilaga B

B1. Avlyssnarkod

```
#!/bin/bash
from time import time


mode=wlan1        #Vilket nätverks kort som skall användas
file=testymeh.csv  #fil som skall sparas till

read -p "Swtich between mode (n=Network, m = Monitor): (n/m)?" reset

if [ "$reset"="$n" ]
then
   echo "Switching Modes..."
   airmon-ng stop $mode                                  #stoppar monitor mode
   systemctl start NetworkManager.service                      #startar upp nätverksanslutningen
   var_attack=0                                          #intern process variabel

fi

if [ "$reset"="$m" ]
then
   SSID=OnePlus5T                                         #Närverksnamn att leta efter
   SCAN_RESULT=$(nmcli -f SSID,CHAN,BSSID dev wifi list | grep $SSID)  #scannar efter nätverksnamn och retunerar kanal och
BSSID
   BSSID=$(echo $SCAN_RESULT | awk '{print $3}')                 #tillhandahåller BSSID
   CHAN=$(echo $SCAN_RESULT | awk '{print $2}')                  #tillhandahåller nätverkskanal

   if [ -z "$SCAN_RESULT" ]
   then
      echo Could not find SSID $SSID
   fi

   if [ -n "$SCAN_RESULT" ]
   then
      echo echo SSID $SSID found on channel $CHAN with BSSID $BSSID
      airmon-ng check kill                                 #avslutar nätverkskortet anslutning
      airmon-ng start $mode $CHAN                                #start monitor mode på hittad kanal på nätverkskortet
      echo "Entering Monitor Mode..."
      xfce4-terminal -e "bash -c 'tshark -i $mode -T fields -e frame.number -e data -e ip.src; exec bash'" #realtidspresentering av fångade
paket
      var_attack=1                                         #intern processing variabel
      if [ "$var_attack"="$1" ] && [ "$reset"="$m" ]
      then
         read -p "Do you wanna start ariodump on chn $1 and BSSID $2 (y/n)?" ario
         if [ "$ario"="$y" ]
         then
            echo ============================================
            echo "Opening airodump in new window..."
            xfce4-terminal -e "bash -c 'airodump-ng -c $CHAN --bssid $BSSID -w /root/new_antenna_06/tcp/test_no_ant_dump $mode; exec
bash'" #startar airodump
            echo ============================================
         fi
         read -p "Get handshake on BSSID? $BSSID (y/n)?" hndshake
         if [ "$hndshake"="$y" ]
         then
            echo "Getting handshake in new window  ..."
            xfce4-terminal -e "bash -c 'aireplay-ng -0 1 -a $BSSID -c B8:27:EB:19:A2:2E $mode; exec bash'" #försöker få krypteringnykel
genom att få enhetr att återansluta
         else
            echo "skipping handshake..."
```

```
        fi
      else
        echo "Failed to enter arioscript..."

      fi
   fi
fi
```

B2. Överföringskod TCP

```
x=03
capfile=/root/new_antenna_06/tcp/test_no_ant_dump-$x.cap
savecsv=/root/new_antenna_06/tcpcsv/tcp_5G_$x.csv



#nedanstående kod läser in capture-fil för att spara undan alla UDP-paket och dess parametar paketnummer, tid, data, återsändningspaket
#,paket som kommit ur ordning, förlorade segment av paket, dubbletter av ACK-paket. Dessa parametrar sparas ner i CSV-fil.
tshark -r $capfile -Y "tcp" -T fields -e frame.number -e _ws.col.Time -e data -e tcp.analysis.retransmission -e tcp.analysis.out_of_order -e
tcp.analysis.lost_segment -e tcp.analysis.duplicate_ack >$savecsv

#Analysering av TCP-paket flaggor, räknar antalet återsänding, förlorade segment, dubletter av ack-paket samt paket som kommit ur
ordning.
tshark -r $capfile -q -z io,stat,3000,\
"COUNT(tcp.analysis.retransmission)tcp.analysis.retransmission",\
"COUNT(tcp.analysis.lost_segment)tcp.analysis.lost_segment",\
"COUNT(tcp.analysis.duplicate_ack)tcp.analysis.duplicate_ack",\
"COUNT(tcp.analysis.out_of_order) tcp.analysis.out_of_order"
```

B3. Överföringskod UDP

```
            x=07
            capfile=/root/new_antenna_06/udp/test_no_ant_dump-$x.cap
            savefile=/root/new_antenna_06/udpcsv/udp_5G_$x.csv

tshark -r $capfile -Y "udp" -T fields -e frame.number -e _ws.col.Time -e data -e wlan.fc.retry -e frame.time_delta_displayed >$savefile

            capinfos -c -y -z -x $capfile
```

B4. Verifieringskod TCP

```
import pandas as pd #Programbilbiotek
import statistics
import sys
import array
import csv
import numpy

readfile='/root/new_antenna_06/tcpcsv/tcp_5G_03.csv' #CSV-fil som kommer från Tshark
print(readfile)
writefile='/root/new_antenna_06/tcpsum5G/tcp5G_03.csv'#sparar undan uppdaterade CSV-fil
pwr_file='/root/new_antenna_06/tcpcsv/test_no_ant_dump-03.log.csv'#CSV-fil som loggar signalstyrka

col=["Number","time","payload","ret","out_of_order","lost_segment","double_ack" ]#intern Column namngivning för CSV-fil
ret_col=["Number", "time", "value", "ret","out_of_order","lost_segment","double_ack", "payload"]#intern Column namngivning för CSV-fil

def hexcount():                                    #Hexadecimal räknare funktion mellan 1-100(0-063 hex).
   counterarray = []                               #Tom array som sparar undan värderna för räknaren
   for i in range(100):                            #fyller counter array med hexadecimaltal från 1-100
```

```
        counterarray.append("00"+"{:02x}".format(i))                    #formaterar så att räknarvärderna matchar räknaren som sker i
klienten (0001, 0002,...0063)
    return counterarray                                                 # skickar tillbaka Arrayen med räknarvärderna


#funktion som letar efter räknarvärden i långa strängar av data
def stringfinder(finderfile, hexecount)->str:                           #matar in CSV-filen som skall processas
    filefind = pd.read_csv(finderfile, sep = '\t', names=col)           #läser in CSV_filen från tshark så den kan processar
    df_file = pd.DataFrame(filefind)                                    # gör om filen till en dataframe för processering
    df_file['value'] =[0]*filefind.shape[0]                             #intern avnänding skapar en tom kolumn för att mata in
matchande räknevärden
    for val in hexecount:                                               #stegar genom arrayen med räknarvärdena
        df_file["new"] = pd.DataFrame(df_file["payload"].str.match(val))    #jämför strängarna av data som fångats mellan sever och klient
med alla räknarvärden
        for index, line in df_file.iterrows():                          #stegar genom skapade Dataframen
            if line["new"] == True and line["value"] == 0:              #Strängen av data matchar ett räknevärde så ersätts platsen
med det räknarvärdet
                df_file["value"].iloc[index] = val                      #ersätter befinligt position med räknarvärde
                break

    ret_file = df_file[ret_col]                                         #CSV-filen formas om så de långa strängarna med data
placeras längst bort i CSV-filen
    return ret_file

#funktion som tar medelvärdet för signalstyrkan mellan avlyssnare och server
def add_pwr(power_file, find_file):                                     #läser in en log-fil där signalvärden sparats undan under en
avlyssningssession.
    file_pwr = pd.read_csv(power_file, sep=',', na_filter=False, usecols=[4])   #skapar en dataframe för att processeras
    power_no_zero = file_pwr[file_pwr != 0].dropna()                    #hoppar över alla rader som innehåller 0 i kolumnen PWR
    pwr_numb = power_no_zero.sum(axis=0)                                #addear ihop alla värden för kolumnen PWR
    pwr_sum = power_no_zero.count()                                     #räknar antalet platsen på kolumnen PWR
    pwr_func = pwr_numb/pwr_sum                                         #dividerar summan med antalet platser för att få medelvärdet
av signalstrykan

    return pwr_func


counter = hexcount()                                                    #kallar på hexadecimal räkar funktionen
findcounter = stringfinder(readfile, counter)                           #kallar på funktione som försöker matcha räknaren med
strängarna av data
w_pwr = add_pwr(pwr_file,findcounter)

count_file = findcounter[findcounter != 0]                              #hoppar över alla rader som innehåller noll
payload = count_file["value"].count()                                   #räknar antalet rader som innehåller ett värde skillt från noll
retrys = count_file["ret"].count()                                      #räknar antalet återsändingar
out = count_file["out_of_order"].count()                                #räknar antalet paket som kommit ur ordning
lost = count_file["lost_segment"].count()                               #räknar antalet paket som förlorat segment av paketen
double = count_file["double_ack"].count()                               #räknar antalet dubletter at ack-paket skett



findcounter.to_csv(writefile)                                           #skriver ner den processade CSV-filen i en ny CSV-fil.

print("pwr: ",w_pwr)
print("payload: ", payload)
print("Retransmission: ", retrys)
print("Out of order: ",out)
print("lost: ", lost)
print("double: ", double)
```

B5. Verifieringskod UDP

```
import pandas as pd
import statistics
import sys
import array
import csv
import numpy


file ='/root/new_antenna_06/udpcsv/udp_5G_06.csv'                    #läser in CSV-fil som skapats av Tshark
print(file)
writefile='/root/new_antenna_06/udpsum5G/udp5_06.csv'                        #CSV-fil att skriva till när processen skett
pwr_file = '/root/new_antenna_06/udpcsv/test_no_ant_dump-06.log.csv'  #CSV-fil som loggar signalstyrka


col=["Number", "time", "payload", "retry","Delta"]                    #namngivning av kolumner
ret_col=["Number", "time", "value", "retry","Delta", "payload"]       #namngivinng av kolumner efter processing

#funktion av hexadecimal räknare
def hexcount():
   counterarray = []                                                  #tom array som skall spara alla värden mellan 1-100 i
hexadecimalt (0-63)
   for i in range(100):                                               #räknare från 1-100
      counterarray.append("00"+"{:02x}".format(i))                    #formatera hexadecimaltalen så dem matachar strängarna
som skickas mellan server o klient
   return counterarray                                                #skickar tillbaka array med räkanarens värde

#funktione som försöker matcha räknarvärdena som finns i strängarna av data.
def stringfinder(finderfile, hexecount)->str:                         #läser in CSV-fil från
   filefind = pd.read_csv(finderfile, sep = '\t', names=col)          #läser in CSV-fil som fås fråm  Tshark
   df_file = pd.DataFrame(filefind)                                   #gör en dataframe för processering
   df_file['value'] =[0]*filefind.shape[0]                            #skapar en ny kolumn
   for val in hexecount:                                              #stegar genom hexadecimala räknaren
         df_file["new"] = pd.DataFrame(df_file["payload"].str.match(val))  #letar efter matching på strängar av data och hexadecimal
räknare
         for index, line in df_file.iterrows():                       #stegar genom CSV-filen från Tshark
               if line["new"] == True and line["value"] == 0:         #kollar ifall matchning är hittat
                  df_file["value"].iloc[index] = val                  #skriver över platsen med det hexidecimala värdet som
matchats
                  break

   ret_file = df_file[ret_col]                                        #möblerar om CSV-filen så strängarna med data kommer
längst bort
   return ret_file                                                    #retunerar ommöblerarde CSV-filen


#funktion som beräknar medelvärdet på signalstrykan
def add_pwr(power_file, find_file):                                   #läser in loggfilen signalstryka
   file_pwr = pd.read_csv(power_file, sep=',', na_filter=False, usecols=[4])  #gör en datafram av filen för processering
   power_no_zero = file_pwr[file_pwr != 0].dropna()                    #hoppar över alla celler med värdet noll
   pwr_numb = power_no_zero.sum(axis=0)                               #summerar ihop alla värden i Kolumnen PWR
   pwr_sum = power_no_zero.count()                                    #räknar antalet platsen i kolumnen PWR
   pwr_func = pwr_numb/pwr_sum                  #dividerar summan med antalet platser för att få medelvärdet på signalstyrkan

   return pwr_func
#beräknar medelvärdet att sända paket, samt långsamaste samt snabbaste tiden.
def average_time(avg_file):
   func_start = numpy.amin(avg_file["time"])
   func_stop = numpy.amax(avg_file["time"])
   avg_time = avg_file.dropna().time.diff()
   favg_mean = avg_time.mean()
   favg_max = avg_time.max()
   favg_min = avg_time.min()
   return favg_mean, favg_min, favg_max, func_start, func_stop
```

```
counter = hexcount() #kallar på Hexadecimalräknaren

findcounter = stringfinder(file, counter) #kallar på matchningsfunktione av strängar av data

w_pwr = add_pwr(pwr_file,findcounter)     #kallar på funktione som räknar ut medelvärde

avg_time, time_min, time_max, time_start, time_stop = average_time(findcounter) #kallar på funktionen som beräknar tid

count_file = findcounter[findcounter != 0]#hoppar över alla rader med värdet noll
payload = count_file["value"].count()     #räknar alla celler som innehåller ett önskad data samt är skilt från 0
retrys = count_file["retry"].count()      #räknar alla återsändningar


findcounter.to_csv(writefile)            #skriver över CSV-filen från tshark till ny CSV-fil
print("========================\n")
print(avg_time,"\n", time_max,"\n", time_min,"\n", time_start,"-->", time_stop)
print("========================\n")
print("payload: ", payload)
print("retrys:", retrys)
print(w_pwr)
```

KÄLLFÖRTECKNING

[1] ”IP-paket model”. [Elektronisk bild]. Tillgänglig: https://www.javatpoint.com/computer-network-tcp-ip-model, Hämtad: 2019-05-25.

[2] ”TCP-paket model”. [Elektronisk bild]. Tillgänglig: https://www.techrepublic.com/article/exploring-the-anatomy-of-a-data-packet/. Hämtad: 2019-05-25.

[3]”UDP-paket model”. [Elektronisk bild]. Tillgänglig: https://www.eetimes.com/document.asp?doc_id=1276768#. Hämtad: 2019-05-29.

[4]”Understanding the IEEE 802.11 Standard for Wireless Network”. [Online]. Tillgänglig: https://www.juniper.net/documentation/en_US/junos-space-apps/network-director3.2/topics/concept/wireless-80211.html. Hämtad: 2019-06-02.

[5]”Internet”, [Online] uppdaterad 29-04-2019. Tillgänglig:https://sv.wikipedia.org/wiki/Internet#Uppbyggnad_och_teknik. Hämtad: 2019-06-02

[6]”Packetshaper and Flow Directions” [Elektronisk bild]. Tillgänglig: https://etherealmind.com/packetshaper-inbound-outbound-outside-inside/. Hämtad: 2019-06-03

[7]”An introduction to Computer Networks”. [Elektronisk bild]. Tillgänglig:https://intronetworks.cs.luc.edu/current/html/tcp.html. Hämtad: 2019-06-03

[8]”Transport Layer Protocol” [Online]. Tillgänglig: https://www.javatpoint.com/computer-network-transport-layer-protocols. Hämtad: 2019-06-03

[9]”Vad är skillnaden mellan WPA och WPA2” [Online]. Tillgänglig:http://www.dator.xyz/Natverk/network-security/75623.html#.XO2pbtMzYWo. Hämtad: 2019-06-03

[10]”Techterms WPA”. [Online]. Tillgänglig: https://techterms.com/definition/wpa. Hämtad: 2019-06-04

[11]”What is a socket”. [Online]. Tillgänglig: https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm. Hämtad: 2019-06-04

[12]”Wireshark”. [Online]. Tillgänglig: https://www.wireshark.org/. Hämtad: 2019-06-05

[13]”Tshark”. [Online]. Tillgänglig: https://www.wireshark.org/docs/man-pages/tshark.html. Hämtad: 2019-06-05

[14]”Python” [Online]. Tillgänglig: https://www.python.org/. Hämtad: 2019-06-05

[15]”Pandas python”. [Online], Tillgänglig: https://pandas.pydata.org/. Hämtad: 2019-06-05.

[16]”kali linux”. [Online]. Tillgänglig: https://www.kali.org/. Hämtad: 2019-06-06

[17]”Aircrack-ng” [Online]. Tillgänglig: https://www.aircrack-ng.org/. Hämtad: 2019-06-06.

[18]”Alfa Networks”.[Online]. Tillgänglig: https://www.alfa.com.tw/. Hämtad: 2019-06-06.

[19][WS] Wireshark network analyzer.[Online]. Tillgänglig: https://wlan-profi-shop.de/datasheet/AWUS051NHv2.pdf Hämtad: 2019-06-06

[20] “wlan-profi-shop datasheets.[Online]. Tillgänglig: https://wlan-profi-shop.de/datasheet/AWUS051NHv2.pdf Hämtad: 2019-06-06

[21] Rokland specs. 2014.[Online]. Tillgänglig: http://www.rokland.com/support/specs/AWUS036ACH.pdf Hämtad: 2019-06-06

[22] Wikipedia 2019.[Online] Tillgänglig: https://en.wikipedia.org/wiki/IEEE_802.11 Hämtad: 2019-06-06