PROCJENA GUŽVE U STUDENTSKOJ MENZI POMOĆU DETEKCIJE WIFI SIGNALA PAMETNIH TELEFONA

Ivanić, Luka

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

PROCJENA GUŽVE U STUDENTSKOJ MENZI POMOĆU DETEKCIJE WIFI SIGNALA PAMETNIH TELEFONA

Rijeka, rujan 2024.

Luka Ivanić
0069091014

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Prijediplomski sveučilišni studij računarstva

Završni rad

PROCJENA GUŽVE U STUDENTSKOJ MENZI POMOĆU DETEKCIJE WIFI SIGNALA PAMETNIH TELEFONA

Mentor: prof. dr. sc. Mladen Tomić

Rijeka, rujan 2024.

Luka Ivanić
0069091014

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 14.03.2024.

Zavod: Zavod za računarstvo
Predmet: Računalne mreže

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Luka Ivanić (0069091014)**
Studij: Sveučilišni prijediplomski studij računarstva (1035)

Zadatak: **Procjena gužve u studentskoj menzi pomoću detekcije WiFi signala pametnih telefona / Estimation of Student Restaurant Congestion by Detection of Smartphones WiFi Signals**

Opis zadatka:

Realizirati sustav detekcije WiFi signala pametnih telefona te način procjene broja studenata u studentskoj menzi. Radi postizanja veće točnosti, uzeti u obzir snagu signala te vrstu i proizvođača uređaja. Eksperimentalno ispitati točnost sustava u različitim uvjetima, odnosno pri različitom broju studenata.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20.03.2024.

Mentor:
prof. dr. sc. Mladen Tomić

Predsjednik povjerenstva za završni ispit:
prof. dr. sc. Miroslav Joler

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2024. _____

Luka Ivanić

Zahvala

Zahvaljujem mentoru na savjetima tijekom pisanja ovoga rada i izuzetnom strpljenju.
Zahvaljujem curi na podršci tijekom pisanja ovoga rada - kako u životu, tako i u pisanju.

SADRŽAJ

1. UVOD

Motivacija za ovaj rad dolazi iz učestalog problema dugog vremena čekanja u redu za dobivanje hrane u studentskim menzama. To je problem koji se svakodnevno javlja u životima prosječnog studenta, koji je ujedno i korisnik usluga studentskih menzi, i rješavanjem tog problema bi se spomenutom prosječnom studentu oslobodilo i do dodatnih sat vremena u danu.

Nada za rješavanje problema dolazi iz opažanja studenata (od sada pa na dalje, u tekstu, sinonim za korisnike menze) da je često gužva u menzama kroz dan veoma nepravilno raspoređena, čak i u manjim periodima. Graf vremena čekanja kroz dan, s malim rasponom apscise (toka vremena kroz jedan dan) od jednoga sata ili manje (< 1h), nerijetko pokriva cijeli raspon ordinate (vremena čekanja) kroz dan, više puta na dan.

Time, pomakom vremena polaska u menzu za na primjer, kratkih petnaest minuta, vrijeme čekanja u redu kroz dan bi se prosječnom studentu, tj. korisniku kombinacije hardverskog i softverskog rješenja na kojemu se temelji ovaj rad, moglo znatno smanjiti.

Kao što je spomenuto, cilj ovoga rada je napraviti hardversko i softversko rješenje, koje u stvarnom vremenu omogućuje uvid u stanje gužve u menzi, te kao dodatna funkcija, i predviđa buduće stanje u kratkom vremenskom roku. Također, dobivena informacija o stanju gužve mora biti dostupna i lako pristupačna studentu korisniku, u bilo kojem trenutku u danu.

2. ZNAČAJKE PROBLEMA

Kako bi se najbolje pristupilo problemu potrebno je definirati dodatne značajke problemu. Relevantne značajke stavljaju naglasak na relevantne detalje problema, što je korisno kod usporedbe potencijalnih pristupa problemu.

Već je spomenut koncept visoke kaotičnosti i nepravilnosti vremena čekanja (velike trenutačne promjene) u razmatranom relativno malom vremenskom periodu. Time, relativno malom promjenom vremena dolaska u menzu (petnaest minuta) moguće je poboljšati subjektivna iskustva čekanja u redu.

Nepravilnost vremena čekanja može se bolje opisati primjerom vremenskog perioda p, kojeg definira nemogućnost jednoznačnog opisivanja stanja gužve, već se sastoji od izmjena različitih stanja. Dakle, različita stanja gužve mogu trajati malo relativnog vremena (<<p) i brzo se izmjenjivati u periodu p. Stanje gužve u jednom trenutku vremena definirano je kao vrijeme čekanja na kraju reda, dok se ne dobije posluga. Odvajajući vremenske trenutke proizvoljnim razmacima od npr. jedne minute, i promatrajući ljude koji su stali u red u minuti 12:00, 12:01, 12:02.., dobilo bi se vrijeme čekanja za svaku minutu u periodu p. Definiranjem ovoga odnosa postaje jasno da je promjena u vremenu čekanja (ordinata) u nekim trenucima velika. Vrijeme čekanja u minuti 12:14 moglo bi biti 0 minuta, dok vrijeme čekanja u minuti 12:15 bi moglo biti 20 minuta, te u minuti 12:25 značajno manjih 10 minuta. Ova razlika u vremenu čekanja bila bi još više istaknuta kada se na nju ne bi gledalo kao na kontinuiranu vrijednost, već kategorijalnu. Kategorijalnim gledanjem bi se svakom minutom moglo izmjenjivati stanje (kategoriju) gužve.

Time je zaključeno da je potrebna veća granularnost pri gledanju gužve tijekom dana, koja omogućuje definiranje koliko je potrebno biti fleksibilan s vremenom polaska u menzu od planiranog ($\pm t$), da bi u na primjer 90% slučajeva izbjegli gužvu (definirano kao vrijeme čekanja veće od 15 minuta). Zatim, kada bi postojala dovoljna pouzdanost kod predviđanja buduće gužve u kratkom vremenskom periodu, vrijeme čekanja u redu (jednog) studenta bi se značajno smanjilo. Uz uvjet da je spreman prilagoditi vrijeme odlaska u menzu za spomenutih $\pm$15 minuta.

Postoji nekoliko razloga zašto su gužve u specifičnim vremenima u danu velike. Jedan od glavnih razloga je što studenti većinom dolaze u skupinama, u trenucima ubrzo nakon završetka predavanja ili ispita koji se provodio na fakultetu. Takve grupne dolaske studenata u menzu ne možemo predvidjeti informacijama iz same menze, već uvidom rasporeda kolegija obližnjih fakulteta.

Broj ljudi koji čekaju u redu, pri vremenu dolaska u red, visoko je koreliran s potrebnim vremenom čekanja za dobivanje posluge, te će se koristiti kao glavna informacija koja upućuje na stvarno stanje reda u menzi, ali je njegova korelacija s vremenom čekanja značajno ovisna o dodatnim faktorima okruženja u menzi, kao što su broj otvorenih kasa, prosječno vrijeme posluživanja jednoga studenta u toj menzi, broju radnika koji u trenutku rade na liniji hrane, i drugi. U daljnjem tekstu upućivanje na korelaciju broja ljudi u redu i vremena čekanja bit će proizvoljno definirano, i vrijednosti u završnoj implementaciji će se možda razlikovati od spomenutih.

Subjektivno iskustvo je pod direktnim utjecajem duljine čekanja u redu, ali odnos nije linearan, već se iskustvo može svrstati u nekoliko kategorija (koje se značajno razlikuju jedna od druge). Lakše i pogodnije je definirati kategorije nakon razvoja glavnog dijela rješenja i utvrđivanja tehničkih sposobnosti razvijenoga rješenja, no i dalje je korisno postaviti okvirne kategorije koje će služiti kao smjernica tijekom razvoja projekta.

Počevši od jako malog broja ljudi koji se nalaze u redu za vrijeme dolaska, postoji razlika u iskustvu kada je red nepostojeći i pristup liniji za hranu je trenutačan, i čak i kada je broj ljudi u liniji mali ($n < 5$). Razlika u iskustvu uzrokovana je velikim omjerom vremena čekanja prvog i drugog slučaja, ako je red prazan i student odmah dobije poslugu, potrošit će t vremena, dok će za svaku dodatnu osobu koja čeka u redu, student morati čekati dodatnih t vremena. Na primjer, za slučaj bez ljudi u redu $t_0 = t$, i za slučaj dodatnih troje ljudi u redu $t_3 = t + 3t = 4t$, omjer čekanja Q bi iznosio:

$$Q = \frac{t_3}{t_0} = \frac{4t}{t} = 4 \qquad (2.1)$$

Rezultat (2.1) govori da je četiri puta duže čekanje u drugom slučaju (t_3), u odnosu na prvi slučaj (t_0). Iako oba slučaja u stvarnosti znače vrijeme čekanje manje od tri minute, percipirano iskustvo je različito. Dakle, postoji razlika između prvog i drugog slučaja, no razlika u subjektivnom iskustvu je i dalje značajno manja od razlika između daljnjih susjednih kategorija, zbog čega su ova dva slučaja spojena u jednu, prvu kategoriju, definiranu brojem ljudi manji od pet ($n < 5$), odnosno „kratak red“.

Sljedeća kategorija, po subjektivnom iskustvu, definirana je kao “srednje” vrijeme čekanja, gdje “srednje” ne označuje nikakvu prosječnu, ni medijansku vrijednost vremena čekanja. Broj ljudi u redu i vrijeme čekanja srednje kategorije nalazi se između dva krajnja ekstrema (kratak i dug red), a definirana je kao više od pet ljudi i manje od petnaest ljudi u redu ($5 < n < 15$). Vrijeme čekanja srednje kategorije i dalje je nedovoljno utjecajno na planove prosječnog studenta, jer se uobičajeno za polazak u menzu rezervira trideset minuta vremena, time ova kategorija kvari jedino raspoloženje prosječnog studenta, ali ne i njegov dnevni plan.

Posljednju kategoriju označava broj ljudi u redu veći od petnaest ($n > 15$). Ovisno o raznim faktorima, broj ljudi u redu u ovoj kategoriji može znatno varirati, i subjektivna iskustva i dalje mogu biti različita, s vremenima čekanja koja variraju od dvadeset i pet minuta do šezdeset minuta ili više, ali sve situacije potiču prosječnog studenta na privremeno odlaganje odlaska u menzu.

Redovi veći od srednje kategorije, koji se sastoje od izuzetno velikog broja ljudi, često nadilaze uvjete ispravnog rada promatranih izvedbi rješenja (npr., lokacija reda unutar prostora menze, ograničen broj ljudi u redu radi mogućnosti detekcije...). Zbog čega nije potrebno definirati više kategorija u tom rasponu, već se mogu dodati tijekom izvedbe po potrebi.

Zaključno, definirajući broj ljudi u redu kao n, početne okvirne kategorije su kratak red ($n < 5$), srednji red ($5 < n < 15$) i dugačak red ($n > 15$).

Dodatan cilj rada je skupljanje podataka kretanja reda tijekom dana, s pomoću čega je moguće definirati konkretnu strategiju za izbjegavanje gužve koju bi student mogao implementirati. Strategiju određuje potrebnu fleksibilnost vremena polaska u menzu (t_f), kako bi se izbjeglo vrijeme čekanja duže od neželjenog (t_n) s postotkom uspješnosti ($P_{uspješnost}$).

3. PRISTUPI

Postoje mnogi kreativni pristupi koji bi bili adekvatni za rješenje problema ovoga rada. Kriteriji za izbor najboljeg pristupa su (1) složenost implementacije pristupa; ne smije biti prelagan, ni presložen za svrhu ovoga rada, (2) skalabilnost rješenja, koja se definira kao mogućnost ponovne izrade u drugim lokacijama, uz zadržavanje kvalitete proizvoda i sprječavanje nastajanja novih problema u izvedbi koji su se mogli predvidjeti prije, (3) cijena rješenja, u najboljem slučaju bilo bi poželjno odabrati pristup najjeftinije cijene, (4) kvaliteta rješenja, koliko je točan završni sustav.

Razmatrat će se pristupi (1) infracrvenog senzora na ulazu kantine, (2) postavljanja kamere u kantinu i procesiranja video sadržaja, (3) korištenja mikrofona za snimanje zvuka i analize, (4) predviđanja gužve gledajući rasporede nastave i (5) pristup ekstrapoliranja duljine reda detektiranjem WiFi signala pametnih telefona.

3.1 Detektor prolaska čovjeka kroz ulaz kantine

Kao jedno od rješenja, postavljanje infracrvenog senzora na ulaz u kantinu bilo bi relativno lako izvedivo i u idealnom svijetu bi brojanjem ulaza i izlaza ljudi pružalo zadovoljavajuće rješenje. Osim predviđanja stanja linije brojanjem, postoje i druge tehnike detektiranja broja ljudi u redu.

Temeljna je ideja, da je brojanjem ljudi koji ulaze ili izlaze iz menze moguće napraviti približnu pretpostavku broja ljudi koji trenutno čekaju u liniji. Detektor bi sam po sebi periodično (reda veličine od 1ms do ~ 50ms) davao čitanja, detektira li ispred sebe objekt (studenta), zajedno s vremenskim žigom (timestamp) za očitano stanje. [1] Procesiranjem i analizom opisane vremenske serije (engl. time series), moglo bi se utvrditi kakav slijed događaja u seriji podataka bi označivao detekciju prolaska čovjeka (uračunavajući i vremenski razmak između događaja).

Detektor ne bi pružao informacije samo o sveukupnom broju ljudi u kantini, već i koliko je ljudi ušlo i u kantinu u zadnjih *t* vremena (i izašlo, što je relevantno za kasniju analizu). Uspoređivanjem podataka stvarnoga stanja linije u menzi i stanja očitanog infracrvenim čitačem (povijest ulaska ljudi u zadnjih *t* vremena), moguće je doći do zadovoljavajuće razine točnosti predviđanja gužve.

Nažalost, postoji nekoliko problema s detektorom prolaska čovjeka, od kojih je prvi slaba povezanost podatka iz kojih se predviđa stanje i pravo stanje u menzi. Za jednu osobu koja čeka u liniji (tj. utječe na vrijeme čekanja) dobivena je jedino informacija o ulasku te osobe u prostoriju, nakon čega je sve ostalo predviđanje. Rješenje bi znatno slabije funkcioniralo u izvanrednim situacijama (npr. dodatna kasa radi, ne radi ni jedna kasa, pauze zaposlenika, neobična brzina kretanja linije, itd..), jer nema nikakav uvid unutar prostorije menze. Osim toga, u slučajevima velikog broja ljudi u redu, red se proteže kroz ulaz u prostoriju menze, što bi efektivno poništilo bilo kakvu vjerodostojnost podataka koje detektor očitava. Također, kada bi se osoba zadržala na ulazu, detektor bi mogao očitavati lažne signale. Nadalje, u slučaju da se red proteže kroz ulaz, konzistentno čitanje "vidim objekt ispred sebe" bi moralo značiti da linija ide do vrata (ili dalje od vrata). Naprotiv, male kretnje osobe, koja se nalazi na ulazu menze, izazivale bi lažna čitanja na detektoru, koje bi zbog kumulativnog načina računanja gužve dovele do nepopravljivog stanja (negativan broj ljudi u menzi, nerealno visoki broj, itd.). Ovo rješenje je i lako prevariti namjernom ometajućom kretnjom ispred senzora.

Već spomenut slučaj zadržavanja čovjeka na ulazu vrata (lokaciji detektora) potaknuo je ideju za drugu tehniku infracrvenim senzorom. U drugoj tehnici, senzor služi kao točka u prostoru do koje red ide. Takav sustav pružao bi informacije o veličini reda, što je dovoljno ili jednako dobro kao i informacija o broju ljudi u redu. Redovi se virtualno uvijek kreću istim pozicijama u prostoru, te problem nepoznavanja oblika reda ne bi igrao ulogu. S druge strane, kako je već spomenuto, definiranje redoslijeda očitanja stanja na određenoj lokaciji, koji bi s dovoljnom sigurnošću garantirao da linija zaista prolazi lokaciju specifičnog detektora je težak zadatak. Linija bi mogla prolaziti dalje od lokacije senzora, dok nijedna osoba ne bi stajala točno ispred senzora, što bi rezultiralo krivom čitanju situacije i brzim promjenama u kategorijama stanja gužve kroz vrijeme. Osim toga, za pojedinu menzu bio bi potreban drugačiji broj i redoslijed detektora, koji bi zatim trebali moći ili samostalno komunicirati WiFi-om, ili alternativnim načinom izmjenjivati

informacije i slati ih na poslužitelj (engl. server), što znatno komplicira izvedbu rješenja i zbog toga nije izabrana.

3.2 Vizualna detekcija gužve kamerom

Postavljanje kamere pruža najpreciznije rezultate od svih rješenja. Potrebno je uspostaviti snimke uživo (engl. live stream), prenoseći ga s kamere na javno web sučelje. Student (korisnik) bi imao vizualan prikaz gužve, s pomoću kojeg bi sam mogao odlučiti ako mu se u trenutku isplati krenuti u menzu, ili pričekati neko vrijeme.

Za predviđanje budućeg stanja reda, potrebno je iz sirove slike (videa) izvući informaciju o stanju gužve na digitalno-čitljiv način. Za taj posao idealno je iskoristiti pogodnost dostupnost postojećih modela strojnog učenja namijenjenih za obradu slike (eng. Machine learning models for image processing), ili točnije detekcije ljudi u slici (eng. Person detection in images). S takvim alatom ne bi samo bio dostupan broj ljudi, već i njihove točne lokacije u prostoriji. S preciznom lokacijom dobila bi se precizna brzina kretanja reda, uz koju se može preciznije predvidjeti buduće stanje gužve. (Bržom kretnjom se red čekanja u kraćem roku smanjuje)

Postavljanje kamere bilo bi najkvalitetnije rješenje za rad, s prilično jednostavnom izvedbom. Ali za postavljanje je potrebno tražiti dopuštenje od vlasnika u svakoj menzi. Većina menzi već ima postavljenje kamere, ali zbog sigurnosti i privatnosti, nepoznato je ako bi se mogao dobiti pristup videu. Osim toga, rješenje zahtjeva specifičnu opremu i profesionalnost potrebnu za postavljanje opreme.

3.3 Detekcija gužve mikrofonom

Postavljanje mikrofona jedna je od jednostavnijih i najjeftinijih ideja. Promatrajući uzorke stvarnoga stanja gužve u menzi i zvuka koji mikrofon zapisuje, bilo bi moguće ekstrapolirati stanje gužve u menzi u dvije široke kategorije, npr. mala i veća gužva. Iznad određenog broja ljudi jačina buke se više ne povećava ili ne može biti detektirana i potrebna je sofisticiranija analiza za preciznije rezultate (ali točnost rješenja bi i dalje imala nizak „strop“). Također, postoji problem privatnosti sa stavljanjem mikrofona u menze (uređaja koji snima zvukove a ujedno i razgovore), što otežava postupak dobivanja dopuštenja od vlasnika menze, za postavljanje uređaja. Potrebno je i točno mjesto stavljanja mikrofona u prostoriji radi bolje točnosti detektiranja gužve, što čini uređaj manje diskretnim. Za razliku od ostalih ideja, puno je teže standardizirati alat za druge menze ili prostorije, što narušava i cilj skalabilnosti.

3.4 Predviđanje gužve s pomoću rasporeda

Predviđanje gužve s pomoću rasporeda predavanja, pristupa problemu na drugačiji način od ostalih rješenja. Moguće je skupiti informacije o tome kada završava nastava na obližnjim fakultetima, što je korisno jer najčešći dolasci u menzu događaju se upravo nakon završetka nastave. Ipak, termini završetaka nastave ne daju dovoljno vjerodostojnu sliku kretanja studenata jer se termini često mijenjaju i popravljanje te informacije je teško dostupno i zahtjeva redoviti ručni unos novih informacija. Također, ne zna se ako i kada će neka predavanja završiti ranije. Iako ideja ne predstavlja potpuno rješenje, može biti korisna dopuna drugom rješenju kod predviđanja gužve. Kvalitetniju informaciju moguće je dobiti od rezervacija učionica u stvarnom vremenu. Točno rezervirani termini pružaju stvarniju sliku kretanja studenata na fakultetu, za razliku od plana nastave, koji se napravi i provede jednom na početku semestra.

3.5 Detekcija gužve s pomoću detekcije WiFi signala pametnih telefona

Svaki student sa sobom nosi pametni telefon, koji se često koristi za vrijeme čekanja u redu. Korištenjem ove informacije, moguće je zaključiti da brojanje (WiFi) uređaja u redu čekanja, dovodi do točnog broja ljudi u redu. Dakle moguće je dobiti uvid u stvarno stanje reda čekanja u menzi detekcijom WiFi signala u prostoriji.

Za detekciju WiFi signala, uz malo računalo, potrebna je jedino dodatna WiFi kartica adekvatne kvalitete. Naravno, većinski nije moguće utvrditi smjer dolaska WiFi signala, time bi kartica bila „slijepa“ za detaljnije stanje rasporeda ljudi u menzi. Tu manu se nadoknađuje drugim informacijama koje se dobiju iz WiFi signala; jačina signala (proporcionalna relativnoj udaljenosti telefona od WiFi kartice), količina vremena bivanja uređaja u menzi, i druge.

Postavljanjem uređaja u blizini uobičajenog lokacijskog središta reda, i uzimanjem signala samo jačih snaga, filtrirali bi se telefoni koji nisu u redu čekanja. Moguće je argumentirati da je nepouzdano računati da će svi telefoni slati WiFi signale. Ta nepouzdanost povećava granicu pogreške rješenja, ali je validno pretpostaviti da omjer detektiranih i ne detektiranih telefona u redu većinski ostaje isti kroz različite slučajeve. Poznavajući omjer telefona, lako je „namjestiti“ krajnji rezultat. Na primjer, omjeru od dva detektirana uređaja $n_d = 2$ naspram jednog ne detektiranog uređaja $n_{nd} = 1$, odgovarao bi kvocijent kompenzacije (Q_k):

$$Q_k = \frac{n_d + n_{nd}}{nd} = \frac{3}{2} = 1{,}5 \qquad (3.1)$$

Broj detektiranih uređaja bi se pomnožio s kvocijentom Q_k, te bi se dobilo pravo stanje gužve u redu. Još jedno moguće poboljšanje bi uključivalo fizičko blokiranje WiFi signala postavljanjem prepreke (aluminijske folije) prema strani neželjenih očitanja telefona, što bi u kombinaciji s filtracijom samo jačih WiFi signala rezultiralo još točnijim podacima.

3.6 Izabrani pristup

Iako su neki od ovih rješenja važeća, detekcija gužve s pomoću brojanja pametnih telefona WiFi signalima u prostoriji pokazala se najboljom opcijom. Grubim limitiranjem detekcije pametnih telefona samo jačih WiFi signala, i kratkim postupkom prilagođavanja rezultata pri postavljanju alata (grubo računajući uređaje od stalnih radnika u kantini i množenje kvocijentom Q_k), moguće je dobiti dovoljno precizne informacije o gužvi.

Na temu dopuštenja stavljanja uređaja u kantinu, ovo rješenje ne bi trebalo pružati probleme, uređaj bi se postavio u standardnu utičnicu blizu reda čekanja. Cijena rješenja je približnih 100 eura, uračunavajući dodatnu marginu. Cijenu čini proizvoljni model Raspberry Pi-a (40 eura), dodatna Wi-Fi kartica (40 eura) i ostale potrebne komponente (margina). [2]

4. ARHITEKTURA

Za detekciju (telefonskih) uređaja detekcijom WiFi signala, potrebni su WiFi kartica i kombinacija hardvera i softvera koji mogu staviti tu WiFi karticu u monitor način rada. Također, moraju moći čitati informacije s WiFi kartice i slati HTTP zahtjeve. Za slanje HTTP zahtjeva potrebno je spojiti uređaj na WiFi, što monitor način rada zabranjuje, zbog čega bi kartica izmjenično bila postavljena u monitor način rada i managed način rada. (U slučaju samo jedne WiFi kartice na uređaju)

4.1 Hardver

Za minimalnu izvedbu navedenih zahtjeva dovoljan bi bio mikrokontroler iz *ATmega* serije, na koji je spojena WiFi kartica i instaliran pripadajući driver. WiFi kartice imaju dva načina rada, takozvani managed način rada, i monitor način rada. Managed način rada je uobičajeni način rada kartice gdje se očitavaju samo WiFi signali namijenjeni MAC adresi uređaja, dok monitor način rada omogućava pregled svih WiFi signala koje kartica detektira. WiFi kartica mora biti dovoljno napredna da podržava monitor način rada. Osim toga, bilo bi potrebno napisati softversko rješenje za postavljanje kartice u monitor način rada. Većina izvedbi postavljanja bilo koje WiFi kartice u monitor način rada upotrebljava postojeće drivere za tu specifičnu karticu. Ti driveri skoro uvijek imaju ograničenja za specifična radna okruženja, tj. operacijske sustave i njihove verzije te alate i njihove verzije. Kako je ATmega resursno limitirana, i ne podržava operacijske sustave, sva komunikacija s WiFi karticom morala bi biti ručno razvijena. Pisanje drivera za WiFi karticu je izrazito kompleksan zadatak, zbog čega ATmega kao centralno upravljačko računalo nije poželjno.

Zbog dostupnosti uređaja, kao hardver je korišten Raspberry Pi 4; jednostavno, minimalno računalo (engl. single-board computer), adekvatno za svrhu projekta. S minimalnim zahtjevima Linux operacijskog sustava za računalnim resursima, čak i Raspberry Pi 4 ih može pokretati sa

svojih 4 GB RAM-a i 32 GB permanentne memorije. Kao operacijski sustav, Ubuntu se pokazao adekvatnom opcijom, zbog popularnosti i velike podrške zajednice. Iako bi se Windows pokazao puno lakšim za postavljanje drivera i za druge operacijske stavke kod namještanja projekta, Windows ne pruža dovoljno nisku razinu privilegije za uključenje monitor načina rada na kartici, zbog čega nije korišten u ovome radu.

Biranje WiFi kartice se možda čini jednostavnim zadatkom, ali u stvarnosti nije trivijalno. Ne podržavaju sve WiFi kartice monitor način rada, i same po sebi WiFi kartice nemaju izraženu karakteristiku mogućnosti ulaženja u monitor način rada. Potrebno je isprobati svaku karticu zasebno. Najbolja tehnika za pronalaženje kartica koje podržavaju monitor način rada (osim biranja najskuplje), je istraživanje osobnih iskustava ljudi na internetu, koji su probali karticu i podijelili svoja iskustva. Nažalost većina izvora informacija dolazi iz SAD-a, gdje je ponuda WiFi kartica u potpunosti drugačija od Europe, a time i Hrvatske. Od najkvalitetnijih informacija na internetu, isprobane su dvije kartice. Kartica TP-link TL-WN725N za koju je proizvođač tvrdio da podržava monitor način rada, bila je dobar izbor zbog male fizičke veličine i niske cijene, no nažalost nije podržavala monitor način rada. Nakon neuspjeha s prvom karticom, izabrana je skuplja kartica Archer T4U, koja se iskazala adekvatnom.

4.2 Softver

Za uporabu WiFi kartice, potrebno je instalirati driver namijenjen za tu karticu. Drivere pruža proizvođač. U Linux operacijskim sustavima, za instalaciju third-party drivera potrebno je skinuti njihov izvorni kod (engl. source code), tj. datoteke s ekstenzijom '.c' (programski jezik C) i njihove “header” datoteke s ekstenzijom '.h'. S kompilacijskim alatom točne verzije, koju specificira proizvođač kratice i drivera, potrebno je kompilirati izvorni kod i rezultirajuću datoteku “postaviti“.

Kako bi uređaj očitao sve “sirove” WiFi signale koja kartica snimi, čak i one koje nisu bile namijenjene za nju (odredišna MAC adresa ne odgovara adresi kartice), potrebno je karticu postaviti u monitor način rada. Aircrack-ng je paket alata koji služi za procjenu sigurnosti Wi-Fi

mreža. Jedan od tih alata je Airmon-ng, koji služi za postavljanje kartice u monitor način rada, i upravljanje (uklanjanje) problematičnih servisa na Ubuntu-u, koji ometaju pravilan rad kartice. Airodump-ng je alat koji upravlja karticom kako bi dobio sve WiFi signale za koje je kartica sposobna uhvatiti, a ne samo namijenjene za njenu MAC adresu. U terminal zapisuje vizualan prikaz WiFi paketa koje kartica hvata, što olakšava i ubrzava razvoj rješenja, jer daje stanje uživo, te nije potrebno zasebno pregledavati izlazne datoteke skripte. Rad Airodump-ng skripte omogućen je tek kad je WiFi kartica postavljena u monitor način rada. S omogućenom opcijom, skripta Airodump-ng također u datoteke proizvoljnog imena ispisuje kumulativni izlaz svoje skripte.

Generirane datoteke iz skripte Airodump-ng potrebno je učitati i obraditi u tip podataka prikladan za analizu. Nadalje, i jedinstveno izolirati sve uređaje od kojih je kartica pokupila WiFi signal(e) i potražiti njihove MAC adrese u službenoj listi jedinstvenih organizacijskih identifikatora (engl. Organizationally unique identifier, OUI), koje pruža organizacija IEEE (Institut inženjera elektrotehnike i elektronike).

S pronalaskom OUI-a iz MAC adrese, tj. s pronalaskom proizvođača WiFi kartice detektiranog uređaja, moguće je pretpostaviti je li taj uređaj pametni telefon. Ako je proizvođač detektirane WiFi kartice jedan od proizvođača Apple, Samsung, Huawei, Lenovo ili Xiaomi, valjano je pretpostaviti da je uređaj pametni telefon. Moguće je da u blizini postoje laptopi priključeni na WiFi, koji imaju iste proizvođače, ali za svrhu ovoga rada navedena klasifikacija je dovoljna. Na primjer, MAC adresa s početkom 60-FD-A6 odgovara proizvođaču Apple Inc., time se za nju pretpostavlja da je pametni telefon.

Nakon filtriranja i brojanja uređaja, zajedno s vremenskom oznakom (engl. timestamp), podaci se trebaju spremiti u lokalnu povijesnu bazu stanja u menzi (records.txt). Daljnje je potrebno obraditi informacije na istom lokalnom uređaju. U slučaju eventualnog skaliranja projekta (na više od jedne menze), bolje je imati centralni sustav za obradu informacija radi lakšeg verzioniranja projekta i direktne dostupnosti podataka, ali to dodatno opterećuje hosting, zbog čega nije implementirano.

Za osposobiti poslužitelj potrebno je uporabiti Node.js, programsko okruženje koje omogućuje izvršavanje Javascript koda izvan web preglednika. Node.js poslužitelj ovoga rada se

treba sastojati od dva endpoint-a, jedan za čitanje trenutnog stanja u menzi (/getCurrentState, kojeg poziva web dio poslužitelja) i jedan za postavljanje trenutnog stanja sa strane Raspberry Pi-a (/postCurrentState, kojeg poziva Raspberry Pi).

Trenutno stanje treba se čuvati u datoteci (currentState.txt). Zadnje što je ostalo je razviti jednostavno web sučelje za čitanje trenutnog lokalnoj stanja s poslužitelja. Za to je potrebna jednostavna HTML i Javascript izvedba, pri čemu se zahtjev šalje za trenutno stanje (/getCurrentState) i rezultat se prikazuje na web stranici tekstom.

Potrebno je razviti i ostala rješenja, kao na primjer, pokretanje programa pri uključivanju Raspberry Pi-a u struju, pokušaj ponovnog uspostavljanja veze s internetom u slučaju prekida i mnogih drugih, kako bi se upotpunila funkcionalnost projekt.

5. TEHNOLOGIJE

5.1 Aircrack-ng

Aircrack-ng je set alata korišten za ispitivanje sigurnosti WiFi mreža. S njim je moguće promatrati pakete koji se šalju WiFi signalima, simulirati napade na mrežu, reproducirati pakete, probijanje WEP i WPA PSK protokola, testirati sposobnosti internet hardvera i drugo. Svi alati koje pruža su bazirani na CLI-u (Command Line Interface), što ih čini savršenim za ovaj rad jer to omogućuje lagano korištenje alata u skriptama.

U radu se ne koriste svi alati pružani Aircrack-ng svitom, već samo nekoliko ključnih. Airmon-ng se koristi za uključivanje monitor načina rad na WiFi kartici i ubija mrežne procese na operativnom sustavu, koji bi mogli utjecati na rad kartice. Airodump-ng se u radu koristi za hvatanje “sirovih” 802.11 okvira, s kojima daje potpuni prikaz internetskih paketa poslanih WiFi-om.

5.2 Ngrok

Ngrok je online servis koji stvara sigurne mrežne tunele do lokalnih mašina, koje se mogu koristiti na primjer kao poslužitelji (engl. serveri). U slučaju ovoga rada, „lokalna mašina“ predstavlja terenski uređaj postavljen u menzu. S Ngrok-om se spaja na uređaj na terenu, bez komplikacija s Firewall-om ili drugim mrežnim ograničenjima. Ngrok je neophodan za ovaj rad jer se uređaj na terenu spaja na internetsku mrežu eduroam, koja je zaštićena mreža s ograničenjima preko koje ne bi bilo moguće uspostaviti port forwarding za rad s SSH-om.

5.3 Node.js

Node.js je Javascript runtime okruženje uz pomoću kojeg je moguće razvijati poslužitelje (engl. servere) i web aplikacije, od kojih su obje potrebne za ovaj rad. Ima veliku podršku zajednice u obliku knjižnica, koja pružaju gotova programska rješenja za širok spektar problema i potreba. Node.js koristi npm (Node package manager), jednostavan CLI alat koji služi za upravljanje njegovih (Node.js) knjižnica.

Za uspostavljanje poslužitelja, u ovome radu koristi se knjižnica *express.js*, minimalan set alata za razvijanje značajki web stranica i REST API-ova. Uz pomoću nje se host-a web aplikacija, koja prikazuje trenutno izračunato stanje u menzi. U isto vrijeme se host-a i REST API, koji služi kao posrednik uređaja na terenu i web aplikacije. Pruža terenskom uređaju endpoint za slanje podataka, koje poslužitelj permanentno sprema u svoj datotečni sustav. Drugim endpoint-om web aplikaciji pruža spremljene podatke.

6. POSTAVLJANJE OKRUŽENJA RADNOG SUSTAVA RASPBERRY PI

Zbog velikog broja odvojenih dijelova u projektu, korisno je razviti/pronaći alate koji olakšavaju razvoj i pronalaženje greški, i omogućavaju ručni uvid u povezanost svih dijelova projekta. Cilj ovoga rada je automatsko djelovanje svih razvijenih alata, te ručan rad s njima nije predviđen, što služi kao dodatna motivacija za postavljanje radnog okruženja za dijagnostiku.

6.1 Instalacija drivera WiFi kartice

Za Linux sustave, izvršni kod (engl. source code) drivera WiFi kartice mora se kompilirati (engl. compile) na okruženju gdje će kasnije taj driver biti instaliran. Proizvođač poslužuje programski paket koji se, među ostalom, sastoji od skripti u programskom jeziku *C* i *Makefile* datoteke koja pruža *make* naredbi potrebne informacije za sastavljanje koda. Nakon uspješnog izvršenja, stvori se *88x2bu.ko* datoteka, koja služi kao „izvršna“ datoteka drivera. Driver 88x2bu.ko se učita pomoću alata *insmod,* koji instalira kompilirani driver u Linux operacijski sustav, i osposobljuje karticu za rad. Za provjeru uspješnosti instalacije drivera koristi se naredba *lsmod,* koja ispisuje sve instalirane drivere na operacijskom sustavu. Ako se 88x2bu.ko nalazi među njima, instalacija je bila uspješna. Doduše, lista drivera u bilo kojem računalnom okruženju najčešće je predugačka za brzi ljudski pregled, time se spomenutoj *lsmod* naredbi može spojiti *grep* naredba, kojom se filtrira ispis u terminalu. [3]

```
$ make clean
$ make
$ insmod 88x2bu.ko
$ lsmod | grep 88x2bu.ko
```

Pri prvim pokušajima pokretanja *'make clean'* i *make* naredbe, došlo je do nekoliko problema. Kod se nije ispravno izvršio zbog toga što je ime skinutog direktorija od proizvođača sadržavalo razmak, te je dolazilo do problema s utvrđivanjem putanji (engl. path) u kodu. Nakon uklanjanja razmaka i ponovnog pokretanja *'make clean'* naredbe, ispisan je novi kod pogreške, koji je upućivao na iste probleme s razmacima, ali u ovom slučaju s datotekama unutar pod-direktorija skinutog programskog paketa. Nakon popravljanja imena datoteka, naredba *make clean* se uspješno izvršila.

```
Makefile:617: arch/aarch64/Makefile: No such file or directory
```

Nadalje, kod naredbe *make* je također dolazilo do kompilacijske pogreške. Problem je dolazio iz nekompatibilnosti skinute skripte i Linux distribucije (verzije) terenskog uređaja. Ime krucijalnog direktorija za stvaranje drivera se u nekim Linux distribucijama zove *aarch64*, a u drugim *arm64*.

```
$ export ARCH=aarch64
```

Skripta je neispravno utvrdila ime direktorija, zbog čega je bio potreban ručan izvoz (engl. export) varijable skripte „*ARCH*" na vrijednost aarch64,. Nakon toga se kod počeo uspješno izvršavati, no skripta je zatim javljala kompilacijsku grešku. Kod nije bio ispravan, ili se nije ispravno izvršavao zbog verzije sustava, kernel verzije ili verzije gcc-a (alat za kompilaciju programskih jezika, u ovom slučaju C-a).

Na stranici proizvođača dokumentirano je da ne garantiraju ispravnost drivera na Linux operacijskim sustavima, ako se ne izvrši na okruženju kojeg su specificirali. U slučaju drivera korištenog u ovome radu, specifikacije su kernel verzija 4.4 i gcc verzija 5.4, što odgovora Ubuntu verzije 16.04 (2016. distribucija). [4] Kako bi se suzili mogući uzroci problema kompilacijskih pogreški, pokušana je instalacija Ubuntu verzije 16.04. Nadalje, došlo je do još jedne prepreke, u procesu montiranja operativnog sustava na Raspberry Pi, često se koristi alat Raspberry Pi Imager [5], koji služi za stvaranje USB-a za pokretanje operacijskih sustava (engl. bootable USB). Na alatu je bio ponuđen izbor Ubuntu instalacija, ali zbog toga što je Ubuntu verzija 16.04 u vrijeme

pisanja ovoga rada bila stara više od pet godina, nije bila ponuđena kao jedna od službenih distribucija prilagođenih za Raspberry Pi 4.

Pokušaj instalacije običnom ISO datotekom distribucije Ubuntu 16.04, također je bio neuspješan. Raspberry Pi zahtjeva .img datoteku za instalaciju, koja nije pronađena na internetu. Nakon dugog slijeda koraka od kojih nijedan nije doveo napredak u radu, privremeno se odlučilo drugačijem pristupu problemu, kako bi se utvrdili razlozi dobivenih problema u posljednjim koracima. Nakon dugotrajnog neuspješnog pokušaja stavljanja Linux operacijskog sustava, uz Windows, kao "dvostruko pokretanje" (engl. dual boot) na osobno računalo (najvjerojatnije zbog nedosljedne kompatibilnosti formata trajnog spremišta SDD-a i Linux distribucije), bilo je potrebno još jednom promijeniti pristup radu. Srećom, još jedno računalo je bilo dostupno, koje nije imalo instalirano operacijski sustav.

Na novom računalu, koje se eksperimentalno koristilo za utvrđivanje uzroka pogreške pri instalaciji drivera, uspješno se instalirala distribucija Ubuntu 16.04. Prateći prijašnje korake, driver se i dalje nije ispravno kompilirao. Detaljnim istraživanja na internetu, utvrdilo se da u slučaju neispravnog drivera, pruženog od TP-link online servisa, je potrebno daljnje potražiti rješenje u nezavisnim third-party Github repozitorijima zasebnih developera, koji su objavili vlastite verzije drivera s popravcima, i ili dodacima. [6] Nakon još nekoliko neuspješnih pokušaja s neslužbenim driverima, na Github-u je pronađen izvorni kod za driver koji se uspješno izvršio.

6.2 Send

Za lakše testiranje slanja HTTP zahtjeva na poslužitelj, korisno je imati prilagođenu naredbu (engl. custom executable), dostupnu kao komanda bilo gdje u terminalu. Navedena naredba imenovana je *send*. Općenito, prilagođena naredba na Linux sustavima izvršava svoju dodijeljenu prilagođenu skriptu (engl. custom script). Na primjer, korištenjem naredbe *ls* se izlistaju poddirektoriji i datoteke trenutnog direktorija.

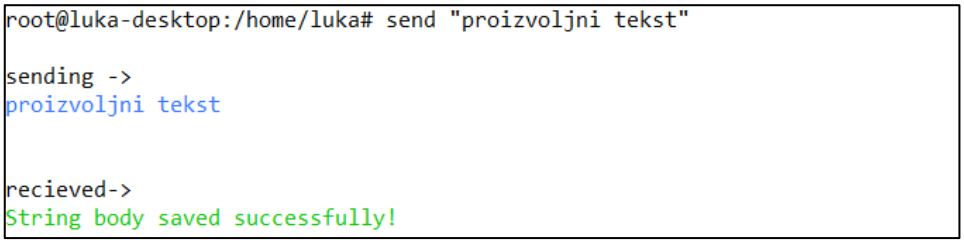
```
#!/bin/bash


message=""

if [ -z "$1" ]; then
        message="example from rasb"
else

        if [ "$1" = "-f" ]; then
                file="$2"
                if [ -f "$file" ]; then
                        message=$(cat $file)
                else
                        echo -e "${YELLOW}File $file does not exist.${NC}"
                        exit -1
                fi
        else
                message="$1"
        fi

fi
Y="\033[0;93m"
A="\033[0;91m"
NC="\033[0m"
printf "\nsending ->\n${Y}$message\n\n${NC}"

response=$(curl -s -X POST  -H "Content-Type: text/plain" -d "$message" https://shielded-atoll-62349-d56f9518b67e.herokuapp.com/save-string-body)

printf "\nrecieved->\n${A}$response\n\n${NC}"

printf "\n${NC}"
```

Slika 6.1 Kod koji se izvrši zvanjem „send“ komande

Prilagođena skripta za naredbu *send* može biti pozvana sama za sebe, uz tekst kao parametar ili uz datoteku kao parametar.

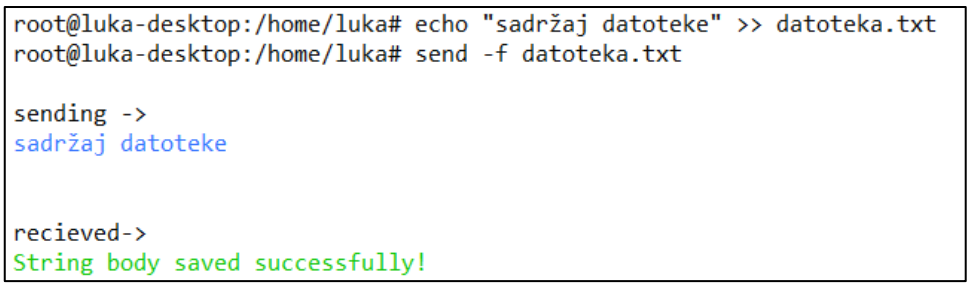
Slika 6.2 Ispis send komande

Kada se skripta izvrši bez dodatnih parametara, pošalje se tekst *"example from rasb"* i ispiše se odgovor od HTTP zahtjeva.



Slika 6.3 Ispis send komande s proizvoljnim tekstom

Korištenjem naredbe send s jednim parametrom, parametar se šalje kao tijelo HTTP zahtjeva (engl. HTTP request body).



Slika 6.4 Ispis send komande s file opcijom

Parametrom *–f* i dodatnim parametrom relativne ili apsolutne putanje datoteke, šalje se sadržaj te datoteke.

Za korištenje naredbe u terminalu, potrebno je imenovati skriptu "send" (bez bash nastavka *.sh*), s komandom *chmod* označiti ju kao izvršivu datoteku (engl. executable), i premjestiti ju na lokaciju */usr/local/bin*. [7]

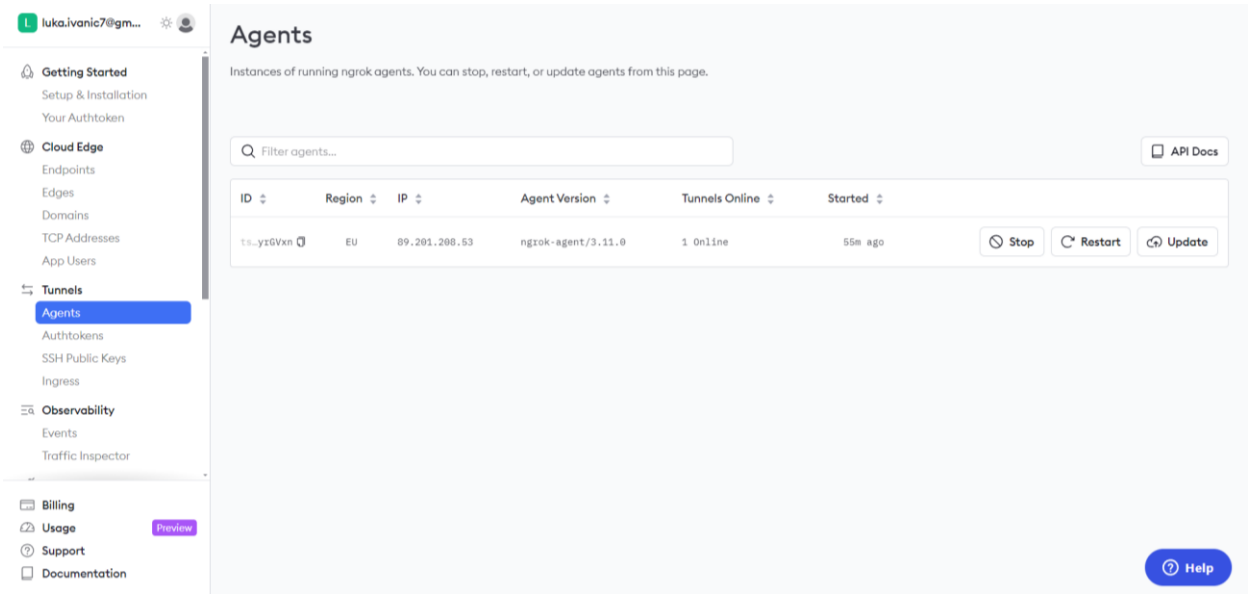
6.3 SSH

Za stalni razvoj aplikacije i za mogućnost upravljanja iz daljine (engl. remote control), potrebno je postaviti pouzdan SSH tunel prema Raspberry Pi-u. Za SSH na specifičnu IP adresu potrebno je namjestiti port-forwarding, ali kako se Raspberry Pi spaja na eduroam (WiFi dostupan studentima unutar fakulteta), administrativni pristup usmjerivaču (engl. router) nije moguć. Alternativno rješenje potrebno. Uz spomenuti Ngrok servis, moguće je namjestiti SSH forwarding (koristeći se „reverse SSH" tehnikom), čime bi se izbjegao Firewall i bilo kakve druge prepreke koje bi sprječavale uspostavljanje SSH tunela na Raspberry Pi.

```
#!/bin/bash
systemctl start ssh
systemctl enable ssh
ngrok config add-authtoken 2iMUhlJkFg0qIsrNMg36Pmt8wMA_7Sd9G7fVBNU6qd8nZZye7
ngrok tcp 22
tmux
```

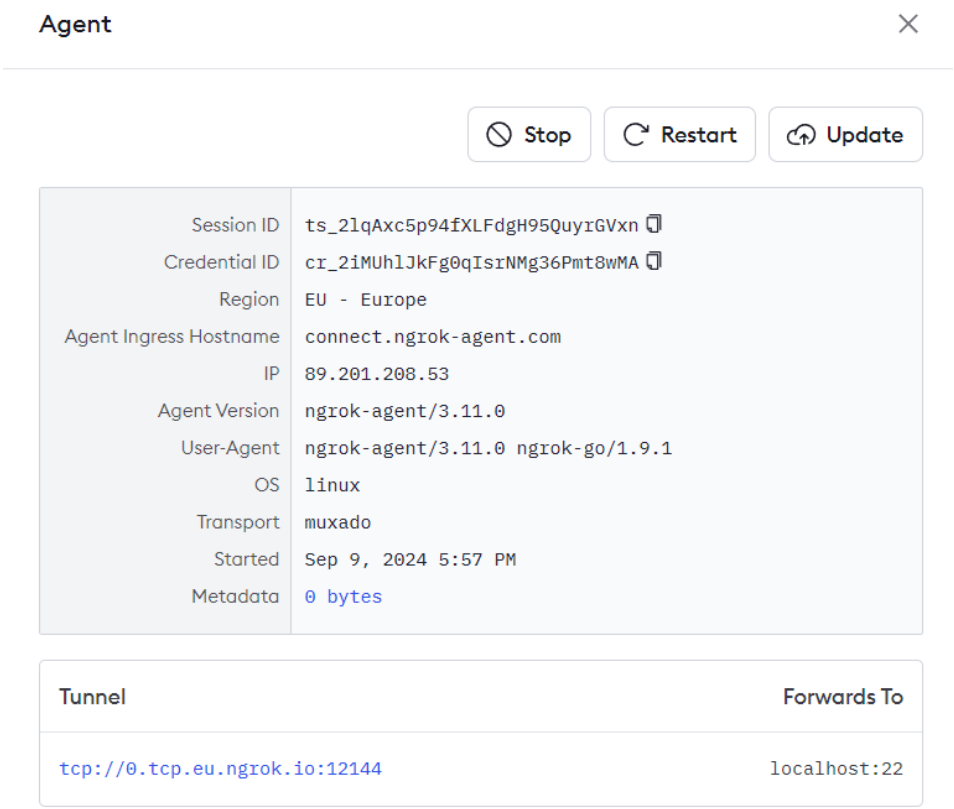
Slika 6.5 Kod za postavljanje Ngrok servisa (skripta sshOnPCStart.sh)

Prije uspostavljanja SSH veze, započne se (*start*) s*sh* servis i omogući mu se (*enable*) izvršavanje pri pokretanju računala tj. terenskog uređaja. Postavlja se auth token, koji se nalazi na stranici Ngrok servisa, te se veza inicijalizira komandom *ngrok tcp 22*.



Slika 6.6 Radna ploča Ngrok servisa

Nakon inicijalizacije, veza je prikazana na stranici Ngrok servisa.



Slika 6.7 Detalji uspostavljenog agenta na Ngrok servisu

Za spajanje na Raspberry Pi, potrebni su IP i port posvećenog tunela, i ime korisnika na Raspberry Pi-u *(luka)*.

```
C:\Users\lukai>ssh luka@6.tcp.eu.ngrok.io -p 16544
The authenticity of host '[6.tcp.eu.ngrok.io]:16544 ([3.66.38.117]:16544)' can't be established.
ECDSA key fingerprint is SHA256:9lc+Kv+oLr6psdPzsFIVHIQ1uWazfT9hEQGM7Hthnic.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '[6.tcp.eu.ngrok.io]:16544,[3.66.38.117]:16544' (ECDSA) to the list of known hosts.
luka@6.tcp.eu.ngrok.io's password:
Welcome to Ubuntu 23.04 (GNU/Linux 6.2.0-1018-raspi aarch64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

131 updates can be applied immediately.
To see these additional updates run: apt list --upgradable

Last login: Mon Sep  9 17:59:35 2024 from 127.0.0.1
luka@luka-desktop:~$
```

Slika 6.8 Postupak spajanja na SSH tunel Ngrok servisa

Nakon upisivanje lozinke za korisnika Raspberry Pi-a, SSH tunel je uspješno postavljen.

Bez automatizacije, i dalje je potrebno prvo ručno uspostaviti vezu od Raspberry Pi-a prema Ngrok-u. Zato je skripta *sshOnPCStart.sh* postavljena na bootup, s pomoću systemd servisa.

```
[Unit]
Description=Start SSH service
After=network.target

[Service]
ExecStart=/home/luka/Documents/util/sshOnPCStart.sh
Type=oneshot
RemainAfterExit=yes

[Install]
WantedBy=multi-user.target
```

Slika 6.9 Postavke dodanog servisa start-ssh.service

Skripta je spremljena pod imenom *start-ssh.service*. Na postavki *'After'* stavljeno je *network.target,* što označava pokretanje nakon uspješnog podizanja internetske veze na računalnom sustavu. Postavka *'ExecStart'* prima putanju skripte koja se pokreće pri pokretanju servisa *start-ssh.service.* (Slika 6.1)

7. RJEŠENJE

Rješenje ovoga rada zahtjeva web sučelje za korisnike i REST API, kojemu Raspberry Pi šalje podatke u pravom vremenu, te je Node.js idealan izbor alata za projekt. Sposoban je u jednoj instanci napraviti i javni dio aplikacije, koja poslužuje web stranicu , i endpoint-ove, za prihvaćanje podataka od Raspberry Pi-a i za posluživanje podataka web stranici.

7.1 Postavljanje Node.js

Za korištenje alata Node.js, potrebno je skinuti instalater i pokrenuti ga [8]. Uz Node.js se instalira i *npm* (Node package manager). Nakon uspješne instalacije Node.js-a, potrebno je birati direktorij u kojem se namjerava razvijati projekt, i u CMD-u je potrebno izvršiti komandu *npm init*. S komandom *npm init* inicijalizira se početni projekt, koji sadrži potrepštine za Node.js projekt *package.json* i *node-modules*. Potrebno je stvoriti Javascript datoteku koja će služiti kao ulazna točka aplikacije. Ime te datoteke može biti proizvoljno dodijeljeno, ali u daljnjem tekstu će biti imenovana "app.js". Za postavljanje skripte *app.js* kao ulaznu točku aplikacije, potrebno ju je navesti na prikladnome mjestu unutar *package.json* datoteke. Nakon što su preduvjeti za korištenje Node.js-a zadovoljeni, aplikaciju je moguće pokrenuti CLI (Command line interface) komandom *node app.js.* Iako je aplikaciju moguće pokrenuti, sama ništa neće pružati, jer je skripta *app.js* prazna. U sljedećoj sekciji, pokazat će se postupak izgradnje REST API-a u Node-u.

7.2 REST API

Za razvijanje REST API-a u Node.js-u, standardni izbor knjižnice je *'express.js'*. Za korištenje *express.js* knjižnice, potrebno ju je dodati (instalirati) u postojeće Node.js module

naredbom *npm install express* na lokaciji projekta. Provjera instalacije vrši se pronalaskom *'express*' direktorija u generiranom nad-direktoriju (engl. parent folder) 'node_modules'. Ili alternativno, korištenjem *require('express')* funkcije u skripti *app.js* i pokretanjem Node.js projekta.

```
const express = require('express');
const app = express();
const bodyParser = require('body-parser');
const fs = require("fs")

app.use(bodyParser.text());

app.post('/uploadState', (req, res) => {
stringBody = req.body;
fs.writeFile('state.txt', stringBody, (err) => {
	if (err) {
			console.error(err);
			res.status(500).send('Error uploading state');
		} else {
			res.send('State saved successfully!');
		}
		});
	});

app.listen(process.env.PORT || 3000, () => {
	console.log('Server started on port 3000');
});
```

Kod *express.js* knjižnice, objekt (varijabla) dobivena pozivom funkcije *express()* se po pravilu naziva '*app'*, kao što je prikazano u kodnom isječku. Za pokretanje Node.js poslužitelja, poziva se funkcija *listen()*, kojoj je potrebno navesti broj port-a i lambda funkciju (koja će se izvršiti nakon uspješnog podizanja poslužitelja). Poslužitelj (engl. server) se zatim podiže na IP adresu računala (ili u slučaju lokalnog podizanja poslužitelja, koristi se adresa localhost, 127.0.0.0), u oba slučaja, uključujući navedeni port.

API endpoint-ovi se stvaraju funkcijama *get*, *post*, *put*, *patch* i *delete*, koje otvaraju endpoint-ove HTTP metoda respektivnih imenima funkcija. U isječku koda, dodan je endpoint s POST metodom, pod imenom "uploadState". Unutar lambda funkcije definira se kod koji se izvrši prilikom primanja zahtjeva tog endpoint-a. Lambda funkciji su dani parametri *req* i *res*, koji predstavljaju primljeni zahtjev (eng. HTTP request) i budući odgovor (eng. HTTP response). Način parsiranja tijela HTTP zahtjeva (engl. request body) definiran je *bodyParser* knjižnicom, točnije *bodyParser.text()* funkcijom. Binarni tok iz zahtjeva (eng. binary stream) izabrani način parsiranja pretvara u znakovni niz. Sadržaj iz tijela zahtjeva (eng. request body) nalazi se u atributu *body* iz objekta *req* (u kodu „*req.body*"). Svrha ovog endpoint-a je prihvaćanje i spremanje stanja gužve s Raspberry Pi-a. Stanje gužve šalje se kao odgovor GET metodi, kojoj će pristupati web dio aplikacije. Za spremište podataka koristi se lokalni datotečni sustav uređaja, koji poslužuje (engl. host) Node.js poslužitelj. Trenutno stanje o gužvi u menzi sprema se u tekstualnu datoteku. Kao što je spomenuto, bilo bi korisno spremati i povijest stanja gužve na poslužitelj, ali bi se dodatno opteretila usluga za hosting poslužitelja (Heroku), što bi povećalo mjesečne troškove razvijenog rješenja. Povijesni podaci se zato zapisuju u trajno skladište SD kartice u Raspberry Pi-u. Pristup povijesnim podacima je i dalje moguć koristeći SSH, tj. jednokratnim slanjem cijele povijesti (komandom *send -f*) na Node.js poslužitelj (engl. server) i skidanjem tih podataka na računalo.

```
app.get('/getState', (req, res) => {
	fs.readFile('state.txt', (err, data) => {
		if (err) {
			console.error(err);
			res.send('No state found');
		} else {
			res.send(data.toString());
		}
	});
});
```

Umjesto *post* funkcije tj. HTTP metode, za dohvaćanje podataka koristi se *get* funkcija, s nazivom endpoint-a „*getState*". Ponovno se koristi *fs* knjižnica za pristup datotečnom sustavu, uz

pomoću koje se čita datoteka state.txt i njen se sadržaj šalje kao tijelo odgovora (engl. response body) na primljen HTTP zahtjev. U slučaju bilo kakve pogreške programa, vraća se rezultat ‘No state found'.

Za provjeru ispravnosti definiranog poslužitelja i endpoint-ova u app.js-u, potrebno je pokrenuti poslužitelj (naredbom *node app.js*), i poslati zahtjeve na endpoint-e, za što je pogodno koristiti alat *curl*.

```
C:\Users\lukai>curl http://localhost:3000/getState
No state found
```

Slika 7.1 Slanje i odgovor s praznim podacima na poslužitelju

Kada još ne postoje trenutačni podaci o statusu gužve u menzi, tj. kada datoteka *state.txt* ima prazan sadržaj, vraća se poruka "No state found".

```
C:\Users\lukai>curl http://localhost:3000/getState
Stanje gužve u menzi
```

Slika 7.2 Slanje i odgovor s podatkom na poslužitelju

Kada postoje trenutačni podaci o stanju gužve u menzi, poruka u odgovoru je sadržaj datoteke *state.txt*.

```
C:\Users\lukai>curl http://localhost:3000/getState
Stanje gužve u menzi
C:\Users\lukai>curl -X POST -H "Content-Type: text/plain" -d "Novo stanje u menzi" http://localhost:3000/uploadState
State saved successfully!
C:\Users\lukai>curl http://localhost:3000/getState
Novo stanje u menzi
```

Slika 7.3 Testiranje ispravnosti rada postavljenih endpoint-a

Testiranje rada endpoint-a ”uploadState“ obuhvaća (1) prvobitnu provjeru sadržaja *state.txt* datoteke zvanjem „getState“ endpoint-a, (2) izmjene tog sadržaja, te (3) ponovnim dohvaćanjem sadržaja i (4) potvrdom jednakosti poslanog i spremljenog sadržaja.

7.3 WEB

Realizacija web stranice u Node.js-u slična je uobičajenom razvoju web stranica koristeći HTML, CSS i JS. Kod ispitivanja ispravnosti namještenog Node.js okruženja, potreban je jedino index.html, koji će na web prikazu prikazivati placeholder tekst.

```
<!DOCTYPE html>
<html>
	<head>
		<title>My Simple HTML Site</title>
	</head>
<body>
	<h1 id="h1">Welcome to my site!</h1>

	<script src="./index.js"></script>
</body>
</html>
```

Unutar radnog direktorija, potrebno je stvoriti pod-direktorij *public*, u kojeg je potrebno smjestiti stvoreni *index.html*. Zaključno, u skripti *app.js* potrebno je postaviti ime direktorija

(*public*) pod kojim se nalazi datoteka *index.html* (koja služi kao početna točka servirane HTML stranice).

```
app.use(express.static('public'));
```

Ovim koracima osposobljena je veza Node.js aplikacije i *index.html* skripte. Trenutni URL za pristup početnoj *index.html* stranici je „http://www.localhost:3000/public/index.html“, kojem se može skratiti *public* u putanji.

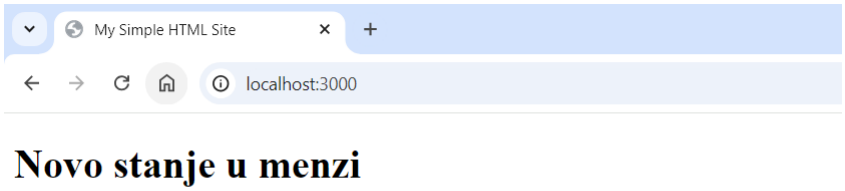
Za postavljanje preusmjerenja svih putanja, tj. zahtjeva za stranicu Node.js poslužitelja, koristi se *app.get()* funkcija.

```
app.get('*', (req, res) => {
	res.sendFile(__dirname + '/public/index.html');
});
```

Stranica trenutno prikazuje samo tekst “Welcome to my site!”. Taj tekst služi kao placeholder pri prvom učitavanju stranice. Kako bi stranica došla do podataka od poslužitelja, potrebna je skripta koja dohvaća podatke, u ovome radu imenovana je *index.js*.

```
const h1 = document.getElementById("h1")
fetch('/getState')
.then((response) => response.text())
.then((response) => {
	console.log(response)
	h1.innerHTML = response
})
.catch((error) => console.error(error));
```

Funkcijom *getElementById* iz objekta *document* dobiva referencu na element h1 definiran u *index.html*-u, koji u trenutno sadrži samo zamjenski (eng. placeholder) tekst. Za slanje HTTP zahtjeva koristi se *fetch* funkcija, koja je integrirana u Javascript i ne zahtjeva dodatne biblioteke iz Node.js paketa. Funkciji *fetch* potrebno je predati parametar URL, tipično se predaje cijela sastavnica URL-a poslužitelja koji poslužuje API (na primjer https://dummyjson.com/test). Unatoč, kako se u ovome slučaju ciljani endpoint nalazi unutar istog Node.js okruženja, dovoljno je navesti samo putanju sub-direktorija URL-a „/uploadState“. Nakon primanja odgovora (engl. response), odgovor se pretvara u znakovni niz (eng. string) funkcijom *response.text()*.



Slika 7.4 Prikaz web aplikacije

Postavljanjem *innerHTML-a* kao tijelo odgovora u nastavku lančanog slijeda lambda funkcija (*h1.innerHTML = response*) nastaje željeni vizualni prikaz na web stranici.

7.4 Softversko rješenje na Raspberry Pi-u

Raspberry Pi je odgovoran za detekciju WiFi signala, obrađivanje podataka, skladištenje obrađenih podataka i slanje obrađenih podataka na poslužitelj. Sve funkcionalnosti su enkapsulirane u svoje skripte, koje su upravljane glavnom skriptom za detektiranje uređaja *detectDevices.sh*.

```
#!/bin/sh
bash startDetecting.sh &

touch files/record.txt
touch files/tempRecord.txt
touch files/messageHistory.txt

sleep 5
while true; do


        current_time=$(date '+%F, %T')
        devicesFound=$(python3 utils/extractDevices.py)

        message=$(echo "\n\n$current_time\n$devicesFound")

        shortMessage=$(echo "$devicesFound" | grep -o "Devices found: [0-9]\+")

        send "$message"

        echo "$message" >> files/messageHistory.txt


        bash wfw.sh files/captures/devices-01.csv > files/tempRecord.txt
        cat files/tempRecord.txt >> files/record.txt

        sleep 60


done
```

Slika 7.5 Kod detectDevices.sh skripte

Skripta *startDetecting.sh* odgovorna je za interakciju s WiFi adapterom i osiguravanjem svih preduvjeta za pravilnu detekciju WiFi signala. Dodatni znak & označuje kernelu pozadinsko

izvršavanje skripte *startDetecting.sh.* Dok se skripta *startDetecting.sh* izvršava u pozadini, program se nastavlja i komandom *touch* se provjerava ako postoje datoteke *record.txt, tempRecord.txt i messageHistory.txt.* U slučaju da datoteke ne postoje, program ih stvori s praznim sadržajem. Time je u skripti osigurano čitanje navedenih datoteka bez mogućnosti izvršne greške. Naredbom *sleep* suspendira se program na 5 sekundi. Čekanje od 5 sekundi daje skripti *startDetecting.sh* dovoljno vremenske margine da se pokrene i izvrši svoje početne funkcije. Skripta *startDetecting.sh* svoje rezultate sprema u zasebne datoteke, koje skripta *extractDevices.py* kasnije čita. Nakon kratke prerade završnih podataka, komandom *send* šalje se trenutno očitano stanje gužve na Node.js poslužitelj. Također, skriptom *wfw.sh* obrađuju se podaci koje je stvorila skripta *startDetecting.sh* i nakon čega se rezultat skripte prikvači na kraj datoteke *record.txt, tj. trajnu povijest.*

```
#!/bin/sh

silent="false"

if [ -z "$1" ]; then
        echo "Will wait for user confirmation to start script.\n"
elif [ "$1"="-s" ]; then
        silent="true"

fi


printf "\n"
bash utils/emptyRecords.sh
bash utils/disableDisruptiveServices.sh
printf "\n"
bash utils/installDriver.sh
printf "\n"
bash utils/correctCardMode.sh
printf "\n"
bash utils/reverseActions/enableNetworkManager.sh
printf "\n"

storeCaptureDir="/home/luka/Documents/wifiDetection/files/captures/devices"
```

Slika 7.6 Prvi dio koda startDetecthing.sh skripte (1/4)

Skripta *startDetecting.sh* ima dva načina rada. Običan način rada služi svrsi ručnog pregleda rada programa, pokazuje se ispis Aircrack-ng alata na ekranu i čeka se potvrda korisnika za početak skripte. Drugi način rada ”silent” izvršava kod bez čekanja korisnika i ne ispisuje izlaz skripte Airodump-ng u terminal.

```
#!bin/bash

mainDir="/home/luka/Documents/wifiDetection/files/captures"

files=$(ls $mainDir)

remove_file_type(){
        local filetype="$1"

        if $(echo "$files" | grep -q $filetype); then
                sudo rm $mainDir/*.$filetype
                printf "Removing $filetype files..\n"
        else
                printf "No $filetype files need to be removed.\n"
        fi


}


remove_file_type "csv"
remove_file_type "cap"
remove_file_type "netxml"
remove_file_type "ivs"
```

Slika 7.7 Kod skripte emptyRecords.sh

Alat airodump-ng sprema detektirane WiFi signale u datoteke te ih indeksira sa svakom iteracijom ponovnog pokretanja alata, što rezultira velikom količinom datoteka u datotečnom spremištu (direktorij relativne putanje files/captures). Pogodno je izbrisati generirane datoteke pri pokretanju glavne skripte kako bi se održala urednost radnog okruženja. Funkcija *remove_file_type()* pregleda sve datoteke u direktoriju spremišta, te briše filtrirane datoteke koje

sadržavaju *filetype* u imenu. Funkcija se zove četiri puta, jedanput za svaki tip datoteke koju skripta Airodump-ng generira.

```
#!/bin/bash

printf "\nDisabling disruptive services, will go offline for a moment..."

if  $(systemctl status NetworkManager | grep -q 'inactive') & $(systemctl status wpa_supplicant | grep -q 'inactive'); then

        printf "\nNetworkManager and wpa_supplicant are already inactive.\n"
else
        systemctl stop NetworkManager
        systemctl stop wpa_supplicant
        printf "\nNetwork manager and wpa_supplicant stoped.\n"
fi


if $(systemctl status avahi-daemon.service | grep -q 'enabled;') | $(systemctl status avahi-daemon.socket | grep -q 'enabled;');then
        systemctl disable avahi-daemon.service
        systemctl disable avahi-daemon.socket
else
        printf "avahi-daemon.service and avahi-daemon.socket are already disabled."
fi



if  $(systemctl status avahi-daemon.service | grep -q 'inactive') & $(systemctl status avahi-daemon.service | grep -q 'inactive'); then

        printf "\navahi-daemon.service and avahi-daemon.socket are already inactive.\n"
else
        systemctl stop avahi-daemon.service
        systemctl stop avahi-daemon.socket
        printf "\navahi-daemon.service and avahi-daemon.socket stoped.\n"
fi
```

Slika 7.8 Kod skripte disableDisruptiveServices.sh

Alat airodump-ng, koji se koristi kasnije u skripti *startDetecting.sh,* stavlja WiFi karticu u monitor način rada i mijenja joj WiFi kanale koje sluša (engl. WiFi channels). Za ispravan način rada potrebno mu je isključiti ostale servise, koji upravljaju mrežom (engl. network) na operativnom sustavu. Isključuju se servisi NetworkManager, wpa_supplicant, avahi-deamon.service i avahi-deamon.socket.

```
#!/bin/bash

if $(lsmod | grep -q '88x2bu'); then
        printf "Wifi adapter 88x2bu already installed."
else
        printf "Installing adapter 88x2bu.."
        sudo insmod /home/luka/Downloads/RTL88x2BU-Linux-Driver-master/88x2bu.ko

fi
```

Slika 7.9 Kod skripte installDriver.sh

Za korištenje WiFi kartice, potrebno je imati postavljen odgovarajući driver. Pokazana skripta *installDriver.sh* izvršava naredbu za postavljanje drivera. (Skripta *installDriver.sh* poziva se u skripti *startDetecting.sh*).

```
#!/bin/bash

if $(iw dev wlx2887ba1ff45d info | grep -q 'monitor'); then
        printf "\nMonitor mode on wlx2887ba1ff45d already set."
else
        airmon-ng start wlx2887ba1ff45d
fi
```

Slika 7.10 Kod skripte correctCardMode.sh

Nakon pokrivanja svih preduvjeta, skripta postavlja WiFi karticu na monitor način rada.

```
#!/bin/bash

systemctl start NetworkManager
```

Slika 7.11 Kod skripte enableNetworkManager.sh

Nakon postavljanja kartice na monitor način rada NetworkManager više ne ometa rad skripte Airodump-ng, te se uključuje natrag naredbom *start*.

```
wait_for_user_input(){

        if [ "$silent" = 'false' ]; then
                printf "\nPress enter to start scanning...\n"
                read -r dummy
        fi


}

run_airodump_with_channels(){
        local channels="$1"
        if [ "$silent" = "true" ]; then
                airodump-ng wlx2887ba1ff45d -w $storeCaptureDir -I 15 -c $channels > /dev/null

        else
                airodump-ng wlx2887ba1ff45d -w $storeCaptureDir -I 15 -c $channels

        fi

}
```

Slika 7.12 Drugi dio koda startDetecting.sh skripte (2/4)

Nastavljajući, unutar skripte *startDetecting.sh* definirane su funkcije *wait_for_user_input() i run_airodump_with_channels().* Prva funkcija je pomoćna, u slijedu *startDetecting.sh* skripte na više mjesta se čeka korisnikova potvrda za nastavak programa. Druga funkcija pokreće alat Airodump-ng, postavljajući mu putanju do direktorija di će spremati izlazne datoteke. Druga funkcija prima parametar *channels,* koja sadrži kanale u obliku znakovnog niza

(eng. string), kod kojeg su kanali odvojeni zarezom (ex. ’3,11,36’). Ovisno o aktiviranosti opcije silent, izlaz alata Airodump-ng se ili ispisuje u terminal, ili prosljeđuje na putanju */dev/null*, gdje datoteka *null* odbacuje bilo kakav upis u sebe. Drugim riječima, ispis Airodump-ng skripte bi se zanemarivao.

```
if [ $# -eq 2 ]; then

        channels=$(printf "1,7,11,36,112")

        if [ "$1" = "-c" ]; then
                channels="$2"
                printf "\nUsing given channels $channels."
        else
                printf "\nOptional parameter -c called but no channel list given (ex. usage '-c 1,6,11)'."
                printf "Defaulting to channels 1, 7, 11, 36 and 112."
        fi

        wait_for_user_input

        run_airodump_with_channels $channels
```

Slika 7.13 Treći dio koda startDetecting.sh skripte (3/4)

U nastavku programa, skripta provjerava ako je korisnik unio dva dodatna parametra (ex. *bash startDetecting.sh -c 3,7,12*). Kao opcionalan način rada, skripti se može ručno unijeti koje WiFi kanale da pregledava. U slučaju da je korisnik skripte dodao opciju –c, ali nije naveo kanale, uzimaju se najčešće korišteni kanali WiFi usmjerivača 1, 7, 11, 36 i 112. (Za cilj rada nije bitno hvatati signale WiFi usmjerivača, ali dok uređaji pokušavaju komunicirati s usmjerivačima (engl. routers), prilagođavaju se kanalu tog usmjerivača, time većina signala poslana iz WiFi uređaja su zapravo na navedenim kanalima usmjerivača). Skripta zatim čeka korisnika ako opcija silent nije uključena, i poziva funkciju *run_airodump_with_channels* s kanalima definiranim u varijabli *channels*.

```
else

        cachedFreqUsed=$(cat files/frequentChannels.txt)

        if [ -z "$cachedFreqUsed" ]; then

                printf "Scanning on all 2.4Ghz and 5Ghz channels.\n"

                wait_for_user_input
                if [ "$silent" = "true" ]; then
                        airodump-ng wlx2887ba1ff45d -w $storeCaptureDir -I 15 -b bga > /dev/null &
                else
                        airodump-ng wlx2887ba1ff45d -w $storeCaptureDir -I 15 -b bga &
                fi
                airoPid=$!

                sleep 12 # Time given to check listen on all channels and capture the ones where wifi communication is happening
                kill $airoPid

                python3 utils/extractChannels.py > files/frequentChannels.txt

                bash startDetecting.sh
        else

                printf "Using cached frequently seen channels $cachedFreqUsed.\n"

                wait_for_user_input

                run_airodump_with_channels $cachedFreqUsed
        fi
fi
```

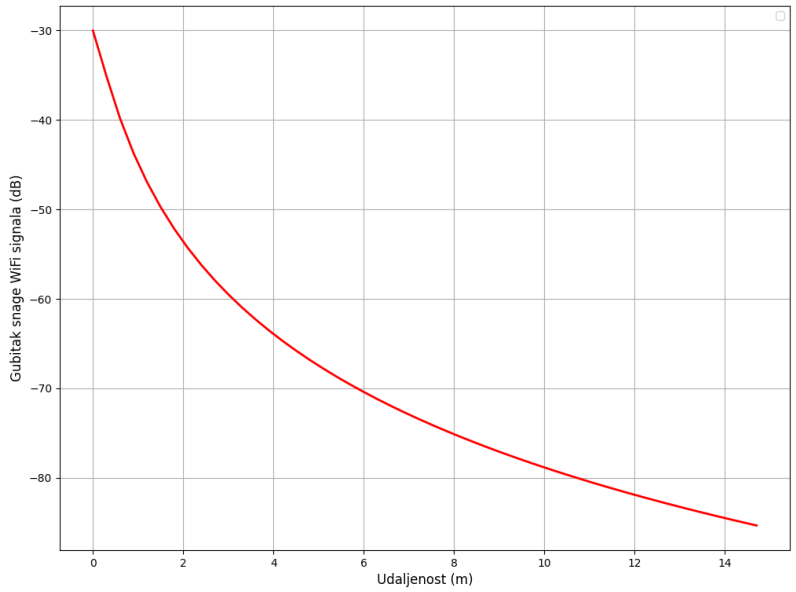
Slika 7.14 Četvrti dio koda startDetecting.sh skripte (4/4)

U slučaju da korisnik skripte nije upisao dodatne parametre, skripta provjerava ako postoje spremljeni podaci o najčešće viđenim kanalima. Ako ne postoji, skeniraju se svi kanali opcijom „–b bga“ (--bands), koja provjerava sve 2.4Ghz kanale („b“ i „a“) i sve 5Ghz kanale („g“). Poziva se skripta Airodump-ng (*run_airodump_with_channels se ne koristi* jer se u ovome slučaju zove dodatna opcija –b, umjesto –c za kanale). Naredbom *airoPid=$!* se dohvaća procesni identifikacijski broj pokrenute Airodump-ng skripte, kako bi se kasnije mogla ugasiti (eng. kill). Nakon perioda čekanja, skripta ugasi Airodump-ng proces, te provjerava na kojim je sve kanalima u tom vremenskom roku adapter detektirao WiFi signale i spremi ih u datoteku *frequentChannels.txt.* Nakon toga, skripta rekurzivno pozove samu sebe (jedanput), ali pri toj će iteraciji datoteka *frequentChannels.txt* imati sadržaj, te će se izvršiti drugi dio IF provjere, gdje će skripta Airodump-ng biti aktivna dok se ne ugasi Raspberry Pi ili ubije proces.

8. TESTIRANJE

S dovršenim postavljanjem Raspberry Pi-a, web aplikacije i API-a, moguće je pokrenuti sve dijelove projekta i prikupiti rezultate. Pri uključivanju terenskog uređaja u struju, on se automatski spaja na namještenu internetsku vezu u blizini, te se spaja na Ngrok servis kako bi omogućio SSH, i pokreće skriptu *detectDevices.sh*.

Na terminalu SSH-a, moguće je vidjeti trenutna očitavanja uređaja i pristupnih točaka (rutera) i njihove poslane pakete. Uz poslane pakete je zapisana i zadnja detektirana snaga signala za pojedini uređaj. Uzimajući pametni telefon za testiranje i pomicanjem u prostoriji, vidi se jasna povezanost snage signala i udaljenosti telefona od terenskog uređaja.



Slika 8.1 Korelacija snage WiFi signala i udaljenosti telefona

S testiranim telefonom, najjači očitani signal, kad bi bio postavljen blizu terenskog uređaja (<0.3m), je imao vrijednost od -30dBm. Daljnjim udaljavanjem signal oslabljuje relativno linearno od -30dBm do -60dBm, za udaljenost od 0.3m do 5m. Daljnje udaljavanje je imalo sve manji utjecaj na slabljenje signala, čak i kroz zidove i objekte. Signalom s najnižom vrijednosti od -90dBm označio je krajnji domet WiFi adaptera, od 15-20m (uključujući nekoliko zidova kao prepreke između telefona i uređaja). (Slika 8.1 Korelacija snage WiFi signala i udaljenosti telefona)

Terenski uređaj je konzistentno detektirao poslane pakete na svim kanalima (3, 11, 36), što pokriva i 2.4Ghz i 5Ghz signale. Najveći uzrok propusta signala je ne slušanje točnog kanala u trenutku prolaska signala, što je neizbježno jer WiFi adapter tipično može slušati samo na jednom kanalu u trenutku vremena.

9. ZAKLJUČAK

Gledajući rezultate testiranja završnog rješenja ovoga rada, vidi se da bi odabran pristup problemu dovoljno precizno detektirao osobne pametne telefone ljudi koji čekaju u red. Postavljanjem terenskog uređaja u blizinu središta reda, i filtriranjem signala po snazi, mogle bi se izbjeći detekcije drugih telefona u prostoriji, te bi procjena stanja gužve u menzi bila adekvatno precizna za svrhe ovoga rada. Terenski uređaj u pravom vremenu šalje podatke na poslužitelj, iz kojeg web aplikacija povlači iste podatke i prikazuje ih korisniku.

LITERATURA

[1] Infrared Sensors and PIR Sensors Breakdown, [Mrežno]. Dostupno na: https://www.getkisi.com/guides/infrared-sensors.

[2] Raspberry Pi products website, [Mrežno]. Dostupno na: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

[3] Geeks For Geeks, [Mrežno]. Dostupno na: https://www.geeksforgeeks.org/chaining-commands-in-linux/.

[4] Wikipedia, 5 9 2024. [Mrežno]. Dostupno na: https://en.wikipedia.org/wiki/Ubuntu_version_history.

[5] Raspberry Pi website, [Mrežno]. Dostupno na: https://www.raspberrypi.com/software/.

[6] TP-link driver download service, [Mrežno]. Dostupno na: https://www.tp-link.com/us/support/download/archer-t4u/.

[7] B. Dhiman, Medium.com, [Mrežno]. Dostupno na: https://bhavyadhiman7.medium.com/how-to-build-custom-commands-in-linux-macos-without-navigating-to-a-specific-directory-19dbccb19ae1.

[8] Node.js website, [Mrežno]. Dostupno na: https://nodejs.org/en.

SAŽETAK

U ovome radu istraživali su se pristupi rješavanja problema redovitog dugog čekanja u redu u studentskim menzama. Definirale su se važne značajke problema, a najvažnija značajke je karakteristika kaotičnog kretanja gužve u čak i malim vremenskim periodima u danu. Dovelo se do zaključka da je zbog karakteristike subjektivnog iskustva tijekom čekanja bolje kategorizirati stanja gužve, nego gledati ih kao na kontinuiranu vrijednost. Stanje gužve u menzi podijelile su se u tri kategorije; kratak red, srednji red i dugačak red. Uz postignute zaključke odabrao se pristup detektiranja gužve čitanjem WiFi signala. Kako svaki korisnik menze sa sobom nosi osobni pametni telefon, detektiranjem poslanih WiFi signala i sakupljanjem jedinstvenih MAC adresa dolazi se do broja ljudi u menzi. Postavljanjem uređaja koji detektira WiFi signale u blizini reda čekanja u menzi i filtriranjem pronađenih uređaja po jačini WiFi signala (tj. po lokacijskoj udaljenosti), broje se isključivo ljudi koji čekaju u redu. Tehničkom izvedbom i testiranjem dovelo se do zaključka da ovaj pristup problemu pruža adekvatnu točnost za potrebe ovoga rada. Uspostavljanjem poslužitelja i web sučelja korisniku se pruža „glatko“ iskustvo pregleda trenutnog stanja gužve u menzi, te polaskom u menzu isključivo tijekom male gužve korisniku se znatno smanjuje vrijeme čekanja u redu.

Ključne riječi: Menza, analiza WiFi mreža, Raspberry Pi, driveri, C, bash, Aircrack-ng, Ngrok, Node.js, Ubuntu, SSH, REST API, curl, Python

ABSTRACT

This thesis investigates different approaches to solving the problem of regular long queue times in student restaurants. The problem's important features were defined, and the most important feature is the characteristic of the chaotic occupancy pattern of the queue, even in relatively small periods. It was concluded that due to the nature of the subjective experience during wait times, it is better to categorize queue lengths, rather than to see them as a continuous value. Queue length was divided into three categories; short, medium and long. With the reached conclusions, the approach of capturing WiFi signals to detect crowds was chosen. As each restaurant user carries a personal smartphone, the number of people in the restaurant can be obtained by capturing WiFi packets and counting unique MAC addresses. By placing the capture device that detects WiFi signals near the restaurant queue, and filtering the detected devices by WiFi signal strength (ie by distance), only people waiting in the queue are counted. Developing and testing the chosen approach, it was concluded that it provides adequate accuracy for the needs of this thesis. By establishing a server and a web interface, the user can check the current state of the queue in the restaurant. By going to the restaurant only during short queues, the user's waiting time in line is significantly reduced.

Keywords: Student restaurant, WiFi network analysis, Raspberry Pi, drivers, C, bash, Aircrack-ng, Ngrok, Node.js, Ubuntu, SSH, REST API, curl, Python