# Aplikacija za analizu mrežnog prometa

**Posarić, Lovro**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Lovro Posarić**

# Aplikacija za analizu mrežnog prometa

**ZAVRŠNI RAD**

**Varaždin, 2021.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Lovro Posarić**

**JMBAG: 0016136124**

**Studij: Informacijski sustavi**

# Aplikacija za analizu mrežnog prometa

**ZAVRŠNI RAD**

**Mentor:**

Izv. prof. dr. sc. Ivan Magdalenić

**Varaždin, lipanj 2021.**

*Lovro Posarić*

**Izjava o izvornosti**

Izjavljujem da je moj završni rad izvorni rezultat mojega rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada korištene su etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

_____

**Sažetak**

Tema ovog završnog rada je izrada aplikacije za analizu bežičnog mrežnog prometa na drugom sloju 802.11 bežičnog protokola. U prvome dijelu rada opisana je teorijska podloga 802.11 protokola te tri glavne vrste zaglavlja koje nalazimo u radu s 802.11 okvirima dohvaćenim adapterom, koji je postavljen u monitorski način rada. U drugome dijelu opisano je programsko postavljanje bežičnog adaptera u monitorski način, dohvaćanje i obrada individualnih okvira, određivanje njihove vrste i podvrste, zajedno s određivanjem čvorova kroz koje je prošao okvir te jačine signala između bežičnog adaptera i tih čvorova. Na kraju je objašnjeno na koji su način rezultati obrade prikazani u samoj aplikaciji u obliku tablica i grafova.

**Ključne riječi:** analiza mreže, Wi-Fi; 802.11 protokol; paket; okvir; monitorski način rada; adapter; statis

# Sadržaj

# 1. Uvod

WiFi adapteri mogu funkcionirati u četiri glavna moda, zajedno s nekim vlasničkim modovima (npr. Mikortik Nstreme). **Master Mode** (još zvan AP ili infrastrukturni mod) koristi se za stvaranje servisa koji izgleda kao tradicionalna pristupna točka te adapter stvara mrežu sa svojim SSID-om i nudi sve povezane servise. Adapteri u master modu mogu komunicirati samo s onima koji su u **Managed** načinu, u kojem se adapter ponaša kao klijent, te ne može komunicirati s drugim koji su u tom modu direktno, već samo preko adaptera u Master načinu s kojim su oba povezana. [1] U Ad-hoc načinu stvara se multipoint-to-multipoint mreža u kojoj ne postoji ni *master* čvor ni zaseban AP, već klijenti komuniciraju direktno sa svojim susjedima.

Posljednji je onaj mod koji nam omogućuje dohvaćanje bežičnog prometa sa svih dostupnih čvorova u blizini, a to je način rada zvan monitorski način (engl. monitor mode). Iako zbog nedostatka autentifikacije za određenu bežičnu mrežu ne omogućuje da čitamo konkretne podatke koji se prenose mrežom, svejedno nam daje mogućnost da očitamo tehničke podatke s podatkovnog sloja te na temelju tih podataka provedemo analizu mreža u čijem smo rasponu.

Monitorski način sa sobom nosi nekoliko problema. [3] Nije definiran ni u IEEE802.11 ni u Wi-Fi standardima. Uz to, trebaju mu upravljački programi te firmware napisan specifično za određeni mrežni kontroler. Nije neobično da proizvođači mijenjaju čipove koje koriste u proizvodu istog naziva. Tako, na primjer, TP-Link je u svojem adapter TL-WN722N iz v1 verziji s čipom Atheros AR9002 zamijenio s Realtek RTL8188 mrežnim kontrolerom u verzijama 2 i 3, a koji, osim posebnih drivera, zahtijeva i posebnu konfiguraciju kako bi se mogli postaviti u monitorski način [3], za razliku od svog prethodnika koji se u monitorski način postavlja kao i svaki drugi adapter koji ga podržava.

# 2. Opis korištenih tehnologija

## 2.1. Parrot Security operacijski sustav

Za izradu sam koristio Parrot Security, izdanje ParrotOS-a namijenjeno sigurnosti i penetracijskom testiranju. ParrotOS jest lightweight distribucija Linuxa bazirana na Debianu, čiji je glavni fokus na privatnosti, što ga čini sličnim Kali Linuxu, najpopularnijem OS-u s naglaskom na kibernetičkoj sigurnosti. ParrotSec dolazi instaliran s mnoštvom alata za testiranje sigurnosnih ranjivosti i mrežnu analizu, uključno s aircrack-ng programskim paketom koji je jedan od popularnijih alata za testiranje bežičnih mreža. Među skriptama sadržanim unutar kao dio aircrack-ng-a jest i airmon-ng, odnosno skripta za postavljanje mrežnog adaptera u monitorski način rada, te nam ona može poslužiti kod testiranja.

Slika 1: ParrotOS logo [8]

## 2.2. Alfa AWUS036ACS bežično mrežno sučelje

U tu sam svrhu odabrao Alfa AWUS036ACS, prilično jednostavan adapter koji podržava i uobičajene 2.4GHz mreže, zajedno s 5GHz mrežama. Adapter je gotovo plug-and-play, osim što se moraju instalirati driveri za njegov mrežni kontroler RTL8812 uz pomoć naredbe **apt-get install -y realtek-rtl88xxau-dkms**.

Slika 2: Alfa AWUS036ACS Wi-Fi adapter [9]

Kako bismo ga postavili u monitorski mod, prvo uz pomoć iwconfig naredbe dohvaćamo popis svih trenutno dostupnih bežičnih sučelja. U mom slučaju, naziv sučelja je wlan0, stoga da ga postavim u monitorski način, koristim sljedeće tri naredbe:

```
ip link set wlan0 down
iw wlan0 set monitor control
ip link set wlan0 up
```

Sve naredbe, dakako, trebaju se pokrenuti sa superuser dopuštenjima. Prva naredba gasi sučelje, druga ga postavlja u monitorski način, a treća ga ponovo pali. Ako ponovo pokrenemo iw dev naredbu, vidjet ćemo kako je sada polje type promijenjeno iz managed u monitor. Sličan rezultat možemo postići i uz pomoć: **iwconfig wlan0 | grep -i mode | awk '{print $4}'**

## 2.3. Qt Framework i Qt Creator

Za izradu aplikacije koristio sam C++ jezik zajedno sa Qt Framework za izradu grafičkog sučelja. Specifičnost Qt Frameworka jest da omogućava razvoj cross-platform aplikacija koje se mogu pokretati na raznim sustavima bez ili s malim promjenama u samom kodu, sve dok su to istovremeno nativne aplikacije s nativnim mogućnostima i brzinom izvođenja koja nije uvjetovana međuslojevima kao kod nekih drugih cross-platform okvira.

Qt podržava izradu korisničkih sučelja uz pomoć WYSIWYG alata, još jedan način jest korištenje QML jezika i modula Qt Quick koji pruža sve osnovne tipove potrebne za izradu sučelja QML-om. QML je baziran na Javascriptu koji ga koristi da bi omogućio prodecuralno programiranje te je u temeljima zapravo deklarativan jezik za opis objekata poput XAML-a koji se koristi za izradu grafičkih sučelja baziranih na .Net razvojnom okviru. QML može biti korišten kao jedini jezik pri izradi Qt aplikacija, ali se najčešće kombinira sa C++-om za izradu samog backenda aplikacije.

Iako se mogu razvijati uz pomoć drugih sustava poput GNU Make, Visual Studio-a te Xcode-a, Qt nudi i svoje integrirano razvojno okruženje zvano **Qt Creator** koji nudi sve uobičajene alate za razvoj aplikacija. Uključno s editorom koda koji podržava C++/QML i EMAscript, označavanje sintakse, završavanje koda, statičku provjeru koda, vizualni debugger, itd.

# 3. Podatkovni sloj 802.11 protokola

U monitorskom smo načinu rada ograničeni na rad s paketima koji dolaze s drugog sloja bežičnog sloja, pa prvo moramo analizirati kako oni izgledaju na razini podatkovnog sloja.

## 3.1. Vrste 802.11 okvira

Okviri bežične mreže dijele se u nekoliko većih kategorija, a to su: **Management, Control, Data i Extension**. Od njih samo Data okviri sadrže podatke iz viših mrežnih slojeva, dok se ostali koriste za upravljanje prijenosom podataka. Najvažnije vrste bit će opisane prema [4].

### 3.1.1. Management okviri

Management okviri omogućuju bežičnim stanicama da se ostvare i održe međusobnu komunikaciju, odnosno podržavaju autentifikaciju, asocijaciju te sinkronizaciju.

Authentication okviri koriste se u 802.11 autenfikaciji koja započinje na način da kontroler bežičnog sučelja (eng. wireless network interface controller, WNIC) šalje autenthication okvir koji sadrži njegov identitet prema AP-u. WNIC šalje samo jedan autethication okvir na koji AP odgovara sa svojim authentication okvirom kojim indicira prihvaćanje ili odbijanje.

Kod autentifikacije temeljene na dijeljenom ključu, nakon što WNIC pošalje inicijalan zahtjev, odgovor AP-a sadrži challenge text. WNIC na to odgovara s kriptiranim challenge textom koji vraća AP-u. AP provjerava je li tekst validan na način da dekriptira te, ovisno o valjanosti, autentificira sučelje ili ga odbija.

Kako bi mobilni čvorovi mogli aktivno tražiti postojeće mreže u svom području, a da ne čekaju da svoje postojanje objave pristupne točke, koriste se paketi zvani **probe request**. Čvor specificira ili SSID specifične pristupne točke ili šalje zahtjev na sve AP-ove uz pomoć sveodredišnog SSID-a.

Na **probe request** odgovara se pomoću probe response paketa. Probe response sadrži sve parametre koji se nalaze i u Beacon okvirima, što omogućuje da se mobilna stanica uskladi i poveže na mrežu.

Pristupne točke s vremena na vrijeme šalju **Beacon** okvire kojima je svrha da objave svoju prisutnost među čvorovima koji su u njenom rasponu, te uključuju parametre potrebne

za povezivanje čvorova na njih. Beaconi sadrže informacije o BSS parametrima i okvire spremljene u međuspremnicima pristupnih točaka.

Čvorovi šalju **association request** okvir prema pristupnim točkama kako bi zahtijevali uključivanje u BSS. U tom okviru čvor uključuje sve svoje mogućnosti spajanja, a od njih se mogu pojaviti samo one koje je AP prethodno objavio u beacon ili probe response okviru. Odgovor na ovakav zahtjev jest **association response** okvir koji uključuje association ID, koji je unikatan za svaki čvor unutar BSS-a.

U slučaju kada je potrebna mobilnost između AP-ova, čvorovi šalju paket sličan asocijacijskom, **reassociation request**. Glavna razlika je u tome što će u reassociation zahtjevu biti sadržan i naziv AP-a na koji je trenutačno čvor povezan. U slučaju da ne dobije povratni odgovor na request (bilo to zbog load balancinga ili čega drugog), čvor će ostati povezan na trenutačni AP te dalje tražiti AP-ove na koje se može spojiti.

### 3.1.2. Control okviri

**Control okviri** omogućuju razmjenu podatkovnih okvira između stanica. Među njima važni su:

**Acknowledgement (ACK) okvir** šalje stanica primatelj kao potvrdu nakon primanja prema stanici koja je poslala podatke. U slučaju da pošiljatelj ne primi ACK unutar predodređenog vremenskog perioda, okvir će ponovno biti poslan.

**Request to Send (RTS) okviri** pružaju uz CTS opcionalnu shemu redukcije kolizija za AP-ove koji se susreću s problemom skrivenog terminala. Stanica šalje RTS okvir kao prvi korak u two-way rukovanju potrebnom prije slanja podatkovnih okvira.

**Clear to Send (CTS) okviri** predstavljaju odgovor na RTS okvir. Oni daju odobrenje da stanica koja ga zahtijeva pošalje podatkovni okvir. Način na koji CTS pruža kontrolu kolizija jest da u sebi sadrži vremensku vrijednost u kojoj sve ostale stanice moraju prestati s transmisijom dok zahtijevajuća stanica prenosi podatke.

### 3.1.3. Data okviri

Većina 802.11 data okvira prenosi stvarne podatke koji se kasnije prosljeđuju protokolima viših slojeva, a ti su podatci obično kriptirani iz sigurnosnih razloga. Ti se podatci još zovu i MAC Service Data Unit (MSDU). Kada to ne bi bio slučaj, svatko bi u monitorskom načinu rada mogao bez problema čitati sve podatke koji teku kroz mreže u njegovom okruženju, a da uopće nije spojen ni na jednu od njih. Postoje i data okviri koji sadrže MSDU payload, a nisu kriptirani, te se označuju s **no data.**

Integracijski servis koji se nalazi unutar AP-a te WLAN kontroleri uzimaju MSDU payload i pretvaraju ga u 802.3 Ethernet okvire.

**Null function** okviri ponekad se koriste da klijentske stanice informiraju AP o promjenama u statusu uštede energije. [6]

## 3.2. Struktura 802.11 okvira

U slučaju 802.11 bežičnih protokola, okvir drugog sloja sastoji se od zajedničkih polja (onih koja su prisutna u svakoj vrsti 802.11 MAC okvira) te specifičnih polja (koja se pojavljuju ovisno o tipu i podtipu određenog okvira).

**Općeniti 802.11 okvir** prikazan je u tablici ispod:

| 2B | 2B | 6B | 6B | 6B | 0 ili 2B | 6B | 0 ili 2B | 0 ili 4B | Varijabilna dužina | 4B |
|---|---|---|---|---|---|---|---|---|---|---|
| Frame Control | Duration/ID | Address 1 | Address 2 | Address 3 | Sequence Control | Address 4 | QoS Control | HT Control | Frame Body | FCS |

Tablica 1: 802.11 okvir (izvor: Spencer, 2020.)

**Duration/ID** sadrži vrijednost koja pokazuje koliko je vremena medij zauzet u μs.

Četiri adrese sadržane u zaglavlju označavaju redom: izvor okvira **SA** (čvor koji je prvi poslao okvir), odredište okvira **DA** (krajnji cilj okvira), prenositelja okvira **TA** (čvor koji je neposredno poslao ovaj okvir), primatelja okvira **RA** (sljedeći čvor preko kojeg se prenosi okvir). Samo jedno od tih polja je obavezno, **Address 1**. Tako, na primjer, CTS okvir ima samo jednu adresu. Koliko će ih biti iskorišteno, ovisi o vrsti okvira:

| | **Control Frame** | **Management Frame** | **Data Frame** |
|---|---|---|---|
| **Address 1** | RA | RA | RA |
| **Address 2** | TA (ne svi) | TA | TA |
| **Address 3** | - | BSSID | BSSID ili SA ili DA |
| **Address 4** | - | - | BSSID ili SA |

Tablica 2: Uloga adresnih polja u okviru ovisno o vrsti (izvor: Spencer, 2020.)

Primjer:

| **CTS okvir** | | | |
|---|---|---|---|
| Frame Control | Duration | RA | CRC |

Tablica 3: Polja u CTS okviru (izvor: Spencer, 2020.)

| RTS okvir | | | | |
|---|---|---|---|---|
| Frame Control | Duration | RA | TA | CRC |

Tablica 4: Polja u RTS okviru (izvor: Spencer, 2020.)

**Sequence Control** sastoji se od dva podpolja, **Sequence Number** (12 bitova) i **Fragment number** (4 bita). Koristi se za filtriranje dupliciranih okvira.

Sam **frame body** sadrži podatke iz trećeg do sedmog sloja koji su enkapsulirani te vrlo često kriptirani, a njihova se veličina razlikuje ovisno o tipu sadržaja koji se prenosi u tom paketu.

Na samom kraju okvira nalazi se frame check sequence, odnosno 32-bitna ciklična redundancijska vrijednost koja se koristi za provjeru je li sadržaj cijelog okvira bio namjerno mijenjan ili bio nenamjerno pokvaren tijekom prijenosa kroz bežični medij. Sve vrijednosti iz zaglavlja okvira i njegovog tijela koriste se u kalkulaciji te se rezultat sprema u FCS polje. Ako primatelj ne dobije isti FCS nakon što provede kalkulaciju, on zaključuje da je paket oštećen, odbacuje ga te neće na njega odgovoriti ACK okvirom. Prema tome, pošiljatelj zna da treba ponovo poslati paket jer nije dobio potvrdu. Često će takve okvire odbaciti već AP prije nego se pošalju do operacijskog sustava, pa stoga uobičajeno nisu niti vidljivi u alatima za analizu prometa. [5]

**Frame control** okvir prikazan jer u tablici ispod:

| B0..B1 | B2..B3 | B4..B7 | B8 | B9 | B10 | B11 | B12 | B13 | B14 | B15 |
|---|---|---|---|---|---|---|---|---|---|---|
| Protocol Version | Type | Subtype | To DS | From DS | More Fragments | Retry | Power Mgmt | More Data | Protected Frame | +HTC/Order |

Tablica 5: Prikaz bitova unutar Frame control polja (izvor: Spencer, 2020.)

**Protocol version** uobičajeno je postavljen na 0, neovisno o kojoj je verziji 802.11 protokola riječ. Tipovi i podtipovi prikazani su u tablici X ispod, zajedno s nazivom vrste paketa koji označavaju.

**To DS/From DS** označava je li izvor, odnosno destinacija bežičnog paketa bežični distribucijski sustav, odnosno mreža koja povezuje više pristupnih točaka, a da one nisu međusobno povezane fizičkom poveznicom.

**More Fragments** bit postavljen je na 0 u slučaju da je trenutačni okvir zadnji (ili jedini) između dijelova većeg okvira, koji je namjerno podijeljen u segmente nazvane fragmentima kako bi se smanjila vjerojatnost događanja grešaka pri prijenosu.

**Retry bit** je postavljen na 1 ako se Data ili Management okvir koristi za retransmisiju prethodno poslanog okvira, dok se kod Control okvira koristi za drukčiju svrhu.

**Power Management** označava hoće li STA otići u sleep stanje (ako je Pwr Mgmt na 1).

**More Data** – samo korišten u managament i data okvirima, uvijek je postavljen na 0 kod control okvira.

**Protected Frame** – koristi li se enkripcija; control okviri nikada nisu zaštićeni, pa je za njih Protected Frame uvijek postavljen na 0.

**Order bit** – kontrolni okviri koriste se kao dio atomičnih operacija zamjene okvira te stoga ne mogu biti slane izvan poretka; ovaj je bit uvijek postavljen na 0.

| **Tip (bit 3 i bit 2 )** | **Podtip (bitovi 7 do 4)** | **Vrsta paketa** |
|---|---|---|
| 00 | 0000 | Association Request |
| 00 | 0001 | Association Response |
| 00 | 0010 | Reassociation Request |
| 00 | 0011 | Reassociation Response |
| 00 | 0100 | Probe Request |
| 00 | 0101 | Probe Response |
| 00 | 0110 | Timing Advertisement |
| 00 | 0111 | Reserved |
| 00 | 1000 | Beacon |
| 00 | 1001 | ATIM |
| 00 | 1010 | Disassociation |
| 00 | 1011 | Authentication |
| 00 | 1100 | Deauthentication |
| 00 | 1101 | Action |
| 00 | 1110 | Action No Ack (NACK) |
| 00 | 1111 | Reserved |
| 01 | 0000-0001 | Reserved |
| 01 | 0010 | Trigger |
| 01 | 0011 | TACK |
| 01 | 0100 | Beamforming Report Poll |
| 01 | 0101 | VHT/HE NDP Announcement |
| 01 | 0110 | Control Frame Extension |
| 01 | 0111 | Control Wrapper |
| 01 | 1000 | Block Ack Request (BAR) |
| 01 | 1001 | Block Ack (BA) |
| 01 | 1010 | PS-Poll |
| 01 | 1011 | RTS |
| 01 | 1100 | CTS |
| 01 | 1101 | ACK |
| 01 | 1110 | CF-End |
| 01 | 1111 | CF-End + CF-ACK |
| 10 | 0000 | Data |
| 10 | 0001 | Data + CF-ACK |
| 10 | 0010 | Data + CF-Poll |
| 10 | 0011 | Data + CF-ACK + CF-Poll |
| 10 | 0100 | Null (no dana) |
| 10 | 0101 | CF-ACK (no data) |
| 10 | 0110 | CF-Poll (no data) |
| 10 | 0111 | CF-ACK + CF-Poll (no data) |

| | | |
|---|---|---|
| 10 | 1000 | QoS Data |
| 10 | 1001 | QoS Data + CF-ACK |
| 10 | 1010 | QoS Data + CF-Poll |
| 10 | 1011 | QoS Data + CF-ACK + CF-Poll |
| 10 | 1100 | QoS Null (no data) |
| 10 | 1101 | Reserved |
| 10 | 1110 | QoS CF-Poll (no data) |
| 10 | 1111 | QoS CF-ACK + CF-Poll (no data) |
| 11 | 0000 | DMG Beacon |
| 11 | 0001 | S1G Beacon |
| 11 | 0010-1111 | Reserved |

Tablica 6: Vrste 802.11 okvira (izvor: Pindoria, 2017.)

Za određivanje vrste paketa koristimo prvi bajt IEE 802.2 zaglavlja. Nakon što smo primili poruku, taj se bajt neće nalaziti odmah na početku paketa jer se prije samog IEE zaglavlja nalazi Radiotap zaglavlje. RadioTap zaglavlja pojavljuju se jedino ako je mrežni adapter namješten u monitorski način rada. Oni nisu dio 802.11 standarda već služe za pružanje dodatni informacija o zahvaćenim paketima. Broj polja nije striktno određen nego, ovisno u implementaciji u samom driveru, može biti više ili manje polja u samom paketu koji smo zahvatili te se zastavice o prisutnosti pojedinog polja nalaze u polju Present. Među njima se nalaze i podatci o jačini signala ili o kanalu koji se koristi za komunikaciju, koji nisu inače sadržani u samom 802.11 zaglavlju.

Radio tap zaglavlje sadrži sljedeća polja: reviziju, padding, duljinu, Present zastavice, MAC vremensku oznaku, Flags zastavice, Data Rate, Channel Frequency, Channel type, SSI Signal, SSI Noise, Antenna te SSI Signal.

Među njima ističu se sljedeća polja:

**Channel frequency** -Tx/Rx frekvencija u MHz

**Channel type** – predstavlja matricu bitova koja označava razna svojstva kanala na kojem se prenose 802.11 okviri

- Turbo – je li kanal proširene duljine (40 Mhz umjesto uobičajenih 20 Mhz)
- CCK – koristi li se CCK shema
- OFDM – koristi li se OFDM shema
- 2 Ghz spectrum
- 5 Ghz spectrum
- Passive – je li dozvoljeno samo pasivno skeniranje
- Dynamic CCK-OFDM – koristi li se dinamička izmjena između CCK i OFDM
- GFSK – koristi li se GFSK shema
- GSM

- Half rate channel – koristi li se kanal u pola normalne širine, tj. 10 Mhz kanal
- Quartare rate channel – koristi li se kanal u četvrtini normalne širine (5 Mhz)

**Rate** – Tx/Rx omjer

**Antenna Signal** – snaga radiofrekvencijskog signala na anteni u dBm
**Antenna Noise** – radiofrekvencijski šum na anteni u dBm

**Zastavice** – svojstva primljenih i poslanih okvira

- CFP – poslano/primljeno tijekom Contention Free Period
- Preamble – poslano/primljeno s dodatkom za pomoć u provjeri redundancije
- WEP – poslano/primljeno s WEP sigurnosnim algoritmom
- Fragmentation – poslano/primljeno sa fragmentacijom
- FCS – uključuje li okvir FCS
- Data Pad – uključuje li okvir dodatak za popunjavanje između 802.11 zaglavlja i payloada do 32-bitne granice
- Bad FCS – je li okvir pao na FCS testu

# 4. Aplikacija za analizu mrežnog prometa

Kao što sam gore napomenuo, aplikaciju sam razvio kao stolnu aplikaciju baziranu na Qt Frameworku; uz to koristio sam QtCharts za prikaz grafova te neke od osnovnih Linux-ovih biblioteka za interakciju s mrežnim adapterom, postavljanje sirovog utikača te obradu podataka dobivenih s utikača.

## 4.1. Dobivanje popisa dostupnih bežičnih sučelja

Programski, iako bismo mogli interpretirati rezultate iwconfig-a, Linux-ova jezgra nudi nam **wireless.h** zaglavlje zajedno s funkcijom **getifaddrs()** iz ifaddrs.h, koja vraća pokazivač na početak vezane liste elemenata **ifaddrs**, odnosno vezane liste svih mrežnih sučelja.

Kako bismo dobili naziv bežičnog protokola, koristimo funkciju **ioctl**(socket, SIOCGIWNAME, &pwrq), gdje je SIOCGIWNAME vrijednost iz wireless zaglavlja koje označava naziv za bežični protokol. Vidimo također da moramo stvoriti TCP socket uz pomoć **socket**(AF_INET, SOCK_STREAM, 0) prije nego pozovemo ioctl. Ako ioctl s tim argumentima nije vratio -1, znamo da je riječ o bežičnom sučelju.

## 4.2. Postavljanje sučelja u monitorski način rada

U monitorski način rada možemo postaviti sučelje i uz pomoć ioctl naredbe, kao što bi to izvela iwconfig sa sljedećim argumentima: **naziv_sucelja mode monitor**

```
bool PostaviUMonitorskiNacin(QString uredaj){
     UgasiSucelje(uredaj);
     bool uspjeh = Postavi(uredaj);
     UpaliSucelje(uredaj);

     return uspjeh;
 }
```

Kao što vidimo iz gore prikazane funkcije, prije nego što možemo postaviti monitorski način, moramo ugasiti sučelje. Isto kao što i uz pomoć ioctl naredbe mijenjamo modove u kojima radi sučelje, putem ioctl-a mijenjamo zastavice da bismo ugasili sučelje. Najlakši način za to je da negiramo postojeće zastavice koje su trenutačno postavljene, što je prikazano ispod u obliku PostaviZastavice funkcije zajedno s UzmiZastavice funkcijom, koja vraća cijeli broj koji prikazuje koje su zastavice postavljene.

```
    void UgasiSucelje(QString uredaj){
        int zastavice;
        UzmiZastavice(uredaj, &zastavice);

        if(zastavice != 0x1003)
            return;

        qDebug() << "Gasim sucelje, zastavice: " << zastavice;
        PostaviZastavice(uredaj, ~zastavice);
    }



    int UzmiZastavice(QString naziv, int* flags){
        int sockfd = -1;

        struct ifreq ifr;
     strncpy(ifr.ifr_ifrn.ifrn_name, naziv.toStdString().
c_str(), IFNAMSIZ);

        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
            return false;
        }

        if(ioctl(sockfd, SIOCGIFFLAGS, (caddr_t) &ifr) != -1){
            *flags = ifr.ifr_ifru.ifru_flags & 0xffff;
        }

        close(sockfd);
        return true;
    }
```

Postavljanje zastavica je vrlo slično uzimanju njihove vrijednosti, osim što ih moramo postaviti unutar ifreq strukture prije nego je pošaljemo kao argument u ioctl funkciju.

```
    static int PostaviZastavice(QString naziv, int flags){
        int sockfd = -1;

        struct ifreq ifr;
     strncpy(ifr.ifr_ifrn.ifrn_name, naziv .toStdString()
.c_str(), IFNAMSIZ);
```

```
        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1){
            return false;
        }

        ifr.ifr_flags = flags & 0xffff;

        if(ioctl(sockfd, SIOCSIFFLAGS, (caddr_t) &ifr) != -1){
            close(sockfd);
            return true;
        }

        close(sockfd);
        return false;
    }
```

Na kraju, kako bismo mogli ostvariti naš glavni cilj, a to je pregled prometa, moramo ga ponovno i upaliti, a to radimo na sličan način na koji smo ga i ugasili, postavljajući zastavice na 0x1003, odnosno postavljamo ih u „UP“ stanje.

```
void UpaliSucelje(QString uredaj){
        int zastavice;
        UzmiZastavice(uredaj, &zastavice);

        if(zastavice == 0x1003)
            return;

        qDebug() << "Palim sucelje, zastavice: " << zastavice;
        PostaviZastavice(uredaj, 0x1003);
    }
```

## 4.3. Dohvaćanje paketa uz pomoć sirovih utikača

Sirovi utikači (engl. raw socket) omogućuju aplikaciji da pristupa paketima direktno s drugog mrežnog sloja, a da nisu enkapsulirani u protokole viših slojeva.

```
int OtvoriSoket(QString nazivSucelja){
        ifreq ifr;
        sockaddr_ll ll;

        int sock_raw = -1;
        sock_raw = socket(PF_PACKET, SOCK_RAW, htons(ETH_P_802_2));
```

```
        if(sock_raw == -1){
            qDebug() << "socket() greska: " << strerror(errno);
            return false;
        }

        strncpy(ifr.ifr_ifrn.ifrn_name,
nazivSucelja.toStdString().c_str(), sizeof(ifr.ifr_name));

        if(ioctl(sock_raw, SIOCGIFINDEX, &ifr) == -1){
            qDebug() << "Socket ioctl() greska: " << strerror(errno);
            return false;
        }

        memset(&ll, 0, sizeof(ll));
        ll.sll_ifindex = ifr.ifr_ifru.ifru_ivalue;
        ll.sll_protocol = htons(ETH_P_ALL);
        ll.sll_family = PF_PACKET;

        if(bind(sock_raw, (sockaddr*)&ll, sizeof(ll)) == -1){
            qDebug() << "Socket bind() greska: " << strerror(errno);
            qDebug() << errno;
            return false;
        }

        return sock_raw;
    }
```

Prvo moramo otvoriti sam *raw socket* na način da pozovemo **socket** funkciju. Kao argumente dodajemo PF_PACKET da bismo stvorili paketni soket, SOCK_RAW kako bismo označili da želimo sirovi soket, te htons(ETH_P_802_2) kako bismo uzimali samo 802.2 pakete.

Paketni soketi služe za slanje i primanje podataka na razini uređaja, odnosno na drugoj OSI razini, a kod njih mogu se koristiti SOCK_RAW, odnosno SOCK_DGRAM za pakete s kojih je maknuto zaglavlje povezničke razine. Mogli smo još iskoristiti **htons(ETH_P_ALL)** da bismo uzeli pakete iz svih vrsta protokola, što ne čini nama nikakvu razliku jer ionako možemo uzimati samo 802.2 pakete u monitorskom načinu rada.

Zatim postavljamo postavke našeg soketa - u struct koji ih opisuje (sockaddr_ll) kopiramo naziv sučelja na kojem želimo slušati promet te uz pomoć ioctl naredbe i SIOCGIFINDEX argumenta uzimamo i indeks tog samog sučelja.

Naposljetku moramo povezati socket s njegovim odgovarajućim imenom uz pomoć funkcije **bind**, a na kraju vraćamo indeks samog soketa.

## 4.4. Određivanje jačine signala

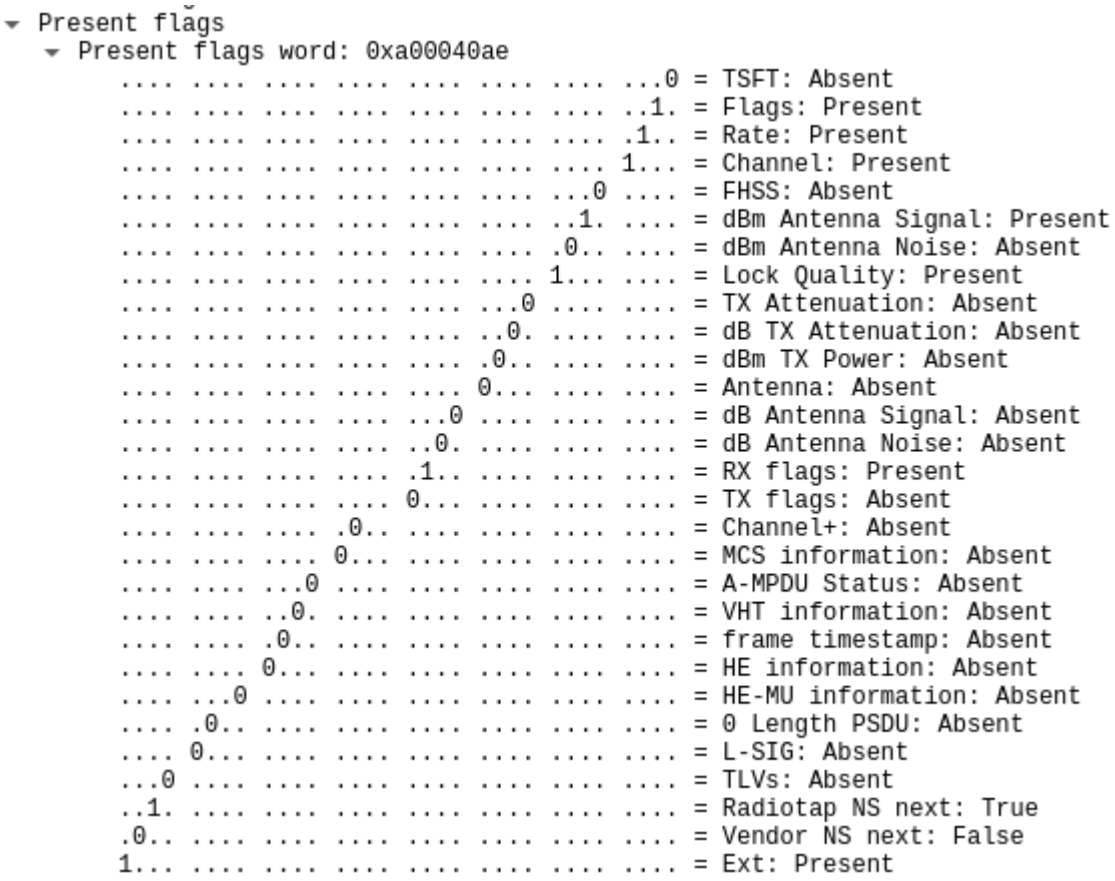
Kako radimo u monitorskom načinu rada, na sam 802.11 okvir dodaje se još jedan dio, a to je Radiotap zaglavlje koje sadrži dodatne informacije o trenutačnom okviru.

RT, između ostalog, sadrži više polja koja opisuju jačinu signala – **dB antenna noise, dB antenna signal, dBm antenna noise, dBm antenna signal**, … No, za razliku od 802.11 zaglavlja, polja nisu fiksne strukture te je potrebno ručno pronaći gdje se željena polja nalaze u uhvaćenom okviru.

Općenito, početak Radiotap zaglavlja je oblikovan u sljedećem obliku:

```
struct ieee80211_radiotap_header {
        u_int8_t        it_version;     /* set to 0 */
        u_int8_t        it_pad;
        u_int16_t       it_len;         /* entire length */
        u_int32_t       it_present;     /* fields present */
} __attribute__((__packed__));
```

Ovisno o implementaciji u upravljačkom programu mrežnog prilagodnika kojeg koristimo, Radiotap nije striktne duljine, već je ona specificirana u **it_len** polju zaglavlja. Na temelju toga možemo zaključiti gdje u našem nizu bajtova završava Radiotap, a gdje počinje 802.11 zaglavlje.
RT zaglavlje nije striktne duljine jer ni polja nisu nužno uvijek prisutna, već je u **it_present** bitmaski specificirano koja su polja prisutna, zapisano u **little-endian** poretku, te ovisno o tome određujemo gdje se nalazi polje koje nam treba. Ako je zadnji bit (**Ext**) postavljen na jedan, još će 4 bajta biti iskorištena kao drugi dio present polja.




Slika 3: Primjer Present bitmaske (izrada autora, 2021.)

Present bitmaski tako može biti više zaredom, odnosno sve dok je Ext bit (31.-vi bit) postavljen na 1 u trenutačnoj maski, iza nje slijedi još 4 bajta koja sadrže još jednu Present bitmasku.

S tim na umu, možemo očekivati da će polja koja sadrže podatke početi u bajtu koji slijedi nakon Ext postavljenog na 0.

```
bool ImaExt(int presentZastavice){
     return presentZastavice & (1<<31);
}


do{
     unsigned int* zastavice_ = reinterpret_cast<unsigned int*>(paketPom);
     zastavice = *zastavice_;
     paketPom += 4; // Pomakni do sljedećeg ili zastavice ili početka polja
   s podatcima
}while(ImaExt(zastavice));
```

Određivanje samih polja radimo na način da prolazimo kroz svaki bit bitmaske te se pomičemo kroz bajtove koje smo uhvatili svaki puta kada u bitmaski pronađemo jedinicu, jer svako od polja ima unaprijed određenu duljinu te znamo koliko bajtova ono zauzima.

Prvo moramo odrediti koliko imamo Present polja kako bismo znali gdje ona završavaju, a gdje počinju sami podaci iz Radiotap zaglavlja. To činimo na način da provjeravamo zadnji bit trenutačne Present bitmaske te u slučaju da je on postavljen na jedan, znamo da još slijedi barem jedna Present bitmaska.

```
int BrojPresenta(unsigned char* bytes) {
      int tren = 0, br = 1;
      bool imaJos = true;

      auto hdrPresent = Split(bytes, 4, 4);
      unsigned int zastavice = ReadInt<unsigned int>(hdrPresent);
      unsigned char* pocetni = bytes + 4;

      do {
        while (zastavice > 0) {
            if (tren == EXT) {
                if ((zastavice & 1) == 1) {
                  br++;

                  hdrPresent = Split(pocetni, 4 * (br - 1), 4);
                  zastavice = ReadInt<unsigned int>(hdrPresent);
                  tren = 0;
                }
                else
                 imaJos = false;
            }
            else {
                if (tren == 15 || tren == 23) {
                  zastavice >>= 1;
                  tren++;
                }

            zastavice >>= 1;
            tren++;

            if (zastavice == 0 && tren != EXT) { // Ako je bitmapa kraća od
            32
```

```
            bita, sigurno poslije nje nema novih
                imaJos = false;
                }
            }
        }
    } while (imaJos);

    return br;
}
```

Zastavice su veličine četiri bajta, odnosno možemo ih tretirati kao običan *int* kroz koji se krećemo bit po bit te ga bitovnim pomacima dijelimo sa dva kako bismo se pomakli od bita do bita. 16-ti i 24-ti bit nisu iskorišteni pa kada do njih dođemo, radimo još jedan pomak povrh.

U slučaju da smo došli do 32.-og bita (tj. EXT iz enumeracije radiotap polja), provjeravamo je li on jednak 1 te u tom slučaju uzimamo sljedeća četiri bajta, povećavamo brojač Present bitmaski te postavljamo brojač trenutačnog bita na 0.

S obzirom na to da idemo s lijeva na desno, ako nam je trenutačni broj jednak 0 (zbog dijeljenja), a nismo još došli do zadnjeg bita, znamo da EXT ne može biti postavljen na 1 te da nema više bitmaski nakon ove.

Nakon toga možemo odrediti vrijednosti nekih od polja koja nam trebaju iz Radiotap zaglavlja.

Duljina cijelog RT zaglavlja zapisana je u trećem polju RT-a te ju možemo lako interpretirati vađenjem dva bajta.

```
int OdrediDuljinuRT(unsigned char* bytes) {
   auto hdrLen = Split(bytes, 2, 2);
   return ReadInt<unsigned short>(hdrLen);
}
```

Prvo spremimo općenite informacije (verziju, duljinu) te uzimamo Present bitmasku koju potom interpretiramo.

```
Radiotap* rt = new Radiotap;

rt->Version = bytes[0];
rt->Length = OdrediDuljinuRT(bytes);

auto hdrPresent = Split(bytes, 4, 4);
```

```
unsigned int zastavice = ReadInt<unsigned int>(hdrPresent);
rt->Present.push_back(zastavice);
```

Kako pozicija određenog polja ovisi o poljima koja dolaze prije njega, za svako polje u present bitmaski moramo se pomaknuti za predodređenu vrijednost. Tako, npr., ako je postavljen TSFT bit, moramo se pomaknuti za 8 bajtova zbog poravnanja, pročitati sljedećih 8 koji predstavljaju tu vrijednost te se ponovno pomaknuti za 8 kako bismo preskočili TSTF polje, nakon kojega uobičajeno slijedi Flags polje koje je veličine samo jednog bajta te se kod njega pomičemo samo jednom za 1 bajt.

```
case (TSFT): {
    trenByte += 8;
    rt->TSFT = ReadInt<long>(Split(bytes, trenByte, 8));
    trenByte += 8;
    break;
}


case (FLAGS): {
    rt->Flags = ReadInt<unsigned char>(Split(bytes, trenByte, 1));
    trenByte += 1;
    break;
}
```

Za nas je ovdje, dakako, najinteresantnije polje **DBM_ANTSIGNAL**, odnosno polje koje sadrži jačinu signala. Kada dođemo do njega, čitamo trenutačni bajt te za njega radimo dvojni komplement tako da dobijemo negativan broj koji predstavlja jačinu signala u dBm.

```
case (DBM_ANTSIGNAL): {
     unsigned char signalByte = ReadInt<unsigned char>(
                                Split(bytes, trenByte, 1));
     rt->DbmSignal = (~signalByte + 1) * -1;
     trenByte += 1;
     break;
}
```

## 4.5. Određivanje vrste paketa

Kao što smo zaključili prethodno kod analize 802.11 zaglavlja, prva dva bajta označavaju Frame Control polje, a prvi od tih dva bajta označava kojem tipu okvira trenutačni okvir pripada. Taj je bajt podijeljen na **podtip-tip-verziju,** odnosno 4 bita predstavljaju podtip, dva tip te preostala dva verziju 802.11 protokola. Za Probe request to izgleda kao sljedeće 01010000, gdje 0101 predstavlja podtip, 00 tip te 00 verziju protokola.

```
Paket* OdrediVrstu(unsigned char* bytes) {
    Paketi paketi;
    auto vrstePaketa = paketi.DohvatiPakete();

    unsigned char* pom = bytes + OdrediDuljinuRT(bytes);
    int tip = pom[0];

    auto paket = std::find_if(std::begin(vrstePaketa),
std::end(vrstePaketa), [&](Paket* paket) { return (*paket).Tip == tip; });

    if(paket == end(vrstePaketa))
    {
        paket =  std::find_if(std::begin(vrstePaketa),
std::end(vrstePaketa), [&](Paket* paket) { return (*paket).Vrsta ==
"Nedefiniran"; });
    }

    return *paket;

}
```

Za svaku vrstu paketa tako možemo spremiti vrstu kao niz bitova, iterirati kroz taj popis te pronaći s kojom se bitovnom vrijednošću poklapa. Implementaciju popisa paketa prikazat ćemo u sljedećem poglavlju.

## 4.6. Određivanje broja trenutačno aktivnih čvorova

Najpraktičniji način da odredimo koliko je čvorova trenutačno aktivno jest da čitamo adresna polja te skupljamo unikatne MAC adrese.

Načelno nas ne zanima koja je vrsta adrese, već samo ima li je u određenoj vrsti okvira, pa koristimo jednostavnu enumeraciju s dvije mogućnosti – Ima/Nema.

```
enum Adrese {
    Ima,
    Nema
};
```

Za svaku podvrstu paketa spremamo i strukturu koja pokazuje koje su adrese prisutne za taj paket.

```
struct AdrPolja {
    char Adr1, Adr2, Adr3, Adr4;

    AdrPolja() {
        Adr1 = ' ';
        Adr2 = ' ';
        Adr3 = ' ';
        Adr4 = ' ';
    }

    AdrPolja(char adr1, char adr2, char adr3, char adr4) {
        Adr1 = adr1;
        Adr2 = adr2;
        Adr3 = adr3;
        Adr4 = adr4;
    }
};
```

Svi paketi imaju nekoliko zajedničkih atributa: Naziv, Vrstu, Tip u binarom obliku te strukturu s adresnim poljima.

```
class Paket {
    public:
    std::string Naziv;
    std::string Vrsta;
```

```
        AdrPolja AdresnaPolja;
        int Tip;

        Paket() { }

        Paket(std::string naziv, int tip) {
              Naziv = naziv;
              Tip = tip;
         }

        AdrPolja DohvatiAdrPolja(int ToDS, int FromDS) { // Samo za Data
        pakete
              if (ToDS == 1 && FromDS == 1)
                 return AdrPolja(Ima, Ima, Ima, Ima);
              else
                 return AdrPolja(Ima, Ima, Ima, Nema);
        }
};
```

Svaka vrsta paketa ima neke svoje specifičnosti u usporedbi s ostalima. Tako kod Data paketa ne specificiramo koja su polja prisutna, već to određujemo na temelju ToDS/FromDS zastavica.

```
class DataPaket : public Paket {
        public:
                DataPaket(std::string naziv, int tip) : Paket(naziv, tip) {
                        Vrsta = "Data";
                }
};
```

Kod management paketa znamo da svi imaju prve tri adrese te u konstruktoru automatski postavljamo adresnu strukturu na odgovarajuće vrijednosti.

```
class MgmtPaket : public Paket {
        public:
                MgmtPaket(std::string naziv, int tip) : Paket(naziv, tip) {
                        AdresnaPolja = AdrPolja(Ima, Ima, Ima, Nema);
                        Vrsta = "Management";
                }
};
```

Kontrolni paketi su sebi svojstveni te svaki od njih ima svoj određeni raspored adresa pa ćemo njih ručno definirati kada dodajemo vrste paketa u listu, a oni su određeni prema [7].

```
class ControlPaket : public Paket {
        public:
            ControlPaket(std::string naziv, int tip, AdrPolja adresnaPolja)
            : Paket(naziv, tip) {
                AdresnaPolja = adresnaPolja;
                Vrsta = "Control";
            }
};
```

Pakete specificiramo u klasi Paketi.

```
class Paketi
{
      private:
                std::vector<Paket*> paketi;
      public:
        Paketi() {
        /* MANAGMENT */
        paketi.push_back(new MgmtPaket("Association Request",
        0b00000000));
        paketi.push_back(new MgmtPaket("Association Response",
        0b00010000));
        paketi.push_back(new MgmtPaket("Reassociation Request",
        0b00100000));
               //...

        /* CONTROL */
        paketi.push_back(new ControlPaket("Block ACK",
        0b10010100, AdrPolja(Ima, Ima, Nema, Nema)));
        paketi.push_back(new ControlPaket("Control Wrapper",
        0b01110100, AdrPolja(Ima, Nema, Nema, Nema)));
               //...

        /* DATA */
        paketi.push_back(new DataPaket("Data", 0b00001000));
        paketi.push_back(new DataPaket("Data + CF-ACK", 0b00011000));
               //...
        }

      std::vector<Paket*> DohvatiPakete() {
```

```
        return paketi;
      }
};
```

Uzimamo prvi bajt koji dolazi nakon RadioTap zaglavlja, a to je dio FrameControl-a, odnosno vrsta paketa, te tražimo u listi paketa paket kojemu je Tip jednak tipu koji smo pronašli u paketu.

Na temelju vrste paketa možemo sada i pronaći u njemu adrese koje su nam potrebne da zaključimo koliko je čvorova aktivno.

Prvo određujemo duljinu RT zaglavlja kako bismo ga znali preskočiti, poslije toga dodajemo još 4 bajta kako bismo preskočili i FrameControl polja. Tada možemo biti sigurni da je trenutačni bajt prvi bajt prvog adresnog polja.

```
std::vector<QString> Citac::OdrediAdrese(unsigned char* bytes, Paket
vrstaPaketa) {
    int rtLen = Procesiranje::OdrediDuljinuRT(bytes);
    int trenByte = rtLen + 4;
    unsigned char* adreseBytes = bytes;

    AdrPolja polja = vrstaPaketa.AdresnaPolja;

    std::vector<QString> macAdrese;

    if (vrstaPaketa.Vrsta == "Data") {
        unsigned char FCZastavice = bytes[rtLen + 1];

        int ToDS = (FCZastavice & 1) == 1;
        FCZastavice >>= 1;
        int FromDS = (FCZastavice & 1) == 1;

        polja = vrstaPaketa.DohvatiAdrPolja(ToDS, FromDS);
    }

    if (polja.Adr1 == Ima) {
        macAdrese.push_back(DodajMAC(Procesiranje::Split(adreseBytes,
trenByte, 6)));
        trenByte += 6;
    }

    if (polja.Adr2 == Ima) {
        macAdrese.push_back(DodajMAC(Procesiranje::Split(adreseBytes,
trenByte, 6)));
        trenByte += 6;
    }

    if (polja.Adr3 == Ima) {
        macAdrese.push_back(DodajMAC(Procesiranje::Split(adreseBytes,
trenByte, 6)));
        trenByte += 6;
    }

    if (polja.Adr4 == Ima)
```

```
        macAdrese.push_back(DodajMAC(Procesiranje::Split(adreseBytes,
trenByte, 6)));

    return macAdrese;
}
```

Ovisno o vrsti trenutačnog paketa te rasporedu adresnih polja koji smo prethodno odredili za tu vrstu, čitamo adresu po adresu, pretvaramo je u tekstualni oblik te dodajemo u listu MAC adresa prisutnih za trenutačni paket.

Data paketi su specifični jer kod njih prisutnost adresa je određena ToDS i FromDS bitovima, pa prvo moramo uzeti njih prije nego odredimo koja su polja dostupna u paketu takve vrste.

Uz to, ne želimo dodati broadcast MAC adresu u listu adresa jer to, naravno, nije poseban čvor.

```
bool JeBroadcastMAC(std::vector<unsigned char> MAC) {
        return std::equal(MAC.begin() + 1, MAC.end(), MAC.begin())
&& MAC[0] == 0xFF;
}
```

Prije nego dodamo čvor u niz, moramo prvo provjeriti je li pravilne duljine, odnosno ima li 6 dijelova od po jedan bajt. Potom pretvorimo svaki dio u string te dodamo „:“ između svakog dijela. Također niti ne želimo imati duplikate pa prvo povjerimo je li trenutačna MAC adresa već u listi.

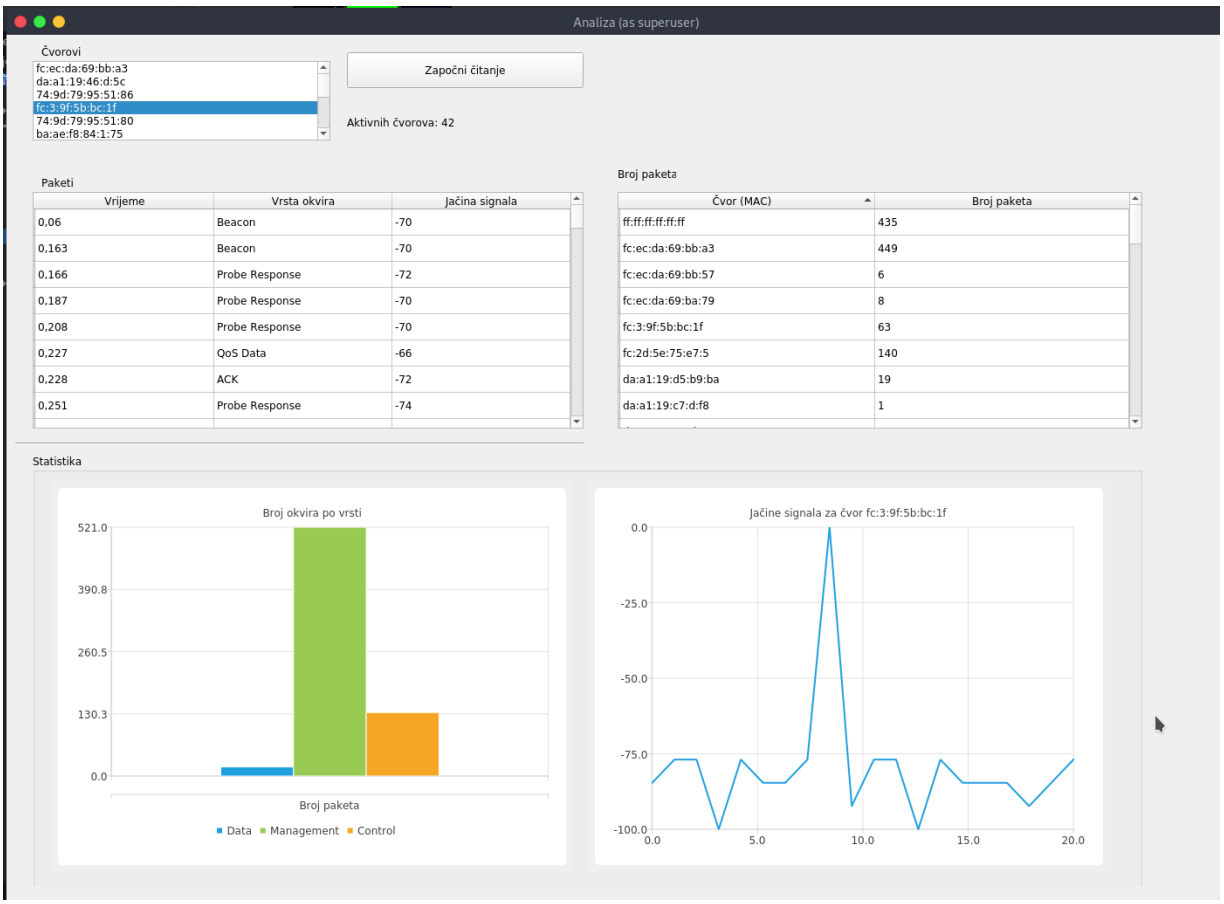
```
QString Citac::DodajMAC(MACNiz MAC) {
    QString adresa = "";

    if(MAC.size() != 6) {
        return adresa;
    }

    for(int i = 0; i < 6;i++){
        adresa += QString::number(MAC[i], 16);
        adresa += ":";
    }

    adresa.chop(1);

    if (std::find(MACAdr.begin(), MACAdr.end(), MAC) == std::end(MACAdr) &&
!JeBroadcastMAC(MAC)) {
        MACAdr.push_back(MAC);

        Cvor cvor;
        cvor.MAC_Adresa = adresa;

        emit noviCvor(cvor);
    }

    return adresa;
```

}

Kada smo ustvrdili da imamo unikatan čvor, šaljemo signal kako bismo ga mogli prikazati na sučelju.

## 4.7. Prikaz rezultata

Sučelje je implementirano kao dva QMainWindow prozora – glavni prozor i prozor za analizu. Na glavnom prozoru korisniku se omogućava izbor između bežičnih sučelja koja su trenutačno spojena na računalo na kojem se izvodi aplikacija. Adapter za to sučelje se pritskom na gumb postavlja u monitorski način te se otvara prozor za analizu. U slučaju da to nije moguće obavještava se korisnika te mu se dopušta da izabere neko drugo bežično sučelje.




Slika 4: Glavno sučelje aplikacije (izrada autora, 2021.)

Prikaz podataka temelji se na dva modela, **CvorModel** i **OkvirModel**. Oba nasljeđuju QAbstractTableModel te se implementiraju metode potrebne za prikaz podataka.

Novi podatci se dodaju uz pomoć signala kao načina za komunkaciju između objekata. Na prozoru za analizu stvaramo poveznicu uz pomoć **connect** funkcije, dok se u klasi Citac nalaze metode noviOkvir i noviCvor koje uz pomoć **emit** naredbe pokrećemo kada smo obradili novi okvir, odnosno kada smo pronašli MAC adresu koju još nismo dodali u listu.

```
connect(citac, SIGNAL(noviOkvir(Okvir)), this, SLOT(dodajOkvir(Okvir)));
connect(citac, SIGNAL(noviCvor(Cvor)), this, SLOT(dodajCvor(Cvor)));
```

Kada se okine noviOkvir, u klasi prozora za analizu poziva se funkcija dodajOkvir te se u nju prosljeđuje novi okvir koji se dodaje na popis te se sučelje automatski osježava.

## 4.8. Statistički prikaz rezultata

Sa svrhom analize podataka, na prozoru s rezultatima imamo i dva statistička prikaza – trakasti graf koji prikazuje broj paketa po glavnim vrstama i statistiku broja paketa po čvoru.

Kako bismo zahtijevali od aplikacije da osvježi statistički prikaz, a da ne stvaramo usko grlo računajući statistiku prilikom dohvaćanja svakog novog paketa, koristimo QTimer mehanizam da bismo u intervalima pozvali funkciju koja će osvježavati prikaz podataka.

```
    statTimer = new QTimer(this);
    statTimer->setInterval(1000);
    statTimer->start();
    connect(statTimer, SIGNAL(timeout()), this, SLOT(osjveziTimerStat()));
```

Interval pozivanja postavlja se uz pomoć setInterval metode na jednu sekundu (tj. 1000 milisekundi), te se uz pomoć prethodno spomenutog slot/signal mehanizma poziva funkcija osvjeziTimerStat() na isteku svakog intervala.

```
void WiFiAnaliza::osjveziTimerStat() {
        emit osvjeziStatReq();
}
```

Glavna svrha te funkcije je da šalje signal prema klasi Citac za čitanje podataka sa soketa kako bi od nje zahtijevala da pošalje nove podatke. Ponovno koristimo slot/signal da napravimo dvostranu komunikaciju između tih klasa, prvi connect da posložimo dio koji šalje podatke, dok drugi služi za slanje zahtjeva prema čitaču.

```
    connect(citac, SIGNAL(osvjeziStatResp(std::vector<Okvir>)), this,
SLOT(osvjeziStat(std::vector<Okvir>)));
```

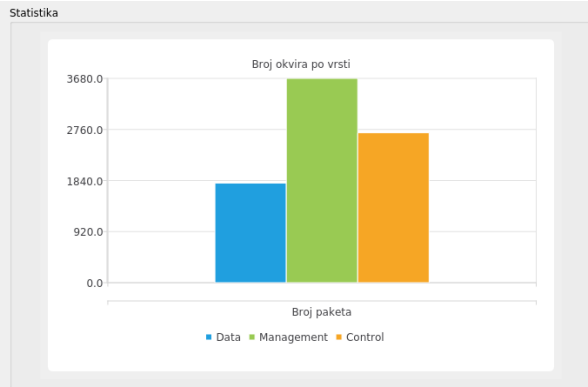
```
    connect(this, SIGNAL(osvjeziStatReq()), citac,
SLOT(osvjeziStatCitac()), Qt::QueuedConnection);
```

S druge strane, Citac šalje signal natrag prema klasi ProzorAnaliza, ali ovaj put s referencom na konkretni vektor s podacima o primljenim okvirima.

```
void Citac::osvjeziStatCitac() {
    emit osvjeziStatResp(sviOkviri);
}

void WiFiAnaliza::osvjeziStat(const std::vector<Okvir> &_okviri) {
    OsvjeziGraf(_okviri);
}
```

### 4.8.1. Graf broja paketa po vrsti

Qt Framework nudi opcionalnu QtCharts biblioteku koja sadrži razne opcije za grafički prikaz podataka u obliku grafova.




Slika 5: Primjer grafa broja paketa po vrsti (izrada autora, 2021.)

Pri prvom postavljanju grafikona za svaku traku trakastog grafa moramo dodati po instancu QBarSet klase kojoj ćemo kasnije pridodavati podatke.

```
    setData = new QBarSet("Data");
    setMgmt = new QBarSet("Management");
    setControl = new QBarSet("Control");
```

Kako u trakastom dijagramu možemo imati više skupova za istu kategoriju na X-osi, moramo dodati i QbarSeries kako bismo ih skupili pod jednom kategorijom.

```
    series = new QBarSeries();
    series->append(setData);
```

```
    series->append(setMgmt);
    series->append(setControl);
```

Samom grafičkom reprezentacijom grafa i pomoćnih elemenata poput legende i osi upravlja instanca QChart widgeta.

```
    chart = new QChart();
    chart->addSeries(series);
    chart->createDefaultAxes();
    chart->setTitle("Broj okvira po vrsti");

    QStringList categories;
    categories << "Broj paketa";
    QBarCategoryAxis *axisX = new QBarCategoryAxis();
    axisX->append(categories);
    chart->addAxis(axisX, Qt::AlignBottom);
    series->attachAxis(axisX);

    chart->legend()->setVisible(true);
    chart->legend()->setAlignment(Qt::AlignBottom);
```

U svrhu prikaza na samom sučelju aplikacije, graf dodajemo na instancu QChartView-a.

```
    chartView = new QChartView(chart);
    chartView->setRenderHint(QPainter::Antialiasing);
    chartView->resize(600, 400);
    chartView->setParent(ui->groupBoxStatistika);
    chartView->move(35, 30);
```

U tom se trenutku još ne prikazuju nikakvi podatci, a za prikaz konkretnih podataka koristimo funkciju `OsvjeziGraf(const std::vector<Okvir> &_okviri)`

```
void WiFiAnaliza::OsvjeziGraf(const std::vector<Okvir> &_okviri) {
    int brojData = std::count_if(std::begin(_okviri), std::end(_okviri),
[](Okvir okvir) { return okvir.paket && okvir.paket->Vrsta == "Data"; });
    int brojMgmt = std::count_if(std::begin(_okviri), std::end(_okviri),
[](Okvir okvir) { return okvir.paket && okvir.paket->Vrsta == "Management";
});
    int brojControl = std::count_if(std::begin(_okviri), std::end(_okviri),
[](Okvir okvir) { return okvir.paket && okvir.paket->Vrsta == "Control";
});

    series->remove(setData);
    series->remove(setMgmt);
```

```
    series->remove(setControl);

    setData = new QBarSet("Data");
    setMgmt = new QBarSet("Management");
    setControl = new QBarSet("Control");

    series->append(setData);
    series->append(setMgmt);
    series->append(setControl);
    chart->removeSeries( series );

    *setData << brojData;
    *setMgmt << brojMgmt;
    *setControl << brojControl;
    chart->addSeries( series );

    chart->axisY()->setRange(0, std::max({brojData, brojMgmt,
brojControl}));
    chartView->update();
}
```

Koristimo **count_if** funkciju da bismo izračunali broj paketa koji su poslani za određenu vrstu glavnih tipova paketa (data/management/control). Potom moramo maknuti trenutačne podatke s grafa, izraditi nove trake za svaku od kategorija, postaviti vrijednosti koje smo prethodno izračunali i dodati ih na sam graf. Na kraju postavljamo raspon Y osi na maksimalnu od tih triju vrijednosti.

### 4.8.2. Statistički prikaz broja paketa po čvoru

Budući da ponovno koristimo QTableView za prikaz podataka u obliku tablice, potrebno je povezati tablicu s njezinim odgovarajućim modelom kako bismo postavili stupce te podatke koji će se u njoj prikazivati.

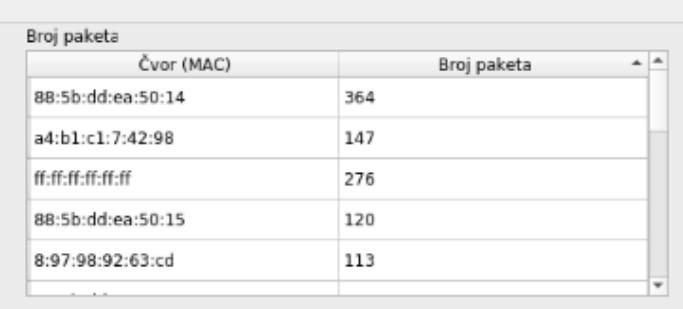
```
    ModelPromet = new CvorPrometModel(this);
    ModelPromet->populateData(promet);

    ui->tableBrojPaketa->horizontalHeader()-
>setSectionResizeMode(QHeaderView::Stretch);
    ui->tableBrojPaketa->setModel(ModelPromet);
    ui->tableBrojPaketa->show();
```

Već smo prethodno postavili da se prilikom dohvaćanja novog okvira poziva `funkcija dodajOkvir(Okvir okvir)` te ju sada možemo iskoristiti da bismo osvježili prikaz broja paketa koji su prošli kroz čvor.

```
void WiFiAnaliza::dodajOkvir(Okvir okvir) {
    if(ModelOkviri != NULL)
     ModelOkviri->dodajOkvir(okvir);

    if(ModelPromet != NULL)
        ModelPromet->dodajPromet(okvir);
}
```

Specifičnost te metode u usporedbi s ekvivalentnim funkcijama u ostale dvije tablice jest u tome da umjesto direktnog dodavanja u niz ovdje provjeravamo ima li već u nizu član kojemu MAC adresa odgovara bilo kojoj iz trenutačnog okvira. Ako je to slučaj, samo povećavamo brojač za taj određeni čvor, inače dodajemo novi element na popis čiji je broj paketa postavljen na jedan.

```
bool CvorPrometModel::dodajPromet(Okvir okvir, const QModelIndex &parent)
{
    for(QString mac : okvir.macAdrese) {
        auto cvor = std::find_if(std::begin(promet), std::end(promet),
[&](CvorPromet &cvorInfo) { return cvorInfo.MAC == mac; });
        if(cvor != promet.end()) {
            cvor->BrojPaketa++;
        }else{
            beginInsertRows(parent, promet.count(), promet.count());
            CvorPromet noviCvor;
            noviCvor.BrojPaketa = 1;
            noviCvor.MAC = mac;
            promet.append(noviCvor);
            endInsertRows();
        }
    }
    return true;
}
```




Slika 6: Primjer tablice broja paketa po čvoru (izrada autora, 2021.)

### 4.8.3. Graf jačine signala odabranog čvora

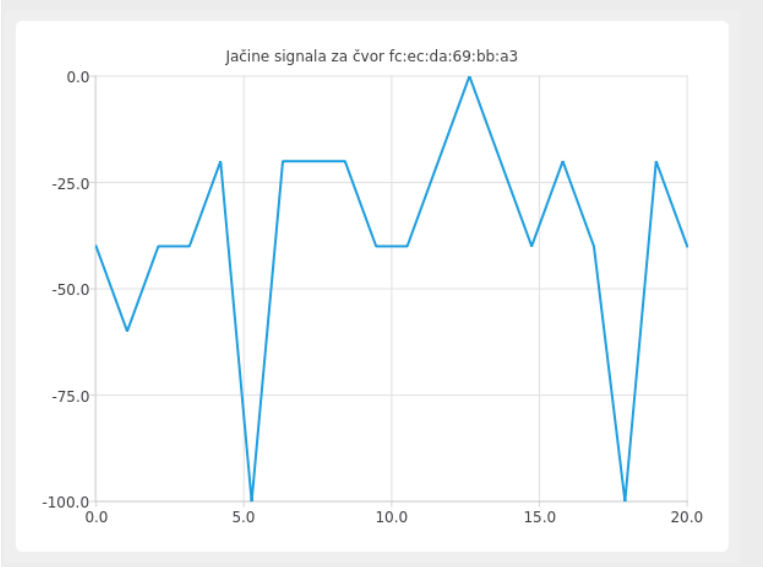
Za određeni čvor moguće je prikazati i jačine čvorova u obliku linijskog grafa. Kao i prethodni graf, ovaj se osvježava na temelju timer-a, u slučaju da je neki čvor odabran u popisu čvorova.

```
void WiFiAnaliza::OsvjeziGrafJacine(const std::vector<Okvir> &_okviri) {
    chartJacine->removeSeries(seriesJacine);

    seriesJacine = new QLineSeries(this);

    QModelIndex index = ui->qListViewCvorovi->currentIndex();
    QString odabraniCvor = index.data(Qt::DisplayRole).toString();

    if(odabraniCvor == "")
        return;

    chartJacine->setTitle("Jačine signala za čvor " + odabraniCvor);

    short brPrikazanih = 0;
    std::vector<Okvir> prikazaniOkviri;

    for (auto it = _okviri.end(); it != _okviri.begin();)
    {
        --it;

        auto okvir = *(it);

        QString sourceMAC = "-1";
        if(okvir.paket->Vrsta == "Control") {
            if(okvir.paket->AdresnaPolja.Adr2 == '1'
                sourceMAC = okvir.macAdrese[1];
        }
        else if(okvir.paket->Vrsta == "Management") {
            sourceMAC = okvir.macAdrese[2];
        }
        else if(okvir.paket->Vrsta == "Data") {
            sourceMAC = okvir.macAdrese[1];
        }

        if (odabraniCvor == sourceMAC)
        {
            prikazaniOkviri.push_back(okvir);
            brPrikazanih++;
```

```
            if(brPrikazanih == 20)
                break;
        }
    }

    brPrikazanih = 0;
    for (auto prikazaniCvor : prikazaniCvorovi) {
         seriesJacine->append(++brPrikazanih, okvir.JacinaSignala);
    }

    chartJacine->addSeries(seriesJacine);
}
```

Najprije s grafa brišemo postojeće podatke te provjeravamo je li odabran neki čvor. Kada je to slučaj, nastavljamo s traženjem okvira u kojima je izvorišna MAC adresa postavljena na MAC adresu odabranog čvora. Prolazimo kroz listu dohvaćenih okvira od kraja kako bismo prvo u obzir uzeli najnovije čvorove te za svaki, ovisno o njegovoj vrsti, uzimamo adresno polje koje sadrži izvorišnu adresu. Tako kod Control okvira Source Address polje imaju samo oni koji imaju i drugo adresno polje (što nije slučaj kod, primjerice, ACK okvira). Za sve podvrste Management paketa struktura je jednaka te se traženo polje nalazi treće u redu adresnih polja, dok za Data pakete tu ulogu ima drugo polje kod svih (zajedno sa specifičnim ulogama koje ima ovisno o ToDS/FromDS zastavicama).

Kada smo ustvrdili da se polja poklapaju, dodajemo vrijednost u zasebnu listu prije nego je dodajemo na graf kako bismo ih mogli prikazati u pravilnom poretku te povećavamo brojač kako ne bismo na grafu prikazali potencijalno prevelik broj čvorova (u ovom je slučaju to ograničeno na dvadeset točaka na grafu). Na kraju prolazimo kroz listu i dodajemo vrijednost na graf.




Slika 7: Primjer grafa jačine signala za čvor (izrada autora, 2021.)

# 5. Zaključak

Velik broj uređaja na jednom mjestu, bilo to u stambenoj zgradi ili studentskom domu, može dovesti do problema s povezivošću i lošim performansama mreže. Zato je važno da se situacija analizira uz pomoć nekog od alata za analizu mrežnog prometa kako bismo utvrdili koji su najaktivniji čvorovi u našem okruženju.

Jedan vrlo potentan alat, koji ne nudi svaki mrežni adapter jer zahtijeva da proizvođač za njega izradi posebne upravljačke programe i firmware, jest monitorski način rada.

Monitorski način rada omogućava da umjesto tipičnog povezivanja sa specifičnom pristupnom točkom mrežni adapter služi kao nadzorni alat za sve događaje koji prolaze mrežom, odnosno umjesto da bude spojen u specifičnu mrežu, on sakuplja sve podatke koji su u njegovom rasponu. To, doduše, znači da ne će biti u mogućnosti čitati payload specifičnog paketa jer nije autentificiran za ni jednu mrežu, no moći će pročitati važne tehničke detalje.

U tome pomaže i Radiotap zaglavlje koje se u monitorskom načinu rada dodaje povrh uobičajenih 802.11 zaglavlja. Iako je definirano koja se polja mogu pojaviti te su ona striktno poredana kako bi se znalo koja su od njih prisutna u određenoj implementaciji Radiotap zaglavlja, moraju se čitati Present bitmaske, odnosno polja od 32 bita, u kojima točno određeni bit postavljen na 1 znači prisutnost određenog polja u RT zaglavlju. Za naše su potrebe posebno interesantna polja Antenna noise/Antenna signal te dB Antenna noise/dB Antenna signal koja nam omogućuju da donosimo zaključke o kvaliteti signala kojem je prenesen određeni okvir.

Osim Radiotap zaglavlja, svaki okvir na podatkovnom sloju dobiva i 802.11 zaglavlje iz kojeg možemo dobiti informacije o samom okviru i njegovom sadržaju. Prvi dio 802.11 zaglavlja je Frame Control iz kojeg možemo pročtiati o kojoj je vrsti i podvrsti paketa riječ. Postoje tri glavne vrste bežičnih paketa – control, management i data, od kojih svaki ima nekoliko desetaka podvrsti. Ostatak zaglavlja ovisi o specifičnoj podvrsti paketa. Naime, kod Management paketa prisutne su uvijek tri MAC adrese, dok kod Data paketa njihova prisutnost ovisi o ToDS/FromDS poljima ili je svaki Control paket specifičan u svom rasporedu, ovisno o tome kako je već definiran u samom 802.11 standardu.

# Popis literature


[1] African Network Operators Group, „Introduction to Wide-Area WiFi“, AfNOG, 2009., [Na internetu] Dostupno na: https://tavaana.org/sites/default/files/introduction_to_wifi_0.pdf [Pristupljeno 20-tra-2021]

[2] M. Kershaw, „Wi-Fi Monitoring & Kistmet“, Sharkfest 2019., 2019. [Na internetu] Dostupno na: https://sharkfestus.wireshark.org/assets/presentations19/32.pdf [Pristupljeno 22-tra-2021]

[3] thatiotguy, „Enable Monitor Mode in TP-LINK TL-WN722N V2/V3“, 2020. [Na internetu]. Dostupno na: https://www.hackster.io/thatiotguy/enable-monitor-mode-in-tp-link-tl-wn722n-v2-v3-128fc6 [Pristupljeno 23-tra-2021]

[4] Tekrtronix, „Wi-Fi: Overview of the Physical Layer and Transmitter Measurements“, 2016. [Na internetu]. Dostupno na: https://download.tek.com/document/37W-29447-2_LR.pdf [Pristupljeno 3-svi-2021]

[5] J. Sharp, „802.11 Frame Types and Formats“, 2020. [Na internetu]. Dostupno na: https://howiwifi.com/2020/07/13/802-11-frame-types-and-formats/ . [Pristupljeno 4-svi-2021]

[6] A. Pindoria, „802.11 Frame Format and Types“, 2017. [Na internetu] Dostupno na: https://dot11ap.wordpress.com/802-11-frame-format-and-types/. [Pristupljeno 4-svi-2021]

[7] Institute of Electrical and Electronics Engineers (IEEE), „IEEE Standard for Information technology-- Local and metropolitan area networks-- Specific requirements-- Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications“, 2012.

[8] *ParrotOS logo* [Slika] (bez dat). Dostupno: www.parrotsec.org [Pristupljeno 21-tra-2021]

[9] *Alfa AWUS036ACS Wi-Fi adapter* [Slika] (bez dat). Dostupno: www.alfa.com.tw [Pristupljeno 21-tra-2021]

# Popis slika

# Popis tablica