VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO GENEROVÁNÍ RÁMCŮ PODLE STANDARDU 802.11

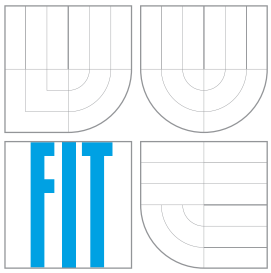
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PAVEL ŠVANDA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO GENEROVÁNÍ RÁMCŮ PODLE STANDARDU 802.11

FRAME GENERATOR BASED ON 802.11 STANDARD

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. PAVEL ŠVANDA
AUTHOR

VEDOUCÍ PRÁCE — Ing. MATEJ KAČIC
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá návrhem a realizací generátoru rámců používaných pro přenos informací v bezdrátových sítích. Na začátku práce je popsán úvod do problematiky přenosu dat v bezdrátorých sítích standardu 802.11 a provedena analýza existujících nástrojů. Poté je popsán postup návrhu jazyka sloužícího k popisu struktury vlastních rámců a návrh vlastního nástroje. Práce se také zabývá následnou realizací výsledného nástroje. Na závěr jsou uvedeny výsledky testování realizovaného nástroje.

Abstract

This thesis describes the design and implementation of the generator frames used for information transfer in wireless networks. At the beginning of the thesis there is described the introduction to the issue of data transmission in wireless networks 802.11 and the analysis of existing tools. Then it is designed language for describing the structure of their own frames and designed their own tools. The work also deals with the subsequent implementation of the resulting tool. Finally the results of the test are mentioned.

Klíčová slova

Generátor rámců, bezpečnost bezdrátových sítí, standard 802.11, RadioTap hlavička, IEEE hlavička.

Keywords

Frame generator, wireless security, standard 802.11, RadioTap header, IEEE header.

Citace

Pavel Švanda: Nástroj pro generování rámců podle standardu 802.11, diplomová práce, Brno, FIT VUT v Brně, 2013

Nástroj pro generování rámců podle standardu 802.11

Prohlášení

Prohlašuji, že jsem tuto práci vypracoval samostatně pod vedením pana Ing. Mateje Kačice

. . . . . .
Pavel Švanda
16. května 2013

Poděkování

Rád bych poděkoval vedoucímu práce panu Ing. Mateji Kačicovi za cenné rady, připomínky a čas, který mi věnoval při vypracování této práce.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V dnešní době si většina z uživatelů nedovede svůj život pomalu představit bez existence Internetu. Ať to jsou velké firmy nebo jednotlivci, využívají Internetu jako každodenního prostředku k získávání informací. Většina z těchto uživatelů používá pro přístup k Internetu bezdrátového připojení. Tímto připojením je ve většině případů bezdrátová síť Wi-Fi, kterou si mnoho z uživatelů oblíbilo. Důvodem její oblíbenosti je pravděpodobně skutečnost, že její použití odprošťuje uživatele od připojení pomocí kabelů a dává jím tak možnost být připojení kdekoliv potřebují.

Vlastnost, pro kterou si Wi-Fi síť získala tolik uživatelů, přináši ale i jedno velké riziko. Tímto rizikem je možnost případného zneužití informací, které jsou v síti přenášeny. Je potřeba vynaložit větší úsilí pro vyšší zabezpečení těchto sítí před případnými útočníky. Ti by se mohli dostat k příslušným informacím pomocí zranitelností, které může tato síť obsahovat. K tomu abychom mohli učinit sítě bezpečnější, je zapotřebí tyto zranitelnosti zkoumat a odhalit dříve než případný útočník. Ale k tomu abychom je mohli zkoumat, je zapotřebí vyvinout odpovídající nástroje. Jedním z těchto nástrojů může být i generátor rámců, které jsou v síti použity pro přenos informací.

Před návrhem a implementací samotného nástroje je zapotřebí získat potřebné informace. Tyto informace jsou zapotřebí především, abychom byly schopni porozumět jak tato síť funguje. Z principu funkčnosti sítě poté můžeme odhalit mnoho zranitelností, které při jejím návrhu mohli vzniknout. Tyto informace budeme dále využívat při vlastní realizaci nástroje. Kapitola druhá v krátkosti představuje standard 802.11. Tento standard je také znám pod označením Wi-Fi sítě. Můžeme zde například naleznout, jakým způsobem probíhá přenos dat v síti a jaké jsou k tomu využity mechanismy. S tímto souvisí i popis rámců použitých při přenosu dat. Můžeme zde naleznout popis RadioTap hlavičky a hlavičky IEEE pomocí nichž je vlastní přenos realizován. Další obsah vyskytující se za těmito hlavičkami je datovým obsahem vlastních rámců. Ten může obsahovat hlavičky dalších protokolů, jejiž popis není náplní práce. Tato data mohou být v otevřené, případně zašifrované podobě. Jednotlivé mechanizmy zabezpečení těchto rámců jsou rovněž představeny v této kapitole.

V současnosti existuje řada nástrojů, které se problematikou zabezpečení bezdrátových sítí zabývají. Kapitola třetí, se zabývá analýzou právě těchto nástrojů. Jsou zde představeny tři aplikace, u kterých je v krátkosti uveden popis hlavní funkcionality a shrnuty nedostatky daného řešení, které u něj shledáváme. Na základě této analýzy byl specifikován cíl práce a představeny funkční a nefunkční požadavky nové aplikace. Z těchto požadavků byl poté proveden návrh tří řešení, která rozšiřují či doplňují stávající nástroje. Jedno z těchto řešení bylo následně doporučeno k dalšímu návrhu a vlastní realizaci.

Nástroj doporučený k realizaci byl rovněž v kapitole třetí popsán z pohledu jeho návrhu. Před samotným návrhem se zde nachází popis použitých nástrojů a knihoven, které byly použity při vlastní realizaci. Na základě volby těchto nástrojů mohl být proveden návrh vlastní aplikace, který

tuto volbu zohledňuje. Důležitou částí celé aplikace je jazyk, který slouží k popisu jednotlivých rámců. Jednotlivé konstrukce tohoto jazyka jsou zde v krátkosti popsány a je vždy ukázáno jak je lze v aplikaci využít. V závěru kapitoly je uveden návrh části aplikace, která se stará o realizaci generování vlastních rámců.

Vlastní realizaci aplikace popisuje kapitola čtvrtá nazvaná Implementace. V této kapitole můžeme naleznout přehled a popis jednotlivých funčností aplikace. V úvodu kapitoly je popsána realizace analyzátorů a interpretu navrženého jazyka. Můžeme zde například naleznout popis, jak jsou jednotlivé rámce v aplikaci reprezentovány a jak je vykonávan program pomocí interpretu. Dále je zde uveden popis jednotlivých kroků, které je zapotřebí vykonat, abychom z popisu rámce byly shopni vygenerovat, zašifrovat a následně zaslat daný rámce.

V rámci práce byly provedeny dva testy realizované aplikace, které jsou popsány v kapitole páté. Tyto testy spočívaly ve volbě a realizaci vhodných útoku na bezdrátovou síť. Pro účely testování byl zvolen jeden jednodušší útok, kterým byl Flood útok. Dále pak byl zvolen jeden již značně složitější útok, který by bez odpovídajících nástrojů bylo možné jen obtížně realizovat. Tímto útokem byl útok vyžívající zranitelnosti nazvané Hole196. Každý námi z vybraných útoků je zde nejdříve v krátkosti představen po teoretické stránce a poté je uveden popis, jak daný útok probíhal. Na závěr každého testu je uvedeno krátké zhodnocení dosažených výsledků, kterých bylo dosaženo. V závěru kapitoly pak můžeme naleznout celkové zhodnocení realizovaných testů a použití aplikace.

Během realizace aplikace jsme narazili na přiležitosti dalšího rozšíření aplikace. Jenotlivé příležitosti byly doporučeny jako možné rozšíření aplikace do budoucna. O těchto rozšířeních pojednává kapitola šestá nazvaná Rozšíření aplikace.

Na závěr práce bylo provedeno krátké shrnutí, které je uvedené v kapitole sedmé nazvané Závěr. Můžeme zde naleznout popis dílčích výsledků, kterých bylo v rámci práce dosaženo, a shrnutí přínosů pro nás samotné.

Kapitola 2

Úvod do bezdrátových sítí standardu 802.11

Počátky bezdrátových sítí standardu **802.11** jsou v roce 1997. V tomto roce publikoval mezinárodní standardizační institut **IEEE** *(Institute of Electrical and Electronics Engineers)* specifikaci bezdrátové sítě pracující v pásmu ISM *(Industrial, Scientific and Medical)*. Pásma ISM na frekvencích 900-929 MHz a 2,4-2,4835 GHz společně s pásmem UNii *(Unlicensed National Information Infrastructure)* na frekvencích 5,15-5,35 GHz a 5,75-5,825 GHz představují nelicencovaná frekvenční pásma. Termín "nelicencovaná"zde znamená, že uživatel, který provozuje bezdrátové zařízení v těchto pásmech, nemusí vlastnit licenci pro funkci daného zařízení. [16]

K standardu definovanému v roce 1997, který je také označován jako původní nebo základní standard, byla postupem času přidána řada dalších rozšíření. Tato rozšíření jsou rozlišována pomocí písmen anglické abecedy a rozšiřují původní standard o nové vlastnosti. Některá tato rozšíření v krátkosti představuje tabulka 2.1. Podrobnější popis rozšíření, která jsou pro tuto práci potřebná, je uveden dále v textu.

Standard	**Popis**
802.11	Původní standard bezdrátových sítí s rychlostí přenosu dat 1 nebo 2 Mbit/s ve frekvenčním pásmu 2,4 GHz.
802.11a	Využívá pásmo 5 GHz s maximální přenosovou rychlostí až 54 Mbit/s.
802.11b	Rozšiřuje původní standard o maximální přenosovou rychlost 11 Mbit/s.
802.11e	Definuje podporu pro kvalitu služeb.
802.11f	Vylepšuje mechanismus spolupráce přístupových bodu.
802.11g	Standard zpětně kompatibilní s 802.11b pomocí něhož dosahujeme rychlostí až 54 Mbit/s v pásmu 2,4 GHz.
802.11i	Dodatek vylepšující zabezpečení bezdrátových sítí.
802.11n	Standard pracující v pásmu 2,4 GHz nebo 5 GHz a dosahující přenosové rychlosti až 600 Mbit/s.
802.11h	Dodatek řešící interference[1] dvou zařízení pracující v témže pásmu.

Tabulka 2.1: Přehled nejznámějších rozšíření standardu 802.11

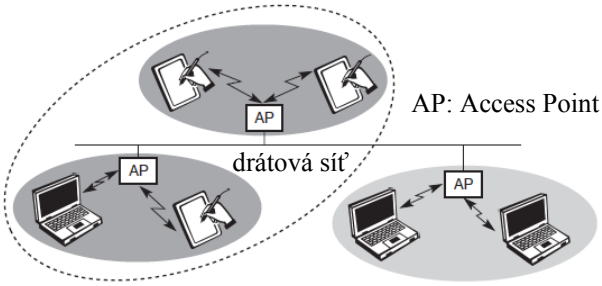
[1] vzájemné ovlivňování, prolínání nebo střetávání dvou signálů

2.1 Architektura sítí

Propojení stanic v bezdrátové síti je možné realizovat dvěma způsoby. Prvním způsobem je vybudovaní infrastruktury podobné jako u drátových sítí Ethernet. Sítě tohoto typu se pak nazývají **Infrastrukturní sítě**. Druhým způsobem je použití **Ad-hoc sítě**, kde jednotlivé stanice spolu komunikují napřímo bez nutnosti prostředníka.

2.1.1 Infrastrukturní síť

V infrastrukturní síti probíhá komunikace mezi dvěma bezdrátovými stanicemi přes prostředníka, kterého nazýváme přístupový bod nebo také AP *(Access Point)*. Přístupový bod umožňuje nejen komunikaci stanic v rámci téže sítě, ale také umožňuje propojení do jiných sítí. Propojení s jinou sítí probíhá přes distribuční systém, který může být tvořen například drátovou sítí Ethernet, tak jak ukazuje obrázek 2.1, nebo také pomocí bezdrátové sítě.

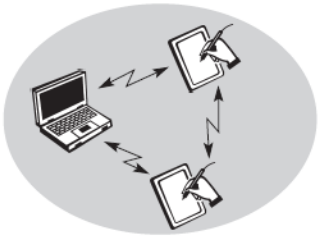


Obrázek 2.1: Infrastrukturní síť [27]

Skupina stanic využívající stejné rádiové pásmo, komunikující s jedním AP, se nazývá BSS *(Basic Service Set)*. Propojením několika těchto sítí pomocí distribučního systému může vzniknout jedna logická síť označována jako ESS *(Extended Service Set)*.

2.1.2 Ad-hoc síť

Stanice v ad-hoc síti spolu komunikují napřímo bez nutnosti prostředníka, tak jak ukazuje obrázek 2.2. Toto má své výhody i nevýhody. Výhodou je, že tato síť je jednodušší oproti síti infrastrukturní. Obsahuje méně funkčních prvků. Nevýhodou je, že jsme omezeni pouze na komunikaci v rámci této ad-hoc sítě. Dalším negativem je i složitost jednotlivých stanic v porovnání s předchozím typem sítí. Zde je zapotřebí, aby stanice udržovala spojení se všemi stanicemi, se kterými chce komunikovat. Toto v infrastrukturní síti není zapotřebí. V ní se udržuje spojení pouze s přístupovým bodem.



Obrázek 2.2: Ad-hoc síť [27]

Skupina stanic komunikující v rámci jedné ad-hoc sítě se označuje jako IBSS *(Independent Basic Service Set)*. [27]

2.2 Přenos dat na fyzické vrstvě

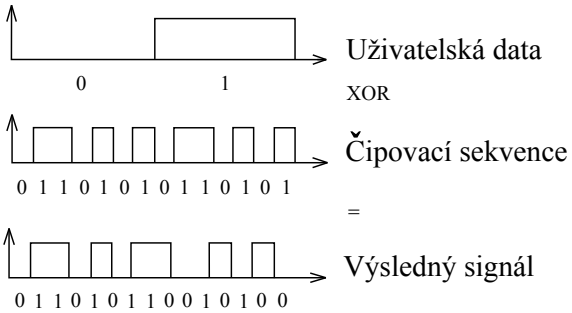
Standard 802.11 definuje přenos dat pomocí infračerveného světla a pomocí rádiových vln. Přenos dat pomocí infračerveného světla není moc rozšířený. Důvodem může být například nízká vzdálenost dosahu, která se pohybuje okolo 10 m, nebo neschopnost průchodu záření většinou materiálů, čímž se zhoršují vlastnosti šíření signálu. Dále se tedy přenosem dat pomocí infračerveného světla nebudeme zabývat. Místo infračerveného světla se používá přenos dat pomocí rádiových vln v bezlicenčním pásmu **2,4 GHz**. V původním standardu jsou definovány dvě metody pro přenos dat pomocí rádiových vln. Obě tyto metody přenosu využívají metod rozprostřeného spektra [27, 26].

FHSS - frequency hopping spread spectrum

FHSS je metoda rozprostřeného spektra, která umožňuje koexistenci více sítí v téže oblasti pomocí mechanismu střídání přenosových kanálů. Střídání kanálů je definováno pomocí tzv. hopping sekvence. Ta definuje, který kanál se použije pro přenos aktuálního rámce. Po přenosu rámce je použit následující kanál určený touto sekvencí. Sekvence je unikátní pro každé dvě komunikující stanice, čímž se minimalizuje pravděpodobnost, že by dvě dvojice stanic komunikovali v tentýž okamžik na stejné frekveci. Pokud by tento případ nastal, přenos by se jednoduše zopakoval na jiné frekvenci.

DSSS - direct sequence spread spectrum

Metoda DSSS je alternativní metodou k FHSS, která místo změny frekvence používá změnu kódu. Vysílaný signál je dán kombinací původního signálu s tzv. čipovací sekvencí, která má pseudonáhodný charakter. V případě sítí 802.11 se k vytvoření této čipovací sekvence využívá Barkerova kódu (+1, -1, +1, +1, -1, +1, +1, +1, -1, -1, -1). Způsob jak je vytvořena výsledná sekvence zachycuje obrázek 2.3.



Obrázek 2.3: Princip výpočtu výsledné sekvence pomocí DSSS

Čipovací sekvence obsahuje několik čipů na jeden datový symbol. Výsledný signál se pak získá pomocí operace XOR se vstupy datového signálu a signálu čipovací sekvence. Datový signál je pak rozprostřen na tuto čipovací sekvenci.

Původní standard 802.11 dosahuje maximálních přenosových rychlostí **1 až 2 Mbit/s** v závisloti na použité modulaci přenášeného signálu. Popis jednotlivých modulací je k dispozici například v publikaci odkazované ze zdroje [27].

2.2.1 802.11a

Nedostatkem původního standardu byla maximální rychlost přenosu dat. S cílem dosažení vyšších přenosových rychlostí bylo v roce 1999 publikováno rozšíření známe jako **802.11a**. V tomto rozšíření je definován přenos dat v bezlicenčním pásmu **5 GHz** s maximální přenosovou rychlostí až **54 Mbit/s**. K dosažení této rychlosti bylo docíleno pomocí technologie **OFDM**.

OFDM - orthogonal frequency division multiplexing

Modulační metoda OFDM spočívá v použití několika desítek nosných frekvencí, které jsou ortogonálně nezávislé. To znamená, že jejich skalární součin je nulový. Přenášený signál se poté moduluje na jednotlivé nosné frekvence. Přenosová rychlost jednotlivých nosných frekvencí je nízká, ale tím že se přenáší více symbolů zároveň dosahuje vyšších rychlostí než za použití jedné nosné frekvence. [17]

2.2.2 802.11b

Ve stejném roce jako bylo definováno rozšíření 802.11a, bylo také definováno rozšíření **802.11b** fungující ve frekvenčním pásmu **2,4 GHz**. Toto rozšíření výužívá metody DSSS a za pomoci několika vícestavových modulací navyšuje rychlost původního standardu až na **11 Mbit/s**.

2.2.3 802.11g

V roce 2003 byl představen nový standard nazvaný **802.11g**. Tento standard pracuje ve frekvenčním pásmu **2,4 GHz** a navyšuje dosavadní maximální rychlost v tomto pásmu z 11 Mbit/s na hodnotu **54 Mbit/s**. Dosažení vyšších přenosových rychlostí bylo použitím metod shodných s 802.11a, pouze v jiném pásmu. K zachování zpětné kompatibility s předchozím standardem 802.11b zahrnuje i přenos pomocí DSSS, ale s rychlostí pouze 11 Mbit/s. [17]

2.3 Metody přístupu k přenosovému médiu

Standard 802.11 definuje celkem tři metody přístupu k bezdrátovému médiu [27]. Z těchto metod jsou výrobci povinni implementovat pouze jednu z nich. Zbývající mohou implementovat volitelně. Tyto metody jsou take nazývány jako DFWMAC *(Distributed foundation wireless medium access control)* metody. Jednotlivé metody budou v krátkosti představeny níže.

2.3.1 DFWMAC-DCF s použitím CSMA/CA

První metoda a zároveň jediná, kterou je výrobce povinen implementovat, se nazývá DFWMAC-DCF s použitím CSMA/CA *(Distributed coordination function using Carrier sense multiple access with collision avoidance)*. Metoda spočívá v naslouchání bezdrátového média s cílem předejít kolizím během přenosu. Pokud stanice požaduje vysílat data a neprobíhá žádná komunikace, může stanice zahájit přenos vlastního rámce. Stanice ovšem nesmí začít rámec vysílat okamžitě. Před každým přenosem musí vyčkat stanovený timeout, nazývaný mezirámcová mezera nebo také IFS *(inter-frame spacing)*. Pokud během této mezirámcové mezery neproběhne žádná komunikace, smí stanice zahájit vysílání rámce. Standard 802.11 definuje tři délky mezirámcových mezer, kterými jsou SIFS, PIFS a DIFS seřazeny od nejkratší po nejdelší. Dle významu rámce se použije daný

interval. Například pokud potřebuje stanice vysílat datový rámec, počká po dobu stanovenou mezerou DIFS. Ale v případě, kdy chce stanice odestal potvrzení přijatého rámce, vyčká pouze krátký interval SIFS.

V případě, kdy komunikace probíhá, vygeneruje si stanice náhodnou hodnotu timeoutu z předem definovaného intervalu, po kterou pozdrží své odeslání. Tento timeout se začne odečítat po ukončení aktuálního přenosu a po vyčkání povinné mezirámcové mezery. Pokud během této doby neproběhne žádná komunikaci smí stanice zahájit přenos vlastního rámce.

2.3.2 DFMAC-DCF s RTS/CTS rozšířením

Předchozí metoda se snažila předejít kolizím pomocí naslouchání přenosového média. V případě bezdrátového media dochází ke kolizím na straně přijímací stanice, nikoliv na straně stanice vysílající. Proto může dojít ke kolizi i v případě, pokud se stanice domnívá, že médium je volné. Tomuto problému se také říká problém skrytého terminálu. Tato metoda se snaží tomuto problému předejít pomocí krátkých rezervujících rámců. Během přenosu těchto rámců může dojít ke kolizi, ale jakmile je stanici pásmo přiděleno probíhá přenos již bezkolizně.

V okamžiku, kdy stanice požaduje zaslat datový rámec, provede nejdříve zaslání rámce RTS *(Request to Send)*. Pro zaslání tohoto rámce platí stejná pravidla jako v metodě předchozí. Po přijetí RTS rámce přijímající stanící, vyšle tato stanice CTS rámec *(Clear to Send)*. Tím dává najevo první stanici, že může zasílat vlastní data, ne však dříve než vyprší interval SIFS. Tento rámec, ale také slyší i ostatní stanice, díky němuž se dozví, že někdo bude v následujícím čase komunikovat. Díky této informaci lze předejí případné kolizi.

2.3.3 DFMAC-PCF s dotazováním

Nevýhodou předchozích dvou řešení je, že negarantují maximální dobu zpoždění nebo minimální šířku přenosového pásma. Toto se snaží řešit následující metoda zvaná DFMAC-PCF s dotazováním *(PCF - point coordination function)*. Metoda počítá s jednou stanicí jako s koordinátorem, proto nelze tuto metodu využít v ad-hoc sítích.

Koordinátor v pravidelných intervalech vysílá rámce typu *beacon*, kterými sděluje ostatním stanicím v síti parametry dané sítě. Čas mezi vysíláním těchto rámců je rozdělen do dvou intervalů. Na první interval – doba bez boje o médium – a druhý interval, kdy probíhá boj o médium za použití předchozích dvou metod. V prvním intervalu koordinator po uplynutí intervalu PIFS periodicky vyzývá ostatní stanice a dává jim najevo, že mají volné přenosové médium. Pokud stanice má data, které potřebuje přenést, provede jejich vyslání po uplynutí intervalu SIFS. V opačném případě koordinátor vyzve další stanici v pořadí.

Pomocí této metody je stanicím garantováná maximální doba mezi jednotlivými přenosy a minimální šířka pásma. Proto je tato metoda vhodná pro zasílání pravidelných datových přenosů.

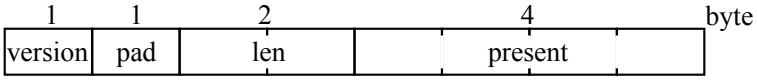
2.4 Struktura rámců

Přenášené rámce se skládají ze dvou hlaviček. Z hlavičky **RadioTap** a hlavičky **IEEE 802.11**. Za těmito hlavičkami se v případě datových rámců nachází přenášená data[2], která mohou být v otevřené nebo zašifrované podobě. Popis jednotlivých šifrovacích mechanismů je popsán v části 2.5 věnující se zabezpečení bezdrátových sítí standardu 802.11.

[2] označovány také jako payload

2.4.1 RadioTap hlavička

Hlavička RadioTap [1] slouží k doplnění informací z ovladačů síťové karty k přenášenému rámci, které jsou předány aplikaci a naopak, tedy předání informací z aplikace směrem k ovladačům. Mezi těmito informacemi lze například naléznout informace o použitém kanálu nebo číslu antény, která byla použita pro vyslání rámce. Výhodou RadioTap hlavičky oproti jiným, jako příklad uveďme hlavičky Prism nebo AVS, je její vysoká flexibilita. Hlavička v základní podobě, tak jak ukazuje obrázek 2.4, obsahuje pouze čtyři povinné položky. Další položky je možné přidávat dle potřeby. Jejich přítomnost je poté poznačena v informačním poli *present*. V případě potřeby je možné kdykoliv další položku doplnit, aniž by to mělo vliv na stávající analyzátory hlavičky. Pro srovnání uveďme hlavičku Prism, která má vždy konstantní délku 144 bytu a není možné ji dále měnit. Tato hlavička neobsahuje informaci o kontrolním součtu FCS *(Frame check sequence)*. Pomocí hlavičky RadioTap lze tento nedostatek jednoduše vyřešit přidáním nové, nepovinné položky a informaci o kontrolním součtu tak zahrnout.



Obrázek 2.4: Hlavička RadioTap

Význam jednotlivých povinných položek hlavičky je následující:

- **version** značí použitou verzi RadioTap hlavičky, která je v současné době rovna hodnotě 0
- **pad** v současné době nemá využití
- **len** určuje celkovou délku RadioTap hlavičky
- **present** je maska obsahující informaci o přítomnosti volitelných položek hlavičky, kde pro každou položku je vymezen jeden bit

Výčet nepovinných položek hlavičky by byl zdlouhavý, a proto jej zájemci mohou nalézt ve zdroji [1] uvedeném v části Literatura.

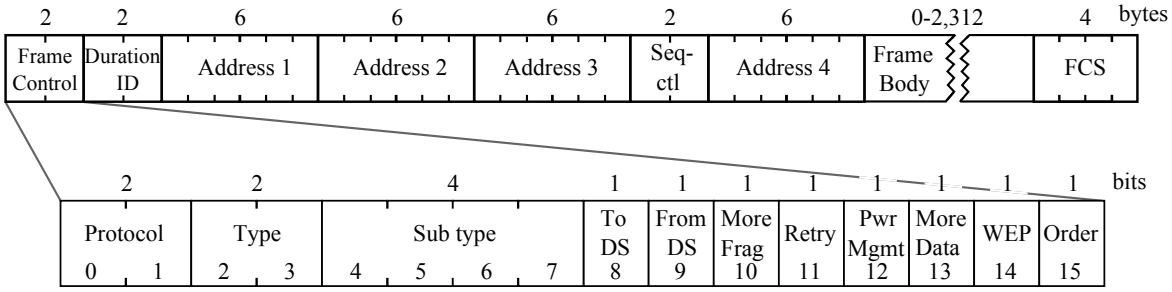
Při tvorbě RadioTap hlavičky je potřeba mít na paměti následující dvě vlastnosti. První vlastností je, že pořadí nepovinných položek je zapotřebí strikně dodržovat. Lze použít libovolnou kombinaci položek, ale jejich pořadí musí vždy odpovídat pořadí, tak jak je uvedeno v poli *present*. Druhou vlastností, kterou je zapotřebí dodržet, je zarovnání jednotlivých informačních polí na jejich přirozené zarovnání. Tedy všechna 8- ,16- ,32- a 64- bitová pole musí začínat na 8- ,16- ,32- a 64-bitové hranici. Pokud tomu tak není, je zapotřebí přidat výplň, aby byla tato vlastnost splněna. Zarovnání RadioTap hlavičky je znázorněno na obrázku 2.5.

```
struct rtapdata {
    uint8_t  antsignal;
    uint8_t  pad_for_tx_attentuation; // <-- přidaná výplň
    uint16_t tx_attenuation;
    uint8_t  flags;
    uint8_t  pad_for_rx_flags;        // <-- přidaná výplň
    uint16_t rx_flags;
} __attribute__ ((packed));
```

Obrázek 2.5: Zarovnání RadioTap hlavičky [1]

2.4.2 IEEE 802.11 hlavička

Pomocí IEEE 802.11 hlavičky *(nebo jen IEEE hlavičky)* [19] se přenáší informace důležité pro činnost bezdrátových sítí tohoto standardu. Mezi informacemi přenášenými v hlavičce nalezneme například adresy kam rámec v síti směřuje, informaci zdali ja rámec zašifrován a mnoho dalšího. Obecnou strukturu hlavičky znázorňuje obrázek 2.6. Tato struktura se ovšem může měnit dle významu rámce. Ne vždy hlavička musí obsahovat všechny adresy. Některé typy rámců mohou obsahovat i další položky. Jako příklad uveďme *beacon* rámec, který obsahuje informace o síti, například typ zabezpečení, podporované rychlosti přenosu dat a mnoho dalšího.



Obrázek 2.6: IEEE 802.11 hlavička

Význam jednotlivých položek je následující:

- **Frame Control** je bitové pole, kde význam jednotlivých bitů je následující:
  - **protocol** udává verzi použitého protokolu. V současné době je to hodnota 0.
  - **type** značí typ přenášeného rámce. Rozlišujeme rámce typu management (00), control (01) a datové (10).
  - **sub type** rozlišuje podtyp přenášeného rámce. Příkladem je beacon rámec (1000), který je typu management, nebo potvrzující rámec *(Acknowledgment nebo ACK)* (1101) typu control. Kompletní seznam podtypů rámců je uveden například v publikaci [19].
  - **to DS** informuje, zdali je rámec přenášen do distribučního systému.
  - **from DS** informuje, zdali je rámec přenášen z distribučního systému.
  - **more frag** je nastaven v případě, kdy jsou data rámce fragmentováná a přenášena postupně.
  - **retry** informuje o opětovném přenášení rámce.
  - **pwr mgmt** informuje příjemce rámce, že jeho odesilatel přešel do úsporného režimu z důvodu šetření energie.
  - **more data** je nastaven, pokud má odesilatel rámce pro příjemce uložených více rámců, které mu potřebuje doručit. Pomocí této informace se informuje příjemce, že může očekávat další data, a nemá tak přecházet do úsporného režimu.
  - **wep** někdy označován také jako *protected*, slouží k informování příjemce, že přenášená data jsou zabezpečena.
  - **order** zaručuje, že rámce a fragmenty rámců budou přenášeny uspořádaně, tak jak byly odeslány, za cenu vyšší režije na straně odesilatele a příjemce.
- **Duration/ID** pole má více použití. Jejich použití je rozlišeno dvěmi nejvíce významnými bity. Jako příklad použití je informace o době v mikrosekundách, po kterou je očekáváno, že bude médium obsazeno pro současný přenos.

- **Address** pole obsahují adresy stanic, které se zúčastní přenosu rámce. Jak jsou adresy nastaveny je dáno bity **to DS** a **from DS** a jejich popis je uveden v tabulce 2.2.
- **Seq-ctl** obsahuje číslo rámců, tak jak jsou vysílány. Každý další rámec má číslo větší o 1 oproti předchozímu. Celkové pole se skládá ze 4 bitů vyhrazených pro číslo fragmetu a 12 bitů vyhrazených pro sekvenční číslo. Pokud je rámec fragmentován, všechny tyto fragmenty mají stejné sekvenční číslo, mění se pouze číslo fragmentu.
- **Frame body** obsahuje data datového rámce. Pokud se jedná o management nebo control rámec může obsahovat další položky hlavičky.
- **FCS** je CRC *(cyclic redundancy check)* kontrolní součet hlavičky IEEE 802.11 a datové části rámce. Přítomnost kontrolního součtu je signalizována v RadioTap hlavičce.

To DS	From DS	Popis	Adresa 1	Adresa 2	Adresa 3	Adresa 4
0	0	komunikace v Ad-hoc síti	DA	SA	BSSID	nepoužita
1	0	komunikace směrem k AP	BSSID	SA	DA	nepoužita
0	1	komunikace směrem od AP	DA	BSSID	SA	nepoužita
1	1	komunikace v distribučním systému	RA	TA	DA	SA

Tabulka 2.2: Nastavení adresních polí rámce [19]

- **DA** - adresa cílové stanice, která se ve většině případů shoduje s adresou přijímající stanice
- **SA** - adresa zdrojové stanice, která se ve většině případů shoduje s adresou odesílající stanice
- **BSSID** - identifikátor sítě, který se v případě infrastrukturní sítě shoduje s adresou AP
- **RA** - adresa přijímající stanice
- **TA** - adresa odesílající stanice

Jak již bylo naznačeno výše, jednotlivé rámce se dělí na management *(rámce správy sítě)*, control *(řídící rámce)* a datové rámce. Každý z těchto rámců dále obsahuje několik podtypů rámců. Jednotlivé rámce obsahují řadu specifických položek hlavičky, které zajišťují funkčnost bezdrátové sítě. Představení všech těchto položek by bylo zdlouhavé, proto jej zájemci mohou naleznout například v publikaci [19] odkazované v závěru práce.

V následující části budou v krátkosti představeny některé typy rámců s popisem jejich významu v souvislosti s 802.11 sítěmi.

Management rámce

Nejrozsáhlejší skupina rámců jsou rámce pro správu sítě. Rámce slouží především k vytvoření spojení mezi klientem a přístupovým bodem. Ale také se zde nachází například rámce sloužící k šíření informací o bezdrátové síti a další typy rámců. Popis vybraných typů rámců se nachází níže.

- **Beacon** rámce vysílá v pravidelných intervalech přístupový bod sítě. Jejich úkolem je oznámení existence sítě, tak aby ji klienti mohli vyhledat a připojit se k ní. Uvnitř beacon rámce se přenášejí informace důležité pro vytvoření spojení mezi klientem a přístupovým bodem.
- **Probe request** slouží k získání informací mobilní stanicí o existujících 802.11 sítích v jejím okolí.
- **Probe response** je odpovědí na probe request. Pokud přístupový bod zachytí požadavek probe request a jeho parametry se shodují s parametry sítě odpovídá na něj pomocí probe response.
- **Association request** použije mobilní stanice při pokusu o připojení se k síti.
- **Reassociation request** je použit v případě, kdy se stanice pohybuje mezi základnovými stanicemi stejné ESS (2.1.1). Jakmile dojde k přechodu od jednoho přístupového bodu k druhému je potřeba, aby se stanice znovu asociovala. Stanice může být vyzvána k opětovné asociaci i v případě, kdy dojde k dočasnému opuštění oblasti pokrytí signálu daného přístupového bodu.
- **Association a reassociation response** je odpovědí přístupového bodu na pokus mobilní stanice se připojit do sítě.
- **Disassociation** rámce slouží k ukončení relace navázané pomocí association request.
- **Authentication** rámec slouží k ověření identity bezdrátové stanice, která se připojuje k síti.
- **Deauthentication** rámce sloužící k ukončení relace navázané pomocí authentication.

Control rámce

Rámce této skupiny pomáhají při doručování datových rámců.

- **Request to Send** *(RTS)* slouží k získání kontroly nad médiem pro přenos datových rámců. Pokud stanice požaduje vysílat datový rámec, vyšle RTS rámec směrem k přijimající stanici. Jakmile přijímající stanice odpoví pomocí rámce typu CTS, stanice má jistotu, že médium je volné k přenosu.
- **Clear to Send** *(CTS)* je odpovědí na rámec RTS.
- **Acknowledgment** *(ACK)* slouží jako pozitivní potvrzení přenosu datového rámce. ACK se zasílá vždy pokud byl datový rámec úspěšně přenesen bez poškození.
- **Power-Save Poll** *(PS-Poll)* zasílá stanice po probuzení z úsporného režimu a dává tak najevo přístupovému bodu, že je schopná přijmout rámce, které jí měli být doručeny během režimu spánku.

Datové rámce

K přenosu dat slouží datové rámce. Data uvnitř rámce mohou být přenášena v otevřené, případně zašifrováné podobě. Popis jednotlivých metod použitých k zabezpeční rámců můžeme naleznout v následující části.

2.5 Zabezpečení bezdrátových sítí standardu 802.11

Bezdrátové sítě s sebou přináši i jedno riziko. Díky použitému médiu, které je přístupné všem, je možné odchytit cizí komunikaci. Pokud tato komunikace není šifrovaná je možné ji interpretovat a zjistit, co je obsahem přenášené zprávy. Běžné bezdrátové sítě využívající rozprostřeného spektra DSSS mohou částečně spoléhat na princip této metody. Tedy komunikaci smí odchytit pouze ten, kdo zná pseudonáhodnou posloupnost tvořící čipovací sekvenci. V případě bezdrátových sítí standardu 802.11 se na toto spolehnout nelze. Posloupnost tvořící čipovací sekvenci je standardizovaná a používá ji každá stanice v síti. Z tohoto důvodu může komunikaci dvou bezdrátových stanic odchytnout kdokoliv další. A proto je potřeba použít jíného mechanismu zabezpečení přenášených dat.

2.5.1 Wired Equivalence Privacy

Standard 802.11 zahrnuje metodu pro šifrování komunikace zvanou **WEP** *(Wired Equivalence Privacy)*. Metoda je založena na šifrovacím algoritmu RC4 s tajným klíčem, jehož délka byla stanovena na 40 bitů. Tato délka klíče se ovšem zdála jako nedostačující, a tak existuje i varianta s velikostí klíče 104 bitů [17].

Princip činnosti metody WEP zachycuje obrázek 2.7. Ze zprávy se vytvoří kontrolní součet pomocí CRC-32 hashovací funkce [25]. Tento kontrolní součet se poté přiloží k přenášené zprávě. Takto nově vzniklá zpráva je poté připravena k zašifrování. Šifrování se provádí pomocí operace XOR s pseudonáhodnou posloupností zvanou key stream. Tento key stream je vytvořen již zmiňovaným algoritmem RC4 z tajného klíče, který je před vstupem do RC4 zkombinován s inicializačním vektorem. Tento inicializační vektor, jehož délka je vždy 24 bitu, tedy i v případě je-li klíč dlouhý 104 bitů, by měl být unikátní pro každý rámec. Tedy i v případě, je-li rámec zasílán opakovaně. Zašifrovaná zpráva společně s inicializačním vektorem se poté přenese pomocí datového rámce.

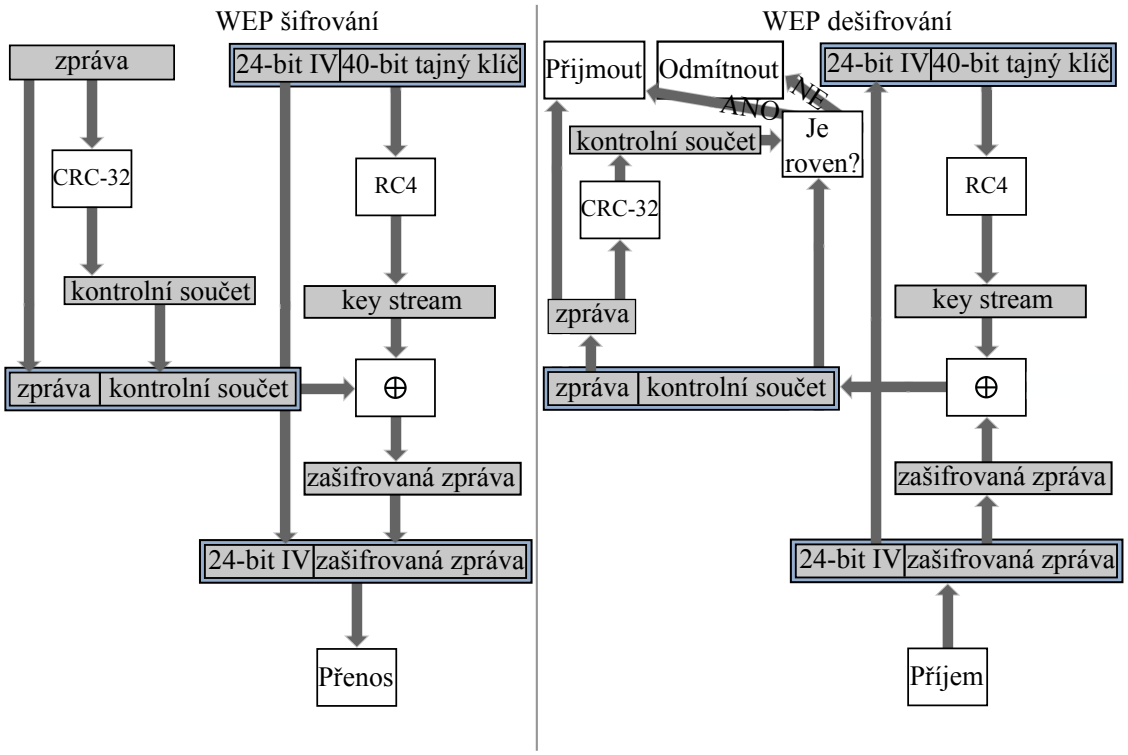
Po přijetí rámce se podobným postupem zpráva dešifruje. Z rámce se vyjme inicializační vektor, který se opět zkombinuje s tajným klíčem, a pomocí RC4 se vygeneruje stejný key stream. Pomocí operace XOR s vygenerovaným key streamem a přijatou zašifrovanou zprávou získáme původní, nezašifrovanou zprávu. Z této nezašifrované zprávy oddělíme kontrolní součet, který následně porovnáme s nově vygenerovaným kontrolním součtem přijaté zprávy. Pokud se tyto součty shodují, zpráva byla přenesena bez porušení a je tedy možné ji doručit. [15, 23]

Nedostatkem metody WEP je znovupoužití inicializačního vektoru, který by se neměl používat opakovaně. Ten se však opětovně použije nejdéle po 2^{24} zašifrovaných rámcích. Toto číslo je ovšem ve skutečnosti mnohem menší vlivem narozeninového paradoxu [30, 14]. V okamžiku, kdy se začne inicializační vektor opakovat, vzniká riziko útoku na bezpečnostní mechanismus WEP.

Standard 802.11 dále definuje dvě metody autentizace. Proces autentizace slouží k ověření totožnosti bezdrátové stanice a rozhodnou tak, zdali může být stanice připojena k síti. Těmito metodami jsou **Open System Authentication** a **Shared Key Authentication** [17]. Metoda jmenovaná jako druhá používá pro základ své činnosti šifrovací metody WEP. Obě tyto metody jsou v krátkosti popsány níže.

Open System Authentication

Ověření stanice za použití této motody je jednoduchý dvoukrokový proces. V prvním kroku zašle stanice, která se chce připojit do sítě, rámec typu *Managemet* a s nastaveným podtypem *Authentication*. Jako autentizační algoritmus je nastaven *"Open System"* a jako identita stanice poslouží její 48 bitová MAC adresa. Tento rámec má nastaveno sekvenční číslo rovno 1. V druhém kroku zašle přijímající stanice výsledek o připojení. Výsledek je zaslán v rámci téhož typu, jako byl rámec



Obrázek 2.7: WEP - šifrování a dešifrování

přijatý, a který má jako autentizační algoritmus nastaveno *"Open System"* a sekvenční číslo rovno 2. Pokud se podařilo stanici připojit k síti bude výsledek roven *"0"*.

Nevýhodou této metody je, že se do sítě může přihlásit každý, kdo zná správnou MAC adresu. Zjistit vhodnou adresu není však nijak složité. Stačí odposlechnout komunikaci na síti a tuto adresu si z přenášených rámců přečíst, protože adresa se přenáší uvnitř rámce vždy jako otevřený text, tedy v nešifrované podobě. Poté si postačuje nastavit MAC adresu síťové karty na adresu zjištěnou z komunikace a přihlásit se do sítě.

Shared Key Authentication

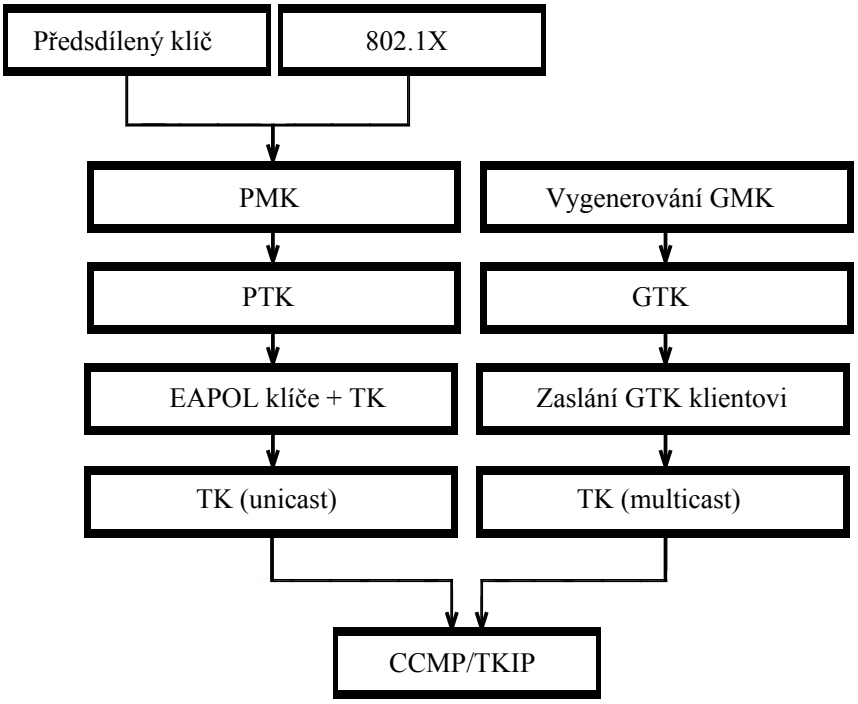
Druhou metodou je metoda *Shared Key Authentication*. Tato metoda provede ověření stanice ve čtyřech krocích. Ve všech krocích se používají rámce stejného typu. Tímto rámcem je *Managemet* rámec s nastaveným podtypem jako *Authentication* a autentizačním algoritmem jako *"Shared Key"*. V prvním kroku zasílá stanice, která se chce připojit do sítě, rámec obsahující 48 bitovou MAC adresu stanice a se sekvenčním číslem 1. Po přijetí tohoto rámce provede přijímající stanice ověření MAC adresy s adresou, kterou má uloženou, a pokud se shodují zašle zpět rámec obsahující *Challenge text*, nazývaný také jako výzva, se sekvenčním číslem 2. Challenge text je náhodné 1024 bitové číslo, které bude použito k ověření stanice. Ve třetím kroku je toto náhodné číslo klientem zašifrováno pomocí WEPu a tajného klíče a zasláno zpět společně s inicializačním vektorem a sekvenčním číslem rovno 3. Jakmile stanice přijme zašifrovanou zprávu, provede ověření pomocí dešifrování. Pokud se dešifrovaný text shoduje s výzvou, je stanice úspěšně autentizována. O výsledku je stanice informována pomocí rámce se sekvenčním číslem 4. [15]

2.5.2 Standard 802.11i

Zabezpečovací a autentizační mechanismy definované v původním standardu 802.11 se staly postupem času nedostačujícími, a tak bylo v roce 2004 schváleno nové rozšíření nazvané **802.11i**. Toto rozšíření s sebou přináší změny v zabezpečení datových rámců, v autentizaci bezdrátových stanic nebo také změny v používání kryptografických klíčů.

Hierarchie klíčů

Jednou ze změn, kterou s sebou přináší standard 802.11i, je i změna v používání klíčů. Ve standardu 802.11i se místo jednoho klíče používá kolekce klíčů, kde každý z klíčů zajišťuje jinou bezpečnostní funkci. Celková kolekce klíčů, označovaná také jako hierarchie klíčů, je zachycena na obrázku 2.8. Pro šifrování multicastových[3] a broadcastových[4] rámců zasílaných AP je určen GTK klíč *(Group Transient Key)*, který je odvozen z GMK klíče *(Group Master Key)*, a mění se vždy, když se do sítě připojí nová stanice. GMK klíč generuje AP a slouží k již zmíněnému odvozování GTK klíčů. [23, 30]



Obrázek 2.8: Hierarchie klíčů

Pro šifrování unicastových[5] rámců slouží PTK klíč *(Pairwise Transient Key)*. Klíč PTK je generován z PMK klíče *(Pairwise master key)*, jehož odvození závisí na použité metodě autentizace. Je-li použita autentizace pomocí 802.1X [15] je PMK klíč odvozen z MSK klíče *(Master session key)* neboli z klíče dodaného autorizačním serverem. V případě použití předsdíleného klíče PSK *(Pre-Shared key)* je PMK klíč tvořen právě tímto klíčem. PTK klíč je rozdělen do třech subklíčů. Do KCK *(Key Confirmation Key)* a KEK *(Key Encryption Key)* klíčů, které se používají při generování

[3] rámce zasílané skupině cílových stanic
[4] rámce zasílané všem stanicím v síti
[5] rámce adresovány jedné stanici

EAPOL rámců, a TK klíče *(Temporal key)* sloužícího k šifrování unicastových rámců. Délky jednotlivých klíčů, které zobrazuje tabulka 2.3, se liší dle použité zabezpečovací metody. [15]

	TKIP TK	**CCMP TK**	**KCK**	**KEK**	**Celková délka**
TKIP PTK	256		128	128	512
CCMP PTK		128	128	128	384
TKIP GTK	256				256
CCMP GTK		128			256

Tabulka 2.3: Délky klíčů použitých v 802.11i [15]

EAPOL rámce slouží k vytvoření bezpečné komunikace a pro bezpečnou výměnu vygenerovaných klíčů mezi klientskou stanicí a AP. Při jejich generování je použito klíče KCK použitého k vytvoření kontrolního součtu rámce a klíče KEK použitého k šifrování vlastního obsahu přenášeného v rámci. [15]

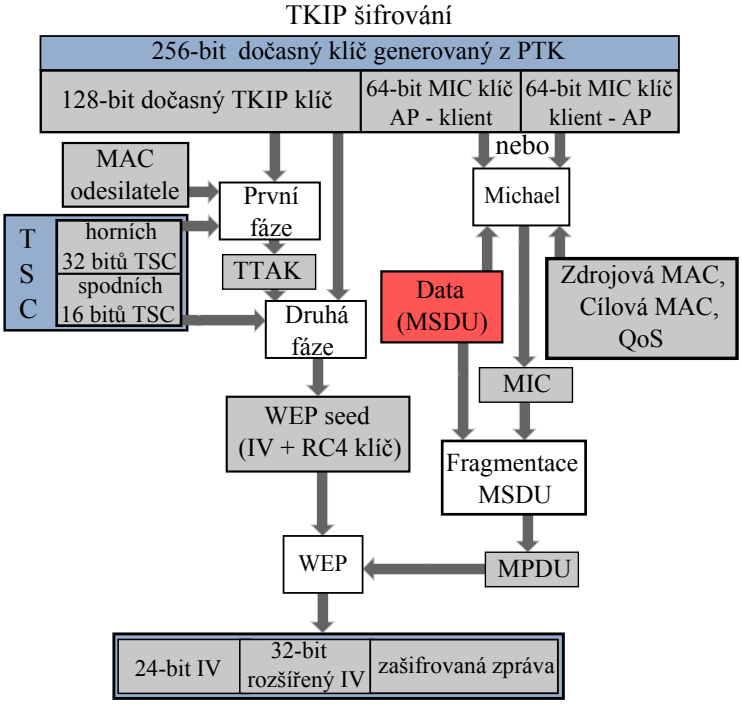
TKIP

Standard 802.11i definuje dvě metody pro šifrování. První metodou je TKIP *(Temporal Key Integrity Protocol)*, která je používaná u zabezpečení WPA *(Wi-Fi Protected Access)*. TKIP byl navržen jako náhrada metody WEP, tak aby pracoval na stejném hardwaru. Tedy, aby se změny týkaly pouze programové části. [23]

První změnou, kterou TKIP přináší je výpočet kontrolního součtu MIC *(Message integrity code)*. Pro výpočet kontrolního součtu je použit algoritmus Michael, jehož vstupy jsou, tak jak zobrazuje obrázek 2.9, zdrojová adresa, cílová adresa, nezašifrovaná zpráva, informace o kvalitě služeb QoS *(Quality of Service)* a příslušný klíč. Na rozdíl od následující metody, je zde použita pro výpočet kontrolního součtu celá nezašifrovaná zpráva, tedy zpráva ještě před případnou fragmentací. Přijímající strana potom musí pro výpočet kontrolního součtu postupně přijmout všechny fragmenty a až poté provést jeho výpočet. Pokud algoritmus Michael narazí při kontrole kontrolního součtu na dva špatné kontrolní součty během jedné minuty, vynutí si 60 vteřinový výpadek sítě a musí být stanoveny nové klíče GTK a PTK. [23]

Druhou změnou, kterou přináší TKIP, je použití klíče pro šifrování. Zatímco u metody WEP byl použit vždy jeden klíč zkombinovaný s inicializačním vektorem, u metody TKIP je pro každý rámec vypočten klíč dočasný. Tento klíč je vypočten ve dvou fázích. V první fázi se použijí statická data, kterými jsou 128 bitu z PTK klíče, MAC adresa odesilatele a horních 32 bitů inicializačního vektoru. Jako hodnota inicializačního vektoru je v případě metody TKIP použita hodnota z čítače TSC *(TKIP Sequence Counter)*, který je s každým novým rámcem inkrementován. Jako vstup druhé fáze je použita hodnota z fáze předchozí, spodních 16 bitů z inicializačního vektoru a opět 128 bitů klíče PTK. Pomocí tohoto postupu získáme dočasný klíč, kterým je zašifrován přenášený rámec. [15, 23]

Šifrování rámce probíhá shodnou metodou jako šifrování u metody WEP. Pomocí RC4, jejímž vstupem je dočasný klíč, označován také jako WEP seed, je vygenerována pseudonáhodná posloupnost. Pomocí této posloupnosti a operace XOR je provedeno zašifrování vlastních dat. Celkový postup výroby klíče, výpočtu kontrolního součtu a následného šifrování zobrazuje obrázek 2.9.



Obrázek 2.9: TKIP - šifrování

CCMP

Druhou metodou definovanou v 802.11i, kterou lze použít pro šifrování rámců, je metoda CCMP *(Counter mode with cipher-block chaining message authentication code)* používaná v zabezpečení WPA 2. Metoda CCMP představuje nejvyšší úroveň zabezpečení a zajištění integrity, kterou 802.11i nabízí. CCMP používá ke své činnosti blokové šifry AES *(Advanced encryption standard)* s velikostí bloku 128 bitů. Blokové šifry AES je použito v CCMP metodě jak pro vytvoření kontrolního součtu MIC, tak i k vlastnímu šifrování dat. V každém případě je ale použit jiný režim této blokové šifry. Pro výpočet kontrolního součtu je použita šifra AES v režimu CBC *(Cipher-block chaining)*. Vlastní data jsou pak šifrována pomocí režimu *Counter mode*.

Cipher-block chaining

V režimu CBC je každý šifrovaný blok před vstupem do blokové šifry AES XORován s předchozím výstupem, tedy se zašifrovaným blokem. První blok je XORován se 128 bitovým inicializačním vektorem. [21]

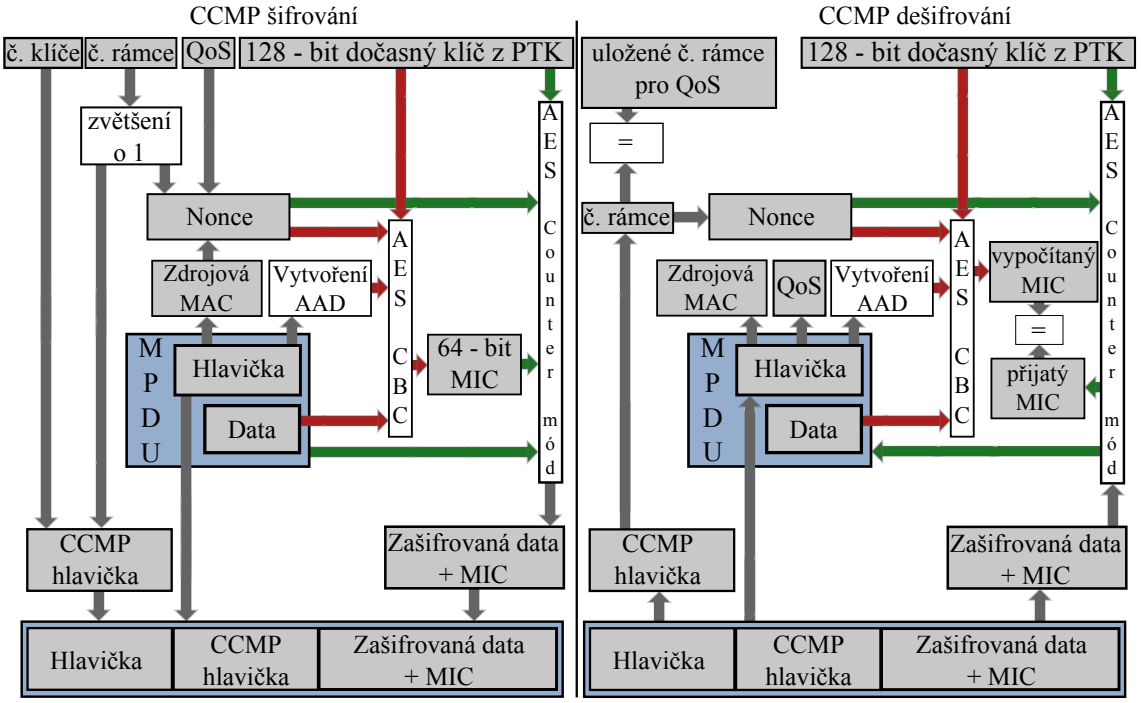
Counter mode

Režim counter mode kombinuje 24 bitový čítač se 104 bitový číslem nonce. Tuto kombinaci čísel následně zašifruje pomocí šifry AES a výsledek XORuje se šifrovaným blokem. Poté inkremetuje čítač a celý postup se zopakuje pro další blok. Výhodou tohoto režimu je, že může být aplikován paralelně a umožnit tak urychlení celého procesu šifrování. [21]

Výpočet kontrolního součtu probíhá z takzvaných AAD hodnot *(Additional authentication data)*. Těmito hodnotami jsou adresní pole, číslo fragmetu a informace o kvalitě služeb QoS. Hodnoty

ADD společně s obsahem zprávy jsou zašifrovány pomocí šifry AES v režimu CBC, čímž získáme 64 bitů kontrolního součtu. Jelikož výsledkem algoritmu AES je zašifrovaný blok velikosti 128 bitů, provede se odstranění dolních 64 bitů z tohoto bloku. [23]

Proces šifrování je u metody CCMP jednodušší než v případě metody TKIP. Postup šifrování a následného dešifrování zachycuje obrázek 2.10. Vstupem do procesu šifrování jsou zabezpečovaná zpráva spojena s vypočteným kontrolním součtem, hodnoty AAD, TK klíč a hodnota nonce. Ta se s každým přenášeným rámcem zvětšuje o jedničku. Proces dešifrování pak probíhá obdobně jako proces šifrování.



Obrázek 2.10: CCMP - šifrování a dešifrování

Tímto byla v krátkosti představena problematika bezdrátových sítí standardu 802.11. Tento standard společně s jeho rozšířeními je natolik rozsáhlý, že jeho představení by vydalo na samostatnou publikaci. Některé specifikace tohoto standardu však nejsou z pohledu práce důležité, proto bylo vybráno a představeno jen několik málo částí, které budeme potřebovat v následujících kapitolách.

Kapitola 3

Analýza a návrh

Bezdrátové sítě standardu 802.11 jsou používány řadou uživatelů především k přístupu do sítě Internet. V předchozí kapitole bylo popsáno, jak je přenos dat v těchto sítích zprostředkován. Vlivem použitého média jsou ale tyto sítě náchylnější na případné útoky než sítě drátové. Cílem takovéhoto útoku může být například snaha útočníka získat citlivé informace, které jsou přenášeny v rámci komunikace. K tomu, aby se případný útočník dostal k citlivým informacím, mu postačuje odposlechnout komunikaci, která právě probíhá prostřednictvím bezdrátového média. Pokud by tato komunikace nebyla dostatečně zabezpečena, mohl by si útočník tyto informace jednoduše přečíst. Dalším případným útočníkovým cílem může být pouhé získání bezplatného přístupu k Internetu. Toho by v případě špatně zabezpečené sítě docílil pomocí pouhého připojení do této sítě.

K jednotlivým útokům by nemohlo docházet, pokud by se v těchto sítích nevyskytovaly zranitelnosti, které jsou následně využity k případnému útoku. Abychom mohli tyto zranitelnosti odhalit, je zapotřebí mít vhodné nástroje, pomocí kterých bychom zkoumali bezpečnost komunikace v síti.

3.1 Rozbor stávajících řešení

V současnosti existuje řada řešení zabývající se bezpečností bezdrátových sítí standardu 802.11. Každé z těchto řešení má různé možnosti jak analyzovat zabezpečení sítě. Některé umožňují pracovat přímo s použitými rámci, jiné nás od nich odprošťují pomocí předdefinovaných funkcionalit. V následující části je představeno několik z nich. V krátkosti je zde popsáno, co jednotlivé aplikace umožňují, jaké mají přednosti před ostatními a co jim naopak schází.

Aircrack-ng

Prvním z řešení, které je možné použít pro analýzu zabezpečení bezdrátové sítě je nástroj aircrack-ng [4]. Tento nástroj je ve skutečnosti sada aplikací sloužící k vytváření útoků na bezdrátové sítě Wi-Fi. Tyto útoky je následně možné využít k již zmíněné analýze zabezpečení.

Mezi aplikacemi můžeme například naleznout aplikaci airodump-ng. Tato aplikace slouží ke sledování a zaznamenávání provozu bezdrátových sítí nacházejících se v okolí počítače, na kterém je aplikace právě spuštěna. Pomocí této aplikace je umožněno zaznamenat komunikaci námi sledované stanice. Tento záznam může být následně analyzován pomocí aplikace aircrack-ng, nesoucí stejné pojmenování jako celá sada nástrojů. Tato analýza je prováděna za účelem získání šifrovacího klíče použitého k zabezpečení komunikace. Aircrack-ng umožňuje získat šifrovací klíč použitý k zabezpečení rámců pomocí metod WEP, WPA i WPA2. V případě metody WEP využívá nástroj zranitelnosti vycházející z návrhu zabezpečení. Podrobnější popis této metody je možné dohledat na domovské stránce nástroje aircrack-ng [4]. V případě metod WPA a WPA2 využívá aplikace

slovníkového útoku za účelem získání požadovaného klíče. Pomocí této metody je procházen seznam hesel, která jsou postupně zkoumána. Pokud se zde heslo použité k zabezpečení sítě nachází, aplikace jej nalezne. V opačném případě se nalezení hesla nezdaří.

Další z řady aplikací je aplikace aireplay-ng. Ta slouží primárně ke generování datového toku neboli rámců použitých při komunikaci ve Wi-Fi sítích. Generovaný datový tok je následně využit k některému z předdefinovaných útoků. Aplikace například umožňuje generovat rámce typu *De-authentication*, které způsobí odpojení klienta ze sítě. Tohoto je využito při útoků snažící se získat přístup do této sítě. Klient se po odpojení pokouší opětovně připojit k bezdrátové síti, čímž generuje datový provoz. Ten je následně zaznamenán pomocí airodump-ng. Z takto zaznamenaného datového toku je následně pomocí aplikace aircrack-ng získán klíč použitý k zabezpečení datových rámců. Generování deauthentication rámců není jediná možnost, kterou aireplay-ng poskytuje. Aireplay-ng obsahuje řadu dalších předdefinovaných útoků, které pracují s generovanými rámci. Celkový výčet je možné naleznout na stránkách aircrack-ng [4].

Aircrack-ng ale neobsahuje pouze výše jmenované aplikace. K dispozici je řada dalších aplikací, které lze využít při bezpečnostní analýze sítě. Jejich použití je však z pohledu této práce méně zajímavé, a proto jejich popis zde nebude uveden.

Řešení aircrack-ng je mocným nástrojem, umožňující analyzovat bezdrátové sítě. Mezi jeho přednosti patří řada aplikací, které lze k tomuto účelu využít. Jedinou jeho nevýhodou je nemožnost vytvářet nové rámce, které by mohli být následně použity. Vlivem tohoto nedostatku jsme omezeni pouze na použití útoků, které jsou v tomto nástroji předdefinovány. Tento nástroj, případně jeho součásti, je možné následně využít v kombinaci s dalšími nástroji, které mohou doplnit absenci některých funkcí.

Scapy

Dalším nástrojem, který může být využit pro bezpečnostní audit bezdrátových sítí, je aplikace Scapy [3]. Tento nástroj v sobě implementuje funkcionalitu řady dalších aplikací. Těmito aplikacemi jsou například arping případně arpspoof sloužící k manipulaci s ARP pakety *(Address Resolution Protocol)*. Dále pak nmap sloužící především ke skenování sítí, nebo nástroj tcpdump sloužící ke sledování a zaznamenávání komunikace na síti. Scapy dále umožňuje pokročilou tvorbu a dekódování paketů široké škály protokolů.

Scapy umožňuje pomocí jednoduchého jazyka definovat hlavičky řady protokolů. Při popisu se zaměřuje především na popis hlaviček protokolů vyšších vrstev. Ale je možné, byť v omezené míře, vytvářet i rámce používané v bezdrátových sítích standardu 802.11. Omezení se týká především počtu položek, které jsme v rámci schopni nastavit, čímž nám například neumožňuje vytvořit kompletní *beacon* rámec, tak jak by byl generován přístupovým bodem sítě. Jak je možné vytvořit rámec v aplikaci scapy znázorňuje ukázka 3.1.

Ukázka 3.1: Definice rámce pomocí Scapy

```
paket = IP(ttl = 10)
paket.dst = "192.168.0.1"
```

Pomocí příkazů výše jsme provedli definici nového IP *(Internet Protocol)* paketu s adresou cíle 192.168.0.1 a hodnotou ttl *(time to live)* rovné deseti.

Po nadefinování rámce, případně paketu, umožňuje i tato aplikace jejich následné zasílání. Scapy definuje řadu funkcí, pomocí nichž je možné nadefinované rámce zaslat. První z těchto funkcí je prosté zaslání na specifikované rozhraní. V tomto případě je paket pouze zaslán. Další možností je zaslání s následnou odpovědí. Pokud zasíláme paket, na který očekáváme odpověď, umožňuje

scapy použít funkci, která provede spárování zaslaného rámce s jeho odpovědí. Po odeslání rámce čeká na přijetí odpovědi, kterou následně uživateli zobrazí.

Scapy obsahuje i podporu pro práci s certifikáty, které je možné využít v protokolech vyšších vrstev. Nikde jsme ovšem nenašli podporu zabezpečení rámců standardu 802.11 pomocí metod WEP, TKIP případně CCMP. Pomocí nástroje scapy jsme schopni definovat mnoho typů rámců a paketů. Mezi nevýhody však patří omezenost definice rámců použitých ve Wi-Fi sítích. Využití tohoto nástroje však může být například ke snadnému generování hlaviček protokolů vyšších vrstev, které bychom mohli následně umístit jako datový obsah rámců standardu 802.11.

Zulu

Třetím představitelem nástrojů, které mohou být využity ke generování Wi-Fi rámců, je aplikace Zulu [7]. Aplikace Zulu je určena ke snadnému generování rámců, díky níž je umožněno rychlé a snadné ladění, případně sondování sítí standardu 802.11. Aplikace se ovládá z terminálu operačního systému Linux. Definice rámců se zadává jako jednotlivé parametry při spuštění aplikace. Ta po svém spuštění tyto parametry zpracuje a vygeneruje výsledný rámec. Tento rámec následně také zašle na specifikované rozhraní. Použití Zulu je velmi snadné. Po spuštění aplikace bez zadaných parametrů se vypíše seznam možných voleb. Ukázka následné tvorby rámce je zachycena na příkladu 3.2 níže.

Ukázka 3.2: Definice rámce pomocí Zulu

```
./zulu -t beacon -i wlan0 --ssid NovaSit
```

V ukázce byl vytvořen rámec typu *beacon*, kterému bylo nastaveno *SSID* na hodnotu "NovaSit". Rozhraní, na které je rámec po vygenerování zaslán, je specifikováno pomocí parametru *i*. Zde bylo použito rozhraní pojmenované *wlan0*.

Zulu je nástrojem, který lze využit pro snadné ladění bezdrátových sítí. Umožňuje poměrně v krátké době jednoduše vytvořit požadovaný rámec. Způsob ovládání, který umožňuje tak rychle a snadno vytvořit požadovaný rámce, se ale stává nepříjemným v okamžiku, kdy požadujeme v rámci nastavit vícero položek. V tomto okamžiku se stává ovládání nevhodné a může vest často k chybám. U aplikace dále nebyla nalezena možnost, která by umožňovala obsah datových rámců zabezpečit některou ze tří metod, již dříve jmenovaných.

Další řešení

Uvedený výčet nástrojů jistě není konečný. Existuje řada dalších aplikací, které lze více či méně použít s cílem ověřování bezpečnosti počítačových sítí. Řada z těchto nástrojů se však specializuje pouze na protokoly vyšších vrstev. Uveďme například aplikaci Nemesis [12], jejímž cílem je injekce paketů protokolů IP, TCP, DNS a řady dalších. Tato ale i další aplikace[1] jsou vhodné k testování počítačových sítí. Řada z nich ale nelze využít ke generování rámců standardu 802.11. Některé z nich pouze v omezené míře.

[1] přehled dalších nástrojů je možné naleznout například ve zdroji [2] odkazovaném v závěru práce

3.2 Cíle práce

V předchozí části bylo představeno několik stávajících řešení, které lze využít při bezpečnostním auditu Wi-Fi sítí. Každé z představených řešení má své přednosti, ale i jisté nedostatky. Bylo by vhodné z předností jednotlivých aplikací vytvořit aplikaci jednu, která by odstraňovala nedostatky každé z nich.

Hlavním nedostatkem, který u většiny z výše jmenovaných nástrojů můžeme naleznut, je absence uživatelské definice rámců standardu 802.11. Některé z nástrojů toto umožňují alespoň částečně. Například pomocí omezené množiny vlastností, které lze u rámců definovat. Tímto nástrojem je například aplikace Scapy. U žádné ze jmenovaných aplikací jsme nenalezli možnost, jak vytvořený datový rámec zabezpečit.

Bylo by tedy vhodné realizovat řešení, které nebude uživatele omezovat v definici struktury rámců. Definice by měla probíhat pomocí vhodného popisu. Tento popis by měl zahrnovat nejenom definici struktury rámců, ale také možnost definovat zabezpečení datových rámců. Dále by tento popis měl být rozšířen o možnost s již popsanými rámci manipulovat. Manipulace by měla zahrnovat generování rámců, zasílání rámců, ale i možnost zobrazit strukturu rámců, ze které by bylo patrné, jak je rámec definován.

3.3 Specifikace požadavků

Vlastnosti, které by měla aplikace splňovat, můžeme shrnout do funkčních a nefunkčních požadavků. V případě funkčních požadavků se jedná o požadavky na funkcionalitu výsledné aplikace. Zde můžeme jmenovat například generování a zasílání navržených rámců, šifrování datových rámců a řadu dalších požadavků. Nefunkční požadavky shrnují požadavky na aplikaci, které se nepřímo týkají její funkcionality. Můžeme zde naleznout například požadavek na snadný a přehledný popis rámců. Shrnutí jednotlivých požadavků můžeme naleznout níže.

Funkční požadavky

- popis IEEE a RadioTap hlavičky
- generování rámců
- šifrování datových rámců
- zasílání vygenerovaných rámců
- ukládání popisu rámců

Nefunkční požadavky

- snadná manipulace s navrženými rámci a jejich opětovné použití
- generování rámců z popisu zadaný uživatelem
- použití programovacího jazyku C případně C++

3.4 Analýza možných řešení

Na základě specifikovaných požadavků můžeme provést návrh možných řešení, které by řešily cíle práce. Popis jednotlivých návrhu můžeme naleznout dále.

Parametry aplikace

Prvním z navrhovaných řešení je použití popisu rámců, který využívá aplikace Zulu popsaná v části 3.1 Rozbor stávajících řešení. Postačovalo by provést rozšíření stávajícího popisu o další vlastnosti. Například pro definici datového rámce bychom přidali další volby sloužící k definici datového obsahu a k definici použité metody zabezpečení.

Ale jak již bylo u popisu aplikace Zulu napsáno, je toto ovládání nevhodné při popisu rámců obsahující větší množství nastavovaných vlastností. Další nepříjemnost, která s tímto popisem vzniká, je komplikované opětovné použití rámce. Uživatel by byl nucen použit vhodného skriptovacího jazyka, který by mu toto umožnil. Proto toto řešení považujeme za nevhodné k dalšímu návrhu.

Textový popis rámců

Dalším řešením popisu rámců je možnost vytvoření popisu rámce pomocí textového editoru. Tento popis by obsahoval veškeré definice, které by byly zapotřebí k vygenerování rámce. Ten by se následně pomocí jednoho parametru předal aplikaci, která by z něj vygenerovala daný rámec. Na závěr by jej umožňovala zaslat na specifikované rozhraní. Výhodou tohoto popisu by byl pohodlnější zápis pro uživatele. Stal by se přehlednějším a umožnil by tak předcházet případným chybám. Pokud by ale v tomto popisu chyba nastala, postačovalo by, aby uživatel popis opětovně otevřel a chybu opravil.

Bohužel i toto řešení přináší komplikaci týkající se opětovného použití navrženého rámce. Uživatel by byl opět nucen využít vhodného skriptovacího jazyka, který by mu umožnil opakovaně používat popsané rámce. Proto i toto řešení považujeme za nevhodné.

Interpret rámců

Posledním z navrhovaných řešení je vytvoření aplikace, která bude obsahovat interpret. Ten bude umožňovat popis nových rámců a následnou manipulaci s již vytvořenými rámci. Podobné řešení představuje nástroj Scapy, který byl popsán již dříve. V tomto interpretu by mělo být možné definovat proměnnou, která bude představovat námi popisovaný rámec. Této proměnné bychom následně mohli nastavovat požadované vlastnosti, mohli bychom definovat, zdali bude rámec zabezpečen a případně jakou metodou. Jednotlivé definice by mělo být možné provést pomocí jednoduchých příkazů.

Tento interpret by následně měl obsahovat sadu příkazů, pomocí níž by bylo možné s takto popsanými rámci pracovat. Měl by, jak bylo stanoveno v požadavcích, umožňovat z tohoto popisu rámec vygenerovat a v případě požadavku jej zaslat na požadované bezdrátové rozhraní.

3.5 Volba použitých nástrojů a knihoven

Následující část se zabývá volbou programovacího jazyka a volbou nástrojů, které byly využity při realizaci vlastní aplikace. Dále zde můžeme naleznost krátký popis použitých knihoven, které byly využity při vlastní realizaci aplikace.

Jazyky C a C++

Jako programovací jazyk, který byl použit k implementaci aplikace, byl zvolen jazyk C++ v kombinaci s jazykem C. Jazyk C++ je objektově orientovaným programovacím jazykem, který byl původně vyvinut jako rozšíření populárního jazyka C [31]. Dalo by se říci, že jazyk C je až na několik definovaných výjimek podmnožinou jazyka C++. Při původním návrhu jazyka na toto byl opravdu kladen důraz, byť ne za každou cenu. V jazyku C existují konstrukce, které v jazyku C++ nelze použít. Dále v obou jazycích existují shodné konstrukce, které mají ale v každém z nich odlišný význam. Těchto rozdílů je ale tak málo a téměř se tyto konstrukce nepoužívají, že je možné zdrojové soubory napsané v jazyku C překládat pomocí překladačů jazyka C++. Velkou výhodou jazyků C a C++ je rychlost aplikace v nich napsaná. Rychlost této aplikace lze srovnávat s aplikací napsanou přímo v asembleru [2].

Analyzátory jazyka

Při realizaci jazyka sloužícího k popisu rámců bylo zapotřebí vyvinout lexikální a syntaktický analyzátor. S cílem ušetřit si práci při realizaci aplikace a zároveň vytvořit si možnost snadné editace jednotlivých analyzátorů, bylo použito nástrojů, které slouží k jejich automatické tvorbě. K tomuto účelu lze dobře využít nástrojů LEX a YACC, které slouží k automatické tvorbě již zmíněných analyzátorů. V následující části budou tyto nástroje v krátkosti představeny.

A Lexical Analyzer Generator

Nástroj *A Lexical Analyzer Generator*, zkráceně LEX, napomáhá při tvorbě lexikálních analyzátorů [5]. Vstupem tohoto nástroje je popis jednotlivých lexikálních symbolů v podobě regulárních výrazů a seznam akcí, které jsou s každým symbolem asociovány. Lex následně z tohoto popisu provede vygenerování lexikálního analyzátoru. Tento analyzátor provádí čtení vstupu uživatele, ve kterém hledá nejdelší shodu s definovaným lexikálním symbolem. Při shodě vykoná odpovídající akci, která byla definována u daného symbolu. Touto akcí může být navrácení odpovídajícího tokenu, který je následně zpracován syntaktickým analyzátorem.

Yet Another Compiler-compiler

Dalším nástrojem, který byl zvolen pro vývoj aplikace, je nástroj *Yet Another Compiler-compiler*, zkráceně YACC [5]. Tento nástroj slouží k tvorbě syntaktického analyzátoru, který je generován z gramatiky zapsané pomocí Backus-Naurovy formy. Analyzátor následně čte výstup lexikálního analyzátoru a kontroluje zapsaná pravidla definovaná uživatelem. Jakmile najde shodu, provede akci, která je asociovaná s daným pravidlem gramatiky.

Oba výše zmíněné nástroje generují zdrojové kódy odpovídajících analyzátorů v programovacím jazyku C. Z tohoto důvodu musela být při volbě jazyka vzata v úvahu i tato skutečnost. Proto byl také jako jazyk použit již zmíněný jazyk C++ v kombinaci s jazykem C.

[2] jazyk symbolických adres

Libpcap

K zasílání a odchytávání rámců byla použita knihovna *libpcap* [11]. Tato knihovna je open source knihovnou, která představuje vysokoúrovňové rozhraní pro práci s pakety počítačových sítí. Knihovna byla vytvořena již v roce 1994 na univerzitě v Berkeley jako součást projektu ke zkoumání a zlepšování TCP protokolu a výkonu brány Internetu. Libpcap je navržena pro použití s jazyky C a C++ a běhu na operačních systémech typu Unix. Pro operační systém Windows existuje upravená verze nazvaná Winpcap [18].

TinyXML

Pro účely ukládání rámců do souborů byl zvolen datový formát XML *(Extensible Markup Language)*. Abychom nemuseli realizovat vlastní analyzátor tohoto formátu, zvolili jsme knihovnu TinyXML [6], která jej již realizuje.Tato knihovna je malá a na použití velmi jednoduchá. Je napsaná v jazyku C++, a proto může být bez jakýchkoliv problému použita v naší aplikaci.

3.6 Popisovaná struktura rámce

Před popisem návrhu jazyka pro popis rámců by bylo vhodné ukázat, jak takový rámec vlastně vypadá. Základní struktura rámce již byla popsána v části 2 Úvod do bezdrátových sítí standardu 802.11. Zde byl rámec představen jako struktura skládající se ze tří částí. Z hlavičky RadioTap, hlavičky IEEE a v případě datového rámce také z vlastního datového obsahu. Na ukázce 3.4, která je uvedena dále, je zobrazen rámec, který byl zachycen pomocí programu Wireshark [8]. Tento program slouží k zaznamenávání a následné analýze datového toku.

Pokud bychom chtěli provést odchycení nějakého rámce v operačním systému Linux, musíme nejprve použít například aplikaci airmon-ng, která je rovněž součástí sady aplikací aircrack-ng popsané v části 3.1 Rozbor stávajících řešení. Aplikace airmon-ng slouží k uvedení bezdrátového rozhraní do monitorovacího módu. V tomto módu je následně možné zachytávat rámce, tak jak jsou přenášeny po síti. Pokud bychom neuvedli rozraní do monitorovacího módu, program Wireshark by nám zobrazil strukturu rámců již bez námi požadovaných hlaviček. Program by rovněž neprovedl zobrazení rámců, které se starají o správu sítě *(například beacon rámce nebo probe rámce)*, a řídících rámců *(například rámce RTS nebo CTS)*. Uvést rozhraní *(v ukázce je použito rozhraní wlan0)* do monitorovacího módu je možné pomocí příkazu z terminálu operačního systému. Tento příkaz je zachycen na ukázce 3.3.

Ukázka 3.3: Uvedení rozhraní do monitorovacího módu

```
airmon-ng start wlan0
```

Aplikace airmon-ng nám pomocí tohoto příkazu vytvoří nové virtuální rozhraní, které bude nejčastěji označeno jako *mon0*. Na tomto rozhraní můžeme posléze pomocí programu Wireshark provést odchycení požadované komunikace.

Pomocí programu Wireshark a postupu, který byl popsán výše byl odchycen datový rámec, který má následující strukturu.

Ukázka 3.4: Příklad struktury rámce

Hlavička RadioTap

```
Version (Header revision): 0
Pad (header pad): 0
Len (header length): 26
Present (present flags): 0x0000482f
TSFT (MAC timestamp): 584219565701
Flags (Flags): 0x10
Rate (Data Rate): 48,0 Mb/s
Channel frequency (Channel frequency): 2437
Channel flags (Channel flags):0x00c0
Antenna signal (SSI Signal): -62 dBm
Antenna (Antenna): 1
RX flags (RX flags): 0x0000
```

Hlavička IEEE

```
FrameControl (FrameControl): 0x4208
Duration (Duration): 44
Address1 (Destination address): 70:f1:a1:59:19:0e
Address2 (BSS ID): 00:21:91:71:54:f2
Address3 (Source address): 00:21:91:71:54:f2
Seq-ctl (Fragment and Sequence number): 3758
FCS (Frame check seqeunce): 0x398fc9f9
TKIP parameters: initialization vector: 0x0000000bc900
                 key index: 0
```

Data

```
0xdc09778bbf3369c19c12c629118b92e0edb73c651b2728ad ...
```

V této struktuře byl proveden popis jednotlivých částí pomocí terminologie, která je používaná v literatuře odkazované u popisu jednotlivých hlaviček. Ta se v některých případech liší od terminologie, kterou používá program Wireshark. Proto je i tato terminologie uvedena v závorce na odpovídajícím řádku. Shodný rámec je vyobrazen na obrázku 3.1. Zde je rámec vyobrazen v hexadecimalní soustavě a jsou zde vyznačeny jednotlivé části rámce.

Version	Pad	Len		Present				TSFT							
0x00	0x00	0x1a	0x00	0x2f	0x48	0x00	0x00	0x85	0x1e	0x33	0x06	0x88	0x00	0x00	0x00
Flags	Rate	Channel freq.		Channel type		Ant.sig.	Antenna	RX flags		Frame control		Duration			
0x10	0x60	0x85	0x09	0xc0	0x00	0xc2	0x01	0x00	0x00	0x08	0x42	0x2c	0x00	0x70	0xf1
Address1				Address2						Address3					
0xa1	0x59	0x19	0x0e	0x00	0x21	0x91	0x71	0x54	0xf2	0x00	0x21	0x91	0x71	0x54	0xf2
Seq-ctl		IV			Key ind	Rozšířený IV				Data					
0xe0	0xea	0xc9	0xeb	0x00	0x20	0x0b	0x00	0x00	0x00	0xdc	0x09	0x77	0x8b	0xbf	0x33

Data						FCS			
0x0c	0xae	0x8b	0x32	0x85	0xee	0x39	0x8f	0xc9	0xf9

Obrázek 3.1: Datový rámec

Jak již bylo napsáno dříve, hlavička RadioTap slouží především k výměně informací mezi aplikací a ovladači bezdrátového rozhraní. Můžeme zde především naleznout informace, které se týkají použitého bezdrátového média. Tyto informace jsou v řadě případů poupravovány ovladači

bezdrátového rozhraní až po přijetí daného rámce. Proto některé položky mohou mít různé hodnoty na dvou zařízeních, na kterých byl rámec odchycen. Jedná se například o položky, které se týkají kvality signálu *(vlastnost Antenna signal)*, použité anténě *(vlastnost Antenna)* nebo například časová značka rámce *(vlastnost TSFT)*. Naopak informace týkající se použitého média, jako příklad uveďme použitou frekvenci *(vlastnost Channel frequency)* nebo například informace o přítomnosti kontrolního součtu *(vlastnost Flags)*, jsou na obou zařízeních totožné.

3.7 Návrh jazyka

K popisu rámců a k jejich následné manipulaci byl navržen jazyk jehož syntaxe vychází z jazyka použitého k popisu paketů v programu Scapy. Důvodem použití podobné syntaxe je, že ho považujeme za jednoduchý, přehledný a zároveň bude tak usnadněno jeho použití pro uživatele, kteří již mají zkušenosti s tímto programem. Ti se nebudou muset učit novou formu zápisu určenou k popisu vlastností rámců.

K zápisu syntaxe jazyka byla použita zjednodušená verze Backus-Naurovy formy *(zkráceně BNF)*[20]. Zápis v BNF je podobný zápisu pomocí bezkontextové gramatiky. Ten rovněž obsahuje terminální symboly, neterminální symboly a pravidla pro přepis neterminálních symbolů. Oproti bezkontextové gramatice však zjednodušuje zápis obvyklých technik jako je například opakování jistého řetězce. Tohoto bychom museli v případě bezkontextové gramatiky docílit pomocí rekurze, čímž by se nám zápis značně zkomplikoval. Pomocí BNF můžeme zápisu rekurze předejít pomocí operátoru "$*$"případně operátoru "+". Rozdíl těchto operátorů spočívá v minimální četnosti výskytu daného řetězce. Zatímco v případě operátoru "$*$"není výskyt řetězce povinen, v případě operátoru "+"je vyžadován alespoň jeden jeho výskyt.

3.7.1 Reprezentace rámce

V navrženém jazyku jsou jednotlivé rámce zastoupeny pomocí proměnné reprezentované textovým identifikátorem. Tento identifikátor byl zapsán pomocí výrazu uvedeného na ukázce 3.5.

Ukázka 3.5: Identifikátor proměnných

```
identifikator: [a-zA-Z][a-zA-Z0-9]*
```

3.7.2 Definice rámce

Vytvoření nového rámce je možné pomocí definice, při níž se specifikuje hlavička, která je právě vytvářena. Tato hlavička může již při tomto zápisu obsahovat definici jednotlivých vlastností. Při definici rámce je umožněno jednotlivé hlavičky za sebou řetězit pomocí operátoru "/". Příklad definice rámce pomocí specifikace hlaviček je zobrazen na ukázce 3.6. V tomto příkladu je vytvořena hlavička RadioTap a hlavička IEEE. U hlavičky RadioTap je zároveň provedeno nastavení příznaku, který indikuje přítomnost kontrolního součtu. Při tvorbě hlavičky IEEE je provedena definice *typu* rámce. Zde je použit typ *beacon* s nastavenou vlastností *SSID*.

Ukázka 3.6: Definice rámce

```
ramec = RadioTap(flags="crc")/IEEE(type="beacon" ssid='bcn_rm')
```

3.7.3 Klíčová slova

V ukázce 3.6 uvedené na předchozí straně si můžeme povšimnout, že pro definici typu rámce a indikaci přítomnosti crc kontrolního součtu, bylo použito klíčového slova. Použitá klíčová slova je zapotřebí uzavřít do uvozovek, tak jak je uvedeno v ukázce. Existují vlastnosti, u kterých lze nastavit více takovýchto klíčových slov. Typickým příkladem je vlastnost *flags* hlavičky RadioTap. Tato vlastnost obsahuje příznaky odesílaných a přijímaných rámců. Ve skutečnosti se jedná o bitmapu, kde každý bit reprezentuje jeden příznak. Těmito příznaky jsou například již zmíněný kontrolní součet *("crc"- odpovídající maska 0x10)*, dále pak informace o zašifrování rámce *("sentReceiveWithWEPencryption"- odpovídající maska 0x04)* a další. U těchto vlastností může být využito řetězení klíčových slov pomocí operátoru "+". Abychom při každé modifikaci této vlastnosti nemuseli vyjmenovávat všechny příznaky, postačuje pouze vyjmenovat nové příznaky, které chceme nastavit, a příznaky které chceme naopak zrušit. Před rušené příznaky poté předřadíme operátor "∼". Ten indikuje odstranění daného příznaku. Použití zřetězení klíčových slov ukazuje ukázka 3.7.

Ukázka 3.7: Použití klíčových slov

```
ramec.flags = "sentReceiveWithWEPencryption + ∼crc"
```

3.7.4 Alternativní definice rámce

V předchozí ukázce 3.7 jsme si mohli povšimnout dalšího způsobu definice hodnot vlastností rámců. Pomocí operátoru "."*(tečka)* je rovněž možné provést definici vlastností rámce. V tomto případě je provedeno nastavení každé vlastnosti odděleně. Pojmenování vlastností a následné přiřazení hodnot zůstává shodné s definicí rámce pomocí předchozího způsobu.

3.7.5 Přiřazení hodnot

Hodnoty odpovídajícím vlastnostem se přiřazují pomocí operátoru "=". V předchozích ukázkách bylo použito klíčových slov a také textových řetězců. Textové řetězce jsou použity u vlastností, kde je očekávána hodnota textového řetězce. Typickým příkladem této vlastnosti je vlastnost *SSID* rámce *beacon*. Jednotlivé řetězce jsou obklopeny pomocí znaku apostrofu.

Dalšími hodnotami, které lze přiřadit, jsou hodnoty číselné. Tyto hodnoty je možné uvádět v dekadickém *(například číslo 14)* případně hexadecimálním tvaru *(například číslo 0xe)*. Posledním typem hodnot, který je v jazyku použit je hardwarová adresa MAC. Její formát zápisu je uveden na ukázce 3.8.

Ukázka 3.8: Definice hardwarove adresy

```
ramec.bssid = "00:21:91:71:54:f2"
```

Pomocí výše definovaných operací bychom měli být nyní schopni popsat celý rámec. Popis nám umožňuje definovat jednotlivé vlastnosti hlaviček rámců, kterým můžeme přiřazovat potřebné hodnoty. Jazyk byl dále rozšířen o možnost zrušení, případně navrácení hodnoty vlastnosti na výchozí hodnotu. Tohoto je docíleno pomocí příkazu **del** aplikovaný na odpovídající vlastnost. Použití tohoto příkazu je demonstrováno na příkladu 3.9.

Ukázka 3.9: Zrušení definované vlastnosti

```
del(ramec.flags)
```

3.7.6 Příkazy k manipulaci s rámci

Další částí námi navrhovaného jazyku jsou příkazy a konstrukce sloužící k manipulaci s rámci. V jazyku je definována řada příkazů, které umožňují s rámci pracovat. Přehled jednotlivých příkazů je shrnut v tabulce 3.1. V ní je uveden pouze základní přehled podporovaných funkcí, které lze použít. V jazyku se vyskytují i další pomocné příkazy a konstrukce, jejichž popis bude uveden dále.

Příkaz	**Popis**
send(identifikator)	Příkaz umožňuje zaslání rámce na předem specifikované rozhraní.
isend(identifikator)	Neblokující varianta předchozího příkazu.
print(identifikator)	Příkaz provede vypsání přehledu nastavených vlastností, které budou použity při generování rámce.
dump(identifikator)	Příkaz provede vypsání rámce v hexadecimální soustavě.
identifikator.load(cesta)	Příkaz provede načtení rámce ze souboru.
identifikator.save(cesta)	Příkaz provede uložení rámce do souboru.
sleep(time)	Příkaz provede uspání aplikace na čas definovaný v proměnné *time*. Jednotkou je jedna vteřina.
msleep(mtime)	Příkaz provede uspání aplikace na čas definovaný v proměnné *mtime*. Jednotkou je jedna milisekunda.
usleep(utime)	Příkaz provede uspání aplikace na čas definovaný v proměnné *utime*. Jednotkou je jedna mikrosekunda.
capture(filtr)	Příkaz provede odchycení rámce splňující vlastnosti filtru.
getiv()	Příkaz provede extrakci inicializačního vektoru z odchyceného rámce.
scapy(popis)	Příkaz provede vygenerování datového obsahu pomocí aplikace Scapy.
key	Příkaz provede spuštení daemona pro načítání klíčů.
getGTK()	Příkaz provede načtení uloženého GTK klíče.
getPTK()	Příkaz provede načtení uloženého PTK klíče.
time start	Příkaz zahájí měření času.
time	Příkaz provede vypsání času, který uběhl od zavolání funkce *time start*.
load(cesta)	Příkaz provede načtení uloženého programu, který následně vykoná.
break	Příkaz provede ukončení cyklu.
list	Příkaz provede vypsání seznamu uložených rámců.
exit	Příkaz provede ukončení aplikace.

Tabulka 3.1: Přehled příkazů jazyka

3.7.7 Proměnné

Jazyk obsahuje mimo proměnných reprezentující jednotlivé rámce také proměnné, do kterých lze uložit číselné hodnoty a textové řetězce. Tyto proměnné jsou reprezentovány shodným identifikátorem, který byl definován k reprezentaci rámců. Použití jednotlivých proměnných je demonstrováno na ukázce 3.10.

Ukázka 3.10: Použití proměnných

```
cislo = 10
retezec = 'textovy retezec'
```

3.7.8 Výrazy a podmínky

Dalšími konstrukcemi, kterými byl jazyk obohacen, jsou výrazy a podmínky. Seznam operátorů, které lze ve výrazech a podmínkách použít, shrnuje tabulka 3.2. Jejich použití je shodné s použitím v běžných programovacích jazycích.

Operátor	Priorita[3]	Význam operátoru
+	7	sčítání
−	7	odečítání
*	8	násobení
/	8	dělení
%	8	zbytek po dělení
++	9	inkrementace
−−	9	dekrementace
<	6	menší než
>	6	větší než
<=	6	menší nebo rovno
>=	6	větší nebo rovno
==	5	rovná se
!=	5	nerovná se
||	2	logická disjunkce
&&	2	logická konjunkce
!	9	logická negace
|	4	disjunkce po bitech
&	3	konjunkce po bitech
=	1	přiřazení

Tabulka 3.2: Přehled operátorů a výrazů

3.7.9 Cykly

Jazyk obsahuje dva typy cyklu. Prvním cyklem je cyklus **while** obsahující podmínku řízení vykonávání cyklu na začátku. Ta je nejprve vyhodnocena a v případě, kdy je podmínka splněna, je provedeno tělo cyklu. To musí být uvedeno ve složených závorkách. Příklad použití cyklu *while* demonstruje ukázka 3.11.

Ukázka 3.11: Cyklus *while*

```
i = 5
while( i >= 0 ) {
        i--
        print(i)
}
```

Tělo cyklu se bude provádět tak dlouho, dokud je splněna podmínka uvedena v kulatých závorkách. Před začátkem cyklu je proměnná *i* inicializována na hodnotu 5. V těle cyklu je tato hodnota pokaždé dekrementována o jedna. Tělo cyklu se tedy v této ukázce provede celkem šestkrát.

[3] čím vyšší číslo, tím vyšší priorita

Dalším typem cyklu, který jazyk podporuje, je cyklus **for**. I tento cyklus obsahuje vyhodnocení podmínky před vykonáním vlastního těla cyklu, které je rovněž zadáno ve složených závorkách. Příklad demonstrující použití tohoto cyklu je uveden na ukázce 3.12 níže.

Ukázka 3.12: Cyklus *for*

```
for(i = 0; i <= 5 ; i++) {
      print(i)
}
```

Před samotným začátkem těla cyklu *for* je provedena inicializace proměnné *i*, které je přiřazena hodnota 0. Následně je provedena kontrola podmínky, zda-li je hodnota menší nebo rovna 5. Pokud je podmínka splněna, provede se tělo cyklu. Na konci cyklu je provedena inkrementace proměnné *i* a cyklus se opakuje. Zde už cyklus začíná vyhodnocením podmínky.

Cyklus for také dovoluje formu zápisu, kde nejsou uvedeny podmínky, takzvaný nekonečný cyklus. Zápis tohoto cyklu je ukázán v ukázce 3.13.

Ukázka 3.13: Nekonečný cyklus

```
i = 0
for(;;) {
      print(i++)
}
```

Jak je možné tento cyklus přerušit, aniž by bylo zapotřebí aplikaci ukončit, je pojednáno v kapitole Implementace.

3.7.10 Podmíněný příkaz

Poslední konstrukcí, kterou jazyk poskytuje, je podmíněný příkaz **if**. Příkaz *if* se skládá z podmínky, těla, které se vykoná v případě, je-li podmínka splněna, a volitelně z těla **else**, které je vykonáno v případě, kdy podmínka splněna není. I v případě podmíněného příkazu jsou těla obsahující příkazy uzavřena do složených závorek. Použití této podmínky, je ukázáno na příkladu uvedeném na ukázce 3.14.

Ukázka 3.14: Podmíněný příkaz *if*

```
i = 0
if((i % 2) == 0) {
   print('cislo je sude')
else {
   print('cislo je liche')
}
```

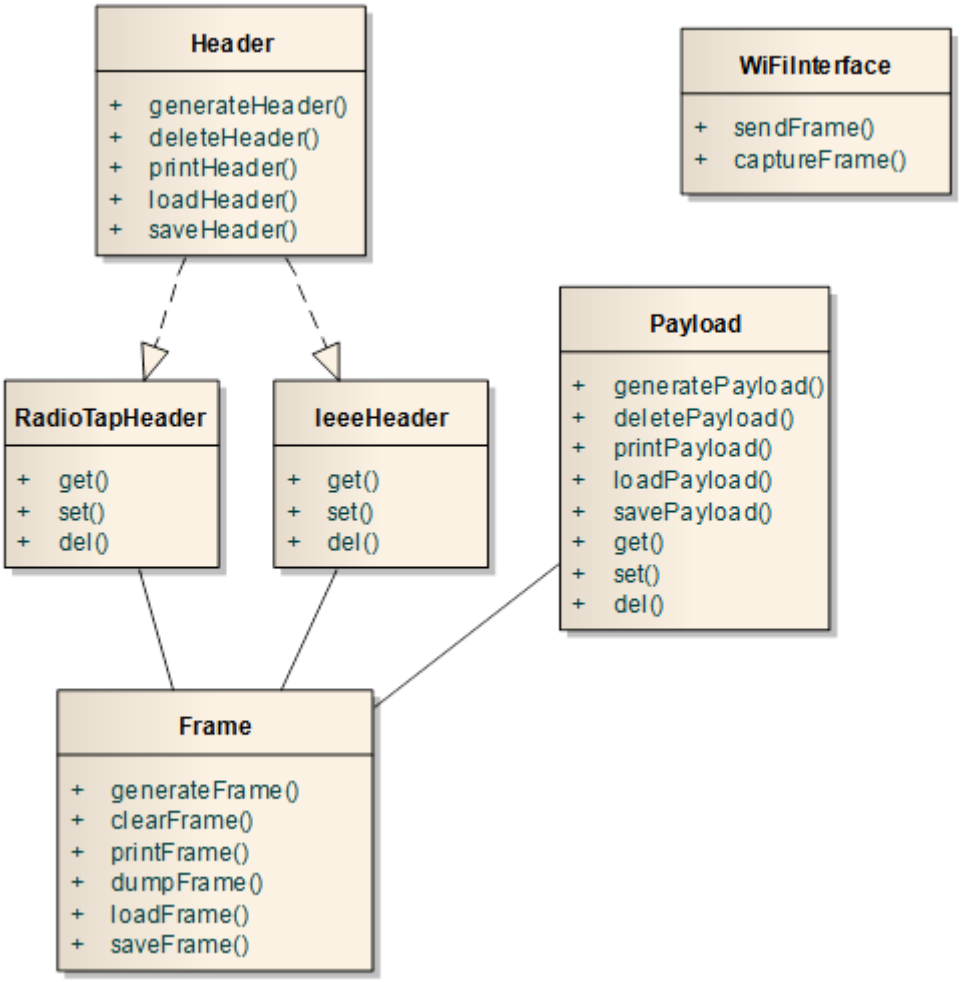
3.7.11 Komentáře

V některý případech, zvláště pokud jde o načítání programu ze souboru, se mohou uživateli hodit komentáře. Do navrhovaného jazyku byl přídán jednořádkový komentář, který většina z uživatelů zná například z programovacího jazyka C případně C++. Tento komentář začíná dvojicí znaků "//". Vše co se nachází za těmito znaky až do konce řádku je považováno za komentář a je při zpracování vynecháno.

3.8 Návrh aplikace

Aplikace je realizována jako konzolová aplikace v programovacím jazyku C++. Při její realizaci bylo využito objektově orientovaného návrhu, pomocí něhož vznikla sada tříd, které byly následně využity ve vlastní aplikaci. Jednotlivé třídy bude dále možné využívat i samostatně jako knihovnu pro tvorbu aplikací využívajících práce s rámci bezdrátových sítí standardu 802.11.

Které třídy a metody bylo zapotřebí realizovat, ukazuje obrázek 3.2. Na něm jsou zachyceny třídy, které jsou nezbytné k tomu, aby aplikace následně mohla provádět generování rámců. Aplikace pro svůj běh potřebuje i další doplňkové třídy. Ty ale nejsou z pohledu generování rámců zajímavé, a proto jsou z tohoto popisu vypuštěny.



Obrázek 3.2: Diagram tříd

Třída Frame

Tato třída reprezentuje vlastní rámec. Objekt této třídy obsahuje jako své interní proměnné objekty hlaviček RadioTap a IEEE a dále objekt datového obsahu rámce. Ten vzniká ze třídy Payload. Třída Frame dále obsahuje potřebné metody, pomocí niž je možné provést vygenerování rámce, jeho zaslání na rozhraní, vypsání v hexadecimální soustavě a vypsání informací o tomto rámci. Pomocí

těchto metod jsou volány odpovídající metody objektů hlaviček. Třída je dále rozšířena o metody *loadFrame* a *saveFrame*, pomocí niž je možné rámec uložit do souboru případně jej z něj načíst.

Header, RadioTapHeader, IeeeHeader

Třída Header je abstraktní třídou, ze které jsou odvozeny vlastní třídy hlaviček. Těmi jsou třídy RadioTapHeader a IeeeHeader. Objekty těchto tříd obsahují jednotlivé interní proměnné uchovávající vlastností hlaviček. Ty jsou nastavovány pomocí odpovídajících metod *set*. Jednotlivé hodnoty je možné získat pomocí odpovídajících metod *get* a jejich smazání, případně navrácení na původní hodnotu pomocí metod *del*. Jednotlivé třídy implementují metody třídy abstraktní, které jsou volány v odpovídajících metodách třídy Frame.

Payload

Podobnou třídou metodě Header je i třída Payload. Objekty této třídy reprezentují datový obsah rámců. Tato třída opět obsahuje vhodné metody pro nastavování, získávání a mazání vlastností uložených v interních proměnných. Ostatní metody mají obdobný význam, jako metody u třídy Header.

WiFiInterface

Poslední důležitou třídou je třída WiFiInterface. Tato třída slouží k práci s bezdrátovým rozhraním. Obsahuje mimo jiné metody sloužící k zasílání rámců a případně k jejich zaznamenávání.

Kapitola 4

Implementace

Následující kapitola popisuje implementační část této práce. V předchozí kapitole bylo ukázáno, jak vypadá navržený jazyk určený k popisu rámců a z jakých klíčových částí v podobě tříd bude aplikace realizována. Toto bude nyní v této části popsáno z pohledu implementace. Bude zde poukázáno na to, co bylo zapotřebí vytvořit, a naopak které části byly realizovány pomocí knihoven.

4.1 Realizace analyzátorů

Jak již bylo napsáno v předchozích kapitolách, k realizaci generátorů bylo využito nástrojů YACC a LEX. Vstupem těchto nástrojů je popis gramatiky navrženého jazyka. V této gramatice se nachází u každého pravidla odpovídající akce, která má být vykonána, nalezne-li analyzátor shodu. Tyto akce představují volání metod v námi použitém programovacím jazyku. Jednotlivé analyzátory jsou popsány níže.

4.1.1 Lexikální analyzátor

Generování lexikálního analyzátoru probíhá z popisu zadaném ve zdrojovém souboru *Grammar.l* v adresáři *grammar*. Tento zdrojový soubor se skládá ze tří částí oddělených symboly "%%". V první části se nachází import hlavičkových souborů a deklarace proměnných. Ve druhé části poté můžeme naleznout vlastní pravidla lexikálního analyzátoru. Jedná se o regulární výrazy, ke kterým je asociovaná vhodná akce. Ve většině případů se však jedná o navrácení tokenu, který je předán syntaktickému analyzátoru. V poslední části lexikálního analyzátoru se nacházejí definice metod a funkcí, které mohou být dále využity v samotném lexikálním analyzátoru případně v některé jiné části aplikace. V našem případě se zde nachází definice funkcí, které se starají o změnu čteného vstupu. O těchto metodách bude zmínka ještě dále v textu.

4.1.2 Syntaktický analyzátor

Druhým z analyzátorů, který bylo potřeba realizovat, je syntaktický analyzátor. Tento analyzátor je generován pomocí nástroje YACC ze zdrojového souboru, který je umístěn opět v adresáři *grammar* a má název *Grammar.y*. Tento soubor je členěn obdobně jako zdrojový soubor lexikálního analyzátoru. V první části můžeme opět naleznout import hlavičkových souborů a deklaraci proměnných. Ve druhé části pak nalezneme přepisovací pravidla gramatiky navrženého jazyka. S každým tímto pravidlem, podobně jako u lexikálního analyzátoru, je asociovaná odpovídající akce. Touto akcí je tvorba uzlů stromu příkazů, který je posléze vykonán pomocí interpretu. Poslední část opět může obsahovat definice funkcí. V našem případě se zde nachází definice těla *main*. V té jsou volány pouze

nezbytně nutné funkce. Touto funkcí je například inicializace bezdrátového rozhraní. V závěru těla *main* se nachází cyklus *while* obsahující volání funkce *yyparse()*. Toto je funkce, pomocí níž se volá syntaktický analyzátor. Při každém novém vstupním řádku se zavolá právě tato funkce. Její volání trvá tak dlouho, dokud analyzátory nenarazí na konec vstupu. Ten je v aplikaci signalizován zadáním příkazu **exit**. V tomto okamžiku jsou ukončeny oba analyzátory.

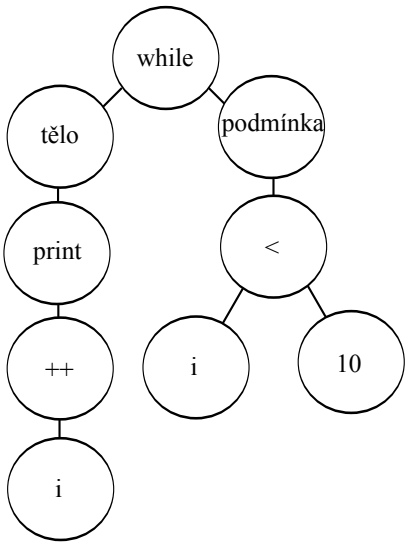
4.2 Interpret jazyka

K vykonávání jednotlivých příkazů byl vytvořen jednoduchý interpret. Tento interpret obsahuje strom příkazů, které jsou po jeho spouštění vykonány. Jak již bylo uvedeno, ke každému pravidlu gramatiky je asociována akce, která má být provedena. V našem případě se jedná o vytvoření nového uzlu stromu příkazů. Tento uzel je následně předán nadřazenému pravidlu, ve kterém je umístěn jako synovský uzel právě vytvářeného uzlu. Posledním uzlem, který je vytvořen, je kořenový uzel. Tento uzel je předán interpretu, který následně provede vykonání celého stromu příkazů pomocí volání funkce *executeStatement* na kořen stromu. Stromem je procházeno a jsou vykonávány příkazy jednotlivých uzlů. Jedním z příkazu je vždy vykonání synovského uzlu. Listové uzly pouze vykonají své tělo a navrací návratovou hodnotu. Krátká ukázka vytvoření tohoto stromu je demonstrována níže.

Pro program uvedený na ukázce 4.1 je vytvořen strom, který je uveden na obrázku 4.1.

Ukázka 4.1: Vykonávaný program

```
while(i < 10) {
        print(i++)
}
```



Obrázek 4.1: Strom příkazů zpracovávaný interpretem

V případě cyklu *while (třída WhileStatement)* obsahuje odpovídající uzel dva synovské uzly. Prvním uzlem je uzel nazvaný tělo *(třída BodyStatement)*. Tento uzel obsahuje seznam synovských uzlů, které jsou postupně vykonány. V našem příkladu zde máme jeden uzel nazvaný *print*. Tento uzel je obecně uzlem příkazu *(třída CommandStatement)*, který obsahuje pouze příznak typu příkazu. Uzel *print* obsahuje odkaz na uzel vykonávající inkrementaci. Opět u tohoto uzlu se jedná

obecně o uzel unární operace *(třída UExpression)* s nastaveným typem operace. Ten má odkaz na uzel proměnné *(třída VariableValueExpression)*. Tento uzel provádí načítání odpovídající hodnoty z tabulky proměnných. Tu následně vrací svému předku.

Druhým uzlem, který obsahuje uzel *while*, je uzel podmínky. Zde se jedná o uzel binární operace *(třída BExpression)*. Ta obsahuje odkazy na dva potomky. Prvním potomkem je opět uzel proměnné *(třída VariableValueExpression)*. Druhým potomkem je uzel číselné hodnoty *(třída NumberValueExpression)*.

Cyklus *while* provádí své tělo, za předpokladu, že je splněna podmínka. Její vyhodnocení proběhne pomocí volání metody *evaluateExpression*, která vrací hodnotu vyhodnoceného výrazu.

4.2.1 Tabulka proměnných

Interpret obsahuje tabulku k uložení proměnných. Tuto tabulku následně pomocí ukazatele předává při vykonání svému kořenovému uzlu. Ten ji dále předá svému potomku. V případě kdy se jedná o uzel, který z ní čte hodnotu, volá odpovídající metodu této tabulky, které předá název proměnné. Pokud se v ní odpovídající proměnná nachází, provede její navrácení. Pokud se v ní nenachází, navrací proměnnou s nastaveným typem *neznámý*.

Každá proměnná v této tabulce obsahuje datový typ. Tento typ může být *číslo*, *textový řetězec* případně *rámec*. Ten je proměnné nastaven při její definici. Při navracení hodnoty proměnné je navrácen i tento datový typ, který je při zpracování hodnoty kontrolován. Pokud příkaz vyžaduje datový typ rámce a dostane číselnou hodnotu, provede interpret vypsání zprávy o neplatném datovém typu.

Proměnné, které jsou aktuálně uloženy v tabulce, je možné zobrazit pomocí příkazu **list** případně pomocí příkazu **list all**. První ze jmenovaných příkazů zobrazí seznam proměnných, které jsou typu *rámec*. Druhý příkaz zobrazí seznam všech proměnných, včetně proměnných uchovávající číselné a textové hodnoty.

4.2.2 Přerušení nekonečného cyklu

Při popisu jazyka bylo uvedeno, že je možné využít takzvaného nekonečného cyklu. Tento cyklus lze využít například v případě, kdy neznáme dopředu přesný počet opakování. U tohoto cyklu ale nastává otázka, co dělat v případě, kdy si přejeme daný cyklus přerušit. Prvním řešením je ukončit celou aplikaci pomocí signálu zaslaný po stisku kombinace kláves CTRL-C. Toto ale může být nevhodné, pokud si přejeme dále s aplikací pracovat. Pro tyto případy byly do aplikace implementovány dva způsoby přerušení tohoto nekonečného cyklu.

Prvním způsobem přerušení cyklu je využití podmíněného příkazu *if* a příkazu **break**. Tento příkaz způsobí přerušení vykonávání cyklu a pokračuje se dále ve vykonávání aplikace. V případě, kdy jsou dva cykly zanořeny do sebe, uplatní se toto přerušení pouze na první z nich. Vnější cyklus poběží dále bez přerušení. V programu je toto realizováno pomocí návratové hodnoty každého příkazu. V těle cyklu se následně kontroluje tato návratová hodnota. Je-li rovna hodnotě *false*, přeruší se vykonávání cyklu. Ten již navrací hodnotu *true*. V opačném případě vykonávání probíhá dále. A právě příkaz *break* vrací hodnotu *false*, která se propaguje přes jednotlivé příkazy, tedy i přes příkaz *if*, až ke zmiňovanému příkazu cyklu.

Druhý způsob ukončení nekonečného cyklu vyžaduje zásah z vnějšku aplikace. Při jejím spuštění je provedeno navěšení obslužné funkce na vhodnou událost. Touto událostí je příjem signálu *SIGUSR1* zaslaný z operačního systému. Tento signál je možné aplikaci zaslat například z druhého okna terminálu pomocí příkazu *kill*. Tento příkaz vyžaduje typ signálu, kterým je zmiňovaný *SIGUSR1*, a číslo procesu naší aplikace. Ten lze získat ze seznamu běžících procesů, případně jej

aplikace vypisuje při jejím spuštěním. Ukončení aplikace pomocí tohoto způsobu je demonstrováno na ukázce 4.2. V ukázce je použito jako číslo procesu číslo *2013*.

Ukázka 4.2: Zaslání signálu SIGUSR1

```
kill -SIGUSR1 2013
```

4.2.3 Načítání programu ze souboru

Abychom ušetřili uživatele používající tento nástroj od opakovaného psaní téhož programu, byla aplikace rozšířena o možnost načítání programů z textového souboru. V tomto souboru si uživatel může připravit vlastní program, tak jak kdyby jej psal přímo v aplikaci. V té následně provede jeho načtení pomocí příkazu **load**. Tento příkaz očekává jeden parametr, kterým je cesta k danému souboru. Parametr může být reprezentován pomocí textového řetězce obsahující danou cestu, případně pomocí aliasu, který tuto cestu zastupuje. O možnosti používání aliasů je pojednáno dále v práci. V tento okamžik nám postačí informace, že se jedná o proměnnou, která obsahuje danou cestu.

Z externího souboru si aplikace přečte jeden řádek obsahující příkaz do interního bufferu. Poté je provedeno přesměrování vstupu lexikálního analyzátoru do tohoto bufferu. Po přesměrování je volána funkce syntaktického analyzátoru *yyparse()*, která zpracuje výstup lexikálního analyzátoru provádějící analýzu vstupu ze zadaného bufferu. Po ukončení syntaktického analyzátoru je provedeno přesměrování lexikálního analyzátoru zpět na čtení z klávesnice. Takto je následně zpracován celý vstupní soubor.

Jedinou výjimkou předchozího zpracování je čtení příkazu cyklu případně podmíněného příkazu *if*. V tomto případě je zapotřebí provést načtení kompletního příkazu, který může být zapsán přes vícero řádků. Tedy i s případným tělem, které bude vykonáváno. Pokud bychom tak neučinili vykonávání aplikace by uvázlo. Lexikální analyzátor by očekával další vstup, který by nemohl být načten dříve, než skončí analyzátor syntaktický. Pomocí načtení kompletního příkazu tomuto můžeme předejít.

Metody, pomocí níž bylo provedeno přesměrování vstupů lexikálního analyzátoru, jsou pro ilustraci uvedeny v ukázce 4.3.

Ukázka 4.3: Změna vstupního toku

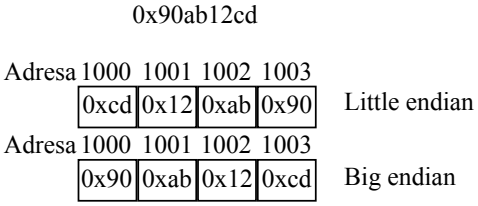
```
void readFromBuffer(char * buf, int lenBuf) {
        include_stack[include_stack_ptr++] = YY_CURRENT_BUFFER;
        yy_switch_to_buffer( yy_scan_buffer(buf, lenBuf) );
        BEGIN(INITIAL);
}

void readFromKeyboard() {
        --include_stack_ptr;
        yy_delete_buffer( YY_CURRENT_BUFFER );
        yy_switch_to_buffer( include_stack[include_stack_ptr] );
}
```

Tímto bylo představeno, jak jsou jednotlivé příkazy v naší aplikaci zpracovávány. V následující části je uveden popis jednotlivých příkazů.

4.3 Generování rámců

Jak jsou jednotlivé rámce v aplikaci uloženy, bylo již ukázáno v kapitole Návrh. Každý rámec je uložený v objektu třídy Frame obsahující jednotlivé hlavičky. Důvodem upřednostnění objektů před použitím struktur bylo, že jednotlivé hlavičky obsahují řadu volitelných položek. V případě hlavičky RadioTap se navíc vyskytují výplňové bajty, na které bylo upozorněno při představení této hlavičky. Ty se mohou v některém rámci vyskytovat a v některém naopak nemusejí. Dále pak hlavička IEEE je velice variabilní. Například rámec typu *RTS* je podobný základní struktuře této hlavičky. Naopak rámec typu *beacon* obsahuje řadu dalších položek, které se opět mohou a nemusejí vyskytovat. Z tohoto důvodu bylo zvoleno použití objektů a tříd. Návrh se stal přehlednějším a realizace byla, alespoň dle našeho názoru, jednodušší. Nicméně použití datových struktur by bylo rovněž možné, ale za cenu menšího komfortu při návrhu a realizaci aplikace.

Na co je dále zapotřebí upozornit, je uložení dat ve vygenerovaném rámci. Jednotlivé položky obou hlaviček, jak RadioTap tak i IEEE, jsou ve výsledném rámci uloženy v pořadí *little endian*. Položky následujících hlaviček, pokud by je bylo zapotřebí generovat, by byly v pořadí *big endian*. Rozdíl obou endianit spočívá v pořadí uložení jednotlivých bajtů na výstupu více bajtového slova. V případě pořadí *little endian* se nachází nejméně významový bajt na nižší adrese a nejvíce významový naopak na adrese vyšší. U pořadí *big endian* je tomuto přesně naopak. Jednotlivá pořadí jsou demonstrována na obrázku 4.2. Na něm je demonstrováno, jak by bylo uloženo 32 bitové číslo v obou uspořádáních.



Obrázek 4.2: Little a big endian

Při vlastním generování rámců je volána metoda *generateFrame* třídy Frame, která volá odpovídající metody tříd hlaviček a metodu třídy Payload. Ty provedou vlastní vygenerování odpovídajících částí. Vygenerovaný rámec se poté uloží do proměnné a zároveň se provede jeho navrácení volající funkci. S cílem ušetřit opakované generování hlaviček, byla třída Frame rozšířena o proměnnou indikující, zdali nedošlo od posledního vygenerování ke změně v hlavičkách případně v payloadu. Pokud ke změně nedošlo, opětovné generování již není prováděno. Provede se pouze navrácení předešlého výsledku.

4.4 Obsah datových rámců

Obsah datových rámců je možné zadat jako hexadecimální řetězec, který obsahuje vlastní data. Tento řetězec si uživatel musí vytvořit sám, například pomocí jiné aplikace. Řetězec musí obsahovat veškeré informace, které se vyskytují za hlavičkou IEEE. Tento řetězec může být ale někdy příliš dlouhý a je možné v něm udělat chybu, díky níž by vznikl nesprávný rámec.

Abychom ušetřili uživatele od nutnosti používat jiné aplikace, byla realizovaná aplikace rozšířena o funkci **scapy**, která uvnitř svého těla volá aplikaci Scapy. Ta byla představena v části 3.1 Rozbor stávajících řešení. Tato funkce je volána s jedním parametrem, kterým je popis obsahu datového rámce. Tento popis je uveden pomocí jazyka, který je použit v samotné aplikaci Scapy, a musí

obsahovat popis jednotlivých částí pomocí hlaviček, které mohou být za sebou zřetězeny. Jak lze vytvořit datový obsah, který se bude skládat z ARP dotazu, ukazuje ukázka 4.4.

Ukázka 4.4: Vygenerování datového obsahu

```
datovyObsah = scapy('ARP(hwsrc = "00:21:91:71:54:f2")')
```

Pomocí příkazu scapy byl vygenerován ARP dotaz, kterému byla nastavena zdrojová MAC adresa. ARP dotaz není ale jediný, který může být pomocí této funkce vygenerován. Jednotlivé hlavičky a protokoly, které lze generovat pomocí aplikace Scapy, je možné naleznout na domovské stránce této aplikace odkazované ze zdroje [3] v části Literatura.

Před samotným obsahem, bezprostředně za hlavičkou IEEE, následuje hlavička LLC *(Logical link control)* [22]. Tato hlavička vždy následuje za hlavičkou IEEE. Pokud by nenásledovala, nebylo by možné vytvořit odpovídající ethernetový rámce používaný v drátových sítích Ethernet. Hlavička tohoto rámce obsahuje číslo protokolu, který za ní následuje. Tato informace v rámcích používaných v bezdrátových sítích WiFi chybí, a proto byla před další obsah přidána hlavička LLC, která toto číslo obsahuje. Aplikace scapy neumožňuje kompletně popsat tuto hlavičku, proto byl její popis zahrnut do realizované aplikace. Popis této hlavičky není náplní této práce, proto její podrobnější popis zde není uveden. Zájemci jej mohou naleznout například ve zdroji [22] odkazovaným v závěru práce.

Aplikace při generování rámce rozlišuje stav, kdy LLC hlavička byla definována a kdy nebyla. V případě, kdy hlavička definována byla, je provedeno její vygenerování a vložení mezi hlavičku IEEE a datový obsah. V opačném případě aplikace předpokládá, že je součástí datového obsahu a její generování neprovádí.

4.5 Šifrování datových rámců

Pro šifrování datových rámců bylo použito metod popsaných v kapitole Úvod do bezdrátových sítí standardu 802.11. Těmito metodami jsou metody **WEP**, **TKIP** a metoda **CCMP**.

Nástroj aircrack-ng již obsahuje implementovanou metodu pro šifrování rámců pomocí metody WEP. Tohoto bylo využito a tato metoda byla z tohoto nástroje převzata. Další šifrovací metody již tento nástroj neobsahuje. Ale obsahuje metodu pro dešifrování rámců pomocí metody TKIP. Ta se od metody pro šifrování téměř neliší. V první části metody je proveden výpočet klíče, dle postupu uvedeného v druhé kapitole. Ten je pro obě metody shodný. Nakonec se provede zašifrování případně dešifrování rámce pomocí metody WEP. Postačovalo tedy pouze pozměnit tuto metodu. Původní metoda pro dešifrování byla nahrazena metodou pro šifrování.

U metody CCMP tomuto tak již není. Zde se metoda již částečně liší a bylo by zapotřebí tuto metodu pozměnit více. Abychom předešli případným chybám, které by při úpravě mohly vzniknout, byla použita implementace této metody z následujícího zdroje [24]. Ta byla přidána do původního hlavičkového souboru nástroje aircrack-ng, který se stal součástí naší aplikace.

Při ověřování správnosti šifrování datových rámců pomocí metod TKIP a CCMP jsme narazili na problém nalezení správného šifrovacího klíče. Je zapotřebí mít na paměti, že při šifrování těchto rámců se využívá dvou rozdílných klíčů. Pokud bychom použili nesprávný klíč, zašifrování se nezdaří. Jak získat jednotlivé klíče můžeme naleznout v následující části.

4.6 Načítání klíčů

K tomu, aby rámec mohl být zašifrován, je zapotřebí správného šifrovacího klíče. Které šifrovací klíče se u dané metody používají, bylo popsáno v kapitole druhé Úvod do bezdrátových sítí standardu 802.11.

Aplikace umožňuje načtení **PTK** a **GTK** klíčů použitých při šifrování metodou TKIP a CCMP. Načítání klíčů je provedeno za pomoci aplikace *wpa_supplicant* [9]. Tato aplikace slouží jako klient pro práci s bezdrátovými sítěmi. Tento klient se pokouší připojit k bezdrátové síti s použitím konfigurace, která je definována v konfiguračním souboru. Ten obsahuje informace potřebné pro připojení k bezdrátové síti.

Načítání klíčů v aplikaci je provedeno pomocí daemona, který běží souběžně s hlavní aplikací. Tento daemon se postará o spuštění aplikace *wpa_supplicant*. Ta při svém běhu vypisuje aktuální informace o stavu připojení. Součástí těchto informací jsou i oba požadované klíče. Spuštění *wpa_supplicantu* je provedeno pomocí příkazu uvedeném v ukázce 4.5. V tomto příkazu je zapotřebí specifikovat rozhraní, které bude použito pro připojení stanice k bezdrátové síti, a cestu ke konfiguračnímu souboru.

Ukázka 4.5: Spuštění wpa_supplicant

```
wpa_supplicant -dd -i wlan0 -c config.conf -K
```

Abychom mohli v aplikaci číst výstup wpa_supplicantu bylo provedeno přesměrování jeho výstupu přes rouru *pipe* na vstup našeho daemona. Ten následně provádí analýzu získaného vstupu a vyhledává potřebné klíče. Ty po jejich nalezení uloží do interních proměnných, ze který je možné si je načíst pomocí příkazů **getPTK** a **getGTK** do vlastní proměnné. Tuto proměnnou, která bude obsahovat načtený klíč, můžeme následně použít pro definici šifrovacího klíče použitého při šifrování rámce. Jak je možné provést načtení klíčů v aplikaci, zobrazuje ukázka 4.6.

Ukázka 4.6: Načtení šifrovacího klíče

```
key 'wlan0' 'config.conf'
gtkklic = getGTK()
ramec.payloadKey = gtkklic
```

V této ukázce bylo provedeno spuštění našeho daemona starající se o načítání klíčů z výstupu *wpa_supplicantu*. Při jeho spuštění je zapotřebí specifikovat použité rozhraní a konfigurační soubor. V okamžiku, kdy je provedeno načtení klíče je možné provést jeho načtení do proměnné a následně ho přiřadit jako šifrovací klíč datového obsahu rámce.

Zdali jsou již klíče načteny nebo ne, je možné zjistit pomocí příkazu **key list**. Tento příkaz provede vypsání obou načtených klíčů. V případě, kdy klíče ještě nejsou načteny, bude výstup obsahovat pouze hodnoty "*0x*".

Námi spuštěný daemon běží na pozadí nepřetržitě po celou dobu spuštění aplikace. V okamžiku, kdy by se v síti změnily klíče, bude toto *wpa_supplicant* signalizovat jejich vypsáním. Tento výstup zaznamená náš daemon a provede jejich uložení. Uživateli pak už postačuje opětovně načíst klíče z interních proměnných.

4.7 Zasílání rámců

Po vygenerování a případně po zašifrování datového rámce umožňuje aplikace provést jeho zaslání přes bezdrátové rozhraní. Toto rozhraní je specifikováno při spuštění aplikace a je tak jediným parametrem, který aplikace vyžaduje. Je zapotřebí, aby toto rozhraní bylo v takzvaném monitorovacím

módu. Jak uvést bezdrátové rozhraní do monitorovacího módu bylo popsáno v kapitole Návrh, kde bylo použito k odchycení ukázkového rámce.

K zasílání nebo též k injekci rámců byla použita knihovna *libpcap*. Pro přístup k bezdrátovému rozhraní skrze tuto knihovnu byla vytvořena třída *WiFiInterface* představena v kapitole Návrh. Po inicializaci rozhraní, je možné provést zasílání vlastních rámců. K zasílání rámců byly implementovány dvě metody.

První metodou je metoda blokující. Ta blokuje aplikaci po dobu zasílání rámce. Tato metoda očekává jako svůj parametr vygenerovaný rámec, který posléze pomocí metod knihovny *libpcap* odešle. Aplikace je během této doby blokována a čeká, až bude rámec odeslán. Jak je možné rámec v aplikaci zaslat pomocí této metody demonstruje příklad na ukázce 4.7.

Ukázka 4.7: Blokující metoda zasílání rámců

```
ramec = IEEE(type="ack")
send(ramec)
```

Druhou metodou, kterou lze v aplikaci použít, je metoda neblokující. Tato metoda na rozdíl od metody předešlé neblokuje aplikaci po dobu odesílání rámce. Při volání této metody je vytvořeno nové vlákno, které se již postará o vlastní zaslání. Tato metoda jako své parametry přijímá odesílaný rámec a dále pak volitelně počet opakování zaslání a případně čas, který bude jednotlivá zaslání oddělovat. Tento čas je uveden v mikrosekundách. Výhodou této metody je, že je možné zasílat dva různé rámce v jeden okamžik. Pro každý ze zasílaných rámců se vytvoří vlákno, které se postará o jejich zaslání. Příklad zaslání rámce pomocí neblokující metody je zobrazen na příkladu v ukázce 4.8. V něm je zasílaný rámec zaslán celkem desetkrát a před každým zasíláním se čeká půl milisekundy.

Ukázka 4.8: Neblokující metoda zasílání rámců

```
ramec = IEEE(type = "beacon" ssid ='frm')
isend(ramec, 10, 500)
```

Abychom si mohli udělat představu o stavu odesílání daného rámce pomocí neblokující metody, umožňuje aplikace zobrazit stav zasílání jednotlivých rámců. Tento stav je zobrazen pomocí funkce **isend list**.

4.8 Záznam rámců

Existuje několik útoků, které ke svému vykonání vyžadují informace z rámců, které byly v síti použity. Typickým příkladem tohoto útoku je útok na zabezpečení WPA2. Aby se útočníkovi podařilo obejít toto zabezpečení, musí mít informace o aktuálně použitém inicializačním vektoru. Více o tomto útoku je možné naleznout v kapitole Testování a zhodnocení.

Abychom mohli tento útok napodobit je zapotřebí získat tento inicializační vektor. Ten získáme pomocí odchycení datového rámce, který jej obsahuje. K tomuto byla rovněž použita knihovna *libpcap*. Ta obsahuje sadu funkcí, pomocí níž je provedeno odchycení požadovaného rámce. Tento rámec je specifikován pomocí filtru, který má charakter textového řetězce. Tohoto jsme využili i v naší aplikaci, kterou jsme rozšířili o funkci **capture**. Tato funkce má jediný parametr, kterým je právě tento textový řetězec. Čím lépe je filtr specifikován, tím je větší pravděpodobnost, že odchytíme právě námi požadovaný rámec. Jednotlivé možnosti, které lze v tomto filtru specifikovat, jsou k vidění v tabulce 4.1. V ní jsou uvedeny pouze výrazy, které provedou filtrování rámců dle IEEE hlavičky. Zbylé výrazy jsou k nalezení například na stránce odkazované v publikaci [10] v závěru práce. Příklad použití funkce **capture** je uveden na ukázce 4.9.

Ukázka 4.9: Odchycení rámce

```
filtr = 'type data and wlan addr1 70:f1:a1:59:19:0e'
ramec = capture(filtr)
```

Funkce **capture** vrací odchycený rámec v podobě textového řetězce obsahující data v hexadecimální soustavě. Tento rámec si můžeme uložit do proměnné pro pozdější využití. Tím může být zobrazení odchyceného rámce pomocí funkce **dump**. Dále z tohoto rámce může být získán již zmiňovaný inicializační vektor. Jak je tohoto docíleno, je popsáno dále.

Výraz	**Popis**
wlan addr1 *MAC adresa*	první adresa rámce *(obvykle adresa cíle)*
wlan addr2 *MAC adresa*	druhá adresa rámce *(obvykle BSSID adresa)*
wlan addr3 *MAC adresa*	třetí adresa rámce *(obvykle adresa zdroje)*
wlan addr4 *MAC adresa*	čtvrtá adresa rámce
type *wlan_type*	typ rámce, *wlan_type* může nabývat hodnot **mgt** *(rámce správy sítě)*, **ctl** *(řídící rámce)* a **data** *(datové rámce)*
type *wlan_type* **subtype** *wlan_subtype*	typ a podtyp rámce, *wlan_subtype* může nabývat hodnot **assoc-req**, **assoc-resp**, **reassoc-req**, **reassoc-resp**, **probe-req**, **probe-resp**, **beacon**, **atim**, **disassoc**, **auth**, **deauth** pro rámce správy sítě ***(mgt)***, **ps-poll**, **rts**, **cts**, **ack**, **cf-end**, **cf-end-ack** pro řídící rámce ***(ctl)*** a **data**, **data-cf-ack**, **data-cf-poll**, **data-cf-ack-poll**, **null**, **cf-ack**, **cf-poll**, **cf-ack-poll**, **qos-data**, **qos-data-cf-ack**, **qos-data-cf-poll**, **qos-data-cf-ack-poll**, **qos**, **qos-cf-poll**, **qos-cf-ack-poll** pro rámce datové ***(data)***
subtype *wlan_subtype*	podtyp rámce, *wlan_subtype* nabývá hodnot shodných s výrazem výše
dir *dir*	určuje směr zaslání rámce, *dir* může nabývat hodnot **nods** *(použito v Ad-hoc sítích)*, **tods** *(směr k AP)*, **fromds** *(směr ke klientovi)* a **dstods** *(použito v distribučním systému)*
and, **or**, **not**	logické spojky pomocí niž je možné jednotlivé výrazy spojovat. Lze využít i spojek & místo **and**, | místo **or** a ! místo **not**.

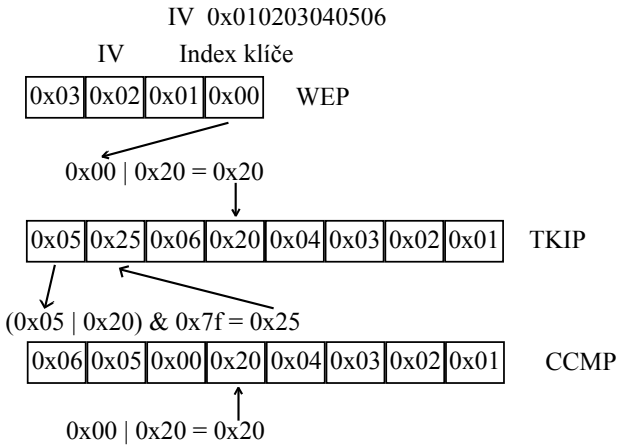
Tabulka 4.1: Výrazy filtru pro odchycení rámců [10]

4.9 Získání inicializačního vektoru

Abychom mohli dokončit útok naznačený v předchozí části je zapotřebí získat onen inicializační vektor. Aplikace v současné době funguje pouze jako generátor rámců a nikoliv jako jejich analyzátor. Tudíž aplikace neumožňuje převod odchyceného rámce na interní reprezentaci, která je použita v aplikaci. Proto bylo zapotřebí, přidat funkci, která z datového rámce získá požadovaný inicializační vektor. Tato funkce má pojmenování **getiv** a volitelně očekává jeden parametr. Tímto parametrem je již dříve odchycený rámec, který byl pomocí funkce *capture* uložen do proměnné jako textový řetězec. Tento parametr, ale může být také vynechán. Funkce *capture* si ve své interní proměnné uchovává poslední odchycený rámec a v případě je-li funkce **getiv** volána bez parametrů je použit právě tento rámec.

Při samotném získávání inicializačního vektoru je provedena analýza předloženého rámce. Zde se nejprve provede přeskočení hlavičky RadioTap, jejíž délka je uložena v proměnné *len*. Dále je pak

provedena analýza hlavičky IEEE. Zde jako první krok je provedena kontrola typu rámce. Pokud typ rámce není roven datovému rámci, vrací funkce hodnotu nula. Tu vrací i v ostatních případech, kdy se nejedná o správný rámec. Po kontrole typu rámce je ověřeno nastavení příznaků *toDS* a *fromDS*, abychom určili správný počet adres v hlavičce. V případě, kdy jsou oba příznaky nastaveny, obsahuje hlavička celkem čtyři adresy. V ostatních případech obsahuje pouze adresy tři. Poté je provedena identifikace přítomnosti informací QoS. Na závěr je ověřeno, zdali je rámec zašifrován a v případě že ano, kterou z dříve jmenovaných metod. Rozlišení jednotlivých metod je možné dle následujícího postupu. Je-li hodnota indexu klíče, nacházející se v rámci za inicializačním vektorem, zvětšena o hodnotu $0x20$ jedná se o šifrování pomocí metod TKIP nebo CCMP. V opačném případě se jedná o metodu WEP. Rozlišení metod TKIP a CCMP je provedeno pomocí upravení inicializačního vektoru. V případě zabezpečení TKIP je hodnota druhého bajtu vektoru po úpravě rovna hodnotě prvního bajtu. Touto úpravou je operace $(iv|0x20)\&0x7f$. Pokud je splněna tato podmínka jedná se o zabezpečení pomocí metody TKIP. V opačném případě se jedná o metodu CCMP. Pro lepší představu je tento rozdíl demonstrován na obrázku 4.3.



Obrázek 4.3: Uložení inicializačního vektoru

Použití metody *getiv* je demonstrováno na ukázce 4.10. Zde je provedeno odchycení rámce a následná extrakce inicializačního vektoru.

Ukázka 4.10: Získání inicializačního vektoru

```
capture('type data')
vector = getiv()
```

4.10 Ukládání rámců do souboru

Aby bylo možné popsané rámce používat opakovaně i po ukončení aplikace, bylo v aplikaci realizováno ukládání rámců do souboru. Ty jsou v aplikaci exportovány do XML *(Extensible Markup Language)* souborů. Z nich je možné rámce opětovně načíst a pracovat s nimi jako před jejich uložením. K tvorbě XML souborů byla použita knihovna *TinyXML*.

Uložení rámce je v aplikaci umožněno pomocí příkazu **save**, který je volán jako metoda ukládaného rámce. Tento příkaz očekává jediný parametr, kterým je cesta ukládaného rámce. Tato cesta může být opět zadaná prostřednictvím textového řetězce případně pomocí aliasu. Aplikace vyžaduje, aby název souboru s vytvořeným rámcem měl příponu *frm*. Uložení rámce je zachyceno na ukázce 4.11.

Ukázka 4.11: Uložení rámce do souboru

```
ramec = IEEE(type = "authentication")
ramec.save('./saveFrame/frame.frm')
```

Uložený rámec je možné opětovně načíst pomocí příkazu **load**. Použití tohoto příkazu je shodné s příkazem pro uložení.

4.11 Aliasy adres

Adresy používané k uložení rámců a k načítání programů mohou být někdy velmi dlouhé. Z tohoto důvody byla aplikace rozšířena o reprezentaci adresy pomocí aliasu. Alias je opět reprezentován pomocí proměnné. Tato proměnná je ovšem uložena v odlišné tabulce, než je tabulka běžných proměnných. V tomto případě se jedná o tabulku aliasů. Z tohoto důvodu aplikace umožňuje použít shodný název proměnné pro alias i pro reprezentaci rámce. Zároveň toto ale znemožňuje uložit si adresu do obyčejné proměnné a tu následně použít při načítání rámce.

Při práci s aliasem je zapotřebí použít klíčového slova **alias**. Aplikace poté ví, že bude pracovat nad odpovídající tabulkou. Pro definici nového aliasu je použito přiřazení, v němž přiřazujeme adresu v operačním systému. Touto adresou může být absolutní i relativní cesta obsahující název rámce případně název programu. Soubor obsahující uložený rámec obsahuje vždy příponu *frm*. Oproti tomu soubor obsahující uložený program osahuje příponu *txt*. Uložený alias je možné ze seznamu rovněž odstranit. Toto se provede pomocí příkazu **del**. Seznam aliasů je možné vypsat pomocí příkazu **list**. I těmto příkazům musí předcházet klíčové slovo **alias**, aby aplikace věděla, nad kterou tabulkou má pracovat. Demonstraci použití aliasů ukazuje příklad v ukázce 4.12. V tomto příkladu je nejdříve provedeno uložení nového aliasu do proměnné *ramec*. Ten je následně použit pro načtení rámce. Proměnná obsahující načtený rámec je ale odlišná od proměnné, ve které je uložen daný alias. Nově vytvořený alias je na závěr odstraněn příkazem **del**.

Ukázka 4.12: Použití aliasu

```
alias ramec = './saveFrame/frame.frm'
ramec.load(ramec)
alias del(ramec)
```

Aby po ukončení aplikace nebyly ztraceny informace o jednotlivých aliasech, je převedeno jejich uložení do konfiguračního souboru. Z tohoto souboru jsou při opětovném spuštění aplikace jednotlivé aliasy načteny.

Kapitola 5

Testování a zhodnocení

Následující kapitola se zabývá testováním realizované aplikace. Pro účely testování byly zvoleny dva útoky na bezdrátovou síť. Touto sítí byla vlastní domácí síť, na kterou se pro účely testování útočilo. Jednotlivé útoky zde můžeme naleznout nejdříve popsané po teoretické stránce. Následně je uveden popis, jak daný útok probíhal a jakých bylo dosaženo výsledků. Na závěr kapitoly můžeme naleznout krátké zhodnocení aplikace a výsledků, kterých bylo pomocí ní dosaženo.

5.1 Flood útoky

Prvním zvoleným útokem byl Flood útok [29]. Tento útok spočívá v generování velkého množství rámců jistého typu s cílem zamezit používání sítě oprávněným uživatelům. Tento typ útoku je také označován jako DoS útok *(Denial of Service)* [28]. Pro účely testování jsme zvolili generování rámců typu RTS *(Request to Send)* a rámců typu CTS *(Clear to Send)*. Jednotlivé útoky pak nesou pojmenování dle použitého typu rámce. V našem případě se jedná o RTS flood útok a CTS flood útok.

Oba zvolené útoky využívají principu, který byl popsán v kapitole druhé v části 2.3 Metody přístupu k přenosovému médiu. Konkrétně se jedná o metodu *DFMAC-DCF s RTS/CTS rozšířením*. Každý z nich však využívá k útoku jiné části této metody.

RTS flood útok

Tento útok spočívá v generování velkého množství RTS rámců, které jsou adresované přístupovému bodu sítě. Pomocí tohoto rámce sděluje útočící stanice přístupovému bodu svůj požadavek na použití přenosového média. Pokud je médium volné, odpovídá přístupový bod rámcem CTS, pomocí něhož přijímající stanici sděluje, že může začít vysílat. Tento rámec přijmou i ostatní stanice v okolí přístupového bodu. Pomocí něj se dozvídají, že bude právě probíhat komunikace a mají tedy své požadavky pozdržet.

CTS flood útok

Při tomto útoku je vynechána první fáze metody přístupu k bezdrátovému médiu. Tedy není prováděno generování rámců RTS, ale jsou již přímo generovány rámce CTS. Zde lze využít skutečnosti, že tyto rámce neobsahují adresu odesílající stanice, ale pouze adresu stanice přijímající. Stanice, které přijmou tyto rámce, si nemohou ověřit, kdo je skutečně zaslal. Proto se domnívají, že byly zaslány přístupovým bodem sítě. Ty jsou opět nuceni pozdržet své vysílání na dobu, než je ukončen následující přenos.

Nevýhodou tohoto útoku je, že generované rámce přijmou pouze stanice, které jsou v dosahu stanice útočníka. Stanice, které jsou mimo jeho dosah, mohou dále komunikovat.

5.1.1 RTS flood útok

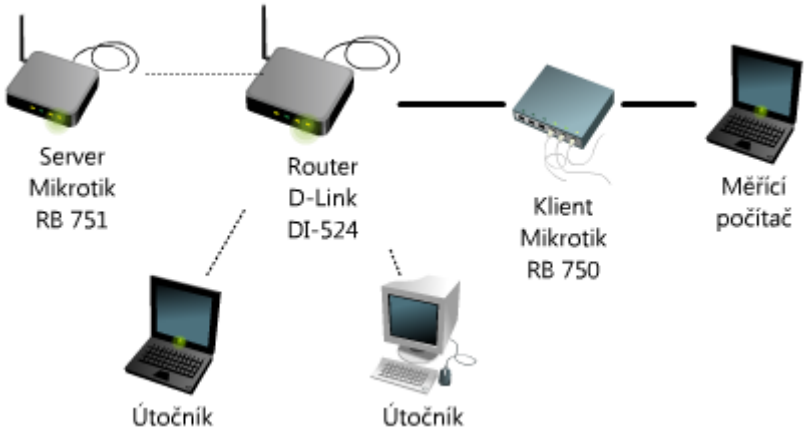
Prvním z realizovaných útoků byl RTS flood útok. Při tomto útoků jsme prováděli generování velkého množství RTS rámců. Popis a následné generování těchto rámců bylo provedeno pomocí realizované aplikace. Program tohoto útoků můžeme vidět na ukázce 5.1. Tato ukázka se skládá pouze z první fáze útoku. Kompletní program útoku je k dispozici na přiloženém CD.

Ukázka 5.1: RTS útok

```
rtsRamec = IEEE(type = "rts")
rtsRamec.duration = 0x44
rtsRamec.receiverAddress = "00:21:91:71:54:f2"
rtsRamec.transmitterAddress = "70:f1:a1:00:11:33"

for(;;) {
    send(rtsRamec)
    usleep(1950)
}
```

Útok byl rozdělen celkem do čtyř fází, kde v každé z ní byl generován odlišný počet rámců. Dále pro první tři fáze útoku bylo použito jediného počítače. Ve čtvrté fázi jsme zapojili počítač druhý, díky němuž jsme dosáhli většího počtu generovaných rámců. Abychom byly schopni generovat v každé fázi rozumný počet rámců, bylo experimentálně zjištěno, jaké musí být zpoždění mezi jednotlivými zasláními.

Pro účely realizace útoku bylo použito několik zařízení. Rozmístění těchto zařízení můžeme vidět na obrázku 5.1. Jejich popis pak můžeme naleznout v tabulce 5.1.

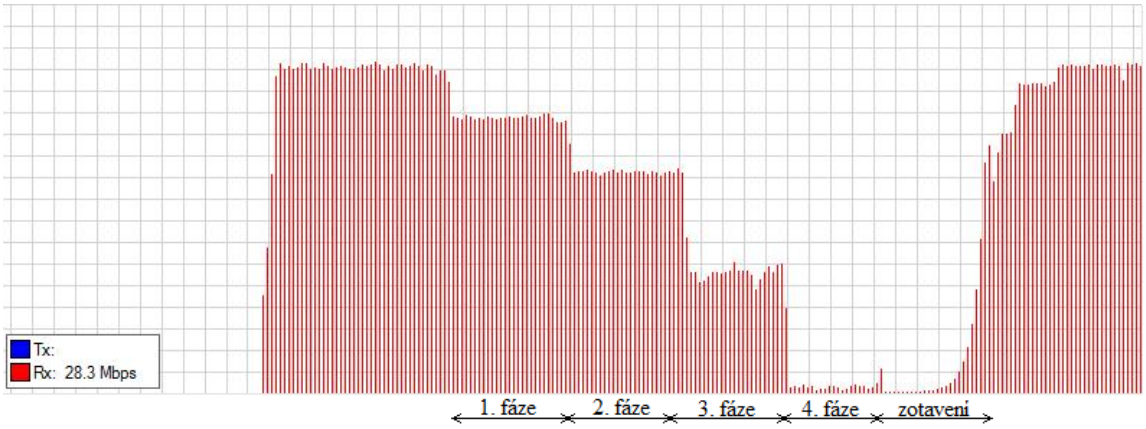


Obrázek 5.1: Schéma rozmístění zařízení při Flood útocích

Zařízení	Účel
D-Link DI-524	AP bezdrátové sítě
Mikrotik RB 751	bezdrátový server poskytující data
Mikrotik RB 750	klient, který stahováním dat měří rychlost sítě
Měřící počítač	počítač, na kterém byla měřena doba odezvy příkazu Ping
Útočník	počítače, ze kterých byl veden útok na síť

Tabulka 5.1: Použitá zařízení při Flood útocích

Při probíhajícím útoku bylo sledováno, jakým způsobem se mění průměrná přenosová rychlost sítě a doba odezvy na příkaz *PING*. Graf změny přenosové rychlosti je možné naleznout na obrázku 5.2. Na něm můžeme pozorovat změnu přenosové rychlosti během různých fází útoků. Můžeme si dále povšimnout, že se nám podařilo změnit přenosovou rychlost sítě z původní hodnoty 28 Mbps *(Mbps - megabitů za vteřinu)* na hodnotu přibližně 400 Kbps *(Kbps - kilobitů za vteřinu)*.



Obrázek 5.2: Změna maximální přenosové rychlosti během RTS flood útoku

Doba odezvy při maximálním počtu rámců vzrostla během útoku z původní průměrné hodnoty 1,5 ms na hodnotu 211 ms. Jednotlivé naměřené hodnoty během útoku jsou shrnuty v tabulce 5.2.

Fáze útoku	Počet zařízení	Počet rámců / s	Rychlost sítě [Mbps]	Odezva[ms]
0	1	0	28,00	1,68
1	1	459	23,80	2,22
2	1	957	19,10	5,25
3	1	1849	10,40	20,39
4	2	3507	0,40	211,69

Tabulka 5.2: Naměřené hodnoty RTS flood útoku

5.1.2 CTS flood útok

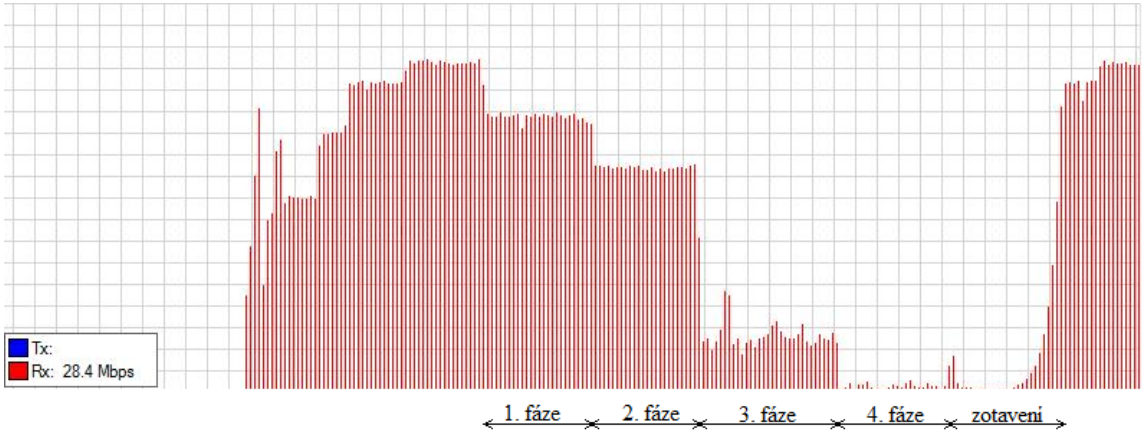
Druhý z realizovaných flood útoků byl CTS flood útok. Postup tohoto útoku byl shodný s předchozím útokem. Jedinou změnou bylo použití odlišného typu rámců. Těmi byly rámce CTS. Popis těchto rámců je uveden na ukázce 5.2. Celý program tohoto útoku je k nalezení rovněž na přiloženém CD.

Ukázka 5.2: Popis CTS rámce

```
ctsRamec = IEEE(type = "cts")
ctsRamec.duration = 0x44
ctsRamec.receiverAddress = "70:f1:a1:00:11:33"
```

Schéma zapojení bezdrátové sítě a použitá zařízení byla shodná s předchozím útokem, a proto již nebudou opětovně uváděny.

I v případě tohoto útoku byla pozorována změna přenosové rychlosti sítě. Ta z původní hodnoty 28,5 Mbps klesla na hodnotu 790 Kbps. Jak se měnila přenosová rychlost sítě v různých fázích útoku můžeme opět pozorovat na obrázku 5.3.



Obrázek 5.3: Změna maximální přenosové rychlosti během CTS flood útoku

Druhou měřenou veličinou byla opět doba odezvy na příkaz PING. Ta se v případě CTS flood útoku změnila z původní průměrné hodnoty 1,2 ms na konečnou hodnotu 237 ms. Shrnutí všech naměřených hodnot můžeme naleznout v tabulce 5.3.

Fáze útoku	Počet zařízení	Počet rámců / s	Rychlost sítě [Mbps]	Odezva[ms]
0	1	0	28,50	1,24
1	1	514	23,80	1,27
2	1	1018	19,30	1,34
3	1	1728	4,80	4,34
4	2	3598	0,79	237,31

Tabulka 5.3: Naměřené hodnoty CTS flood útoku

5.1.3 Vyhodnocení flood útoků

V rámci testování pomocí flood útoků byly provedeny dva různé útoky. Při každém z nich byly generovány rámce odlišného typu. V obou případech se nám podařilo snížit přenosovou rychlost sítě na hodnotu, při které by i běžný uživatel zaznamenal, že není se sítí něco v pořádků. Dosažení maximálního minima se nám podařilo, ale až při paralelním použití dvou počítačů. V případě, kdy jsme prováděli generování rámců na dvou různých rozhraních téhož počítače, se nám toto nepodařilo. Zde jsme dosáhli maximálních hodnot shodných při použití jediného rozhraní počítače. Toto můžeme pravděpodobně přisoudit nedostatečnému výkonu počítače, ze kterého byl útok veden. V případě menšího počtu generovaných rámců se použití více rozhraní na daném počítači projevilo a celková sílá útoku se tak zdvojnásobila.

5.2 Hole196

Druhým typem útoku, který byl v rámci testování realizován, byl útok využívající zranitelnosti zabezpečovacího protokolu WPA2 [13]. Tato zranitelnost je nazývána Hole196. Útok využívající této zranitelnosti je již v porovnání s předchozími Flood útoky složitějším a bez potřebných nástrojů bychom jej jen obtížně realizovali.

5.2.1 Zranitelnost Hole196

Této zranitelnosti smí využít pouze útočník, který je přihlášen do dané bezdrátové sítě. V kapitole druhé v části 2.5 Zabezpečení bezdrátových sítí standardu 802.11 bylo popsáno, jak jsou jednotlivé rámce šifrovány a jaké jsou k tomu využívány klíče. Hole196 využívá zranitelnosti používání klíče GTK, pomocí něhož jsou šifrovány multicastové a broadcastové rámce. Pokud je útočník přihlášen do bezdrátové sítě zná také tento klíč. Poté mu je umožněno generovat validní rámce šifrované právě tímto klíčem.

Aby se mu podařilo vytvořit validní rámec, který by ostatní stanice přijaly, musí vlastnit odpovídající inicializační vektor. Tento vektor musí být větší než inicializační vektor použitý k zašifrování posledního broadcastového případně multicastového rámce. Zároveň však nesmí být moc velký, jinak by stanice rámec odmítly. V okamžiku, kdy útočník vlastní správný inicializační vektor a zná potřebný GTK klíč, smí generovat validní broadcastové rámce, tak jako kdyby je generoval samotný přístupový bod sítě. Klientské stanice následně tyto rámce přijmou a provedou jejich zpracování.

5.2.2 Realizace útoku

Při tomto útoku jsme prováděli generování datových rámců, které obsahovaly jako datovou část ARP dotaz. Tento dotaz slouží k získání MAC adresy *(Media access control)* stanice vlastnící danou IP adresu. Tento dotaz je zasílán pomocí broadcastového rámce všem stanicím, které jsou připojeny k dané bezdrátové síti. Stanice, která vlastní tuto IP adresu, na tento dotaz reaguje pomocí vygenerování ARP odpovědi. V ní sděluje dotazující stanici svoji MAC adresu. Zároveň si ukládá adresu dotazujícího do své ARP cache paměti jako dvojici IP adresa a odpovídající MAC adresa.

Abychom mohli vygenerovat validní rámec, bylo zapotřebí provést načtení aktuálního GTK klíče a získání posledního použitého inicializačního vektoru. K tomuto bylo využito odpovídajících funkcí, které byly popsány v části Implementace. Po získání těchto informací jsme mohli provést vygenerování nového rámce a jeho následné zaslání. Program, který byl k tomuto útoku použit, můžeme vidět na ukázce 5.3.

Ukázka 5.3: Útok využívající zranitelnosti Hole196

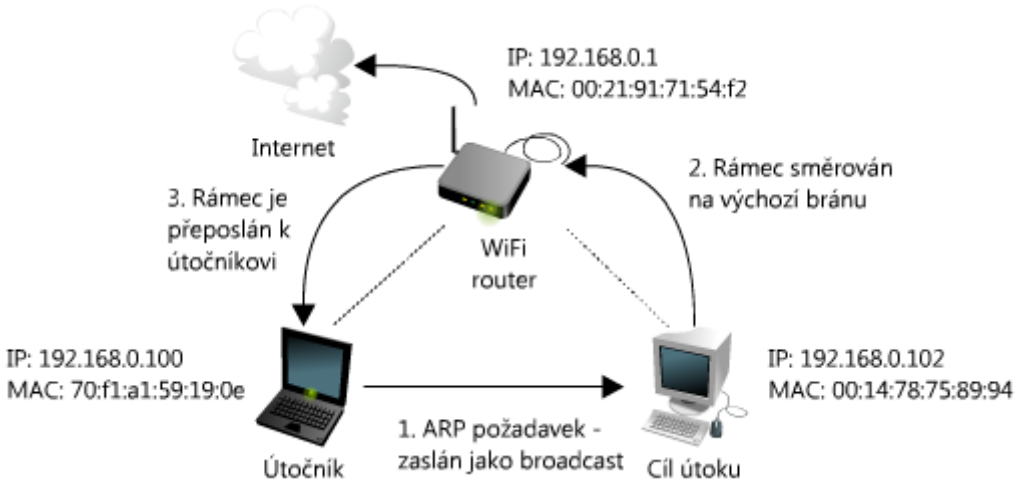
```
ramec= RadioTap(flags = "crc") /
       IEEE(frameControl = 0x4208
            duration = 0
            addr1 = "ff:ff:ff:ff:ff:ff"
            addr2 = "00:21:91:71:54:f2"
            addr3 = "70:f1:a1:59:19:0e"
            seq=2065) /
       LLC(llcType="apr")
arpDotaz = scapy('ARP(psrc="192.168.0.1", pdst="192.168.0.102",
                      hwsrc="70:f1:0a1:59:19:0e")')
ramec.payloadData = arpDotaz
ramec.payloadKeyIndex = 1
ramec.payloadCipher = "ccmp"

key 'wlan0' 'config.conf'

for(;;) {
   odchycenyRame = capture('type data and
                            wlan addr1 ff:ff:ff:ff:ff:ff and
                            wlan addr2 00:21:91:71:54:f2 and not
                            wlan addr3 70:f1:a1:59:19:0e and
                            dir fromds')
   iv = getIV(capturedFrame)
   frame.payloadVector = iv + 1
   klic = getGTK()
   ramec.payloadKey = klic
   send(ramec)
}
```

Tento útok dále vyžaduje správné nastavení MAC adres rámce a ARP dotazu. První adresa rámce je adresa, na kterou se daný rámec zasílá. V našem případě je to broadcastová adresa obsahující hodnoty "$0xff$". Druhá adresa obsahuje adresu *BSSID* sítě a třetí adresa obsahuje adresu odesílající stanice. Rámec musí dále obsahovat nastavený příznak *FromDS*, který říká, že rámec směřuje z distribučního systému.

Jak již bylo uvedeno, datovým obsahem rámce je ARP dotaz. Tomuto dotazu předchází hlavička LLC *(Logical link control)* [22], která je vždy obsažena v datovém obsahu. Důvod její přítomnosti byl popsán v části 4.4 Obsah datových rámců kaptitoly Implementace. Samotný ARP dotaz poté obsahuje MAC adresu a IP adresu odesilatele a IP adresu příjemce. MAC adresa příjemce je vyplněna hodnotami "$0x00$", protože je v daný okamžik neznámá. Příjemce provede jejich nahrazení za vlastní MAC adresu a tu odešle nazpět v ARP odpovědi. Použité stanice během vlastního útoku a jejich odpovídající adresy můžeme vidět na obrázku 5.4.

5.2.3 Vyhodnocení útoku

Pomocí útoku popsaném výše se nám podařilo pomocí ARP dotazu pozměnit obsah ARP cache paměti oběti útoku. Obsah ARP cache paměti je k vidění na obrázku 5.5. Ta obsahovala před začátkem útoku záznam, kde byla u IP adresy 192.168.0.1 *(adresa výchozí brány)* uvedena MAC adresa 00:21:91:71:54:f2. Po provedení útoku se nám podařilo tuto MAC adresu změnit na adresu



Obrázek 5.4: Schéma rozmístění stanic během útoku

70:f1:a1:59:19:0e. Od tohoto okamžiku by veškerá komunikace směřující na výchozí bránu byla přeposílána přístupovým bodem na adresu, kterou se nám podařilo umístit do cache paměti oběti.

```
root@pavel-desktop:/home/pavel# arp -n
Adresa HWtyp HWadresa Příznaky Maska Rozhraní
192.168.0.1              ether   00:21:91:71:54:f2   C                     wlan1
root@pavel-desktop:/home/pavel# arp -n
Adresa HWtyp HWadresa Příznaky Maska Rozhraní
192.168.0.1              ether   70:f1:a1:59:19:0e   C                     wlan1
```

Obrázek 5.5: Obsah ARP cache paměti oběti útoku

5.3 Zhodnocení výsledků

Pomocí aplikace se nám podařilo realizovat dva odlišné útoky s odlišnou složitostí. Realizace těchto útoků například pomocí programovacího jazyka C by nám zabrala mnoho času a byla by potřeba napsat řada programových řádků. Pro srovnání uveďme útok využívající zranitelnosti *Hole196*. Realizace tohoto útoku v jazyku C by trvala méně zkušenému uživateli asi jeden týden. Do tohoto času počítáme i čas využitý na nastudování potřebných informací a čas potřebný na realizací a odladění použité aplikace. Při jeho realizaci by byl uživatel nucen napsat více jak sto padesát programových řádků. Celková délka programu by se ale také odvíjela od použitých nástrojů a množství použitých knihovních funkcí. Zde by bylo zapotřebí zdůraznit, že ne všechny metody potřebné pro realizaci útoku je možné v knihovnách naleznout a bylo by je potřeba teprve realizovat. Proto výsledná délka programu by mohla být tedy i delší. Pomocí realizované aplikace byl tento útok uskutečněn pomocí krátkého programu, který má patnáct řádků a jeho napsání trvalo deset minut.

Nyní je už pouze na uživateli, který útok by pomocí této aplikace realizoval. Před sebou má nástroj, pomocí něhož je možné vytvářet i složité útoky v poměrně krátkém čase. Těchto útoků by ale neměl zneužívat. Nástroj byl realizován s cílem napomoci při odhalování zranitelností, které by bylo zapotřebí ošetřit, tak aby nemohly být dále zneužívány.

Kapitola 6

Rozšíření aplikace

V následující části je v krátkosti pojednáno, jaké aplikace poskytuje možnosti pro její budoucí rozšíření. Jednotlivé uvedené možnosti naznačují, co by bylo dále vhodné doplnit, případně co by bylo zapotřebí poupravit, tak aby bylo možné aplikaci využívat v plné míře.

6.1 Uživatelské rozhraní

První rozšíření aplikace se týkají uživatelského rozhraní. Aplikace je realizována jako konzolová aplikace, která s sebou přináší i typický způsob ovládání. Tím je zadávání příkazů prostřednictvím dané konzoly. Pro aplikaci tohoto typu je ale toto ovládání dostačující.

Nedostatkem naší aplikace, který v ní shledáváme a jsme si toho vědomi, je nedokonalost zadávání jednotlivých příkazů. Uživateli prozatím není umožněno používat historii příkazů a je mu znemožněn posun kurzoru po napsaném příkazu, pomocí něhož by v něm například opravil překlep. Aplikace by tedy v brzké době měla být o toto rozšířena. Dále by pak bylo vhodné realizovat nápovědu možných příkazů, které by se doplňovaly například po stisku klávesy *tabelátor*.

Jednotlivá rozšíření jsou v krátkosti shrnuta v následujícím seznamu:

- uchování historie příkazů
- možnost pohybu kurzoru po aktuálním příkazu
- kontextová nápověda klíčových slov, která by byla vkládána po stisku vyhrazené klávesy

6.2 Analýza rámců

Druhý typ rozšíření se týká analýzy již vygenerovaných rámců. Aplikace v současné době pracuje pouze jako jejich generátor. Ze zadaného popisu umožňuje vygenerovat odpovídající rámec. V některých případech by bylo ale vhodné, kdyby aplikace umoňovala i jejich obrácený převod. Tedy umožnít z již vygenerovaných rámců získat jejich odpovídající popis. Aplikace by ale k tomuto potřebovala vhodný analyzátor, který by prováděl daný převod. Nevýhodou hlaviček RadioTap a IEEE je jejich rozmanitost. Dostáváme řadu rámců, které jsou mnohokrát velmi odlišné. Z tohoto důvodu není realizace tohoto analyzátoru jednoduchá a nebylo jej možné zahrnout do realizované aplikace.

Využití tohoto analyzátoru by bylo možné například pro následnou analýzu odchycených rámců pomocí funkce *capture*. Z nich bychom následně mohli získat jednotlivé položky, které bychom mohli přiřazovat našemu popisovanému rámci. Příkladem této položky je inicializační vektor, který

je v současnosti získáván pomocí připravené funkce. Ta ale umožňuje získat pouze inicializační vektor. Někdy bychom ale mohli potřebovat i jiné položky než pouze inicializační vektor. Těmito položkami mohou být například adresy rámce a řada dalších informací.

Navrhovné rozšíření je opět uvedeno níže:

- analyzátor vygenerovaných rámců

6.3 Statistika útoků

Posledním z navrhovaných rozšíření je realizace vylepšené statistiky útoků. Abychom mohli lépe vyhodnocovat realizované útoky je zapotřebí vlastnit statistická data, která by nám napověděla, jak byl daný útok úspěšný. V současnosti umožňuje aplikace pouhé měření počtu zaslaných rámců v daném časovém intervalu. Toto měření je zprostředkováno pomocí příkazu pro zahájení měření času a inkrementace proměnné obsahující počet odeslaných rámců. Po ukončení útoku je měření času ukončeno pomocí odpovídajícího příkazu. Uživatel si poté sám může provést například přepočet průměrného počtu rámců za jednu vteřinu. Toto v řadě případů postačuje, ale existují případy, kdy bychom mohli potřebovat tuto statistiku podrobnější. A nemusí se jednat například jen o informaci počtu odeslaných rámců.

Dále pak například při útoku využívající zranitelnosti Hole196 by bylo vhodné vlastnit informaci, zdali oběť útoku daný rámec přijala. Tuto skutečnost by signalizovala pomocí zaslání odpovědi prostřednictvím ACK rámce. Aplikace by tento rámec měla být schopná zpracovat a uživateli říci, zdali přijetí rámce proběhlo úspěšně.

Navrhovaná rozšíření jsou i v tomto případě shrnuta do seznamu uvedeném níže:

- informace o průměrném počtu zaslaných rámců
- vhodné statistity útoků
- zpracování ACK rámců

Kapitola 7

Závěr

Náplní této práce byla realizace nástroje umožňující generování rámců používaných v bezdrátových sítích standardu 802.11. Po prostudování principů funkčností těchto sítí a následné analýze již realizovaných aplikací byl proveden návrh tří realizací nové aplikace. Z těchto realizací byla zvolena jedna, která byla doporučena k její implementaci. Touto realizací byl návrh nazvaný Interpret rámců, který realizuje popis generovaných rámců pomocí navrženého jazyka.

Výsledkem práce je realizovaná aplikace, která umožňuje popisovat rámce skládající se z hlaviček RadioTap a IEEE pomocí navrženého jazyka. V případě datových rámců je pak možné definovat datový obsah, který je umožněno zabezpečit jednou ze tří používaných metod. Jednotlivé metody byly představeny v části práce, která se věnovala popisu bezrátových sítí standardu 802.11. Z popisu specifikující jednotlivé rámce umožňuje aplikace provést jejich vygenerování a v případě potřeby jejich zaslání na předem specifikované rozhraní.

Použití této aplikace bylo demonstrováno na dvou útocích popsaných v kapitole Testování a zhodnocení. Pro tyto účely byly zvoleny dva útoky. Prvním byl jednodušší útok, jehož cílem je zabránit ostatním uživatelům používání dané sítě. Výsledkem bylo, že po zahájení útoku se stala tato síť téměř nepoužitelná vlivem poklesu přenosové rychlosti. Druhým útokem byl již složitější útok využívající zranitelnosti Hole196, který by bez odpovídajících nástrojů bylo jen obtížně možné realizovat. Naším cílem bylo pomocí této zranitelnosti pozměnit záznam v ARP cache paměti oběti útoku. Cíl útoku se zdařil a nám se podařilo pozměnit adresu výchozí brány, jež měla za následek přesměrování komunikace na útočící stanici. Podrobnější popis jednotlivých útoků je k nalezení rovněž v kapitole Testování a zhodnocení.

Pomocí této práce nám bylo umožněno získat nové znalosti a dovednosti v oblasti zabývající se bezdrátovými sítěmi. Mohli jsme si udělat představu, jak je komunikace v síti zprostředkována a co je zapotřebí realizovat, aby byla uskutečněna. Na závěr jsme si mohli vyzkoušet, jak je možné využít zranitelností, které obsahují návrh a realizace těchto sítí.

Literatura

[1] Radiotap. [online], cit. 2012-12-27.
URL http://www.radiotap.org/

[2] Manipulace síťového provozu a aktivní MITM. [online], cit. 2013-01-04.
URL http://wiki.airdump.cz/Manipulace_s%C3%AD%C5%A5ov%C3%A9ho_provozu_a_aktivn%C3%AD_MITM

[3] Scapy. [online], cit. 2013-01-04.
URL http://www.secdev.org/projects/scapy/

[4] Aircrack-ng. [online], cit. 2013-01-05.
URL http://www.aircrack-ng.org/

[5] The Lex & Yacc Page. [online], cit. 2013-01-05.
URL http://dinosaur.compilertools.net/

[6] TinyXML. [online], cit. 2013-01-05.
URL http://sourceforge.net/projects/tinyxml/

[7] Zulu. [online], cit. 2013-04-10.
URL http://zulu-wireless.sourceforge.net/

[8] Wireshark. [online], cit. 2013-04-11.
URL http://www.wireshark.org/

[9] Linux WPA/WPA2/IEEE 802.1X Supplicant. [online], cit. 2013-04-13.
URL http://hostap.epitest.fi/wpa_supplicant/

[10] Manpagez: man pcap-filter. [online], cit. 2013-04-13.
URL http://www.manpagez.com/man/7/pcap-filter/

[11] TCPdump & libpcap. [online], cit. 2013-04-14.
URL http://www.tcpdump.org/

[12] Nemesis. [online], cit. 2013-05-08.
URL http://nemesis.sourceforge.net/

[13] Ahmad, M.: Wpa too. [online], cit. 2013-04-15.
URL http://www.defcon.org/images/defcon-18/dc-18-presentations/Ahmad/DEFCON-18-Ahmad-WPA-Too-WP.pdf

[14] Anderson, R.: *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2010, ISBN 978-1-118-00836-2.

[15] Benton, K.: *The Evolution of 802.11 Wireless Security*. UNLV Informatics, 2010.
URL http://itffroc.org/pubs/benton_wireless.pdf

[16] Bigelow, S. J.: *Mistrovství v počítačových sítích*. Computer Press, 2004, ISBN 80-251-0178-9.

[17] Earle, A.: *Wireless Security Handbook*. Taylor & Francis, 2010, ISBN 0-8493-3378-4.

[18] Garcia, L. M.: Programming with Libpcap - Sniffing the Network From Our Own Application. [online], cit. 2013-04-23.
URL http://recursos.aldabaknocking.com/libpcapHakin9LuisMartinGarcia.pdf

[19] Gast, M. S.: *802.11 Wireless Networks - The Definitive Guide*. O'Reilly, 2002, ISBN 0-596-00183-5.

[20] Habiballa, H.; Volná, E.; Fojtík, R.: Od teorie formálních jazyků k jednoduchému překladači. [online], cit. 2013-04-14.
URL http://www1.osu.cz/home/Habibal/files/mfi5big.pdf

[21] Hanáček, P.; Staudek, J.: *Bezpečnost informačních systémů*. Úřad pro státní informační systém, Praha, 2000, ISBN 80-238-5400-3.

[22] IEEE: The Structure and Coding of Logical Link Control (LLC) Address: A Tutorial Guide. [online], cit. 2013-04-24.
URL http://standards.ieee.org/develop/regauth/tut/llc.pdf

[23] Lehembre, G.: Bezpečnost Wi-Fi - WEP, WPA a WPA2. [online], cit. 2012-12-31.
URL http://www.hsc.fr/ressources/articles/hakin9_wifi/hakin9_wifi_CZ.pdf

[24] Malinen, J.: CTR with CBC-MAC Protocol (CCMP). [online], cit. 2013-04-13.
URL https://github.com/cozybit/hostap-sae/blob/master/wlantest/ccmp.c

[25] Menezes, A.; van Oorschot, P.; Vanstone, S.: *Handbook of Applied Cryptography*. Taylor & Francis, 2010, ISBN 978-0-849-38523-0.

[26] Nichols, R.; Lekkas, P.: *Wireless Security*. McGraw-Hill Education (India) Pvt Limited, 2006, ISBN 978-00-7138038-6.

[27] Schiller, J.: *Mobile Communications*. Addison-Wesley, 2003, ISBN 0-321-12381-6.

[28] Shinder, D.; Cross, M.: *Scene of the Cybercrime*. Elsevier Science, 2008, ISBN 978-1-59749-276-8.

[29] Singh, R.: Wireless Network Security - Main threats at different layers. [online], cit. 2013-04-15.
URL http://www.rgsociety.org/journals/index.php/ijwwc/article/download/329/154

[30] Stallings, W.: *Cryptography and Network Security: Principles and Practice*. Prentice Hall PTR, 2011, ISBN 978-0-136-09704-4.

[31] Virius, M.: *Jazyky C a C++*. Grada, 2006, ISBN 80-247-1494-9.

Příloha A

Obsah přiloženého CD

```
CD
|
+- Dokumentace
|        +- zdrojové soubory této práce
+- Příručka
|        +- Zdrojové soubory
|        |      +-zdrojové soubory příručky
|        +- prirucka.pdf - instalační příručka aplikace
+- Útoky
|        +- zdrojové soubory útoků
+- Wifi packet generator
|        +- realizovaná aplikace
+- xsvand00.pdf - tato technická zpráva
```