VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

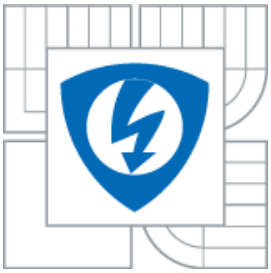
KOMUNIKACE MEZI ZAŘÍZENÍMI POMOCÍ TECHNOLOGIE BLUETOOTH

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE ONDŘEJ GASIOR
AUTHOR

BRNO 2015

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

KOMUNIKACE MEZI ZAŘÍZENÍMI POMOCÍ TECHNOLOGIE BLUETOOTH

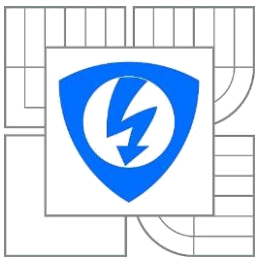
COMMUNICATION BETWEEN DEVICES USING BLUETOOTH TECHNOLOGY

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — ONDŘEJ GASIOR
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. IVO LATTENBERG, Ph.D.
SUPERVISOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student: Ondřej Gasior ***ID:*** 154252
Ročník: 3 ***Akademický rok:*** 2014/2015

NÁZEV TÉMATU:

Komunikace mezi zařízeními pomocí technologie Bluetooth

POKYNY PRO VYPRACOVÁNÍ:

Navrhněte aplikaci pro zařízení s operačním systémem Android, která bude umožňovat uživatelům rychlou výměnu textových zpráv (instant messaging). Vytvořte také verzi aplikace pro operační systém MS Windows. Naprogramujte ji v jazyce C# s využitím platformy .NET. Navrhněte vlastní protokol vhodný pro výměnu textových zpráv. Aplikace budou komunikovat prostřednictvím technologie Bluetooth. Komunikovat mohou mezi sebou adresně dvě zařízení, případně je možno vytvořit skupinovou komunikaci, kdy zprávy pak budou dostávat všechna zařízení ve skupině.

DOPORUČENÁ LITERATURA:

[1] MURPHY, M. L. Android 2 Průvodce programováním mobilních aplikací, Nakladatelství Computer Press, a.s. 2011, 376 stran, ISBN 978-80-251-3194-7
[2] WATSON, B., C# 4.0 - řešení praktických programátorských úloh, Zoner Press, 2010, 656 s., ISBN 978-80-7413-094-6
[3] VIRIUS, M., C# 2010 Hotová řešení, Computer Press, 2012, 424 s., ISBN 978-80-251-3730-7

Termín zadání: 9.2.2015 ***Termín odevzdání:*** 2.6.2015

Vedoucí práce: doc. Ing. Ivo Lattenberg, Ph.D.
Konzultanti bakalářské práce:

doc. Ing. Jiří Mišurec, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Tato práce se zaměřuje na komunikaci mezi mobilními zařízeními s operačním systémem Android a počítači s operačním systémem MS Windows pomocí technologie Bluetooth. Hlavní náplní je vytvoření aplikací, pomocí níž je vytvořena síť, reprezentující virtuální místnost, sloužící k výměně textových zpráv (tzv. instant messaging). To zahrnuje vytvoření protokolu nezávislého na použité platformě a následně jeho konkrétní implementaci. První část textu se detailně zabývá technologií Bluetooth a jejími principy. Dále jsou pak představeny obě podporované platformy, včetně programovacích jazyků, které využívají. Závěr je pak věnován popisu navrženého protokolu, obou aplikací za pomoci zdrojových kódů a také ukázce formátu zasílaných zpráv.

KLÍČOVÁ SLOVA

Bluetooth, chat, IM, instant messaging, Android, MS Windows, C#, Java, aplikace, .NET, programování

ABSTRACT

This thesis focuses on communication between mobile devices running the Android operating system and the PCs running MS Windows via Bluetooth. The main task is to create an applications which creates network, representing virtual room, for the exchange of text messages (ie. instant messaging). This involves creating a platform independent protocol and consequently its actual implementation. At first, Bluetooth technology and its principles are discused in detail, then both supported platforms are introduced, including their programming languages. The conclusion focuses on designed protocol and the analysis of both application with their source code using including sample of format of sending messages.

KEYWORDS

Bluetooth, chat, IM, instant messaging, Android, MS Windows, C#, Java, application, .NET, programming

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma Komunikace mezi zařízeními pomocí technologie Bluetooth jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu mé bakalářské práce doc. Ing. Ivu Lattenbergovi, PhD. za pomoc, ochotu a cenné rady při jejím zpracování.

V Brně dne ....

(podpis autora)

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

V Brně dne ....

(podpis autora)

GASIOR, O. *Komunikace mezi zařízeními pomocí technologie Bluetooth.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2015. 67 s. Vedoucí bakalářské práce doc. Ing. Ivo Lattenberg, Ph.D..

OBSAH

SEZNAM OBRÁZKU

ÚVOD

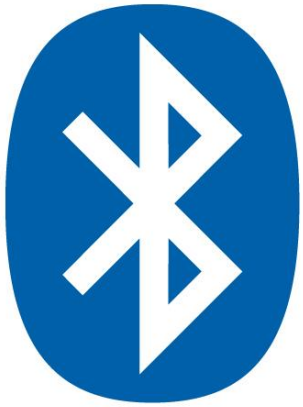
Technologie Bluetooth existuje již téměř dvě desetiletí. Za tuto dobu se díky malé energetické náročnosti a velkým možnostem využití stala již běžnou součástí počítačů a mobilních zařízení. Jejich propojením lze získat zajímavá řešení.

Cílem této práce vytvořit dvě verze aplikace, která pomocí univerzálního protokolu bude po vytvoření Bluetooth spojení a virtuální místnosti schopna výměny textových zpráv a to adresně mezi dvěma zařízeními a případně v hromadné komunikaci pro celou místnost. Jedna verze aplikace, vytvořena v jazyce Java, bude určena pro zařízení s operačním systémem Android a druhá, vytvořena pomocí jazyku C# s využitím platformy .NET, pro operační systém MS Windows.

Následující text pojednává o klíčových tématech, kterých se tato práce týká. Kapitola 1 detailně popisuje fungování technologie Bluetooth. Kapitola 2 pojednává o operačním systému MS Windows a programovacím jazyku C# včetně .NET Frameworku. Kapitola 3 se věnuje operačnímu systému Android a také programovacímu jazyku Java, který tato platforma využívá. Kapitola 4 se zabývá popisem navrženého protokolu pro výměnu textových zpráv, který implementují obě aplikace. Kapitola 5 se věnuje praktické části práce. Nejprve jsou uvedeny nástroje a zařízení použité při vývoji. Následuje představení obou vytvořených aplikací za pomoci důležitých částí kódu. Na závěr je uvedená ukázka zasílaných zpráv při simulaci navrženého scénáře. Kompletní programy jsou přiloženy na CD jak ve spustitelné formě, tak ve formě zdrojových kódů.

1 BLUETOOTH

Bluetooth je v dnešní době zavedeným standardem pro bezdrátovou komunikaci. Lze na něj narazit doslova na každém kroku. Ročně se prodá přes 2,5 miliardy zařízení s touto technologií. Možnosti využití jsou obrovské, a proto se lze s technologií Bluetooth setkat například v mobilních telefonech, tabletech, počítačích, fotoaparátech, televizích nebo i v hodinkách [1].



Obr. 1.1: Oficiální logo technologie Bluetooth [2].

1.1 Vývoj technologie Bluetooth

První zmínky o této technologii se objevují roku 1998, kdy byla založena skupina Bluetooth SIG - Special Interest Group firmami Ericsson, Nokia, IBM, Intel a Toshiba. Již za nedlouho se přidali například 3Com, Motorola, Lucent či Microsoft. V roce 2014 měla skupina Bluetooth SIG již přes 24 000 členů. Původním záměrem této skupiny bylo vytvořit technologii, která by nahradila kabeláž, měla dostatečný dosah a to vše při minimálních nárocích na energii a výrobní náklady. To je důvodem, proč je technologie Bluetooth vhodná pro malá lehká zařízení napájená baterií, jako jsou například mobilní telefony [3].

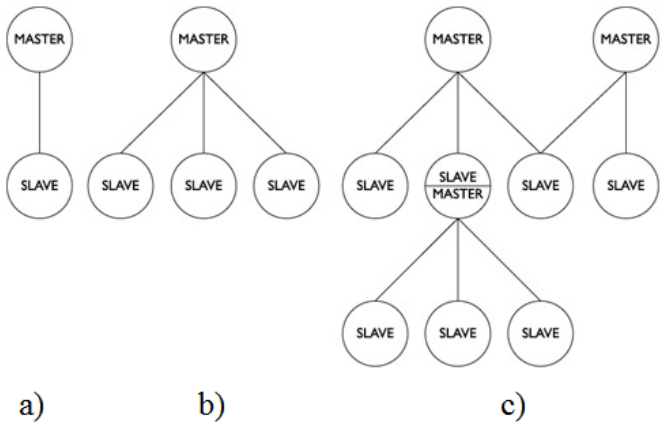
Již o rok později, v červenci 1999, se objevila první verze této technologie Bluetooth 1.0, zde se však vyskytovaly problémy se vzájemnou kompatibilitou či s jednoznačným přiřazením rolí master a slave. Z tohoto důvodu byla již v únoru 2001 vydána verze 1.1, která chyby opravovala a byla již základem pro použití v komerčních produktech. Velmi důležitým bodem v historii Bluetooth byl rok 2002, kdy se technologie chopila organizace IEEE (Institute of Electrical and Electronics Engineers), přesněji jedna z jejích pracovních skupin 802.15 zabývající se technologiemi WPAN (Wireless Personal Area Network), do které Bluetooth spadá. Technologie Bluetooth byla standardizována jako IEEE Std 802.15.1 a tato pracovní skupina se nadále aktivně podílela na vývoji. V listopadu roku 2003 se objevila od základů přepracovaná verze 1.2, která byla rozšířena o možnost rychlého vytvoření připojení a také o technologii Adaptive Frequency Hopping. Následuje verze 2.0 vydaná roku 2004 přinášející

přenosové rychlosti až 2,2 Mbit/s. Verze 2.1 z roku 2007 mimo jiné vylepšuje rychlost párovaní zařízení [3].

Velkým pokrokem byla verze Bluetooth 3.0 z roku 2009, která dosahuje přenosových rychlostí až 24 Mbit/s. Tohoto bylo dosaženo tím, že pomocí Bluetooth se naváže spojení a samotný přenos dat je uskutečněn pomocí technologie Wi-Fi. Zatím poslední specifikací je zpětně kompatibilní Bluetooth 4.0, která již nevylepšuje přenosové rychlosti, ale zaměřuje se jak na zabezpečení, tak hlavně na co nejnižší spotřebu [1][4].

1.2 Síťová architektura

Systém tvoří malé síťové struktury, označované názvem piconet. Každé mobilní nebo pevné elektronické zařízení, které je součástí této sítě, obsahuje malý terminál, v němž je umístěn rádiový vysílač a přijímač. V jedné piconetové struktuře může mezi sebou vzájemně komunikovat až 8 zařízení za použití tzv. topologie Ad-hoc, což znamená, že jednotlivá zařízení jsou rovnocenná, avšak zařízení, které první iniciuje sestavení sítě, se stává řídící jednotkou tzv. master a plní řídící funkci spočívající v identifikaci účastníků, zajištění jejich vzájemné synchronizace, atd. Ostatní zařízení se pak stávají podřízenými jednotkami tzv. slave. Tyto funkce jsou však dočasné a zanikají s ukončením spojení. Důležité ovšem je, že každá síť piconet může mít pouze jedno řídící zařízení. Pokud ve stejné oblasti pracují i další sítě piconet, vzniká rozptýlená Ad-hoc topologie (scatter Ad-hoc, scatternet), jejíž příklad je nakreslen na obrázku 1.2c. Libovolné zařízení pak může být součástí několika sítí piconet, ovšem pouze jednou jako master [5].

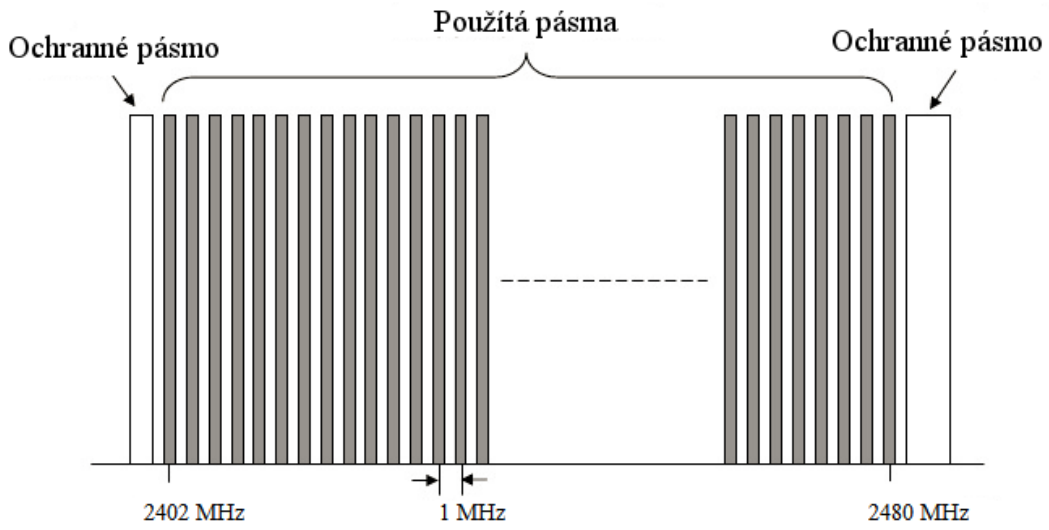


Obr. 1.2: Možné topologie Bluetooth sítě: a) Point-to-point b) Point-to-Multipoint c) Scatternet [6].

1.3 Přenos signálu

Účelem technologie Bluetooth je tedy bezdrátově propojit různorodá zařízení a to i bez přímé viditelnosti. Pracuje v nelicencovaném pásmu 2,4 GHz, toto pásmo se nazývá ISM (Industrial, Scientific, Medical) a je určeno pro průmyslové, vědecké a medicínské aplikace. Ve většině zemí světa se z tohoto pásma využívá 79 kanálů, pro jejichž střední frekvenci platí f_c = 2402 + k MHz, kde k nabývá hodnot od 0 do 78, ovšem například ve Francii je díky regulacím ISM pásma kanálů jen 23 a pro jejich střední frekvence platí f_c = 2402 + k MHz, kde k nabývá hodnot od 0 do 22. Z toho vyplývá, že šířka kanálu je 1 MHz [3].

Aby se předešlo rušení, používá se technika FHSS (Frequency Hopping Spread Spectrum), při které probíhají neustálé frekvenční skoky mezi výše zmíněnými kanály, v této implementaci konkrétně 1600 skoků za sekundu. Sekvence skoků je určená pseudonáhodně podle unikátní 48 bitové adresy (přidělené výrobcem) řídícího zařízení a jsou vynechány kanály již obsazené vysíláním jiného zařízení díky technice Adaptive Frequency Hoping. Vysoká energetická účinnost a malé nároky na energii nemohou být dosaženy pomocí klasické kvadraturní amplitudové modulace, proto se využívá modulace GFSK (Gaussian Frequency Shift Keying). Její princip spočívá v omezení šířky pásma potřebného pro přenos signálu filtrem typu dolní propust Gaussovského typu a následném převodu logické 1 na kladný posun frekvence a logické 0 na záporný posun. Kmitočtový posun se používá v rozmezí 140 až 175kHz [3].



Obr. 1.3: Ilustrace využitých kanálů [7].

1.4 Rozdělení do tříd

Jak lze vyčíst z tab. 1, Bluetooth zařízení lze dělit dle vysílacího výkonu do třech tříd. Třída 1 má maximální vysílací výkon 20 dBm (100 mW) a minimální 0 dBm (1 mW). Třída 2 má

maximální vysílací výkon 4 dBm (2,5 mW) a minimální výkon -6dBm (0,25 mW). Mezi těmito hodnotami se lze pohybovat a docílit úspory energie snížením vysílacího výkonu. Maximální výkon třídy 3 je 0 dBm (1 mW), minimum již ovšem není definováno. Maximální dosah pro třídu 1 je 100 metrů, pro třídu 2 až 10 metrů a pro třídu 3 pouze 1 metr. Je zřejmé, že dosah spojení je přímo úměrný vysílacímu výkonu. Tyto hodnoty jsou pouze orientační, protože velmi záleží na prostředí. Nejvyšších hodnot lze dosáhnout ve volném prostoru [3].

Tab. 1.1: Různé třídy zařízení Bluetooth [3].

Třída	Maximální vysílací výkon		Minimální vysílací výkon		Přibližný dosah [m]
	[mW]	[dBm]	[mW]	[dBm]	
1	100	20	1	0	100
2	2,5	4	0,25	-6	10
3	1	0	-	-	1

1.5 Protokolová struktura Bluetooth

Bluetooth je definováno vrstvovou protokolovou architekturou, kterou je možno vidět na obrázku 1.4. Mezi protokoly tvořící jádro systému patří:

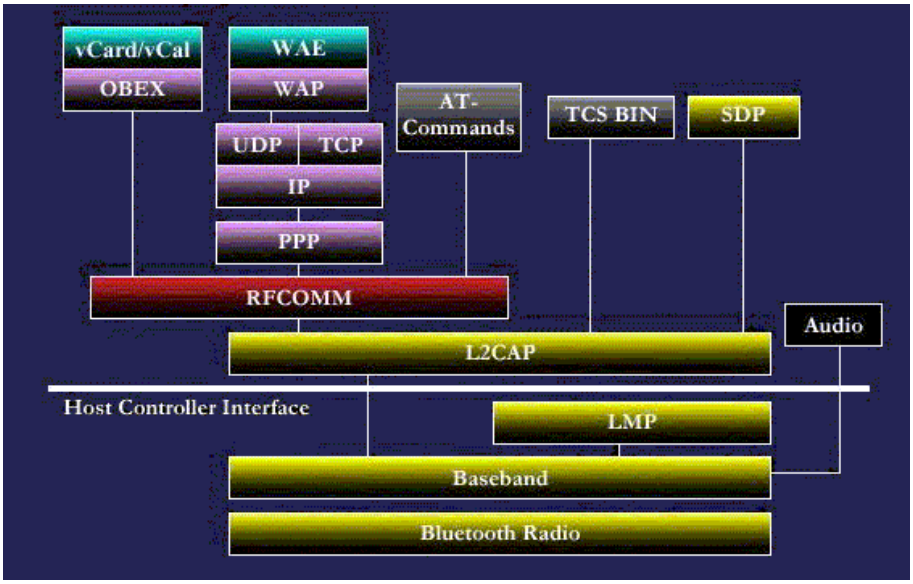
- **Radio**: Specifikuje vysílací rozhraní, frekvenci, včetně skoků a vysílací výkon.
- **Baseband**: Řeší sestavování nových spojení v piconet síti, adresaci, formát paketu, časování a správu energie.
- **LMP (Link Manager Protokol)**: Řídí veškerou komunikaci, tedy sestavení, kontrolu i ukončování spojení s dalšími zařízeními.
- **L2CAP (Logical Link Control and Adaptation Protocol)**: Propojuje protokoly vyšších vrstev s vrstvou baseband. Zajišťuje jak spojově, tak i nespojově orientovanou službu. Toto zahrnuje například také autentizaci a šifrování. Řídí rozdělování, nebo naopak sestavování delších paketů.
- **SDP (Service Discovery Protocol)**: Vyhledává informace o zařízeních, jaké služby poskytují. Informuje o charakteristikách těchto služeb. Definuje také to, jak vyhledávání probíhá [8].

Poté jsou zde protokoly, které mají za úkol nahradit kabelové propojení, a protokoly, které se starají o realizaci telefonních přenosů mezi zařízeními. Mezi tyto protokoly patří:

- **RFCOMM**: Tento protokol reprezentuje virtuální sériový port, který je používán vyššími vrstvami.
- **Telephony Control Protocol**: Jedná se o bitově orientovaný protokol, který definuje signalizaci pro sestavení hlasového hovoru mezi Bluetooth zařízeními [8].

Dále jsou definovány tzv. převzaté protokoly. Ty jsou převzaty z jiných systémů a včleněny do Bluetooth architektury. Filozofií Bluetooth je vymýšlet pouze nutné protokoly a využít již existující standardy kdykoliv je to možné. Převzaté protokoly jsou tedy následující:

- **PPP**: Jde o tzv. Point-to-Point Protocol. Tento Internetový protokol je standardem pro přenos IP datagramů přes linky typu bod-bod.
- **TCP/UDP/IP**: Jedná se o protokoly ze síťového modelu TCP/IP. Tyto protokoly Bluetooth používá k zpřístupnění služeb Internetu.
- **OBEX**: Neboli Object Exchange Protocol byl vyvinut skupinou Infrared Data Association (IrDA) pro výměnu datových objektů. Funkčně je tento protokol podobný HTTP, například také používá model klient-server, ovšem je ještě o něco jednodušší. Může sloužit například k výměně vizitek nebo k synchronizaci kalendáře.
- **WAE/WAP**: Celým názvem Wireless Application Protocol. Slouží k zpřístupnění internetu mobilním zařízením.
- **Audio**: Definuje služby pro přenos zvuku mezi Bluetooth zařízeními [8].

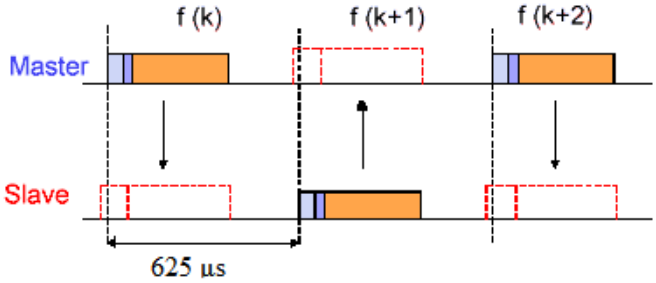


Obr. 1.4: Protokolová sada Bluetooth [9].

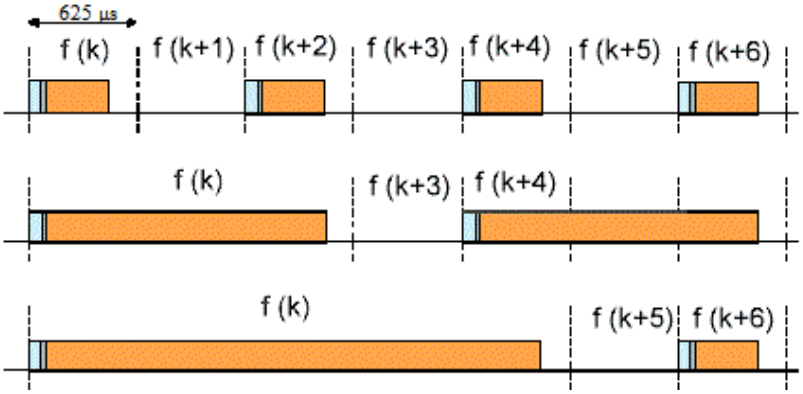
1.6 Přenos dat

Jak již bylo řečeno, při vysílání se používá technika FHSS s frekvencí skoků $f_{FH} = 1600\ s^{-1}$. Z toho vyplývá, že doba vysílání jedné nosné se rovná $T_{FH} = 1/f_{TH} = 625\ \mu s$. Tedy každý rádiový kanál je rozdělen na časové úseky (timesloty) délky 625 µs. Tzv. timesloty jsou pak číslovány podle hodinového signálu řídící jednotky. Rozsah číslování se pohybuje v rozmezí od 0 do 2^{27}-1, takže jeden cyklus má délku 2^{27} timeslotů. Pro komunikaci mezi terminály se používá tzv. duplex TDD (Time-Division Duplexing). Řídící jednotka vysílá v každém sudém timeslotu, zatímco

podřízená vysílá pouze v lichých, jak je naznačeno na obrázku 1.5. Přenos je uskutečněn pomocí paketů, které se vkládají do timeslotů, přičemž platí, že každý paket je vysílán na jiné nosné. Jeden paket bývá obvykle přenášen v jednom timeslotu, ovšem může být rozšířen tak, že je přenesen ve třech nebo pěti timeslotech, jak je znázorněno na obrázku 1.6. V tomto případě je vysílání celého paketu provedeno na téže nosné [5].

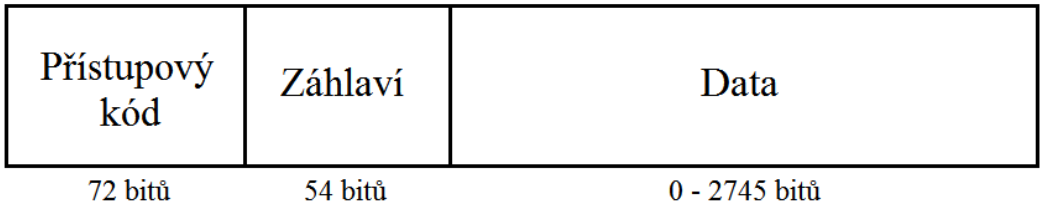


Obr. 1.5: Sudé a liché timesloty [10].



Obr. 1.6: Ilustrace různých velikostí paketů [11].

Používané pakety mají formát naznačený na obrázku 1.7. Na začátku každého paketu je přístupový kód tvořený 72 bity, za ním následuje 54 bitové záhlaví a nakonec jsou přenášena uživatelská data o velikosti 0 až 2745 bitů. Přístupový kód má pseudonáhodné vlastnosti a je unikátní pro každou síť piconet. Určuje jej řídící stanice a slouží k synchronizaci a autorizovanému přístupu do sítě. V záhlaví se přenáší 18 informačních bitů, které jsou zabezpečeny kanálovým kódováním FEC na výsledný počet 54 bitů. Obsahují adresu řízení přístupu na medium, informace o typu paketu, řídící bity a bity pro kontrolu chyb v záhlaví. Celková délka paketu se tedy může pohybovat v rozmezí 126 až 2871 bitů [5].



Obr. 1.7: Rozvržení Bluetooth paketu [5].

1.6.1 Typy linek

Mezi řídícím a podřízeným zařízením mohou být sestaveny dva různé typy linek. Oba typy je možné využít k zajištění přenosů dat podle požadavků jednotlivých jednotek. Pro zajištění komplikovanějších přenosů kombinujících oba typy linek je možné v průběhu spojení měnit jejich typ.

- **Synchronous Connection Oriented (SCO)**: Tento typ zajišťuje konstantní šířku pásma pro přenos, proto jsou SCO linky využívány hlavně pro výměnu dat citlivých na zpoždění. Doručení ovšem není potvrzováno a žádná data se neposílají znovu. Tento přístup je vhodný například pro posílání audio dat, která nejsou citlivá na menší ztráty. Konstantní šířka pásma je zajištěna řídícím zařízením, které rezervuje potřebné timesloty. Tyto linky existují pouze mezi řídícím zařízením a jedním podřízeným zařízením, jedná se tedy o linky typu bod-bod .
- **Asynchronous Connectionless (ACL)**: Tento typ linek je typu bod-více bodů mezi řídícím zařízením a všemi podřízenými zařízeními v síti piconet. V timeslotech nerezervovaných pro SCO linky si řídící zařízení může vyměňovat pakety s jakýmkoliv podřízeným zařízením v piconetu, včetně těch, které jsou již součástí SCO linky. V jednu chvíli může existovat pouze jedna ACL linka. Pro většinu typů posílaných paketů na této lince funguje opětovné odesílání při jejich ztrátě. Šířka pásma není konstantní a není nijak rezervována [8].

1.6.2 Oprava chyb při přenosu

Technologie Bluetooth využívá k opravě chyb na přenosové cestě těchto mechanismů:

- **FEC (Forward Error Correction):** Samoopravný kód s kódovým poměrem 1/3, který je používán na 18 bitové záhlaví paketu.
- **FEC:** Samoopravný kód s kódovým poměrem 2/3, který je využíván na datovou část paketu. Kodér používá Hammingův kód s parametry (15, 10), který je schopen opravit jednu chybu a detekovat 2 chyby v každém kódovém slově.
- **ARQ (Automatic Repeat Request):** Mechanismus, který je využívaný u linek ACL, a který se řídí následujícími pravidly:
    a) **Rozpoznání chyby**: Pokud příjemce rozpozná chybu, která není opravitelná, paket je zahozen.
    b) **Potvrzení bezchybného přijetí**: Pokud příjemce dostane paket v pořádku, odešle zpět potvrzení.
    c) **Přeposlání po vypršení času**: Pokud odesílatel odešle paket a do určitého časového intervalu nedostane potvrzení o bezchybném přijetí, paket je odeslán znovu.

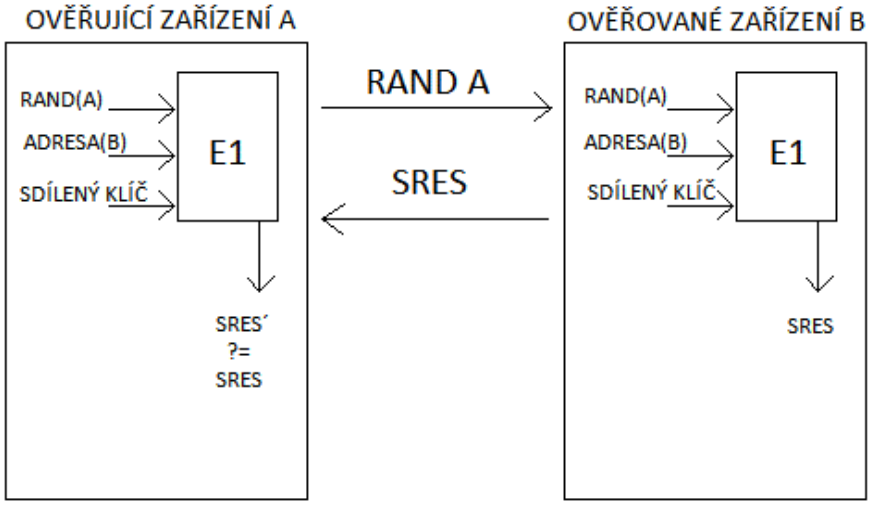
d) **Oznámení o přijetí chybného paketu a přeposlání**: Pokud příjemce detekuje chybu v přijatém paketu, oznámí to odesílateli a ten paket opět pošle [8].

1.6.3 Zabezpečení přenosu

Za účelem zabezpečení je před vlastním zahájením přenosu dat zajištěna autentizace protilehlého zařízení. Přenášená data pak mohou být také šifrována. Algoritmy k těmto činnostem využívají těchto čtyř položek:

- **Adresa zařízení**: Jedinečná 48 bitová adresa, která je veřejná.
- **Autentizační (sdílený) klíč**: Tajný 128 bitový klíč.
- **Šifrovací klíč**: Tajný klíč o délce 4 až 128 bitů.
- **Náhodné číslo**: Pseudonáhodné 128 bitové číslo generované samotným zařízením [3].

Účelem autentizace je ověření identity zařízení, která spolu komunikují. Tím se předchází neautorizovanému přístupu k datům. Proces autentizace je ilustrován na obrázku 1.8.



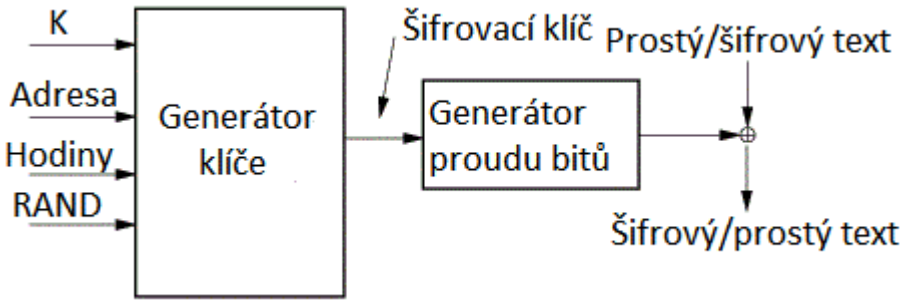
Obr. 1.8: Proces autentizace [12].

Proces autentizace obnáší ověření znalosti sdíleného klíče (tzv. link key). Ten je v několika fázích generován z tzv. inicalizačního klíče, který je odvozen z PINu zadaného uživatelem při zahájení procesu autentizace neboli tzv. párování. Spárovaná zařízení jsou tedy taková, která již disponují sdíleným klíčem, který může být použit při více spojeních. Párování je zrušeno jeho smazáním a při dalším spojení musí být vygenerován klíč nový [3].

V tomto případě zařízení A autentizuje zařízení B. Nejprve zařízení A vygeneruje náhodné číslo RAND A a zašle jej zařízení B. Obě zařízení použijí autentizační algoritmus E_1 k vygenerování 32 bitového podpisu SRES pomocí hodnoty RAND A, 48 bitové adresy zařízení B a tajného sdíleného klíče. Algoritmus E1 je založen na šifrovacím algoritmu SAFER. Poté co

zařízení B vygeneruje SRES, zašle jej zpět zařízení A a to porovná přijatou hodnotu s vlastní vypočítanou. Pokud se shodují, zařízení B je úspěšně ověřeno [8].

Posílaná data mohou být chráněna šifrováním datové části paketu, přístupový kód ani záhlaví šifrovány nejsou. K tomuto účelu je využíván šifrovací algoritmus známy jako E_0. Postup šifrování lze vidět na obrázku 1.9 [8].



Obr. 1.9: Proces šifrování [13].

Pokud je šifrování během komunikace povoleno, řídící zařízení pošle náhodné číslo RAND podřízenému zařízení a poté je pro každý vysílaný paket určen nový šifrovací klíč generovaný pomocí náhodného čísla RAND, adresy řídícího zařízení, aktuálního stavu hodin a sdíleného tajného klíče K. Tento šifrovací klíč je použit jako vstup pro algoritmus E_0, jehož úkolem je vytvářet proud bitů. Data jsou pak šifrována, či na straně příjemce opět dešifrována, pomocí operace XOR. Proměnnost klíčů je zajištěna měnícím se stavem hodin jakožto jednou z proměnných při výpočtu [8].

1.7 Profily

Tzv. profily zajišťují vzájemnou kompatibilitu zařízení s Bluetooth. Specifikují, jaká část protokolové sady bude použita. Zařízení, která spolu komunikují, musí podporovat stejný profil, jelikož tento profil určuje jaký druh dat a jakým způsobem se bude přenášet. Základ tvoří následujících třináct profilů.

1. **Cordless Telephony Profile (CTP)** - Profil určený pro bezdrátové telefony.
2. **Dial-up Networking Profile (DUN)** - Profil umožňující připojení zařízení, jako je notebook nebo tablet prostřednictvím mobilního telefonu k internetu.
3. **Fax Profile (FAX)** - Tento profil je určen pro dobře definované rozhraní mezi mobilním telefonem nebo pevnou linkou a PC s nainstalovaným faxovým softwarem.
4. **File Transfer Profile (FTP)** - Profil, který přes Bluetooth poskytuje přístup ke složkám a souborům v jiném zařízení.

5. **Generic Access Profile (GAP)** - Poskytuje základ pro všechny ostatní profily. Definuje, jakým způsobem se dvě zařízení s Bluetooth objeví a naváží spolu spojení.
6. **Generic Object Exchange Profile (GOEP)** - Slouží k vzájemné výměně datových objektů mezi zařízeními s Bluetooth. Je základem pro další profily jako FTP nebo SYNCH.
7. **Headset Profile (HSP)** - Nejpoužívanější běžný profil, který poskytuje podporu pro populární Bluetooth sluchátka s mikrofonem.
8. **Intercom Profile (ICP) -** Umožňuje, aby dva telefony fungovaly jako vysílačky a přímo mezi sebou přenášely hlas. Jedná se o doplněk k profilu CTP.
9. **LAN Access Profile (LAP)** - Umožňuje zařízení s Bluetooth přístup k sítím LAN, WAN nebo Internetu přes jiné zařízení, které má fyzické připojení k síti.
10. **Object Push Profile (OPP)** - Základní profil, který používá profil GOEP pro přenášení drobných "objektů", jakými jsou například obrázky nebo vizitky.
11. **Serial Port Profile (SPP)** - Definuje nastavení virtuálních sériových portů pomocí Bluetooth. Představuje tak bezdrátovou náhradu klasického sériového propojení s rychlostí do 128 kb/s.
12. **Service Discovery Application Profile (SDAP)** - Zjišťuje, jaké služby jsou k dispozici na zařízení s Bluetooth, ke kterému se chcete připojit.
13. **Synchronization Profile (SYNCH)** - Profil, který používá GOEP k synchronizaci kalendáře a kontaktů mezi dvěma Bluetooth zařízeními [14].

Tento počet profilů ovšem není konečný, vznikají nové, které rozšířily možnosti použití Bluetooth. Samotní výrobci mohou vytvářet další specifické profily pro jednotlivá řešení, než je přijat nový standardní profil. Mezi další využívané profily patří:

1. **Advanced Audio Distribution Profile (A2DP)** - Slouží k bezdrátovému přenosu hudby ve stereo kvalitě.
2. **Basic Imaging Profile (BIP)** - Jsou jím vybavené některé dražší digitální fotoaparáty a kamery. Umožňuje tisk fotografií nebo automatické zálohování.
3. **Basic Printing Profile (BPP)** - Profil pro vzdálenou komunikaci s tiskárnou.
4. **Hands-Free Profile (HFP)** - Profil používaný v Handsfree sadách. Je rozšířením Headset Profilu, proti kterému nabízí více možností při vzdáleném ovládání telefonu. Kromě příjmu a odmítnutí hovoru můžete například vytočit poslední volané číslo, aktivovat hlasového vytáčení, regulovat hlasitost a další.
5. **Health Device Profile (HDP)** - Profil, který používají lékařské přístroje pro odesílání a příjem dat.

6. **Human Interface Device Profile (HID)** - Profil sloužící k připojení počítačových periferií, jako jsou například klávesnice, myši nebo herní ovladače.
7. **Video Distribution Profile (VDP)** - Profil, který umožňuje přenos videa v reálném čase [14].

2 MS WINDOWS

2.1 Microsoft

Microsoft je softwarová akciová společnost sídlící v Redmondu ve státě Washington ve Spojených státech amerických založená roku 1975 Billem Gatesem a Paulem Allenem. Zaměstnává přes 127 000 lidí po celém světě. Mezi její hlavní produkty patří balík kancelářských programů Microsoft Office, herní konzole Xbox, či vyhledávač Bing, ovšem nejvýznamnějším produktem této firmy se stal operační systém Microsoft Windows, který je s více než 90 % tržního podílu nejpoužívanějším operačním systém v oblasti osobních počítačů [15].

2.2 Historie MS Windows

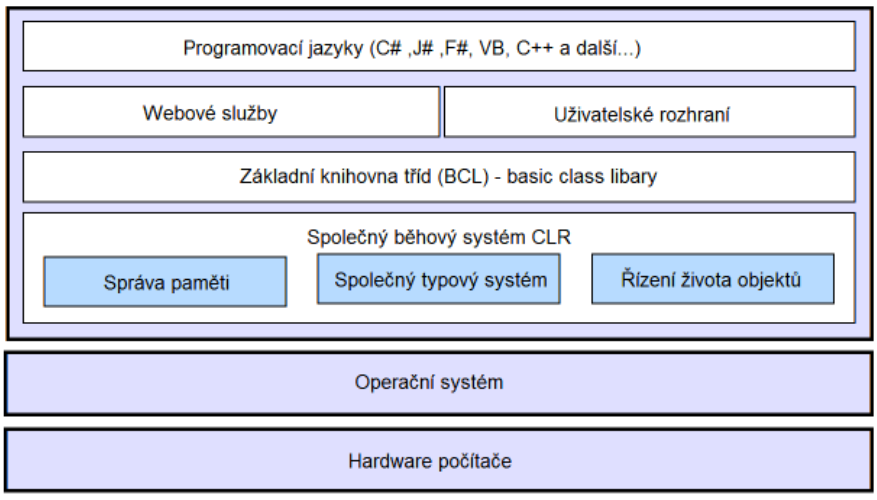
Předchůdcem MS Windows byl systém MS-DOS, který byl uveden na trh již v roce 1981, 6 let po založení firmy. Díky ručnímu psaní příkazů není MS-DOS uživatelsky přívětivý a proto roku 1985 Microsoft přichází s Windows 1.0. Ten přináší jednoduché uživatelské rozhraní ve formě rozevíracích nabídek, posuvníků a hlavně oken, ve kterých jsou zobrazeny programy a podle kterých systém dostal svůj název. Jedná se v podstatě o nástavbu na systém MS-DOS. Obsahoval již notoricky známé programy jako Malování či Poznámkový blok. Roku 1987 na něj navazuje Windows 2.0, který přináší novinky jako ikony na ploše, možnost překrývání oken, či velice užitečné klávesové zkratky. Mimo jiné díky tomu se z Microsoftu stává největší softwarová firma na světě. Již o 3 roky později je uveden operační systém Windows 3.0 následovaný verzí 3.1 a dosahuje rekordních prodejů. Systém se již instaluje z disket. Díky technickému pokroku a měl tento systém mnohem větší výkon s pokročilejší grafikou než jeho předchůdci. Vydání nové sady pro vývoj softwaru Windows (SDK) také zvyšuje jeho oblibu mezi vývojáři [16].

Windows NT se objevuje roku 1993. Oproti přechozím 16 bitovým verzím se již jedná o 32 bitový operační systém. Objevuje se zde preemptivní multitasking, který zabraňuje zamrzání systému vlivem nesprávně naprogramovaných aplikací. Tato verze není založena na MS-DOS, která používala tzv. nepreemptivní multitasking. Velká reklamní kampaň roku 1995 ohlásila příchod Windows 95. Kromě nového tlačítka Start přichází tato verze s podporou e-mailů, faxů a multimédií. Je zde integrovaná podpora internetu pomocí telefonického připojení a objevuje se první verze prohlížeče Internet Explorer. V dalších letech se objevují verze Windows 98 s podporou USB a čtením DVD, Windows Me zlepšující podporu multimédií a stabilitu. Tyto verze jsou poslední založené na systému MS-DOS, další verze již vychází z verze Windows NT. Windows 2000, již založen na Windows NT, dále zlepšuje stabilitu systému či lepší podporu technologie Plug and Play [16].

Patrně nejoblíbenějším systémem všech dob se však stal Windows XP, který podporuje již 25 světových jazyků. Vylepšen je po všech stránkách a objevuje se první 64 bitová verze operačního systému od Microsoftu Windows XP-64 Edition roku 2001. Pro zajímavost Windows XP narostl do velikosti 45 miliónů řádků kódu. S velkým důrazem na zabezpečení přichází Windows Vista roku 2006, přináší například nástroj na ochranu dat BitLocker. Windows Vista má bohužel vysoké nároky na parametry počítače a proto o 3 roky později přichází velmi oblíbený Windows 7 s nižšími nároky a lepší stabilitou. Přináší podporu dotykového ovládání. Malou revoluci roku 2012 přináší kontroverzní Windows 8. Představuje nové uživatelské rozhraní přizpůsobené k ovládání dotykem a obchod s aplikacemi Windows Store. Je představena i verze Windows RT speciálně pro přenosná zařízení jako jsou tablety. Aktuálním operačním systémem firmy Microsoft je nyní Windows 8.1, který opravuje nedostatky předchozí verze a přidává například lepší propojení s cloudem [16].

2.3 .NET Framework

Když hovoříme o platformě .NET Framework, mluvíme o jednom ze systémů infrastruktury .NET. .NET je název strategie firmy Microsoft pro software, který je nezávislý na operačním systému a hardwaru. Projekty .NET tedy nejsou určeny pouze pro stolní počítače, ale také například pro mobilní zařízení nebo servery. Pro tyto projekty je díky univerzálnosti platformy .NET možno použít dokonce několik jazyků, například C#, F#, Visual Basic, C++ nebo J#. Nástrojem pro vývoj aplikací na platformu .NET je Visual Studio. .NET Framework obsahuje jazykově nezávislé běhové prostředí (Common Language Runtime, CLR) a knihovnu tříd (Basic Class Library) [17].



Obr. 4.2: Hierarchie .NET frameworku[18].

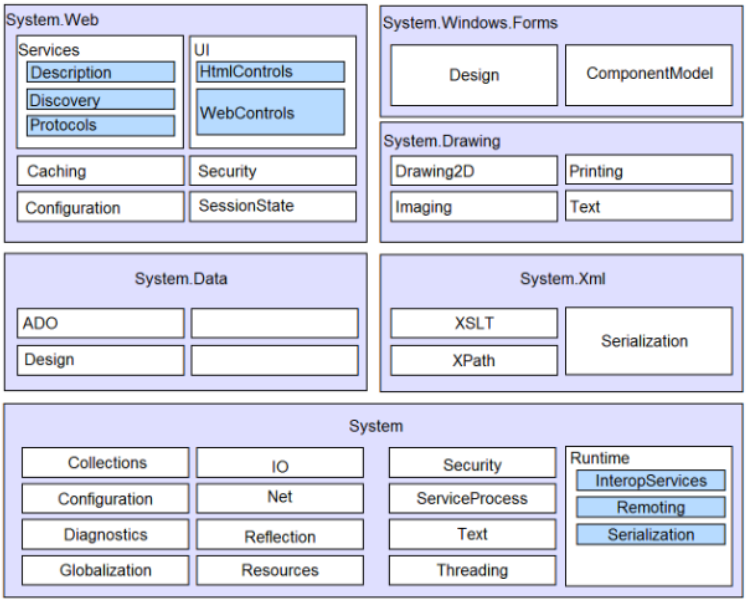
2.3.1 Common Language Runtime

Běhový systém CLR si lze představit jako virtuální stroj, ve kterém pracují aplikační funkce platformy .NET a kde mají všechny jazyky k dispozici knihovny tříd systému. CLR také pracuje

jako prostředník mezi překladačem a konečnými instrukcemi strojového jazyka, jelikož na ně převádí kód zprostředkovacího jazyka MSIL (Microsoft Intermediate Language) vytvořený překladačem. Mezi výhody CLR patří řízený kód, jednotný typový systém a správa paměti (Garbage Collector) [19].

2.3.2 Knihovny .NET Framework

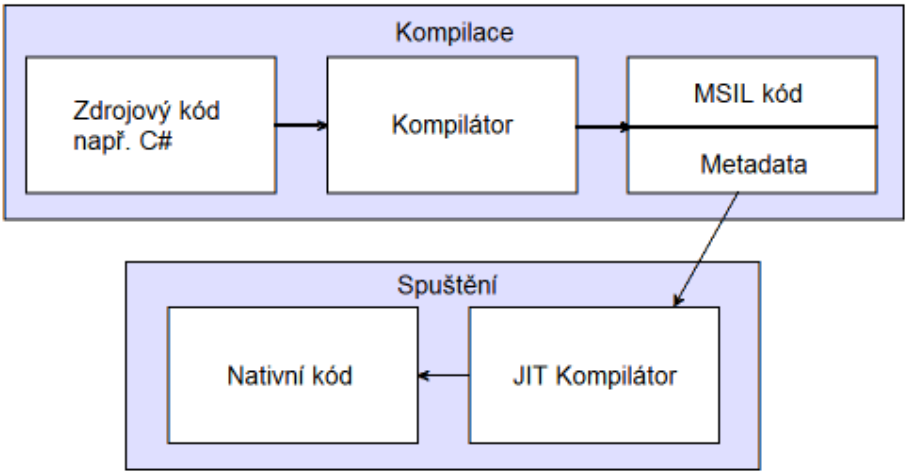
Pro všechny podporované jazyky existuje společná knihovna tříd, což v praxi znamená, že všechny programovací jazyky, které ji používají, mají teoreticky naprosto stejné možnosti. Výhodou je například podobná syntaxe programovacích jazyků, což má za následek jejich lehčí zvládnutí člověkem [19].



Obr. 4.3: Ukázka tříd knihovny BCL [20].

2.3.3 Microsoft Intermediate Language – MSIL

Pro snadnější včlenění jazyka do platformy .NET, vyvinul Microsoft jazyk MSIL, který je podobný assembleru. Při kompilaci programu pro systém .NET bere překladač jako vstup zdrojový kód příslušného jazyka a výstupem je právě aplikace v MSIL. CLR zajistí, že při prvním spuštění aplikace se program z MSIL pomocí JIT (Just In Time) kompilátoru přeloží do strojového kódu. Teoreticky tedy lze vyvíjet programy i v jazyce MSIL, ovšem je to nepraktické [19].



Obr. 4.4: Průběh spouštění programu v .NET frameworku [21].

2.4 Programovací jazyk C#

Jazyk C# je objektově orientovaný programovací jazyk, který vytvořila firma Microsoft zároveň s platformou .NET. Poprvé se objevil roku 2000 jako součást Visual Studia .NET. Je to vlastně zjednodušená, vylepšená a čistě objektová verze programovacího jazyka C++. C# se využívá hlavně k tvorbě databázových programů, webových aplikací, webových služeb, formulářových aplikací ve Windows a podobně. Aktuální verze tohoto jazyka je 5.0 vydaná v srpnu 2012 [22].

3 ANDROID

Pojem Android je označení pro open source počítačovou platformu určenou pro zařízení, jako jsou chytré telefony, navigace, tablety či televize. Jeho součástí je operační systém, který je založen na jádře Linux, uživatelské prostředí a také aplikace. Největší silou této platformy je univerzálnost, protože jádro operačního systému Android bylo navrženo pro běh na různém hardwaru. Systém může být použit bez ohledu na použitý chipset či velikost obrazovky. V současné době se jedná o nejpoužívanější mobilní platformu. Aplikace pro lze zakoupit v obchodě Google Play, kde jich je většina dostupná zdarma. Tyto aplikace jsou vyvíjeny v programovacím jazyce Java [23].

3.1 Google

Google je softwarová společnost se sídlem v kalifornském Silicon Valley ve Spojených státech amerických založena roku 1998 Larry Pagem a Sergejem Brinen, která původně začala jako internetový vyhledávač. Její hlavní příjmy pochází z reklamy (služba AdWords), ovšem věnuje se mnoha projektům jako například sociální síti Google+, videoportálu YouTube nebo mapám Google Maps a díky četným akvizicím i mnoha dalším. Od roku 2007 se ve spolupráci s asociací Open Handset Alliance podílí na vývoji platformy Android [24].

3.2 Vývoj platformy Android

Společnost Android Inc. byla založena Andy Rubinem a jeho společníky roku 2003 a později byla převzata společností Google, která platformu Android dále vyvíjela. V říjnu roku 2008 bylo vydáno první zařízení s Androidem T-Mobile G1. Jednalo se o mobilní telefon s dotykovou obrazovkou. Již v první verzi zde byly známé ikony, integrované služby Googlu, jako například Gmail a hlavně obchod s aplikacemi. Verze 1.1 v únoru roku 2009 přinesla hlavně opravu mnoha chyb. O dva měsíce později přichází verze 1.5, která vylepšuje podporu Bluetooth, také přináší přepracované uživatelské prostředí v čele s novou softwarovou klávesnicí na obrazovku. Verze 1.6 v září 2009 přináší podporu rozličných rozlišení displeje a o měsíc později verze 2.0/2.1 přináší podporu Bluetooth 2.1, HTML 5, více uživatelských účtů a navigaci pomocí Google Maps. V květnu 2010 přináší verze 2.2 lepší podporu technologie Flash, nebo také USB Tethering a funkci Wi-Fi Hotspot. Android 2.3 ještě téhož roku přináší podporu pro velká rozlišení obrazovky a efektivnější zacházení s energií. Přelomovou se stává verze 3.0 květnu roku 2011, která je určená pro tablety. Od této verze se už v zařízeních s Androidem nepočítá s hardwarovými ovládacími prvky, jsou nahrazeny ovládáním gesty či jejich ekvivalenty na dotykové obrazovce. Téhož roku byla vydána verze 4.0, která sjednocuje předchozí verze. Další verze 4.1-4.4 přinášejí stovky vylepšení, co se týče stability systému, výkonu, nebo uživatelského rozhraní. Verze 4.4W z roku 2014 přináší podporu pro funkční doplňky jako například chytré hodinky a náramky. Nejnovější představenou verzí je 5.0, která

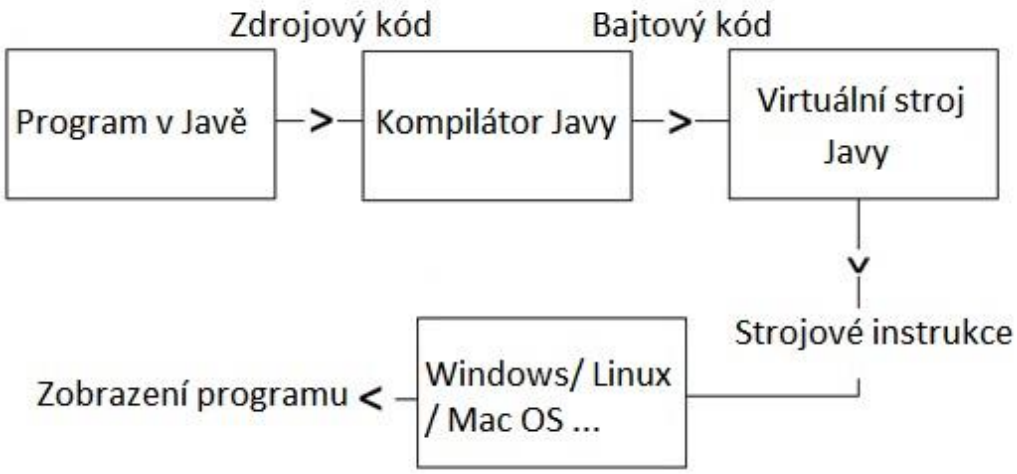
již podporuje 64 bitové procesory, vektorový design, nebo širší využití USB [25][26].

3.3 Java

Java jedním z nejrozšířenějších programovacích jazyků současnosti. Jedná se o moderní objektově orientovaný jazyk, jenž nachází uplatnění hlavně při vývoji webových aplikací, nebo také právě v aplikacích pro Android. Co se týče syntaxe, vychází tento jazyk ze staršího C++. V roce 1995 vydala společnost Sun Microsystems verzi Java 1.0, která byla v té době revoluční. Vývoj nadále pokračoval a dnes je Java ve své 8. verzi a jedná se již o open source projekt. Pro vývoj aplikací lze použít například vývojové prostředí Eclipse [27].

3.3.1 Přenositelnost

Hlavní motivací pro vytvoření Javy byla potřeba jazyka, který by byl nezávislý na použité platformě a který by se dal použít pro tvorbu softwaru vsazovaného do nejrůznějších zařízení spotřební elektroniky. Problém byl v tom, že většina počítačových jazyků byla v té době navržena tak, aby se kompilovala pro určitý cíl. S příchodem internetu byla potřeba přenositelnosti programů mezi platformami ještě větší [27].



Obr. 3.2: Spouštění programu v Javě [28].

Tohoto tvůrci Javy dosáhli tím, že výstupem z kompilátoru Javy není spustitelný kód, ale tzv. bajtový kód. Bajtový kód je vysoce optimalizovaná sada instrukcí navržených k provádění běhovým systémem Javy, který se nazývá virtuální stroj Javy (Java Virtual Machine). Program v Javě přeložený do bajtového kódu tak lze mnohem snadněji spouštět v celé škále prostředí, protože pro každou platformu stačí implementovat pouze virtuální stroj Javy. Pokud by se program v Javě zkompiloval do nativního kódu, musely by pro každý typ výpočetní jednotky připojené k Internetu existovat různé verze téhož programu. To ale samozřejmě není proveditelné řešení. Provádění bajtového kódu virtuálním strojem Javy je tedy nejsnazším způsobem pro vytváření skutečně přenositelných programů [27].

4 NAVRŽENÝ PROTOKOL

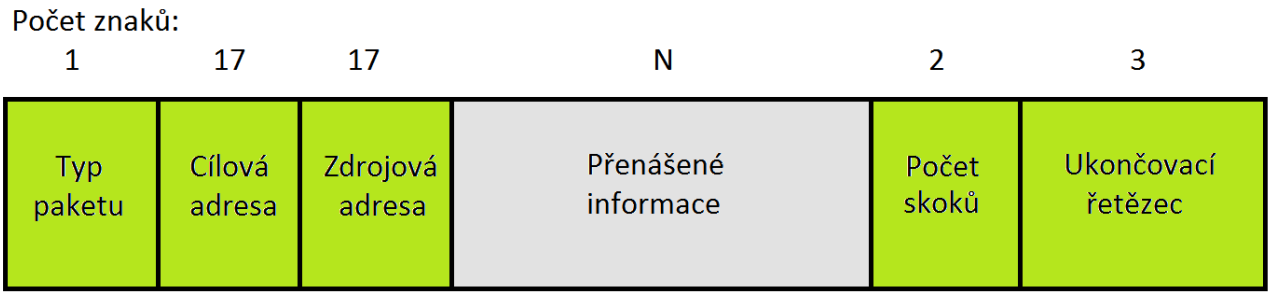
Obecně lze protokol definovat jako soubor pravidel a postupů při vzájemné komunikaci mezi zařízeními. Navržený protokol bere v potaz možnosti technologie Bluetooth a obou platforem. Je zde snaha o univerzálnost a rozšiřitelnost. Veškerá uživatelská komunikace je potvrzovaná. Protokol definuje dva základní pilíře komunikace a to vytvoření sítě, tedy virtuální místnosti, za účelem komunikace, a také vlastní předávání zpráv. Protokol k vzájemné komunikaci využívá datové jednotky nazvané pakety. Podle dohodnuté syntaxe jsou pomocí paketů přenášeny informace z jednoho zařízení na druhé.

4.1 Datové jednotky

Při tvorbě protokolu bylo vytvořeno šest typů paketů a to MSG – paket nesoucí zprávu mající příznak „a“, ACK – paket nesoucí potvrzení s příznakem „b“, UPD – paket nesoucí informace o síti s příznakem „c“, DUPD – paket informující o rušení spojení s příznakem „d“, EMSG – paket představující hromadnou zprávu s příznakem „e“ a EACK – paket s potvrzením hromadné zprávy a příznakem „f“. Pakety jsou textově orientované, jedná se tedy o řetězec znaků. Dále je definováno, že při přijetí špatného paketu je paket zahozen a příjemce paketu odesílatele odpojí.

4.1.1 Společná část

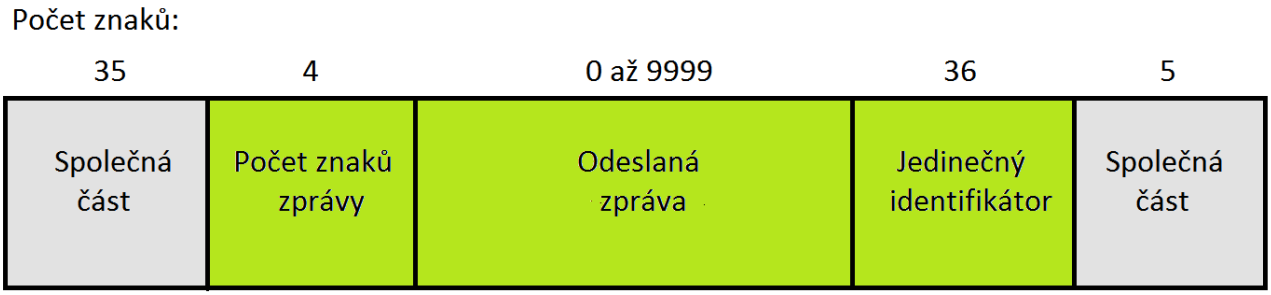
Všechny pakety mají shodné položky, které zapouzdřují volitelná data. Na obrázku 4.1 lze vidět uspořádání tohoto společného formátu. V prvním poli se nachází jeden znak označující, o který z paketů se jedná. Například pro paket ACK zde bude znak „a“. Následují cílová a zdrojová adresa ve formátu 01:23:45:67:89:AB, tedy celkem 17 znaků na jednu adresu. Jedná se o fyzické adresy příslušných Bluetooth adaptérů, které by měly být jedinečné. Protokol nedefinuje nic jako broadcast, všechny pakety jsou adresovány na konkrétní zařízení. Jelikož se jedná v podstatě o přepínanou síť, docházelo by k broadcastovým bouřím. Poté následuje N znaků, kde N je počet znaků volitelných dat v paketu. Následuje opatření proti nekonečnému přeposílání paketu ve formě dvouznakového pole „Počet skoků“, které nese hodnotu určující kolikkrát bude paket přeposlán, než bude zahozen. Při vytvoření paketu je tato hodnota nastavena na 64 a poté při průchodu každým zařízením snižována o 1. Jako ochrana před paketem špatného formátu je na konec každého umístěn řetězec „end“. Pokud se při dekódování paketu na příslušné pozici nenachází přesně tento řetězec, je označen za špatný a je zahozen.



Obr. 4.1: Společná část paketů.

4.1.2 Paket typu MSG – adresovaná zpráva

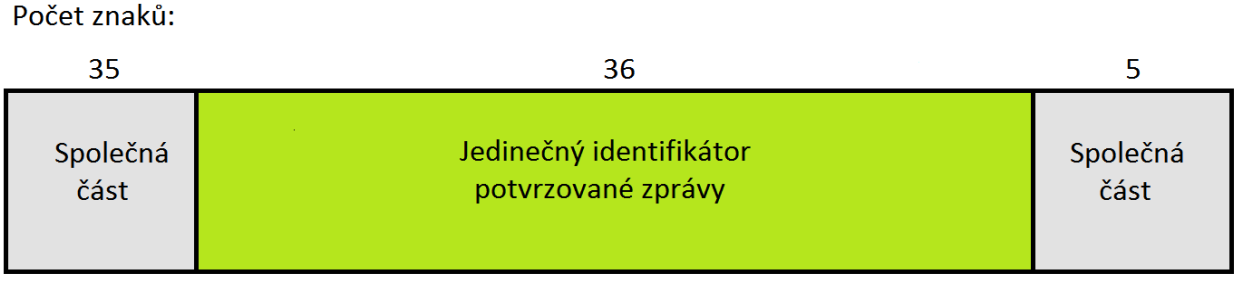
Tento typ paketů v sobě skrývá informaci o odeslané zprávě. Formát, ve kterém je tato informace přenášena, lze vidět na obrázku 4.2. Sestává se z části reprezentované čtyřmi znaky, která udává, kolik znaků obsahuje zaslaná zpráva. Za ní následuje vlastní zpráva o maximální délce 9999 znaků. Jelikož jsou zprávy potvrzované, je zde obsažen také jedinečný identifikátor zprávy o délce 36 znaků ve formátu 01234567-89AB-CDEF-GHIJ-KLMNOPQRSTUV.



Obr. 4.2: Uspořádání paketu typu MSG.

4.1.3 Paket typu ACK – potvrzení adresované zprávy

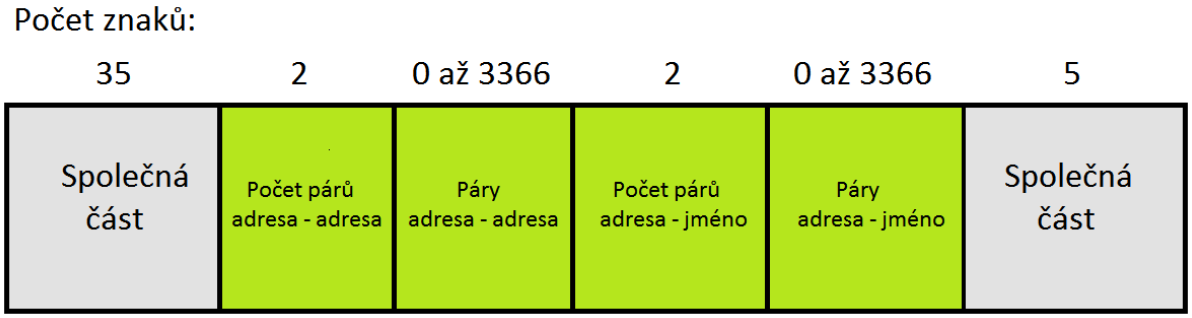
K potvrzování doručených zpráv slouží paket typu ACK. Po přijetí zprávy je ihned odesláno potvrzení na zdrojovou adresu. Tímto je odesilatel informován o tom, že zpráva byla úspěšně doručena. Jak lze vidět na obrázku 4.3 tento paket nese pouze jedinečný identifikátor potvrzované zprávy.



Obr. 4.3: Uspořádání paketu typu ACK

4.1.4 Paket typu UPD – informace o síti

Počet informací nesených paketem typu UPD je již poněkud obsáhlejší. Tento paket slouží k předávání informací o existenci jednotlivých účastníků a také o topologii, kterou vytvářejí. Je zasílán všem zařízením v síti po vytvoření jakéhokoliv spojení. Jeho rozložení je patrné z obrázku 4.4. Nejprve jsou zde 2 znaky sloužící ke stanovení počtu dvojic adresa – adresa. Tyto dvojce reprezentují všechna spojení mezi zařízeními a na základě těchto informací si následně protokol vytváří topologii sítě. Pokud je vytvořeno spojení mezi 2 zařízeními A a B, potom jsou v paketu obsaženy dvě dvojice adresa a to A – B a také B – A. Velikost jednoho tohoto bloku je tedy násobkem 34, tedy velikosti dvou za sebou jdoucích adres ve formátu 01:23:45:67:89:AB01:23:45:67:89:AB. Jelikož pro člověka není běžné pracovat s adresami v tomto formátu, a protože každé Bluetooth zařízení disponuje také jménem, následuje blok, který umožní zařízením přiřadit k adrese také jméno. Uživatel tedy již nepracuje s adresami, nýbrž pouze se jmény. Jelikož výsledná síť není nijak velká, předpokládá se, že jsou jména stejně jako adresy jedinečná. Podobně jako v předchozím případě jsou zde 2 znaky určující počet párů adresa – jméno následovaná příslušným počtem těchto dvojic o délce 34 znaků ve tvaru 01:23:45:67:89:AB0123456789ABCDEFG. Z toho vyplývá, že jména jsou reprezentovaná 17 znakovým řetězcem. Pokud je jméno delší, je ořezáno, pokud je kratší je pro účely přenosu doplněno prázdnými znaky.

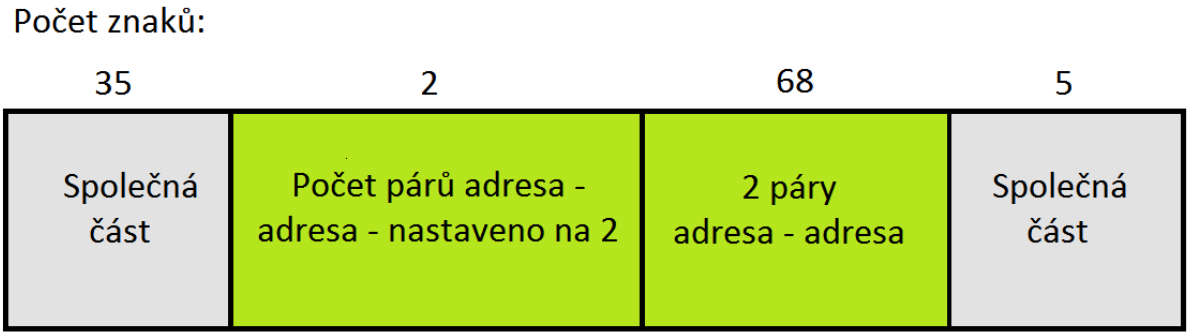


Obr. 4.4: Uspořádání paketu typu UPD.

4.1.5 Paket typu DUPD – informace o zaniklých spojeních

Po sestavení spojení následuje také situace, kdy se některý z uživatelů rozhodne odpojit. Za předpokladu, že se tímto celá síť nerozpadne, ve smyslu nemožnosti předávání informací v důsledku neexistujících spojení, je nutné na tuto změnu reagovat. Zařízení, které v síti zůstává, ale jiné, k němu přímo připojené zařízení bylo odpojeno, zasílá paket typu DUPD na všechna zbývající zařízení v síti. Tento paket, jak lze vidět na obrázku 4.5, obsahuje dvouznakové pole, které je defaultně nastaveno na hodnotu dvě, protože se předpokládá, že je zpráva zasílána ihned po odpojení a tedy je zde informace pouze o jednom spojení. Následují

2 páry adresa - adresa, reprezentující zaniklé spojení, tedy 68 znaků. Na základě toho je pomocí výpočtu každé zařízení schopno sestavit topologii sítě a případně pokud neexistuje cesta k zařízení, smazat i dvojici adresa - jméno.



Obr. 4.5: Uspořádání paketu typu DUPD.

4.1.6 Paket typu EMSG – hromadná zpráva

Jelikož součástí zadání je také možnost zaslat zprávu všem, byl za tímto účelem definován paket typu EMSG. Formát tohoto paketu je shodný s formátem paketu typu MSG, ovšem pokud chce zařízení odeslat zprávu všem, nastaví typ paketu na „e“. Toto umožní, aby příjemce identifikoval, že se jedná o hromadnou zprávu a také díky zdrojové adrese je v hromadné konverzaci možno rozlišit původce zprávy.

4.1.7 Paket typu EACK – potvrzení hromadné zprávy

Pokud zařízení odešle hromadnou právu v paketu typu EMSG je důležité, aby bylo informováno, kdo tuto zprávu obdržel. Za tímto účelem je jako potvrzení hromadné zprávy zasílán zpět paket typu EACK shodného formátu jako paket typu ACK, nesoucí jedinečný identifikátor hromadné zprávy.

4.2 Postup pro nalezení cesty v síti

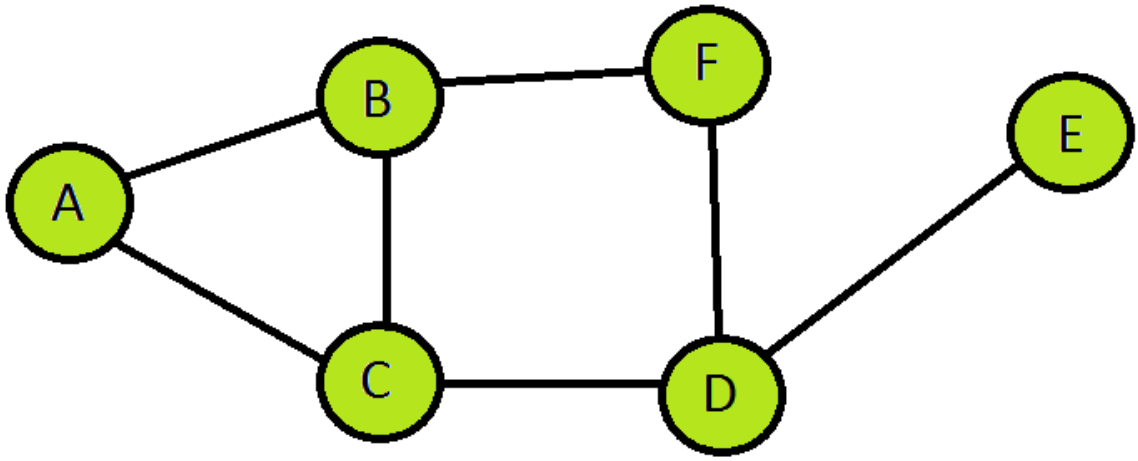
Při využití protokolu pro vytvoření spojení mezi více zařízeními tedy není vždy jisté, že adresát je vždy přímo připojen a tedy, že je vždy jasné, kudy paket poslat. Teoretické maximum účastníků v jedné místnosti je určeno množstvím dvojic adres, které lze přenést v paketu typu UPD. Pokud počítáme se dvěma páry adres na jedno spojení a topologií, kdy jsou všechna zařízení propojena v jedné řadě, je maximální počet zařízení v síti 50. V konkrétní implementaci toto může být programově omezeno.

Aby zařízení vždy vědělo komu předat paket, který není určen přímo jemu, je nutné z dostupných dat zjistit vhodnou cestu. Vstupními daty jsou informace předávané v paketu typu UPD. Tento paket obsahuje informaci o, tom jaká zařízení se v síti vyskytují a na základě toho si z dostupných dat vytvoří její topologii. Pro každé zařízení je vytvořena položka ve slovníku ve tvaru <adresa zařízení> - <seznam adres přímo připojených zařízení>. Pokud je

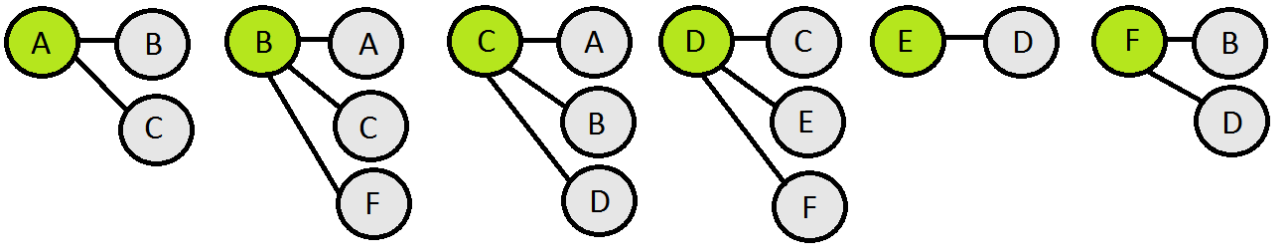
přijat paket typu DUPD, je příslušná informace o spojení odstraněna. Po přijetí jednoho z těchto dvou typů paketů vždy následuje přepočítání všech cest. K tomu slouží algoritmus, jehož vstupními informacemi jsou vytvořená topologie a seznam zařízení v síti. Výstupem je pak opět slovník ve tvaru <adresa> - <adresa dalšího zařízení v cestě k cíli>, neboli směrovací tabulka.

4.2.1 Ukázka algoritmu

Mějme obecnou topologii o 6 zařízeních, kterou můžeme vidět na obrázku 4.6. Tato zařízení si označíme písmeny A – F. Právě se do sítě připojilo zařízení F a zařízení A o tom dostalo zprávu ve formě paketu typu UPD. Z této zprávy zjistí, že se v síti nalézá celkem 6 zařízení a na základě informací o všech spojeních vytvoří slovník, viz obrázek 4.7.



Obr. 4.6: Obecná topologie pro ilustraci postupu pro výpočet cesty.



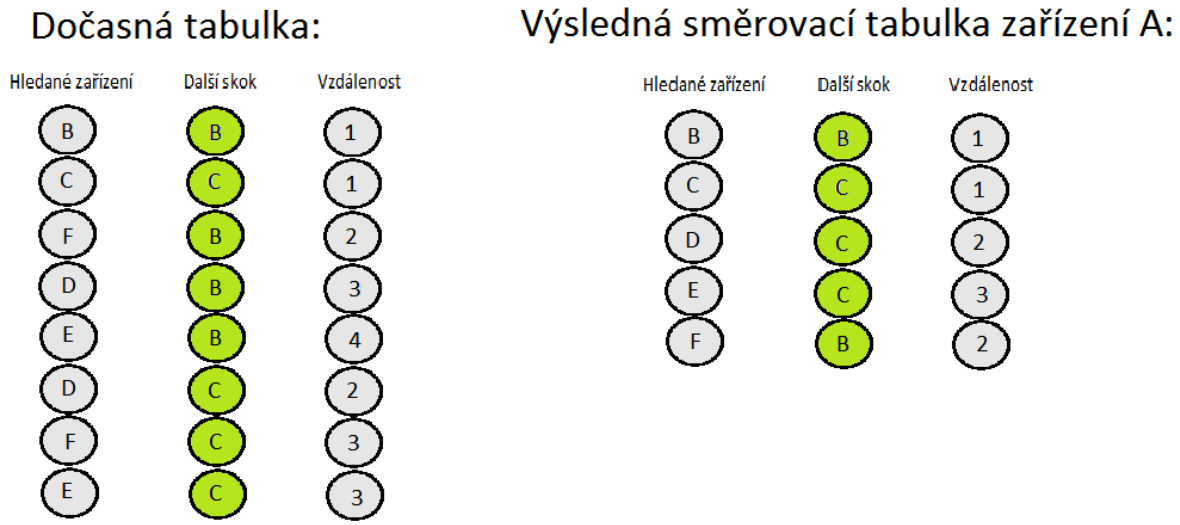
Obr. 4.7: Vytvořený slovník.

Úkolem algoritmu je tedy najít nejkratší cestu ke všem ostatním zařízením (v tomto případě k B, C, D, E a F) a uložit další skok do směrovací tabulky. Toto se děje ve 2 krocích:

1. Přidání přímo připojených do dočasné tabulky se vzdáleností 1 a vyloučení těchto zařízení z hledání.

2. Pro každého souseda je započato vyhledávání. Počínaje přímo připojenými zařízeními si algoritmus vytváří strom, ve kterém hledá. Základní premisou je, že přímo připojená zařízení, přes které by již mohla vést jen delší cesta, se neberou při hledání v potaz. Dále platí, že v každém uzlu hledání pokračuje do všech k němu připojených zařízení, ovšem ne již zpět ve stromu. Pokud je nalezeno zařízení, které pro daný strom ještě není přidáno v dočasné směrovací tabulce, je tam přidáno s příslušnou vzdáleností od hledajícího zařízení a jako další skok je nastaven soused, u kterého vyhledávání začalo.

Tedy nejprve jsou z hledání, po přidání do dočasné tabulky, odstraněny zařízení B a C. Poté se algoritmus podívá, s kým sousedí zařízení B. Zařízení C je přímo připojeno k A, do dočasné směrovací tabulky se přidá tedy pouze F, se vzdáleností 2. K zařízení F je dále připojeno B, to se nebere v potaz, jednak jelikož je přímo připojeno, a také protože se nevyhledává zpět ve stromu. Zařízení D je přidáno se vzdáleností 3 a následně zařízení E se vzdáleností 4. Obdobným způsobem je provedeno vyhledávání také pro strom s počátkem u zařízení C. Výsledkem tohoto procesu je tzv. dočasná směrovací tabulka. Z té je vždy ke každému zařízení vybrána nejkratší cesta reprezentovaná dalším skokem, viz obrázek 4.8.



Obr. 4.8: Dočasná a výsledná směrovací tabulka pro zařízení A

Jelikož všechna zařízení mají stejné informace o topologii sítě a využívají k výpočtu cesty stejný algoritmus, neměly by zde vznikat smyčky. Pokud ano, je paket zahozen, když je jeho počet skoků nastaven na 0.

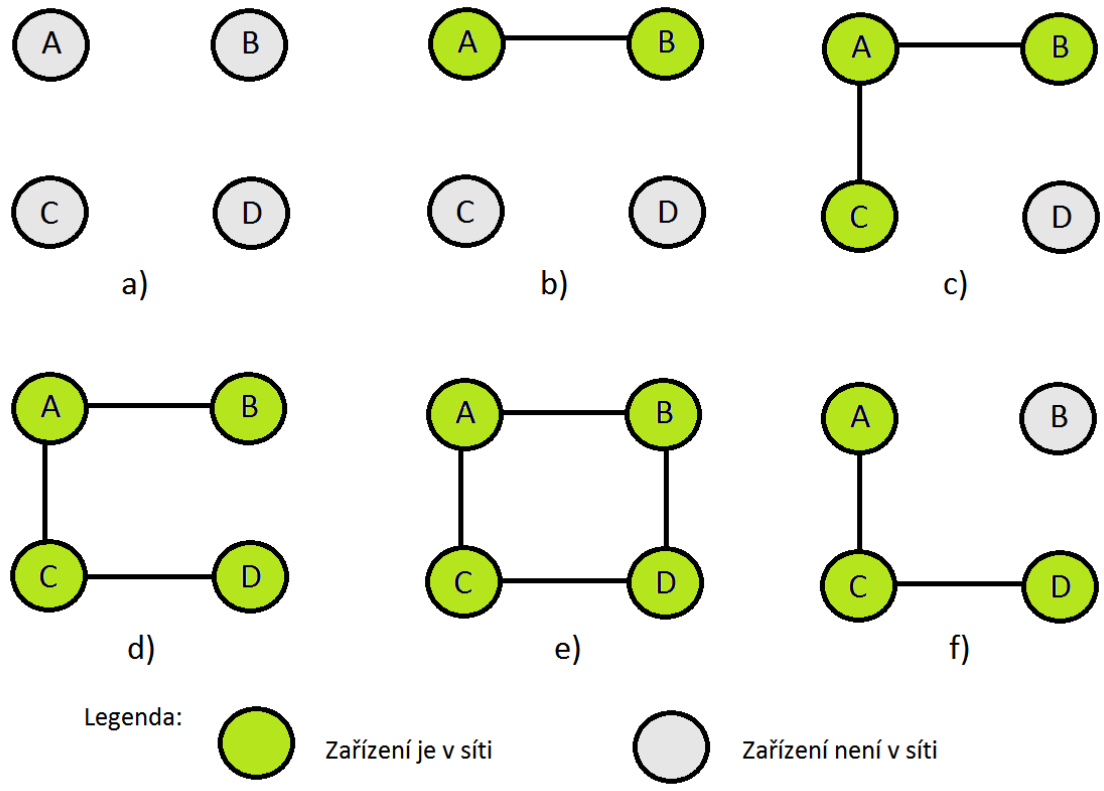
4.3 Vytvoření sítě

Na základě předchozích informací lze přistoupit ke konkrétnímu popisu ustanovení virtuální místnosti. Mějme 4 zařízení využívající aplikaci s tímto protokolem. Označíme si tato zařízení písmeny A až D. Obrázek 4.9 znázorňuje jednotlivé fáze, ve kterých se může případná virtuální místnost ocitnout.

V 1. fázi mají všechna zařízení spuštěnou aplikaci. V tomto momentě naslouchají a očekávají připojení, případně mohou samostatně vyhledat zařízení v okolí a pokusit se navázat spojení.

Ve 2. fázi nejprve zařízení A iniciuje spojení se zařízením B na úrovni technologie Bluetooth. Následuje sestavení spojení na úrovni navrženého protokolu. Protokol přijímá informaci o úspěšném sestavení spojení a zařízení si vyměňují informace o topologii. Virtuální místnost o minimální velikosti byla tedy právě vytvořena. Zařízení si již mohou vyměňovat zprávy, případně spojení ukončit a vrátit se tak do 1. fáze.

Ve 3. fázi se zařízení A připojí k zařízení C. Následuje odeslání paketů typu UPD na obě připojená zařízení. Také zařízení C odesílá informace, které má k dispozici. Platí, že obě zařízení, která právě vytvořila spojení zašlou o této skutečnosti informace všem.



Obr. 4.9 : Jednotlivé fáze spojení: a) 1.fáze b) 2. fáze c) 3. fáze d) 4. fáze e) 5. fáze f) 6.fáze.

Ve 4. a 5. fázi byla postupně vytvořena finální podoba sítě, kdy pokud dojde k odpojení jednoho zařízení, stále může pokračovat komunikace. Jedná se tedy o necentralizovanou síť, kde, pokud je to ošetřeno, může se původní iniciátor sítě odpojit a ostatní účastníci mohou v konverzaci pokračovat. Starost o redundanci byla tedy přenechána na uživateli. Ze snah o automatické připojování zařízení v síti, a tedy vytvoření redundance protokolem, bylo upuštěno z důvodu různého přístupu obou platforem k zabezpečení. U některých zařízení musí dojít k instalaci vzdáleného zařízení, jiná jsou pak třeba viditelná pouze při uživatelském nastavení a to pouze dočasně.

Pokud tedy v 6. fázi dojde například k vypnutí aplikace na zařízení B a ukončení spojení, zařízení A i D odešlou všem zbývajícím zařízením paket typu DACK s informací o této skutečnosti. Po přepočítání směrovacích tabulek mohou ostatní účastníci dále pokračovat ve výměně textových zpráv.

4.4 Předávání zpráv

Veškeré předchozí procedury měly za cíl vytvořit prostředí umožňující potvrzované posílání textových zpráv a to jak jednomu zařízení adresně, tak i všem zařízením, která jsou dostupná.

Vlastní předávání zpráv v již vytvořené virtuální místnost je již relativně snadné. Vstupem od uživatele je zpráva ve formě textového řetězce a označení cíle, a to buď jménem cílového zařízení, nebo klíčovým slovem pro hromadnou komunikaci.

Pokud se jedná o zprávu adresovanou jednomu zařízení, je pomocí uchovávaného slovníku přeloženo jméno na adresu. Následně je ve směrovací tabulce vyhledán záznam o dalším skoku a textová zpráva opouští zdrojové zařízení zapouzdřená jako paket typu MSG. Každé zařízení v cestě k cíli zkontroluje, zda paket není určen jemu a případně jej předá dál. Cílové zařízení zašle zpět paket typu ACK a informace o doručení je předána uživateli.

Pokud protokol obdrží žádost o doručení hromadné zprávy, jednoduše pošle zprávu všem zařízením ve své směrovací tabulce adresně, ovšem zapouzdřenou do paketu typu EMSG. To umožní příjemci zjistit, že se jedná o hromadnou zprávu, na kterou reaguje potvrzovacím paketem typu EACK. Odesílatel má tedy podle příchozích paketů EACK informaci o tom, komu všemu byla zpráva doručena.

5 VÝVOJ APLIKACÍ

V následujícím textu budou představeny obě aplikace. Nejprve jsou uvedeny prostředky použité při vývoji, poté je popsána jejich struktura a pomocí vybraných zdrojových kódů je ukázáno, jak konkrétně pracují. Následuje ukázka fungování obou aplikací při simulaci scénáře z kapitoly 4.3.

5.1 Použité prostředky

Pro vývoj aplikace pro Windows bylo využito následujících prostředků:

- **Microsoft Visual Studio 2013:** Vývojové prostředí společnosti Microsoft pro vývoj aplikací pro platformu .NET. Ve verzi Express je po registraci zdarma ke stažení na stránkách společnosti Microsoft [29].
- **Knihovna InTheHand.Net.Personal.dll:** Knihovna projektu 32feet.NET, která pracuje mimo jiné s technologií Bluetooth. Tuto knihovnu lze stáhnout na oficiálních stránkách projektu [30].
- **Knihovna Microsoft.Expression.Interactions.dll a System.Windows.Interactivity.dll**: Knihovny firmy Microsoft přidané do projektu za účelem lepší funkcionality XAML. Po registraci ke stažení zde [31].

Vývoj a testování aplikace pro OS Windows probíhal na notebooku Lenovo E530 s aktuálním operačním systémem MS Windows 8.1, dále byl při testování použit notebook Lenovo Z580 s operačním systémem MS Windows 7.

Veškeré prostředky pro vývoj aplikací pro OS Android jsou zdarma ke stažení z oficiálních webových stránek pro vývojáře Android Developers [32]. Dále jsou uvedeny konkrétní využité prostředky:

- **Eclipse IDE**: Open source vývojové prostředí primárně určené k vývoji aplikací v jazyce Java.
- **JDK (Java Development Kit**): Soubor základních nástrojů pro vývoj aplikací pro platformu Java.
- **ADT (Android Development Tools)**: Plugin pro vývojové prostředí Eclipse IDE, který přidává funkce nutné pro vývoj aplikací pro Android.
- **SDK (Software Development Kit) Platform Android 4.4 (API 19)**: Vývojové nástroje pro Android aplikace ve verzi 4.4.

Aplikace byla vyvíjena a testována za pomoci zařízení Lenovo IdeaTab S8-50 a Google Nexus 7 II využívající verzi Androidu 4.4 a 5.0.2.

5.2 Struktura aplikací

Obě aplikace jsou rozděleny na 3 logické části. První je vlastní implementace protokolu, na kterou je pomocí druhé, propojovací části, navázána poslední část, uživatelské rozhraní. Základní myšlenkou rozdělení na tyto části je vytvoření snadno udržovatelné vrstvové architektury.

5.3 Implementace protokolu

Celý protokol je opět rozdělen do tří částí, vrstev, jež odlišují jejich funkce. První část se stará o vlastní spojení na úrovni technologie Bluetooth. Další se pak, využívající služeb první, stará o směrování ve vytvořené síti a komunikaci se třetí vrstvou, která se stará o vlastní předávání textových zpráv. Implementace protokolu je na obou platformách do jisté míry podobná, proto je pro demonstraci využit prioritně kód jazyka C# z důvodu lepší čitelnosti.

```
// Komentáře popisující kód

// Popis vynechaných částí kódu
```

5.3.1 Namespace Connections

Třída Scanning

Základem pro vytvoření spojení je možnost vyhledat zařízení v okolí. K tomuto slouží statická třída Scanning. V jazyce C#, za použití knihovny InTheHand.Net.Personal.dll, je vyhledávání uskutečněno pomocí metody DiscoverDevicesInRange() zavolané z instance třídy BluetoothClient. Návratovou hodnotou je pole BluetoothDeviceInfo[], z kterého lze následně získat objekty, které lze použít pro komunikaci na úrovni Bluetooth.

```
public static BluetoothClient ClientConnection {get; set;}        // Instance třídy
                                  // mající metodu pro vyhledávání zařízení v okolí
public static BluetoothDeviceInfo[] Devices {get; set;}          // Pole nalezených
                                                                        // zařízení
public static List<string> InRangeList {get; set;}    // Seznam nalezených zařízení
public static event OnScanDone scanDone;        // Událost při ukončení vyhledávání
public delegate void OnScanDone();                                 // Delegát události
public static void scan()                                  // Při zavolání metody scan()
{                                                                    // se do pole „Devices“
    ClientConnection = new BluetoothClient();       // uloží informace o nalezených
    Devices = ClientConnection.DiscoverDevicesInRange();                // zařízeních
    InRangeList.Clear();       // Následně je pro každé uložena informace o jménu a
    foreach (BluetoothDeviceInfo c in Devices)                  // adrese do seznamu
        InRangeList.Add(c.DeviceName + "\n" + c.DeviceAddress.ToString("C"));
    done();                                                  // Zavoláním metody done()
}
public static void done()
{                                                          // je zkontrolováno, zda je
    if (scanDone != null)          // někdo přihlášen na odběr události, pokud ano,
    scanDone();                     // je zavolán její delegát, událost je vyvolána
}
```

V jazyce Java na platformě Android je situace poněkud odlišná. Vyhledávání zde funguje za pomoci instance třídy BroadcastReceiver a BluetoothAdapter. Do proměnné „bluetoothAdapter“ je pomocí metody getDefaultAdapter() přiřazena reference na defaultní Bluetooth adapter v zařízení. Následně po zavolání metody startDiscovery() je započato vyhledávání. Pomocí zaregistrovaných filtrů, instancí třídy IntentFilter, je v metodě onReceive() reagováno na události generované při vyhledávání.

```
public static void startDiscovery()
{
    if(bluetoothAdapter.isDiscovering())          // Pokud adaptér stále vyhledává,
        bluetoothAdapter.cancelDiscovery();              // je vyhledávání ukončeno
    bluetoothAdapter.startDiscovery();                    // Začátek vyhledávání
}
private static void registerFilters(MainActivity main)
{                                          // Vytvoření filtru a registrace filtru
    filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);      // k příslušnému
    main.registerReceiver(receiver, filter);                          // receiveru
    // Další filtry a registrace
}
public static void startReceiving(final MainActivity main,final Handler handler)
{                                  // Po zavolání statické metody startReceiving()
    listOfDevices.clear();
    foundDevices.clear();
    startDiscovery();                                              // začne vyhledávání
    receiver = new BroadcastReceiver()
    {
        @Override
        public void onReceive(Context context, Intent intent)
        {                              // Při zachycení události je zjištěn její typ
            String action = intent.getAction();
            if (BluetoothDevice.ACTION_FOUND.equals(action))
            {               // Při nalezení nového zařízení je přidáno do seznamů a
                BluetoothDevice device =
                intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                foundDevices.add(device);      // objekt listOfDevices je následně
                listOfDevices.add(device.getName() + "\n"+              // odeslán
                device.getAddress());        // s označením 1 do instance handleru
                handler.obtainMessage(1, listOfDevices).sendToTarget();
            }                                  // Princip handleru bude popsán později
            else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action))
            {    // Po ukončení vyhledávání je poslána zpráva do instance handleru,
                handler.obtainMessage(0, new Object()).sendToTarget();
                cancelDiscovery();                          // je ukončeno vyhledávání
                unregReceiver(main);                // a je odregistrován receiver
                return;
            }
            else if (BluetoothAdapter.ACTION_STATE_CHANGED.equals(action))
            {        // Pokud bylo během vyhledávání Bluetooth vypnuto, je vyvolána
                turnOnBluetoothRequest(main);              // žádost o jeho zapnutí
            }
        }
    };
    registerFilters(main);                                         // Registrace filtrů
}
```

Třída Listening

Aby se dalo k zařízení připojit, musí na to být připraveno. Toho je na obou platformách dosaženo v principu stejně. Po zavolání metody setupListener() začne instance třídy BluetoothListener přijímat žádosti o vytvoření spojení. Jedinečný identifikátor „MUUID“ slouží k identifikaci, že se jedná o Bluetooth spojení. Na obou stranách musí být tento identifikátor totožný.

```
public Guid MUUID =                                          // Jedinečný identifikátor
new Guid("00001101-0000-1000-8000-00805F9B34FB");              // Bluetooth připojení
public BluetoothListener bluetoothListener { get; set; }          // Instance třídy
                                                                 // BluetoothListener
public void setupListener()
{
    bluetoothListener = new BluetoothListener(MUUID);      // Inicializace instance
    bluetoothListener.Start();             // s příslušným identifikátorem a začátek
    bluetoothListener.BeginAcceptBluetoothClient               // přijímání spojení
    (this.BluetoothListenerAcceptClientCallback, bluetoothListener);
}
private void BluetoothListenerAcceptClientCallback(IAsyncResult result)
{             // Toto je vykonáváno asynchroně, naslouchání tedy neblokuje aplikaci
    try                 // Pokud je někdo připojen, zavolá se tato metoda, ve které
    {                                                  // je opět započato naslouchání
        BluetoothListener bluetoothListener =
        (BluetoothListener)result.AsyncState;
        bluetoothListener.BeginAcceptBluetoothClient
        (this.BluetoothListenerAcceptClientCallback, bluetoothListener);
        BluetoothClient client =
        bluetoothListener.EndAcceptBluetoothClient(result);
        Room.onConnect(client);                  // Reference na připojené zařízení
    }                                                        // je předána vyšší vrstvě
    // Ošetření vyjímek
}
```

Třída Connecting

Tato třída slouží k vytvoření Bluetooth spojení. V konstruktoru je instanci třídy předána reference na zařízení vyhledané pomocí třídy Scanning. Pomocí této reference je vytvořeno spojení, instance BluetoothClient, které je předáno vyšší vrstvě. Lze si povšimnout, že je použit stejný identifikátor, jako na druhé straně spojení.

```
public Guid MUUID =                                          // Jedinečný identifikátor
new Guid("00001101-0000-1000-8000-00805F9B34FB");              // Bluetooth připojení
private BluetoothDeviceInfo deviceInfo;         // Informace o zařízení k připojení
public Connecting(BluetoothDeviceInfo DeviceInfo)                    // Konstruktor
{
    deviceInfo = DeviceInfo;
}
public void connectToDevice()
{
    BluetoothClient client = new BluetoothClient();         // Pomocí nové instance
    client.BeginConnect(deviceInfo.DeviceAddress, mUUID,      // BluetoothClient je
    this.BluetoothClientConnectCallback, client);      // vyvolán pokus o připojení
}
```

```
private void BluetoothClientConnectCallback(IAsyncResult result)
{
    try
    {                                   // Výsledná reference na připojené zařízení je
        BluetoothClient client = (BluetoothClient)result.AsyncState;
        client.EndConnect(result);
        Room.onConnect(client);                              // předána vyšší vrstvě
    }
    catch (SocketException)
    {                         // Pokud se spojení nezdaří vyšší vrstva je informována
        Room.Status = "Connection\nwas not succesfull";
        Room.status();
    }
}
```

Třída InputOutput

Tato statická třída slouží k odesílání a přijímání zpráv. Obsahuje 2 hlavní metody. Metoda Write() slouží k zasílání zpráv. Metoda GetStream() pak k vytvoření nového vlákna, ve kterém se ve smyčce čeká na příchozí zprávy. Aby nedocházelo k blokování aplikace, má každé připojené zařízení své vlákno. Jako kódování je použit Unicode, tedy 16 bitů na znak.

```
public static void Write(string address, string packet)
{
    try
    {
        byte[] bytesToSend;        // Přiřazení reference na další skok k příslušné
        BluetoothClient client  = RoutingTable.Table[address];            // adrese
        bytesToSend = Encoding.Unicode.GetBytes(packet);             // Převedení do
        Stream str = client.GetStream();                // Unicodu a získání streamu
        str.Write(bytesToSend, 0, bytesToSend.Length);          // Zápis bytů paketu
    }                                                                      // do streamu
    catch (Exception)
    {                 // Případná chyba je předána výš s referencí na chybné spojení
        RoutingHandler.badPacketReceived(address);
    }
    // Další vyjímky
}
public static void GetStream(BluetoothClient client)
{
    var thread  = new Thread(() => StreamThread(client));      // Vytvoření a start
    thread.Start();                                                // nového vlákna
}
private static void StreamThread(BluetoothClient client)
{
    Stream readingStream;
    string receivedStream;
    while (true)
    {
        try
        {                 // Získání streamu pro zápis a čtení do bufferu velikosti
            readingStream = client.GetStream();               // nejdelšího možného
            byte[] received = new byte[40150];                                // paketu
            readingStream.Read(received, 0, received.Length);             // Řetězec
            receivedStream = Encoding.Unicode.GetString(received);
            if (receivedStream[0] == '\0')
                throw new FormatException();
            RoutingHandler.routePacket(receivedStream, client);          // získaný
```

```
        }                                       // z bufferu je předán ke směrování

        catch (Exception)
        {                                   // Při chybě je toto oznámeno vyšší vrstvě
            RoutingHandler.badPacketReceived(address);
            client.Close();
            break;                                              // Smyčka je ukončena
        }
    }
}
```

5.3.2 Namespace Routing

Třída Packet

Jedná se o abstraktní třídu, která definuje společné součásti všech typů paketů pomocí konstant a vlastností, které přebírají pomocí dědičnosti. Jelikož je třída abstraktní, tak vynucuje přepsání svých metod ve svých potomcích. Využití konstant definovaných na jednom místě usnadňuje změnu parametrů.

```
public string DestinationAddress {get; set;}                    // Zdrojová adresa
public string SourceAddress {get; set;}                           // Cílová adresa
public int Ttl {get; set;}                                          // Počet skoků
public Guid PacketID {get; set;}                 // Jedinečný identifikátor paketu
public char TypeOfPacket {get; set;}                                 // Typ paketu
private const string trailer = "end";                          // Ukončovací řetězec
private const char b = '\0';                                      // Vyplňovací znak
private const int addressLength = 17;                               // Délka adresy
private const int guidLenght = 36;              // Délka jedinečného identifikátoru
private const int typeLenght = 1;                     // Délka označení typu paketu
private const int ttlLength = 2;        // Definuje počet cifer údaje o počtu skoků
private const int dictionaryMaxLenghtOrder = 2;       // Počet cifer délky slovníku
private const int nameLenght = 17;                                   // Délka jména
private const int maxLenghtOrder = 4;                  // Počet cifer délky zprávy

public abstract string getString();       // Metoda převede paket do podoby řetězce
public abstract Packet getPacket(string toPacket);     // Metoda převede řetězec na
                                                                  // objekt typu Packet
```

Třída MessagePacket

Jedná se o třídu, která dědí z třídy Packet a je použita k přenosu zprávy. Hlavními metodami jsou přepsané metody rodičovské třídy getString() a getPacket(). Metoda getString() vrací řetězec definovaného formátu. Metoda getPacket() pak vrací objekt typu Packet, který obsahuje údaje obsažené v přijatém řetězci. K rozdělení řetězce, je použita metoda Substring(), která vrací nový řetězec tvořený částí původního řetězce.

```
public override string getString()              // Metoda vracející řetězec z paketu
{
    string MessageLength = MessageCharCount.ToString();    // Převedení počtu znaků
    while (MessageLength.Length != MaxLenghtOrder)          // na řetězec a přidání
    MessageLength += B;                    // vyplňovacího znaku do požadované délky
    string TTL = System.Convert.ToString(Ttl);        // Analogicky pro počet skoků
    while (TTL.Length != TtlLength)
```

```
    TTL += B;
    return (TypeOfPacket + DestinationAddress + SourceAddress    // Metoda následně
    + MessageLength + Message + PacketID + TTL + Trailer    // vrací součet řetězců
    );                                                    // jednotlivých vlastností
}
public override Packet getPacket(string toPacket)              // Metoda vracející
{                                                        // Paket vytvořený z řetězce
    MessagePacket p = new MessagePacket();                          // Nový objekt
    p.TypeOfPacket = toPacket[0];
                                                       // Typ paketu jako první znak
    p.DestinationAddress = toPacket.Substring                      // Cílová adresa
     (p.TypeLenght, p.AddressLenght);

    p.SourceAddress = toPacket.Substring                           // Zdrojová adresa
     (p.TypeLenght + p.AddressLenght, p.AddressLenght);

    p.MessageCharCount = Int32.Parse(toPacket.Substring             // Délka zprávy
     (p.TypeLenght + 2 * p.AddressLenght, p.MaxLenghtOrder)     // převedena na typ
    .Trim(p.B));                              // int po odstranění vyplňovacích znaků


    // Další dekódování řetězce

    if (toPacket.Substring(p.TypeLenght + 2 * p.AddressLenght
    + p.MaxLenghtOrder + p.MessageCharCount + p.GuidLenght + TtlLength,
    p.Trailer.Length) == p.Trailer)           // Pokud se na přesném místě na konci
    {     // řetězce nachází ukončovací řetězec „end“, je vrácen objekt typu Packet
        return p;
    }
    else
    {                                      // V opačném případě je vyvolána vyjímka,
        throw new FormatException();           // která je zpracována vyšší vrstvou
    }
}
```

Třídy AcknowledgementPacket a UpdatePacket

Tyto dvě třídy opět dědí z rodičovské třídy Packet.cs a používají podobnou logiku k přepsání abstraktních metod. Instance třídy AcknowledgementPacket slouží jako potvrzovací pakety, kdy je jedinečný identifikátor nesený paketem předán vyšší vrstvě. Z instance třídy UpdatePacket je možno získat informace, které jsou následně použity pro výpočet cesty k ostatním zařízením v síti.

Třida RoutingTable

Jedná se o statickou třídu, která udržuje veškeré informace o síti. Zapouzdřuje členy, pomocí kterých je možný překlad údajů nutných pro směrování. Nejdůležitější je slovník „Table“, který je vytvářen pomocí algoritmu za použití informací ve slovníku „AddressListAddressPairs“, který udržuje informace o spojeních v síti, tedy její topologii. Pomocí slovníku „Table“ je umožněn překlad adresy zařízení na objekt „BluetoothClient“, který reprezentuje další skok. Tento objekt je použit k odeslání paketu. Dále pak třída definuje metody pro podmíněné odstraňování záznamů ze slovníků. Například pokud zanikne spojení se sousedem, ale existuje k němu i nadále cesta, není odebrán ze členů sítě.

```
public static Dictionary<string, List<string>> // Slovník udržující topologii sítě
AddressListAddressPairs {get; set;}
public static Dictionary<string,string>              // Slovník umožňující překlad
AddressNamePairs {get; set;}                                    // adresy na jméno
public static Dictionary<string, string>             // Slovník umožňující překlad
NameAddressPairs {get; set;}                                    // jména na adresu
public static Dictionary<string, BluetoothClient>    // Slovník umožňující překlad
AddressDirectlyConnectedPairs {get; set;}      // adresy na objekt BluetoothClient
public static Dictionary<string, BluetoothClient>    // Slovník umožňující překlad
Table {get; set;}   // adresy na objekt BluetoothClient, reprezentující další skok
public static List<string>                                  // Seznam jmen členů sítě
MemberNames {get; set;}
```

Třída RoutingAlghorithm

Třída RoutingAlgotithm obsahuje jednu hlavni metodu run(), která využívá informací obsažených ve třídě RoutingTable k vytvoření směrovací tabulky používané při odesílání paketů.

```
public static void run()
{
    // Inicializace proměnných
    // „topology“ je kopií AddressListAddressDictionary, z které
    // jsou mazány uzly, které již algoritmus prošel
    // „toFind“ je seznam adres, ke kterým je třeba nalézt cestu
    // „costTable“ je dočasná směrovací tabulka, která obsahuje i počet skoků

    foreach (var a in RoutingTable.AddressDirectlyConnectedPairs)
    {                                   // Všichni přímo připojení jsou přidání
        if (toFind.Contains(a.Key))
        {
            costTable.Add(a.Key, new Dictionary<int, BluetoothClient>());
            costTable[a.Key].Add(1, a.Value);                // do dočasné tabulky
            toFind.Remove(a.Key);          // se vzdáleností 1 a vyřazeni z hledání
        }
    }

    if (toFind.Count == 0)                      // Pokud byli již všichni nalezeni
    {
        setTable(costTable);                     // je nastaven slovník „Table“ ve
        return;                           // třídě RoutingTable a funkce je ukončena
    }


    foreach (var b in RoutingTable.AddressDirectlyConnectedPairs)
    {                                    // Pro každého souseda, je vytvořen strom
        int i = 1;                                          // ve kterém jsou hledány
        getTopology(topology);                    // cesty ke všem zařízením v síti
        removeDirectlyConnected(topology);
       find(toFind, topology, b.Key, b.Key, b.Value, i, costTable);
    }
    setTable(costTable);              // Nakonec jsou do finální směrovací tabulky
}                                                   // zkopírovány nejkratší cesty
```

Metoda find() slouží k vyhledávání v topologii. Jedná se o rekurzivní metodu, která prochází jednotlivé uzly, přidá je do dočasné směrovací tabulky a následně pro každého kdo s tímto uzlem sousedí volá sama sebe, dokud není prohledán celý strom.

```
private static void find(List<string> toFindList, Dictionary<string, List<string>>
topology, string node, string hop, BluetoothClient client, int i,
Dictionary<string, Dictionary<int, BluetoothClient>> costTable)
{
    if (topology[node].Count != 0)              // Pokud je uzel k někomu připojen
    {
        i++;                                          // je vzdálenost zvýšena o 1
        foreach (var a in topology[node])                 // a pro každého souseda
        {                                                  // platí, že pokud není
            if (toFindList.Contains(a))
            {                                    // hledaný v dočasné tabulce s nižší
                if(!costTable.ContainsKey(a))                          // vzdáleností
                    costTable.Add(a, new Dictionary< BluetoothClient,int>());
                if (!costTable[a].ContainsKey(client))             // je tam přidán
                    costTable[a].Add(client,i);
                if (costTable[a][client] > i)
                    costTable[a][client] = i;
            }
        removeNode(topology, node);       // Následně je uzel odstraněn z topologie
        find(toFindList, topology, a, hop, client, i, costTable);           // a je
        }                                    // spuštěno vyhledávání pro další uzel
    }
}
```

Třída RoutingHandler

Tato statická třída obsahuje veškerou logiku směrování ve vytvořené síti. Základem je metoda routePacket(), která přečte první znak příchozího řetězce a určí, o jaký se jedná paket. Na základě toho je volána příslušná metoda, které je paket předán k dalšímu zpracování.

```
public static void routePacket(string packet, BluetoothClient client)
{
    switch (packet[0])
    {
        case 'a':                               // Pokud je typ paketu „a“ je směrován
            routeMessagePacket(packet, client);       // jako paket nesoucí zprávu
            break;
        case 'b':                                                  // Paket typu „b“
            routeAckPacket(packet, client);                // pak jako potvrzovací
            break;
        // Další typy paketů
        default:                        // V případě, že se paket neshoduje se žádným
            badPacketReceived(client.RemoteEndPoint.Address.ToString("C"));
            break;         // podporovaným typem, je mětodě předána adresa souseda,
    }                                                      // který zaslal vadný paket
}
```

Obecně platí, že je zkontrolována cílová adresa paketu. Pokud se tato cílová adresa neshoduje s adresou místního Bluetooth zařízení, je paket předán dál, na základě informací ze směrovací tabulky. V případě shody je paket se zprávou, či potvrzením poslán vyšší vrstvě. Paket se směrovacími informacemi slouží k přepsání informací ve třídě RoutingTable a je

spuštěn algoritmus pro výpočet cesty. Dále platí, že pakety nesoucí zprávu, tedy s příznakem „a“ a „e“, využívají třídy MessagePacket, pakety s příznakem „b“ a „f“ AcknowledgementPacket a pakety nesoucí informace o topologii, tedy s příznakem „c“ a „d“, pak UpdatePacket.

```
private static void routeMessagePacket(string packet, BluetoothClient client)
{
    MessagePacket msg = new MessagePacket();        // Nová instance MessagePacket
    try
    {
        msg = (MessagePacket)msg.getPacket(packet);            // je naplněna daty
    }                                                                  // z řetězce
    catch (FormatException)
    {                                                // Pokud má řetězec špatný formát
        badPacketReceived(client);         // je metodě badPacketReceived() předána
    }                             // reference na souseda, který poslal špatný paket
    if (msg.DestinationAddress ==      // Pokud se lokální adresa shoduje s cílovou
    BluetoothRadio.PrimaryRadio.LocalAddress.ToString("C"))      // adresou paketu
    {                                 // je předána vyšší vrstvě, buď jako hromadná,
        if (msg.TypeOfPacket == 'e')                  // pokud je typ paketu „e“,
            Room.newMessageReceived("Everyone",
            RoutingTable.AddressNamePairs[msg.SourceAddress],
            msg.Message, msg.PacketID);
        else                                                  // případně adresně.
            Room.newMessageReceived(msg.SourceAddress,
            RoutingTable.AddressNamePairs[msg.SourceAddress],
            msg.Message, msg.PacketID);
        sendAck(msg);
    }
    else                                    // Pokud má paket jinou cílovou adresu
        reSend(msg);                                              // je poslán dál
}
private static void reSend(Packet packet)
{                            // Při přeposlání je počet skoků paketu snížen o jeden
    packet.Ttl -= 1;                              // Pokud je i poté větší než jedna
    if(packet.Ttl > 0)                                  // paket je poslán dále
        sendToDestination(packet.DestinationAddress, packet.getString());
}
private static async void sendToDestination(string address, string packet)
{                                // Asynchroně je volána metoda write(), která předá
    await Task.Run(() =>InputOutput.Write(address, packet));
}                                                        // paket k dalšímu skoku
```

Pokud od vyšší vrstvy přijde požadavek na odeslání zprávy, nebo směrovacích informací na určitou adresu, je vytvořena nová instance příslušného paketu, která je naplněna informacemi a v podobě řetězce je předána k odeslání.

```
public static void SendUpdate(string address)
{
    UpdatePacket p = new UpdatePacket();               // Nové instanci paketu je
    p.TypeOfPacket = 'c';                               // přiřazen typ „c“, celá
    p.AddressAddressPair = RoutingTable.AddressListAddressPairs;      // topologie,
    p.AddressNamePair = RoutingTable.AddressNamePairs;        // členové místnosti,
    p.SourceAddress = BluetoothRadio.PrimaryRadio.      // zdrojová adresa místního
    LocalAddress.ToString("C");                                        // zařízení,
    p.DestinationAddress = address;                          // cílová adresa, délka
    p.AddressNamePairCount = RoutingTable.AddressNamePairs.Count;           // obou
```

```
    p.AddressAddressPairCount = RoutingTable.getAddressPairCount();     // slovníků
    p.Ttl = 64;                                                        //a počet skoků
    sendToDestination(p.DestinationAddress, p.getString());            // Odeslání
}
```

V případě, že má příchozí paket špatný formát, nebo pokud nastane jiná chyba při spojení, je pomocí metody badPacketReceived() zajištěno odpojení příslušného souseda a předání zprávy ostatním členům sítě.

```
public static void badPacketReceived(string address)
{
        foreach (string s in RoutingTable.Table.Keys)           // Pro každého, ke
        {                       // kterému je známá cesta, kromě právě odpojovaného
            if (s != address)
                SendDeleteUpdate(BluetoothRadio.PrimaryRadio.         // Je poslána
                LocalAddress.ToString("C"),                  // zpráva o přerušení
                address, s);                                          // spojení
        }
        if(RoutingTable.AddressNamePairs.ContainsKey(address))       // Je nastaven
            Status = RoutingTable.AddressNamePairs[address] +             // status
            "\nhas been disconnected";                  // pro zobrazení uživateli
        if(statusEvent!=null)
        {                               // a je vyvolána událost, pro změnu statusu
            statusEvent();
        }                               // dále je pak soused odstraněn z topologie
        removeClient(address);                          // a je spuštěn algoritmus
}                                                  // pomocí metody removeClient()
```

5.3.3 Namespace Conversations

Tato vrstva, jak název napovídá, využívá služeb podřízených vrstev a udržuje informaci o vyměňovaných zprávách, ze kterých za pomoci identifikačních údajů ostatních zařízení vytváří konverzaci.

Třída Message

Základní jednotkou konverzace je zpráva. Třída Message zapouzdřuje vlastnosti každé takové zprávy. Tyto vlastnosti popisují kromě obsahu vlastní zprávy také identifikační údaje o zprávě samotné, o jejím původci a také o jejím stavu. Na základě těchto vlastností je pak se zprávou nakládáno. Třída dále obsahuje 2 konstruktory, které nastaví jednotlivé vlastnosti na základě toho, zda jde o zprávu přijatou, nebo odeslanou.

```
public string Address {get; set;}                                  // Adresa protějšku
public string Name {get; set;}                                           // Jeho jméno
public string Text {get; set;}                                  // Vlastní text zprávy
public string Time {get; set;}                                 // Čas odeslání/přijetí
public bool Delivered {get; set;}                     // V případě adresné komunikace
public List<string> DeliveredTo {get; set;}       // V případně hromadné komunikace
public bool FromHere {get; set;}       // Udává, jestli je zařízení původcem zprávy
public Guid ID {get; set;}                           // Jedinečný identifikátor zprávy
```

Třída Conversation.cs

Tato třída zapouzdřuje celou jednu konverzaci, dialog. Obsahuje v sobě seznam zpráv v konverzaci („InMessageConversation“). Pokud je zpráva poslána adresně, z tohoto zažízení, je zařazena do slovníku „ToDeliver“, jehož klíčem je jedinečný identifikátor zprávy. Po přijetí potvrzení o doručení je pak z toho slovníku odstraněna a vlastnost „Delivered“ této zprávy je nastavena na hodnotu „true“. K zobrazení konverzace je zapotřebí řetezec „InStringConversation“.

```
public List<Message> InMessageConversation {get; set;}        // Seznam všech zpráv
public Dictionary<Guid, Message> ToDeliver {get; set;}       // Slovník ID - Zpráva
public string InStringConversation {get; set;}   // Konverzace převedená do řetězce
```

Pokud do konverzace přibude nová zpráva, případně je odeslaná zpráva potvrzena, dochází k vygenerování řetězce. Tento řetězec je generován na základě vlastností všech zpráv, které jsou obsaženy v „InMessageConversation“ pomocí metody refreshConversation().

```
public void refreshConversation()                          // Při zavolání této metody
{
    InStringConversation = string.Empty;        // je smazán starý řetězec a následně
    foreach (Message m in InMessageConversation)        // je procházena každá zpráva
    {                                                                  // v konverzaci
        if (m.FromHere && !m.Delivered)                // Odeslaná, nedoručená zpráva
        {                                    // K řetězci jsou přičteny údaje o zprávě
            InStringConversation += m.Time + " " + m.Name +      // (čas, odesilatel)
            " says: (not delivered)\n" + m.Text + "\n";              // a text zprávy
        }
        else if (m.FromHere && m.Delivered)              // Odeslaná, doručená zpráva
        {
            InStringConversation += m.Time + " " + m.Name +
            " says: (delivered)\n" + m.Text + "\n";
        }
        else                                                          // Přijatá zpráva
        {
            InStringConversation += m.Time + " " + m.Name +
            " says: \n" + m.Text + "\n";
        }
    }
}
```

Tato metoda slouží k sestavení řetězce pro adresnou komunikaci. Pro hromadnou konverzaci je ještě nutno v řetězci uvést, komu byla zpráva doručena. Metoda ovšem funguje na podobném principu.

Třída Room.cs

Statická třída Room je abstrakcí virtuální místnosti. Zapouzdřuje slovník, umožňující přeložit adresu na konverzaci, a také seznam se jmény všech účastníků. K interakci s uživatelským prostředím je v jazyce Java využit tzv. Handler, kterému je poslána zpráva, obsahující identifikaci události a případně i objekt. V jazyce C# jsou využity události zastoupené delegáty. Třída také obsahuje řetězec, reprezentující aktuální stav místnosti. Slouží k informování uživatele o události, jako je odpojení účastníka, nebo nemožnosti navázat spojení.

```
public static Dictionary<string, Conversation>                          // Slovník
Conversations {get; set;}                                             // s konverzacemi
public static List<string> Members {get; set;}              // Jména členů místnosti
public static string Status { get; set; }                 // Aktuální status místnosti
public static event OnNewMessage message;           // Událost značící novou příchozí
public delegate void OnNewMessage(string address);                        // zprávu
public static event OnConnect connectionDone;       // Událost vyvolaná při vytvoření
public delegate void OnConnect();                                     // nového spojení
public static event NotConnected connectionFailed;           // Událost vyvolaná při
public delegate void NotConnected();                                 // selhání spojení
public static event setStatus statusEvent;             // Událost značící změnu stavu
public delegate void setStatus();                                          // místnosti
```

Za účelem obsluhy vstupů uživatele a možnosti komunikace s nižší vrstvou tato třída definuje několik metod. Jednou z nich je metoda onConnect(), která je volána, pokud je vytvořeno nové spojení.

```
public static async void onConnect(BluetoothClient client)           // Po připojení
{
    Status = client.RemoteMachineName + "\nhas connected";          // je změněn stav
    status();                                          // místnosti a je vyvolána událost

    RoutingTable.addAddressListAddressPair                      // Spojení je zaneseno
    (client.RemoteEndPoint.Address.ToString("C"),                     // do topologie
    BluetoothRadio.PrimaryRadio.LocalAddress.ToString("C"));

    // Další přidávání do slovníků

    newConversation                                      // Je vytvořena nová konverzace
    (client.RemoteEndPoint.Address.ToString("C"));                  // s tímto členem

    await Task.Run(() => InputOutput.GetStream(client));          // Je započat příjem
    RoutingAlgorithm.run();                                 // a jsou přepočítány cesty

    foreach (string s in Routing.RoutingTable.Table.Keys.ToArray())
    {                                        // Poté je všem zaslána aktuální topologie
        if (s != BluetoothRadio.PrimaryRadio.LocalAddress.ToString("C"))
            Routing.RoutingHandler.SendUpdate(s);
    }
    getRoomMembers();                                  // Obnoven seznam členů v místnosti
    connected();                               // Vyvolána událost značící nové spojení
}
```

Dále třída obsahuje metody pro příjem a pro posílání zpráv. Metoda sendMessageTo() přidá do příslušné konverzace zprávu a předá ji včetně adresy nižší vrstvě. Případně potvrzení je zpracováno metodou delivered(), která vyvolá přepočítání řetězce konverzace. Případná odpověď je zpracována metodou newMessageReceived().

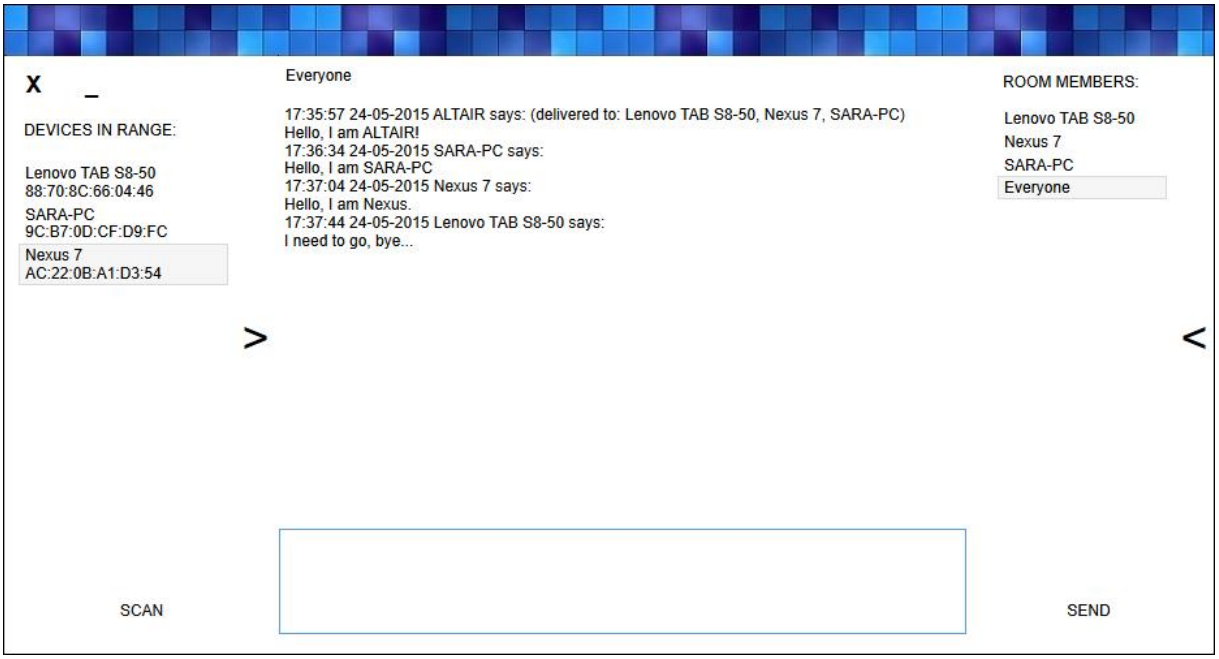
```
public static void sendMessageTo(string name, string text)           // Při požadavku
{                                                                    // na poslání zprávy
    string toAddress ;
    if(name == "Everyone")
        toAddress  = "Everyone";
    else                                                          // je přeloženo jméno
        toAddress = RoutingTable.NameAddressPairs[name];                  // na adresu
```

```
 Message m = new Message                                    // Je vytvořena nová zpráva,
    (toAddress, BluetoothRadio.PrimaryRadio.Name, text, true, Guid.NewGuid());

    RoutingHandler.sendMessageTo(toAddress, text, m.ID);      // která je odeslána,
    Conversations[toAddress].ToDeliver.Add(m.ID, m);      // přidána do konverzace,
    Conversations[toAddress].InMessageConversation.Add(m);
    Conversations[toAddress].refreshConversation();       // která je přepočítána a
    refreshConversation(toAddress);                  // je vyvolána událost značící
}                                                                  // změnu konverzace

public static void delivered(string address, Guid deliveredID)
{                                                            // Pokud přijde potvrzení,
    if (Conversations.ContainsKey(address))
    {                                              // je zpráva označena jako doručená,
        Conversations[address].wasDelivered(deliveredID);          // konverzace je
        Conversations[address].refreshConversation();                // přepočítána
        refreshConversation(address);                         // a následuje událost
    }
}

public static void newMessageReceived                     // Při příchodu nové zprávy
(string fromAddress, string fromName, string text, Guid id)
{                                             // je přidána do příslušné konverzace,
    Message m = new Message(fromAddress, fromName, text, id);
    Conversations[fromAddress].InMessageConversation.Add(m);
    if (fromAddress == "Everyone")
    {                                                          // která je přepočítána
        Conversations[fromAddress].refreshEveryoneConversation();
        refreshConversation(fromAddress);                 // a je vyvolána událost
    }
    else
    {
        Conversations[fromAddress].refreshConversation();
        refreshConversation(fromAddress);
    }
}
```

5.4 Uživatelské rozhraní

Část uživatelského rozhraní je nedílnou součástí každé aplikace. Uživatel pomocí něj předává své požadavky programu, který na ně reaguje. Uživatelské rozhraní pro Android je vytvářeno pomocí značkovacího jazyku XML. Pomocí značkovacích tagů jsou definovány jednotlivé uživatelské prvky, jejich poloha a jejich chování. Uživatelské rozhraní pro .NET aplikace typu WPF se vytváří za pomoci značkovacího jazyka XAML, který je upravenou verzí univerzálního XML. Jazyk XAML nabízí více možností práce s uživatelským rozhraním a také podporuje tzv. binding, tedy že vlastnost nějakého prvku uživatelského rozhraní je provázána s proměnnou v programu v jazyce C#.

5.4.1 Uživatelské rozhraní .NET WPF



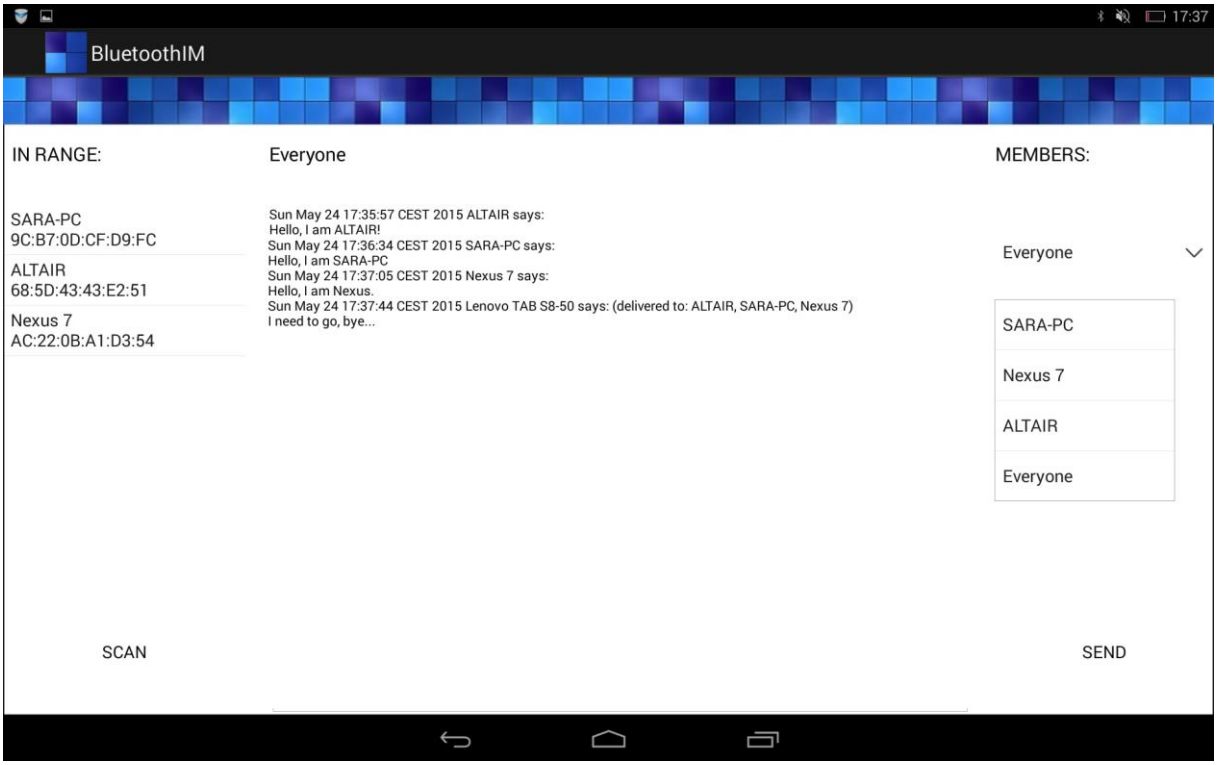
Obr. 5.1: Vzhled aplikace pro Windows

Ve verzi aplikace pro Windows jsou veškeré uživatelské prvky definovány v souboru MainWindow.xaml. Definice prvku funguje na jednoduchém principu, kdy tag prvku, například <Button>, obklopuje jednotlivé své vlastnosti, například Width="169", a také svůj obsah. Vlastnost Command definuje tzv. příkaz, který je vykonáván při interakci s prvkem, například při stisknutí tlačítka. Vlastnosti jako ItemsSource či Text pak definují obsah, který bude zobrazen v prvku. Binding pak odkazuje na konkrétní proměnnou v programu.

```
<Window
    x:Name="Window"                                          // Jméno prvku
    x:Class="BluetoothIM.View.MainWindow"          // Třída prvku a jmenné prostory
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:i="http://schemas.microsoft.com/expression/2010/interactivity"
    // Další jmenné prostory
    WindowStyle="None"                        // Odstranění defaultního rámečku okna
    Width="{Binding Path=WindowWidth}"                                  //Šířka okna
    WindowState="{Binding Path=WindowState}"    // Stav okna, například „Minimized“
    Icon="icon.png"                                              // Ikona programu
    BorderBrush="Black"                                  // Jednopixelový černý
    BorderThickness="1px"                                        // rámeček okna
    // Další vlastnosti
>

    <Window.DataContext>                        // Definice tzv. datacontextu, tedy
    <vm:IMViewModel/>                    // třídy, s kterou je prvek Window svázán
    </Window.DataContext>                                    // pomocí bindingu

    // Definice chování prvku Window při tažení, či zavírání, dále definice
    // „ButtonStyle“ upravující vzhled a chování tlačítek

    <Grid Margin="0,0,0,0">                     // Plocha, ve které jsou další prvky
```

```
        <TextBox                                // Textové pole pro výpis konverzace
            x:Name="Input"                      // Jméno prvku a svázání s proměnnou
            Text="{Binding Path=InputText, Mode=OneWay}">             // InputText
            <TextBox.Style>                       // Definice stylu textového pole,
                <Style>
                    <Style.Triggers>                    // kdy pokud se změní text
                        <EventTrigger
                            RoutedEvent="TextBox.TextChanged">
                        <SoundPlayerAction                              // je přehrán
                            Source="beep.wav"/>                             // soubor
                        </EventTrigger>                               // beep.wav, jako
                    </Style.Triggers>                               // jako upozornění
                </Style>
            </TextBox.Style>


        <i:Interaction.Triggers>                      // Dále pokud je změněn text
            <i:EventTrigger EventName="TextChanged">
                <si:CallMethodAction                  // je skrolováno až na konec
                    TargetObject="{Binding ElementName=Input}"
                    MethodName="ScrollToEnd">                     // textového pole
                </si:CallMethodAction>
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </TextBox>

        <ListBox                              // Prvek zobrazující zařízení v dosahu
            x:Name="InRange"            // jménem InRange má provázány vlastnosti s
            ItemsSource="{Binding Path=InRangeCollection}"         // datacontextem
            SelectedIndex="{Binding Path=ToConnect}">
            <i:Interaction.Triggers>                      // Při dvojkliku je volán
                <i:EventTrigger EventName="MouseDoubleClick">    // příkaz _Connect
                    <i:InvokeCommandAction Command="{Binding Path=_Connect}"/>
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </ListBox>
        // Další prvky uživatelského rozhraní
    </Grid>
</Window>
```

5.4.2 Android layout



Obr. 5.2: Vzhled aplikace pro Android

Soubor activitiy_main.xml

Obdobně jako v XAML je i zde pomocí XML definováno uživatelské rozdaní pomocí tagů a jejich vlastností. Aby mohla aplikace fungovat na různých rozlišeních, je nutné, aby rozměry všech prvků byly definovány jako poměr velikosti obrazovky. Za tímto účelem je v rodičovském prvku <RelativeLayout> rozdělena plocha pomocí prvků <LinearLayout> do mřížky, v které jsou umístěny prvky jako <Button>, nebo <ListView>. Velikosti prvků je dána vlastností „weight“, která definuje, kolik procent zabírá prvek ve svém rodiči.

```
<RelativeLayout                                          // Rodičovský prvek
    // Další vlastnosti
    android:background="@drawable/background">       // Obrázek na pozadí ve složce
                                                          // „drawable“
    // Další prvky <LinearLayout>
    <LinearLayout                    // Prvek LinearLayout obaluje uživatelské prvky
        android:layout_height="fill_parent"      // Na výšku vyplňuje svého rodiče
        android:orientation="vertical"                    // Jedná se o sloupec,
        android:layout_width="0dp"             // šířka je tedy nastavena na 0 a je
        android:layout_weight="0.18">        // definována vlastností weight na 18%
        // Další prvky uživatelského rozhraní
        <Spinner                                          // Rozbalovací nabídka
            android:id="@+id/Members"                        // má id Members
            android:layout_width="match_parent"       // Šířka je shodná s rodičem
            android:layout_height="0dp"                   // a výška je nastavena
            android:layout_weight="0.15"                     // na 15% rodiče
            android:textColor="#000000" />                // Barva písma je černá
```

```
        <Button                                  // Tlačítko sloužící k odeslání zprávy
            android:id="@+id/Send"                                    // má id „Send“
            style="?android:attr/borderlessButtonStyle"       // Odstranění rámečku
            android:textColor="#000000"             // Odkaz na definované chování v
            android:background="@drawable/button_style"     // souboru button_style
            android:layout_width="match_parent"         // Šířka je shodná s rodičem
            android:layout_height="0dp"
            android:layout_weight="0.20"              // Výška pak 20%, při kliknutí
            android:onClick="SendClicked"          // je volána metoda SendClicked()
            android:text="SEND"/>                          // Tlačítko má text „SEND“
    </LinearLayout>                                                  // Uzavírací
</RelativeLayout>                                                        // tagy
```

Třída MainActivity

Jelikož Android defaultně nepodporuje obdobu bindingu, jsou při startu aplikace předány reference na prvky, které je třeba programově ovládat, třídě ViewController. Ta pak může za běhu nastavovat vlastnosti prvků a zpracovávat uživatelské vstupy.

```
ViewController controller;                          // Pokud je zapnuto Bluetooth
// Zjištění zda je zapnuto Bluetooth
controller = new ViewController         // je předána reference na jednotlivé prvky
    ((ScrollView) findViewById(R.id.Scroll),        // prvky uživatelského rozhraní
    (Button)findViewById(R.id.Send),
    (ListView) findViewById(R.id.InRange),
    // Další prvky
    (Button)findViewById(R.id.Scan),this);
```

Soubor AndroidManifest.xml

Kromě definice prvků uživatelského rozhraní musí také každá aplikace pro Android obsahovat tzv. manifest. Tento soubor, psaný v XML, obsahuje důležité informace o aplikaci, které jsou nezbytné pro její fungování, jako například informaci o Java balíčku, který obsahuje aplikaci, nebo o její verzi:

```
<manifest                                                              // Jmenný
xmlns:android="http://schemas.android.com/apk/res/android"          // prostor XML
package="bluetoothIM.View"                       // Java balíček obsahující aplikaci
android:versionCode="1" android:versionName="1.0">               // Verze aplikace
```

Určuje pro jaké verze Androidu je aplikace určena pomocí verze API (Application Programming Interface), která se pro každou verzi liší:

```
<uses-sdk
android:minSdkVersion="14"                     // Nejstarší podporovaná verze (4.0)
android:targetSdkVersion="19" />                     // Cílená verze (Android 4.4)
```

Určuje, jaká oprávnění aplikace ke své funkci potřebuje:

```
<uses-permission android:name=                                  // Vyžaduje svolení
"android.permission.BLUETOOTH_ADMIN" />                  // k používání Bluetooth
<uses-permission android:name=
"android.permission.BLUETOOTH" />
<uses-permission android:name=                          // Vyžaduje svolení ke čtení
"android.permission.WRITE_EXTERNAL_STORAGE" />          // a zapisovaní na úložiště
<uses-permission android:name=
"android.permission.READ_EXTERNAL_STORAGE" />
```

5.5 Propojení protokolu a uživatelského prostření

V obou verzích aplikace od sebe byly odděleny vrstvy uživatelského rozhraní a protokolu pro výměnu zpráv. Oba tyto celky je nutné nějak propojit. Za tímto účelem je na platformě Windows přidán projekt ViewModel, na platformě Android toto zastává již dříve zmíněná třída ViewController.

5.5.1 ViewModel

Tento projekt obsahuje hlavní třídu IMViewModel, která implementuje rozhraní INotifyPropertyChanged, které při změně proměnné aktualizuje obsah prvku uživatelského rozhraní, s kterým je proměnná svázána. Dále obsahuje tzv. commandy, ve kterých je vykonávána úloha vyžádána interakcí uživatele. Dále je zde třída IMViewModelLogic, která je přihlášena k událostem, které produkuje protokol a reaguje na ně.

Třída IMViewModel

```
public class IMViewModel : INotifyPropertyChanged
{
    public Scan _Scan {get; set;}                                   // Command typu Scan
    // Další commandy
    public IMViewModelLogic logic {get; set;}          // Instance logiky ViewModelu
    private List<string> inRangeCollection;              // Seznam zařízení v dosahu
    private int toConnect;       // Index zařízení vybraného uživatelem k připojení
    // Další vlastnosti
    public event PropertyChangedEventHandler PropertyChanged;          // Při změně
    private void OnPropertyChanged(string propertyName)      // hodnoty je vyvolána
    {                           // událost PropertyChanged, zpracovaná touto metodou
        PropertyChangedEventHandler handler = PropertyChanged;
        if (handler != null)
        {
            handler(this, new PropertyChangedEventArgs(propertyName));
        }
    }

    public IMViewModel()                                                // Konstruktor
    {
        inRangeCollection = new List<string>();
        logic = new IMViewModelLogic(this);                    // Předání referencí
        _Scan = new Scan(this);                                             // na sebe
        // Inicializace dalších vlastností
    }

    public int ToConnect
    {
        get {return toConnect;}
        set                                             // Při změně hodnoty vlastnosti
        {
            toConnect = value;
            OnPropertyChanged("ToConnect");                 // je vyvolána událost
        }
    }
    public List<string> InRangeCollection
    {
```

```
    get {return inRangeCollection;}
    set
    {
        inRangeCollection = new List<string>();
        foreach (string s in value)            // Nastavená hodnota je zkopírována
        {
            if (!inRangeCollection.Contains(s))                  // do vlastnosti
                inRangeCollection.Add(s);
        }
        ScanText = "SCAN";            // Skenování dokončeno (změna ze „SCANNING“)
        OnPropertyChanged("InRangeCollection");             // Je vyvolána událost
    }
}
```

Třída Scan

Tzv. commandy jsou vykonávány při interakci s uživatelským rozhraním. V konstruktoru jim je předána reference na instanci třídy představující tzv. datacontext uživatelského rozhraní a proto s ní mohou pracovat. Nutná je implementace rozhraní ICommand, které obsahuje předepsané metody.

```
public class Scan : ICommand                                  // Implementace rozhraní
{
    IMViewModel vm;                                       // Reference na IMViewModel
    public Scan(IMViewModel VM)
    {
        vm = VM;                                            // předaná v konstruktoru
    }
    public bool CanExecute(object parameter)    // Před vykonáním je zkontrolováno,
    {                           // jestli lze příkaz vykonat, metoda může obsahovat
        return true;                                  // podmínky, které toto upravují
    }                                                                  // Událost při
    public event EventHandler CanExecuteChanged;          // změně vykonavatelnosti
    public void Execute(object parameter)           // Metoda je volána po kontrole
    {                                                                  // vykonavatelnosti
        if(vm.ScanText == "SCAN")                      // Pokud skenování neprobíhá
        {
            vm.ScanText = "SCANNING...";             // je nastaven text tlačítka a
            vm.Logic.ScanningThread = new Thread(Scanning.scan);         // v novém
            vm.Logic.ScanningThread.Start();                // vlákně je vyhledáváno
        }
    }
}
```

Třída IMViewModelLogic

Třída tedy obsahuje logiku pracující mezi protokolem a datacontextem a řídí jejich spolupráci. Při události vyvolané protokolem nastaví příslušnou vlastnost datacontextu a naopak při uživatelské akci předává pokyny protokolu v podobě volání jeho funkcí.

```
public class IMViewModelLogic                                  // Třída s logikou
{
    private IMViewModel vm {get; set;}        // obsahuje referenci na datacontext,
    private Thread scanningThread {get; set;}         // dále pak vlákno, ve kterém
    private Listening listener {get; set;}              // probíhá skenování a také
                                    // Listner, pro inicializaci přijímání spojení
    public void init()     // Konstruktor volá tuto metodu, kde je třída přihlášena
    {                                       // k událostem, které generuje protokol
```

```
    Scanning.scanDone += () =>                  // Například při zachycení události
    {                                                   // při skončení skenování
        List<string> s = new List<string>();                  // je kopie seznamu
        foreach (string t in Scanning.InRangeList)                  // nalezených
        s.Add(t);                                                     // zařízení
        Vm.InRangeCollection = s;                       // přiřazena do datacontextu
    };
    // Přihlášení k dalším událostem
    BluetoothRadio.PrimaryRadio.Mode                    // Zviditelnění zařízení,
    = RadioMode.Discoverable;                         // které je pak možno vyhledat
    listener = new Listening();                             // Zahájení přijímání
    listener.setupListener();                                           // spojení
    }
    public static void Connect(int ToConnect)                     // Pro připojení
    {                              // je metodě předán index v nalezených zařízeních
        Connecting c = new Connecting(Scanning.Devices.ElementAt(ToConnect));
        c.connectToDevice();           // K tomuto zařízení se pak protokol připojí
    }
}
```

5.5.2 Třída ViewController

Tato třída představuje úplné rozhraní mezi protokolem a uživatelským rozhraním na platformě Android. Komunikace s protokolem probíhá přes tzv. Handler, jehož reference je mu předána. Pomocí metody obtainMessage() a jejich parametrů „what“ a „obj“. První parametr identifikuje událost a v druhém může být volitelně přenesen jakýkoliv objekt, který lze přetypovat pro další využití. Na základě toho může třída promítnout protokolem generované události do prvků uživatelského rozhraní, které jí byly předány v konstruktoru. Při interakci uživatele s prvky rozhraní zde také dochází k volání metod. K demonstraci rozdílu mezi platformami je k ukázce vybrána podobná část logiky propojovací vrstvy.

```
public ViewController(ScrollView scview, TextView conversation,      // Konstruktor
Button send, Spinner spinner, ListView inRange, TextView input,
EditText output, Button scan, MainActivity main)
{
    // Přiřazení do dalších vlastností
    InRange = inRange;                      // Prvku zobrazujícímu zařízení v dosahu
    InRange.setOnItemClickListener(this);              // je přiřazen tzv. Listener
    Scan = scan;
    Scan.setOnClickListener(new OnClickListener())          // Tlačítku Scan obdobně
    {
        @Override                                    // Při kliknutí na toto tlačítko
        public void onClick(View v)
        {
        scan();                                    // dochází k volání metody scan()
        }
    });
    handler = new Handler()                                   // Nová instance handleru
    {
        @Override                              // V konstruktoru definuje chování při
        public void handleMessage(Message msg)                    // přijetí zprávy
        {
            switch (msg.what)
            {
            // Různé další případy
```

```
            case 0:                          // Pokud „what“ je rovno nule, značí to
            Scan.setText("SCAN");                                  // konec skenování
            break;

            case 1:                                      // Pokud „what“ je rovno 1,
            InRange.setAdapter(new ArrayAdapter<String>(Main,
            android.R.layout.simple_list_item_1,
            (ArrayList<String>) msg.obj));                                 // je prvku
                                     // nastaven obsah ve formě nalezených zařízení,
                                                 // které jsou předány ve zprávě
            break;
            // Různé další případy
        };
    Thread threadL = new Thread()                          // V novém vlákně dochází
    {
        public void run()
        {
            Listening.setupListener(handler);               // k přijímání spojení
        }
    };
    threadL.start();
    Room.setHandler(handler);                      // Předání reference na Handler
}
@Override
public void onItemClick(AdapterView<?> parent, View view, int position,
long id)                                          // Při výběru z nabídky nalezených
{
    Connecting connecting;                         // dochází k pokusu o připojení
    connecting = new Connecting(Scanning.getFoundDevices().get(position));
    connecting.connectToDevice(handler);
}
private void scan()
{
    if (Scan.getText().equals("SCAN"))                              // Metoda scan()
    {
        threadS = new Thread()                              // provede v novém vlákně
        {
            public void run()                                 // vyhledávání zařízení
            {
                Looper.prepare();
                Scanning.startDiscovery();
                Scanning.startReceiving(Main, handler);
            }
        };
        threadS.start();
        Scan.setText("SCANNING...");                        // pokud právě neprobíhá
    }
}
```

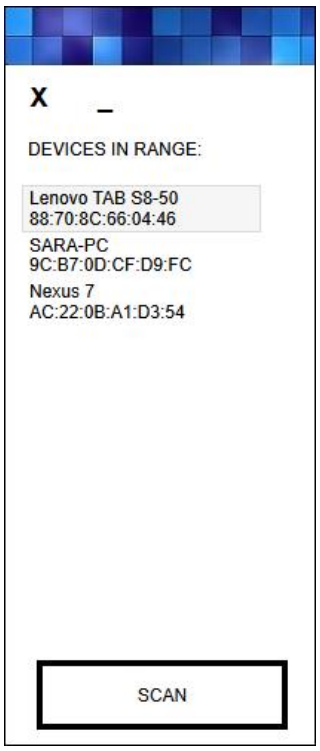
5.6 Simulace scénáře z kapitoly 4.3

Jako praktická ukázka funkčnosti poslouží simulace scénáře z kapitoly 4.3 za pomoci reálných zařízení. Scénář bude probíhat z pohledu zařízení A, u kterého budou uvedeny příchozí a odchozí pakety s časovou značnou. Z toho si lze utvořit představu, jak konkrétně komunikace funguje. Z textové podoby paketu jsou odstraněny výplňové znaky a pro lepší přehlednost jsou bloky odděleny novým řádkem. Simulace se účastní zařízení uvedená v tabulce 5.1.

Tab.5.1 – Parametry zařízení

Označení	Model	Operační systém	Jméno	Adresa
A	Lenovo E530	Windows 8.1	ALTAIR	68:5D:43:43:E2:51
B	Lenovo S8-50	Android 4.4	Lenovo TAB S8-50	88:70:8C:66:04:46
C	Asus Nexus 7	Android 5.0.2	Nexus 7	AC:22:0B:A1:D3:54
D	Lenovo Z580	Windows 7	SARA-PC	9C:B7:0D:CF:D9:FC

5.6.1 První fáze



Obr. 5.3: Výsledky vyhledávání na zařízení ALTAIR

V první fázi neexistuje žádné spojení, zařízení se mohou navzájem vyhledat. Na obrázku 5.3 je zobrazen výsledek vyhledávání zařízení A. Je třeba upozornit, že pokud zařízení vyhledává, nebývá samo viditelné. Po výběru zařízení z nabídky je zahájen pokus o spojení.

5.6.2 Druhá fáze

Po výběru zařízení B z nabídky je sestaveno spojení, které následuje výměna směrovacích informací. Po tomto je již možno zadat zprávu a pomocí tlačítka „SEND“ ji odeslat. Při sestavování spojení jsou vyměněny tyto pakety:

```
A -> B - 17:34:56.837                                    // Směr a časový otisk
c                                                                 // Typ paketu
88:70:8C:66:04:4668:5D:43:43:E2:51                  // Cílová a zdrojová adresa
2                                                              // Počet spojení
88:70:8C:66:04:4668:5D:43:43:E2:51                 // Adresy reprezentující
68:5D:43:43:E2:5188:70:8C:66:04:46                                // spojení
2                                                             // Počet členů
68:5D:43:43:E2:51ALTAIR                                          // Členové
88:70:8C:66:04:46Lenovo TAB S8-50
64                                                            // Počet skoků
end                                                  // Ukončovací řetězec
```

```
B -> A - 17:34:56.900
c
68:5D:43:43:E2:5188:70:8C:66:04:46
2
88:70:8C:66:04:4668:5D:43:43:E2:51
68:5D:43:43:E2:5188:70:8C:66:04:46
2
88:70:8C:66:04:46Lenovo TAB S8-50
68:5D:43:43:E2:51ALTAIR
64
end
```

5.6.3 Třetí fáze

V této fázi zařízení A sestavuje spojení se zařízením C. Po sestavení spojení zašle zařízení A oběma připojeným všechny své směrovací informace. Zařízení B a C se o sobě tedy dozví, vypočítají si k sobě cestu a také si, přes zařízení A, vymění pakety typu UPD. Celkem je tedy vyměněno 5 paketů.

```
A -> B - 17:35:03.097
c
88:70:8C:66:04:4668:5D:43:43:E2:51
4
88:70:8C:66:04:4668:5D:43:43:E2:51
68:5D:43:43:E2:5188:70:8C:66:04:46
68:5D:43:43:E2:51AC:22:0B:A1:D3:54
AC:22:0B:A1:D3:5468:5D:43:43:E2:51
3
68:5D:43:43:E2:51ALTAIR
88:70:8C:66:04:46Lenovo TAB S8-50
AC:22:0B:A1:D3:54Nexus 7
64
end
```

```
A -> C - 17:35:03.099
c
AC:22:0B:A1:D3:5468:5D:43:43:E2:51
4
88:70:8C:66:04:4668:5D:43:43:E2:51
68:5D:43:43:E2:5188:70:8C:66:04:46
68:5D:43:43:E2:51AC:22:0B:A1:D3:54
AC:22:0B:A1:D3:5468:5D:43:43:E2:51
3
68:5D:43:43:E2:51ALTAIR
88:70:8C:66:04:46Lenovo TAB S8-50
AC:22:0B:A1:D3:54Nexus 7
64
end
```

5.6.4 Čtvrtá fáze

Nyní je vytvořeno spojení mezi zařízeními C a D. V síti jsou již tedy 4 zařízení, která spolu mohou komunikovat. Zařízení A po této události přesměruje několik paketů. Paket adresovaný pro něj má formát:

```
C - > A - 17:35:13.110
c
68:5D:43:43:E2:51AC:22:0B:A1:D3:54
6
68:5D:43:43:E2:51AC:22:0B:A1:D3:54
68:5D:43:43:E2:5188:70:8C:66:04:46
88:70:8C:66:04:4668:5D:43:43:E2:51
AC:22:0B:A1:D3:5468:5D:43:43:E2:51
AC:22:0B:A1:D3:549C:B7:0D:CF:D9:FC
9C:B7:0D:CF:D9:FCAC:22:0B:A1:D3:54
4
68:5D:43:43:E2:51ALTAIR
88:70:8C:66:04:46Lenovo TAB S8-50
AC:22:0B:A1:D3:54Nexus 7
9C:B7:0D:CF:D9:FCSARA-PC
64
end
```

Nyní může zařízení A zaslat zprávu zařízení D, které následně odešle potvrzení:

```
A -> D - 17:35:20.136
a
9C:B7:0D:CF:D9:FC68:5D:43:43:E2:51
13                                                          // Délka zprávy
Hello SARA-PC                                                    // Zpráva
ecd923bd-09d8-4971-9823-fa470e8ac53c                 // Jedinečný identifikátor
64
End

D -> A - 17:35:20.952
b
68:5D:43:43:E2:519C:B7:0D:CF:D9:FC
ecd923bd-09d8-4971-9823-fa470e8ac53c          // Identifikátor potvrzované zprávy
63
end
```

5.6.5 Pátá fáze

Aby byla zajištěna redundance a při výpadku jednoho spojení mohla komunikace zbylých stále probíhat je vytvořeno i spojení mezi B a D. Zařízení A o tom dostane mimo jiné tuto zprávu:

```
B -> A - 17:35:38.919
c
68:5D:43:43:E2:5188:70:8C:66:04:46
8
AC:22:0B:A1:D3:5468:5D:43:43:E2:51
AC:22:0B:A1:D3:549C:B7:0D:CF:D9:FC
88:70:8C:66:04:4668:5D:43:43:E2:51
88:70:8C:66:04:469C:B7:0D:CF:D9:FC
68:5D:43:43:E2:5188:70:8C:66:04:46
68:5D:43:43:E2:51AC:22:0B:A1:D3:54
9C:B7:0D:CF:D9:FCAC:22:0B:A1:D3:54
9C:B7:0D:CF:D9:FC88:70:8C:66:04:46
4
AC:22:0B:A1:D3:54Nexus 7
88:70:8C:66:04:46Lenovo TAB S8-50
68:5D:43:43:E2:51ALTAIR
9C:B7:0D:CF:D9:FCSARA-PC
64
end
```

Na obrázcích 5.1 a 5.2 lze vidět uživatelské rozhraní po výměně několika hromadných zpráv. Pokud zařízení A odešle hromadnou zprávu, odešle na každé zařízení paket typu EMSG a dostává potvrzení ve formě paketu EACK, například se zařízením D:

```
A -> D - 17:35:57.250
e
9C:B7:0D:CF:D9:FC68:5D:43:43:E2:51
19
Hello, I am ALTAIR!
0cd837a6-2d8c-4115-83dc-f708dd5b97fa64
end

D -> A - 17:35:58.639
f
68:5D:43:43:E2:519C:B7:0D:CF:D9:FC
0cd837a6-2d8c-4115-83dc-f708dd5b97fa
63
end
```

5.6.6 Šestá fáze

Nyní nastává situace, kdy je využita redundance v síti. Zařízení B se odpojuje, nicméně ostatní mohou v konverzaci pokračovat. Tato událost u zařízení A vede k zaslání dvou paketů typu DUPD zbývajícím členům sítě. Jelikož se zařízením B sousedilo také zařízení C, je obdržen také tento paket:

```
C -> A - 17:38:09.579
d
68:5D:43:43:E2:51AC:22:0B:A1:D3:54
2
AC:22:0B:A1:D3:549C:B7:0D:CF:D9:FC                    // 2 páry adres reprezentující
9C:B7:0D:CF:D9:FCAC:22:0B:A1:D3:54                              // zaniklé spojení
0
64
end
```

5.7 Soubory přiložené na CD

Na přiloženém CD se nachází kompletní zdrojové kódy k oběma aplikacím. K jejich editaci je potřeba dříve představených nástrojů. Dále se zde nachází i spustitelné verze pro obě platformy. V textovém souboru ObsahCD.txt je popsáno jejich umístění.

Pro spuštění aplikace pro Windows je třeba spustit soubor BluetoothIM.exe. Pokud je zapnuto Bluetooth, tak po stisku tlačítka „SCAN“ začne vyhledávání. Dvojklikem na jméno nalezeného zařízení je vytvořeno spojení, okno se zvětší a objeví se šipky, kterými lze ovládat jeho šířku. Po výběru adresáta na pravé straně aplikace je možno odesílat zprávy tlačítkem „SEND“ nebo klávesou „ENTER“. Uprostřed je zobrazena vlastní konverzace.

Soubor BluetoothIM.apk představuje instalační soubor pro zařízení s OS Android. Nejprve je třeba povolit instalaci externích aplikací a poté po výběru souboru dojde k nahrání aplikace na zařízení. Ikona s aplikací by se měla objevit v menu. Po spuštění je ovládání obdobné jako u Windows.

Před startem aplikace je doporučeno, aby bylo na zařízení zapnuto Bluetooth, v opačném případě je zobrazena žádost nebo upozornění. Dále je lepší, pokud jsou všechna zařízení, která budou přímo připojena předem spárována a nainstalovaná, vyžaduje-li to systém. Nedodržení těchto doporučení může vést k nestandardním situacím vlivem nekompatibility různých zařízení a systémů, která nebyla otestována. Při nestandardním chování aplikace je doporučeno Bluetooth adaptér na příslušném zařízení restartovat.

ZÁVĚR

Tato práce se zabývala přenosem dat pomocí technologie Bluetooth mezi platformami Android a MS Windows. Hlavním cílem této práce bylo seznámit se s technologií Bluetooth a na základě získaných znalostí navrhnout dvě verze aplikace, které po vytvoření virtuální místnosti budou uživateli nabízet možnost přenášet zprávy a komunikovat s ostatními připojenými uživateli (tzv. instant messaging). Potvrzované zprávy je možno zasílat adresně i hromadně pro celou místnost.

V teoretické části práce byla detailně rozebrána témata týkající se dané problematiky. Nejprve byla popsána technika přenosu dat pomocí technologie Bluetooth. Poté byly představeny obě platformy, jejich programovací jazyky a nakonec také navržený protokol.

Praktická část se věnovala vlastní realizaci obou aplikací. To zahrnovalo popis prostředků, kterých bylo použito při jejich vývoji a testování. Poté zde byla představena struktura obou aplikací s přiloženými ukázkami důležitých částí kódu. Nakonec byly přiloženy ukázky zasílaných paketů při simulovaném scénáři.

Výstupem této práce jsou tedy dvě aplikace pro výměnu textových zpráv, které mohou sloužit ke komunikaci na vzdálenost v řádu desítek metrů při zpoždění v řádech stovek milisekund.

SEZNAM ZKRATEK

A2DP - Advanced Audio Distribution Profile - Bluetooth profil
ACL - Asynchronous Connectionless - Typ Bluetooth linky
ACK - Acknowledgement - Typ paketu
ADT - Android Development Tools - Nástroje pro tvorbu aplikací
API - Application Programming Interface - Programovací rozhraní
ARQ - Automatic Repeat Request - Technika přenosu
BCL - Basic Class Library - Základní knihovna .NET
BIP - Basic Imaging Profile - Bluetooth profil
BPP - Basic Printing Profile - Bluetooth profil
CD - Compact Disk - Přenosové médium
CLR - Common Language Runtime - Běhové prostředí .NET
CTP - Cordless Telephony Profile - Bluetooth profil
DVD - Digital Versatile Disc - Přenosové médium
DUN - Dial-up Networking Profile - Bluetooth profil
DUPD - Delete Update - Typ paketu
EACK - Everyone Acknowledgement - Typ paketu
EMSG - Everyone Message - Typ paketu
FAX - Fax Profile - Bluetooth profil
FEC - Forward Error Correction - Samoopravný kód
FHSS - Frequency Hopping Spread Spectrum
FTP - File Transfer Profile - Bluetooth profil
GAP - Generic Access Profile - Bluetooth profil
GFSK - Gaussian Frequency Shift Keying - Typ modulace
GOEP - Generic Object Exchange Profile - Bluetooth profil
HDP - Health Device Profile - Bluetooth profil
HFP - Hands-Free Profile - Bluetooth profil
HID - Human Interface Device Profile - Bluetooth profil
HSP - Headset Profile - Bluetooth profil
HTTP - Hypertext Transfer Protocol - Internetový protokol
ICP - Intercom Profile - Bluetooth profil
IDE - Integrated Development Environment - Vývojové prostředí
IEEE - Institute of Electrical and Electronics Engineers - Standardizační organizace
IP – Internet Protocol - Internetový protokol
IrDA - Infrared Data Association - Technologická asociace

ISM - Industrial, Scientific, Medical - Přenosové pásmo
JDK - Java Development Kit - Nástroje pro Javu
JIT - Just In Time - Název kompilátoru
JVM - Java Virtual Machine - Běhové prostředí pro Javu
L2CAP - Logical link control and adaptation protocol - Protokol Bluetooth
LAP - LAN Access Profile - Bluetooth profil
LMP - Link Manager Protocol - Protokol Bluetooth
MS - Microsoft - Softwarová firma
MSG - Message - Typ paketu
MSIL - Microsoft Intermediate Language - Jednotný jazyk
OBEX - Object Exchange Protocol - Protokol Bluetooth
OS - Operační Systém - Software
OPP - Object Push Profile - Bluetooth profil
PPP - Point-to-Point Protocol - Bluetooth protokol
RFCOMM - Radio Frequency Communication - Bluetooth protokol
SCO - Synchronous Connection Oriented - Typ Bluetooth linky
SDAP - Service Discovery Application Profile - Bluetooth profil
SDK - Software Development Kit - Nástroje pro vývoj aplikací
SDP - Service discovery protocol - Bluetooth protokol
SIG - Special Interest Group - Skupina, která vytvořila Bluetooth
SPP - Serial Port Profile - Bluetooth profil
SYNCH - Synchronization Profile - Bluetooth profil
TCP - Transmission Control Protocol - Sítový protokol
TDD - Time-Division Duplexing - Typ dělení kanálu
UDP - User Datagram Protocol - Bluetooth protokol
UPD - Update - Type paketu
USB - Universal Serial Bus - Přenosový standard
VDP - Video Distribution Profile - Bluetooth profil
WAE/WAP - Wireless Application Protocol - Bluetooth protokol
WPAN - Wireless Personal Area Network - Typ sítě
WPF - Windows Presentation Foundation - Typ aplikace
XML - Extensible Markup Language - Značkovací jazyk

POUŽITÉ ZDROJE

[1] Bluetooth*: History of the Bluetooth* [online]. [cit. 2014-11-30]. Dostupné z: http://www.bluetooth.com/Pages/History-of-Bluetooth.aspx

[2] Flickr Software: Bluetooth logo. [online]. [cit. 2014-12-01]. Dostupné z: http://flicksoftware.com/wp-content/uploads/2013/04/bluetooth-logo.png

[3] COOKLEV, Todor. *Wireless communication standards*. New York: IEEE Press, 2004, xxii, 360 s. ISBN 07-381-4066-X.

[4] PC World: Základy technologie Bluetooth. [online]. [cit. 2014-12-01]. Dostupné z: http://pcworld.cz/hardware/Zaklady-technologie-Bluetooth-puvod-a-rozsah-funkci-6635

[5] HANUS, Stanislav. *Bezdrátové a mobilní komunikace*. 1. vyd. Brno: VUT, 2001, 134 s. ISBN 80-214-1833-8.

[6] Programming4: Bluetooth Technical Architecture. [online]. [cit. 2014-12-01]. Dostupné z: http://programming4.us/image/052011/Bluetooth%20Technical%20Architecture_1.jpg

[7] Intelligent Hospital Today: Frequency Hopping Spread Spectrum. [online]. [cit. 2014-12-01]. Dostupné z: http://intelligenthospitaltoday.com/wordpress/wp-content/uploads/2013/05/FHSS.png

[8] STALLINGS, William. *Wireless communications and networking :*. Singapore: Pearson Education, 2002, 584 s. ISBN 81-780-8560-7.

[9] Tutorial Reports: Protocol Stack. [online]. [cit. 2014-12-01]. Dostupné z: http://www.tutorial-reports.com/sites/default/files/bluetoothprotocolstack.gif

[10] Swedetrack: How timeslots are used. [online]. [cit. 2014-12-01]. Dostupné z: http://www.swedetrack.com/images/masla1.gif

[11] Swedetrack: Multi-slot packets. [online]. [cit. 2014-12-01]. Dostupné z: http://www.swedetrack.com/images/masla2.gif

[12] Labinf.polito: Description of the authentication process. [online]. [cit. 2014-12-01]. Dostupné z: http://www.labinf.polito.it/~s100675/bt_sec6.gif

[13] Di.unisa: Bluetooth Encryption. [online]. [cit. 2014-12-01]. Dostupné z: http://www.di.unisa.it/~ads/corso-security/www/CORSO-0203/Bluetooth/Immagini/pag159_1.gif

[14] Mobilenet: Bluetooth sjednotilo bezdrátovou komunikaci. [online]. [cit. 2014-12-01]. Dostupné z: http://mobilenet.cz/clanky/techbox-bluetooth-sjednotilo-bezdratovou-komunikaci-12085

[15] Microsot: Profil společnosti Microsoft Česká republika. [online]. [cit. 2014-12-01]. Dostupné z: http://www.microsoft.com/cs-cz/news/inside_ms.aspx

[16] Microsot: Historie Windows. [online]. [cit. 2014-12-01]. Dostupné z: http://windows.microsoft.com/cs-cz/windows/history#T1=era0

[17] LATTENBERG, Ivo. *Objektově orientované programování*. 1. vyd. Brno: VUT, 2012, 141 s. ISBN 978-80-214-4447-8.

[18] Programujte: Přehled .NET Framework. [online]. [cit. 2014-12-01]. Dostupné z: http://programujte.com/galerie/2008/12/200812072009_framework.png

[19] Programujte: .NET Framework. [online]. [cit. 2014-12-01]. Dostupné z: http://programujte.com/clanek/2008120700-net-framework/

[20] Programujte: Basic Class Library. [online]. [cit. 2014-12-01]. Dostupné z: http://programujte.com/galerie/2008/12/200812072009_bcl_small.png

[21] Programujte: Kompilace. [online]. [cit. 2014-12-01]. Dostupné z: http://programujte.com/galerie/2008/12/200812072010_jit.png

[22] Csharp: Něco málo o C#. [online]. [cit. 2014-12-01]. Dostupné z: http://csharp.aspone.cz/

[23] Android Developers: Android, the world's most popular mobile platform. [online]. [cit. 2014-12-01]. Dostupné z: http://developer.android.com/about/index.html

[24] Google: Podrobná historie společnosti. [online]. [cit. 2014-12-01]. Dostupné z: http://www.google.com/about/company/history/

[25] Engineers Garage: Android. [online]. [cit. 2014-12-01]. Dostupné z: http://www.engineersgarage.com/articles/what-is-android-introduction

[26] The Verge: Android: A visual history. [online]. [cit. 2014-12-01]. Dostupné z: http://www.theverge.com/2011/12/7/2585779/android-history

[27] SCHILDT, Herbert. *Java 7: výukový kurz*. 1. vyd. Brno: Computer Press, 2012, 664 s. ISBN 978-80-251-3748-2.

[28] Boloji: Java program execution. [online]. [cit. 2014-12-01]. Dostupné z: http://cms.boloji.com/articlephotos/Java%20Virtual%20Machine.jpg

[29] Visual Studio: Express. [online]. [cit. 2014-12-01]. Dostupné z: http://www.visualstudio.com/en-us/products/visual-studio-express-vs.aspx

[30] 32feet.NET: Download. [online]. [cit. 2014-12-01]. Dostupné z: http://32feet.codeplex.com/releases/view/88941

[31] Blend Interactivity WPF [online]. [cit. 2015-05-25]. Dostupné z: https://www.nuget.org/packages/Blend.Interctivity.WPF.v4.0/

[32] Android: Get the Android SDK. [online]. [cit. 2014-12-01]. Dostupné z: http://developer.android.com/sdk/index.html