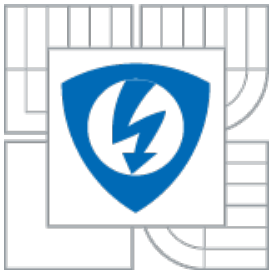
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATIONDEPARTMENT OF TELECOMUNICATION

EXPERIMENTY S TECHNOLOGIÍ LIGHT WEIGHTMESH

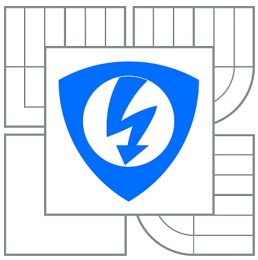
Lightweight Mesh Experiments

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — Richard Nováček
AUTHOR

VEDOUCÍ PRÁCE — Ing. Milan Šimek Ph.D.
SUPERVISOR

BRNO 2014



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Richard Nováček	***ID:***	134097
Ročník:	3	***Akademický rok:***	2013/2014

NÁZEV TÉMATU:

Experimenty s technologií Light Weight Mesh

POKYNY PRO VYPRACOVÁNÍ:

Cílem bakalářské práce je navrhnout a experimentálně ověřit laboratorní úlohy do předmětu MSSY zabývající se novou technologií Light Weight Mesh. Student bude mít k dispozici původní úlohy s technologií Zigbee a dle těchto úloh připraví nové s technologií LWM. Cílem bakalářské práce je vytvořit sadu minimálně čtyř laboratorních úloh, ve kterých budou studenti seznámeni s technologií LWM. V laboratorních úlohách se očekává tématika základní práce s periferiemi, vytvoření komunikačního spoje bod bod, mesh komunikace a pokročilá nastavení LWM pro mesh komunikaci, např. práce se zabezpečením komunikace. Ke každému návodu bakalářské práce bude dodán funkční, komentovaný a otestovány firmware. Návody do laboratorních cvičení budou součástí textu práce.

DOPORUČENÁ LITERATURA:

[1] Stojmenovic I., Handbook of Sensor Networks, Wiley,ISBN:13 978-0-471-68472-5, 2005.
[2] FARAHANI, Shahin. Zigbee Wireless Networks and Transceivers. [s.l.] : Elsevier, 2008. 329 s. ISBN 978-0-7506-8393-7

Termín zadání:	10.2.2014	***Termín odevzdání:***	4.6.2014

Vedoucí práce: Ing. Milan Šimek, Ph.D.
Konzultanti bakalářské práce:

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

UPOZORNĚNÍ:

Abstrakt

Bakalářské práce se zabývá novoutechnologii Lightweight Mesh a její implementaci na laboratorní úlohy do předmětu Bezdrátové senzorové sítě a vytvoření patřičné dokumentace, která bude sloužit jako podklad studentům. Součástí je použití sériové komunikace, časovače a AD převodníku, bezdrátové komunikace se standardem 802.15.4 v Atmel Studiu. Měření maximální propustnosti a parametrů vysílacího výkonu a indikátoru kvality služeb.

Klíčvá slova

Lightweight, mesh, Atmel, Zigbee, Timer, 802.15.4, RSSI, LQI

Abstract

Bachelor thesis deals new technology Lightweight Mesh and its implementation on a laboratory tasks to the subject of wireless sensor networks and the creation of appropriate documentation, which will serve as a basis to students. Also included is the use of serial communication, timer and ADC, wireless comunication with standard 802.15.4 in Atmel Studio. Measure maximal throughput and measurement parameters of transmitting power and indicator quality of service.

Keywords

Lightweight, mesh, Atmel, Zigbee, Timer, 802.15.4, RSSI, LQI

Bibliografická citace mé práce:

NOVÁČEK, R. *Návrh experimentu s technologií Light Weight Mesh.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2014. 39s. Vedoucí bakalářské práce Ing. Milan Šimek, Ph.D..

Prohlášení

Prohlašuji, že svoji bakalářskou prácit na téma Návrh experimentu s technologií Lightweight Mesh vypracoval samostatně pod vedením vedoucího semestrálního projektu 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 tohoto projektu jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 1. května 2014 .....

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Milanu Šimkovi Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mého semestrálního projektu.

V Brně dne 1. května 2014 .....

podpis autora

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic

http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

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.

Brno . . . . . .

. . . . . .
(podpis autora)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OBSAH

1 Úvod 7
2 Přehled protokolu Lightweight Mesh 8
2.1 Architektura Lightweiight Mesh 9
2.2 Směrování 11
3 První laboratorní úloha – základní periferie 15
3.1 Návod pro Práci s LED diodami 15
3.2 Použití časovače 16
4 Druhá laboratorní úloha - použití USART 17
4.1 Použití AD převodníku 18
5 Třetí laboratorní úloha - bezdrátový přenos 19
5.1 Odesílání dat 20
5.2 Příjímání dat 21
5.3 Návod pro samostatnou práci 22
5.4 Senzor osvětlení 25
6 Čtvrtá laboratorní úloha - automatizované měření 26
6.1 Postup pro samostatnou práci 27
7 Měření parametrů bezdrátového přenosu 31
7.1Měření propustnosti 32
7.2 Měření ztrátovosti 33
7.3 Měření RSSI a LQI 34
8 Závěr 35
9 Seznam použité literatury 36
10 Seznam použitých zkratek a symbolů 37
11 Příloha 37

1 Úvod

Cílem bakalářské práce je navrhnout a experimentálně ověřit laboratorní úlohy do předmětu MSSY (bezdrátové senzorové sítě) zabývající se novou technologií Lightweight Mesh (dále LWM). Stávající úlohy pracují s technologií Zigbee od firmy Atmel, tento standard schválený od roku 2004 vznikl jako nadstavba standardu IEEE (Institute of Electronics Engineers) 802.15.4, který definoval fyzickou vrstvu a vrstvu přístupu k médiu. Dle těchto úloh připravit nové s technologií LWM. Cílem bakalářské práce je vytvořit otestovaný a ověřený firmware pro laboratorní úlohy a napsat k nim patřičnou dokumentaci. Součástí práce je použití základních periferií, jako ovládání LED diod, komunikaci s PC přes terminál. Použití časovače procesoru pro periodicky opakující se procesy. Pomocí fotorezistoru a AD převodníku změřit okolní hodnotu osvětlení a bezdrátově přeposlat. Samotný princip bezdrátového přenosu v LWM, včetně použití zabezpečení a základního směrování. V poslední části bylo provedeno automatizované měření skutečných parametrů přenosu, konkrétně propustnosti, ztrátovosti, vysílacího výkonu a indikátoru kvality služeb. Ty měření probíhali mezi dvěma až pěti moduly a na různou vzdálenost.

Realizace bude provedena v Atmel Studiu a součástí práce je implementace LWM na platformu EduBoard obsahující modul Zigbit s integrovaným mikroprocesorem Atmega1281 a AT86RF230 radiovým čipem, tento modul je dostupný v laboratořích.

2 PŘEHLED PROTOKOLU LIGHTWEIGHT MESH

Protokol Lightweight Mesh byl navržen firmou Atmel s ohledem na potřeby celé řady bezdrátových aplikací. Některé z těchto aplikací zahrnují:

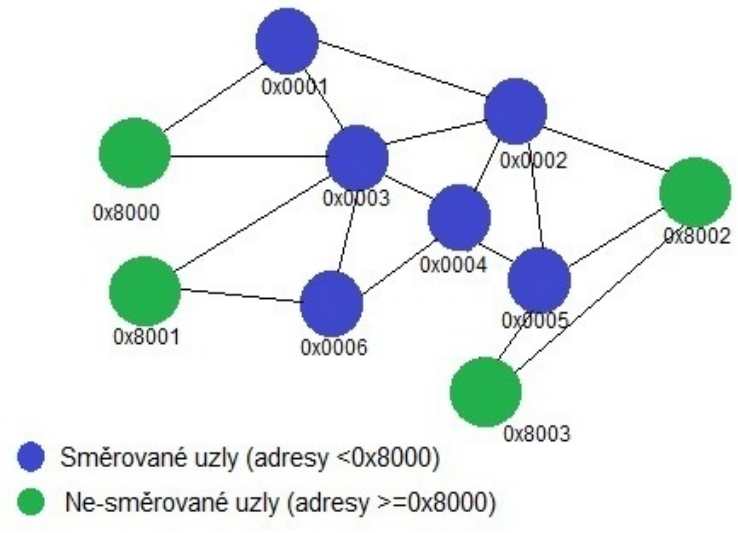
- Dálkové ovládání
- Alarmy a zabezpečení
- Domácnosti a komerční budovy s automatizací
- Hračky a vzdělávací zařízení

Vlastnosti

- Jednoduchá konfigurace a použití
- Až 65 535 uzlů v jedné síti
- Po zapnutí uzlu je připraven k odesílání a přijímání dat; není nutné připojování
- Malá velikost (8KB Flash a 4 KB RAM pro běžné aplikace)
- Směrovací tabulka je automaticky aktualizovaná na základě odeslaných a přijatých rámců

Síťová topologie

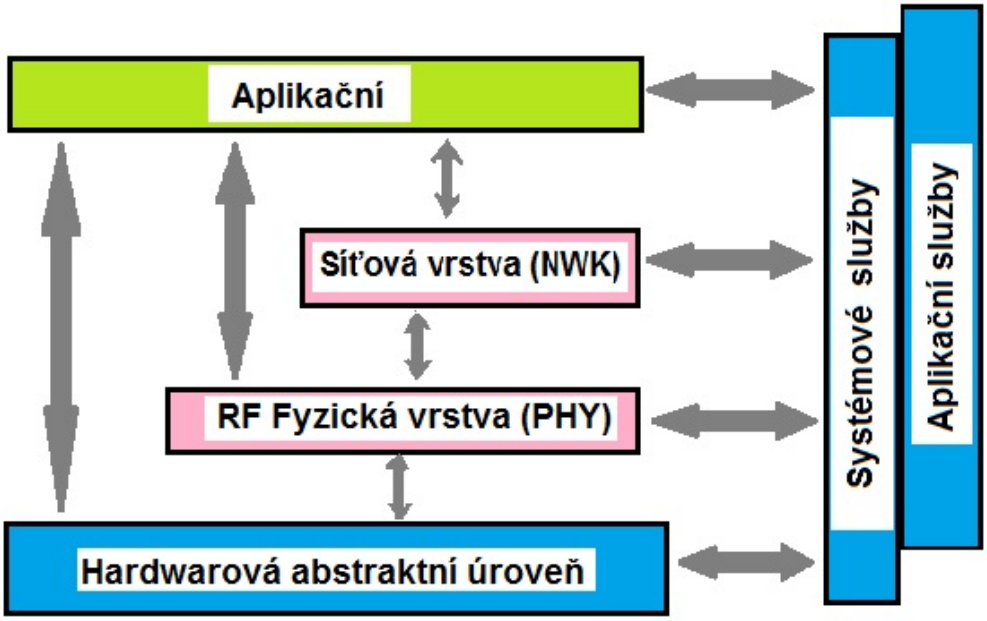
Topologie sítě a možné typy zařízení jsou na Obr. 2.1. Modře označené uzly jsou směrované uzly, které tvoří typické jádro sítě a očekává se, že jsou napájeny. Zeleně označené ne-směrované uzly, jsou součástí sítě a mohou a odesílat a přijímat data, pokud jsou v dosahu, ale neočekává se, že budou k dispozici po celou dobu k dispozici. Typicky jsou umístěny na koncích sítě.



Obr. 2.1: Topologie sítě a typy zařízení.

2.1 ARCHITEKTURA LIGHTWEIIGHT MESH

Architektura Lightweight Mesh je na Obr.2.2. Kód je rozdělen do několika logických úrovní z nich každý poskytuje sadu API (zkratka pro Application Programming Interface) přístupné pro uživatele. Stack je navržen aby poskytoval pouze funkce naprosto nezbytné pro bezdrátovou komunikaci, a očekává se, že zbytek bude vytvořen uživatelem.



Obr.2.2: Lightweight Mesh architektura.

Konvence

Lightweight Mesh používá dobře definovaný soubor konvence, která dělá velmi dobře čitelný zdrojový kód, a sníží dobu vývoje aplikace. Zde jsou základní pravidla:

- Každý název funkce API je prefix názvu vrstvy, kde je umístěna funkce. Například: *NWK_SetAddr ()* je funkce síťové vrstvy zásobníku
- Popisný název funkce může mít *Reg*, *Ind* nebo *Conf* příponu, indikuje následující:
  - *Req* odpovídá na požadavky z uživatelské aplikace do zásobníku (například, *NWK_DataReq ())*
  - *Ind* odpovídá asynchronní příznaky událostí, propagované do uživatelské aplikace do zásobníku (například *NWK_DataInd ())*
  - *Conf* odpovídá uživatelem definovaný callback pro asynchronní požadavky, které potvrzuje požadavky provedení
- Každá struktura a typ názvu nese *_T* příponu

Formát rámce

Lightweight Mesh rámec se skládá ze standardu IEE 805.15.4 MAC hlavičky, síťové hlavičky, aplikačních dat,volitelně (MIC-message integrity code) a kontrolní součet CRC. Obecný formát rámce je zobrazen na obrázku 2.3.

16	8	16	16	16
Frame Control	Sequence number	PAN ID	Destination address	Source address
MAC Header				

8	8	16	16	4	4	0/16
Frame Control	Sequence number	Source address	Destination address	Source Endpoit	Destination Endpoit	Multicast Header
Síťová Hlavička						

Variable	0/32	16
Variable	MIC	CRC
Payload	MIC	CRC

Obr. 2.3: Obecný formát rámce.

Rámec kontrolního pole

Rámec kontrolního pole (Frame Control field) je 1 byte velký a obsahuje kontrolní informace o rámci. Rámec kontrolního pole je zobrazen na obr. 2.4.

BITS:0	**1**	**2**	**3**	**4.7**
Ack Request	Security Enabled	Link Local	Multicast	Reserved
Network Frame Control				

Obr. 2.4: Lightweight Mesh rámec kontrolního pole (Frame Control field).

2.2 SMĚROVÁNÍ

Nativní směrování

Jedná se o originál Lightweight Mesh algoritmus,je to jednoduchý, kompaktní a nepoužívá další příkazy k objevování trasy. Ale tento algoritmus nemůže zaručit, že objevené trasy jsou optimální, protože provádí pouze místní optimalizace. směrovací algoritmy používají směrovací tabulky pro jejich provoz.Směrovací tabulka se skládá ze záznamů popsané v Tab. 2.1.

Tab. 2.1: Směrovací tabulka.

Název	Velikost, bity	Popis
fixed	1	Označuje pevnou položku,která nemůže být odstraněna
multicast	1	Označuje multicast záznam
reserved	2	Vyhrazeno a měla by být 0
score	4	Pokud je hodnota 0, záznam je vymazán z tabulky
dstAddr	16	Cílová adresa
nextHopAddr	16	Síťová adresa dalšího uzlu
rank	8	Udává jak často je záznam použit
lqi	8	Udává kvalitu trasy

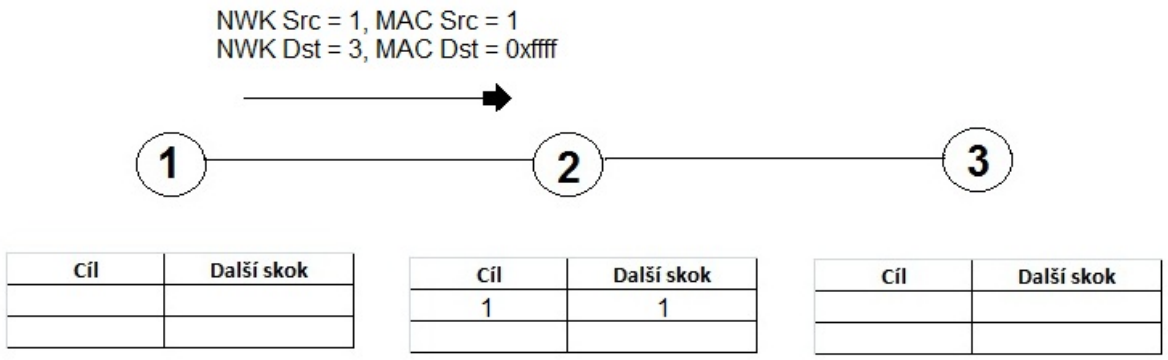
S nativním směrováním neexistuje žádná speciální cesta k objevu trasy, trasa je objevena v rámci běžného poskytování údajů. Princip objevu trasy je znázorněn níže. Uzly označené 1-2-3 jsou směrovací a za těchto předpokladů:

- Uzel 1 chce poslat data do uzlu 3
- Směrovací tabulky na všech uzlech jsou prázdné
- Neexistuje žádná přímá cesta mezi uzly 1 a 3

Počáteční konfigurace je naObr. 2.5.

Obr. 2.5: Počáteční konfigurace sítě.

Prvním krok přenosu dat zahrnuje objev trasy. Viz. Obr. 2.6.Uzel 1 odešle rámec s cílovou adresou (3) , a cílová MAC adresa je 0xffff.

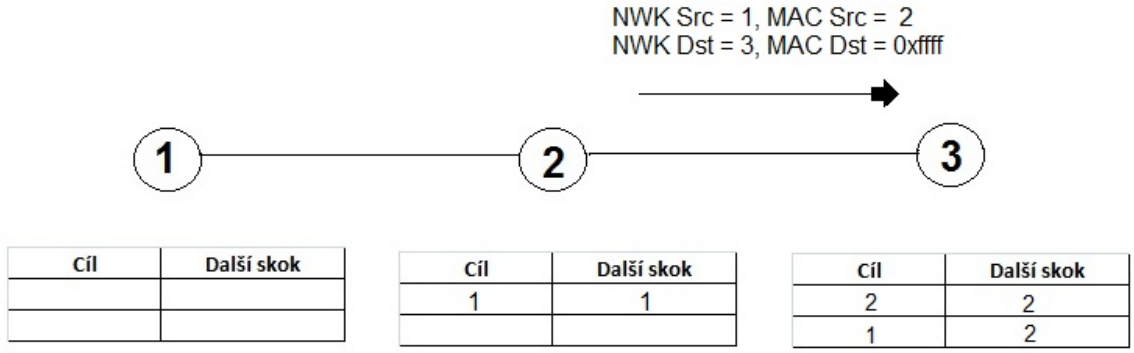


Obr. 2.6: První krok objevu trasy.

1. Uzel 2 přijme rámec a přidá záznam o uzlu 1 do směrovací tabulky.

Druhý krok:Obr. 2.7:

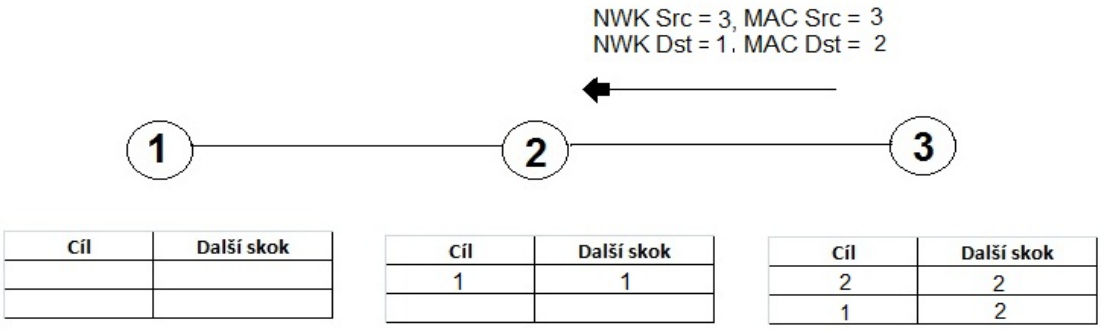
2. Uzel 2 pošle broadcast rámec (protože cílová MAC adresa je nastavena na 0xffff- neznámá)
3. Uzel 3 obdrží tento rámec a přidá záznam o uzlu 2 do své směrovací tabulky.
4. Uzel 3 přidá záznam o uzlu 1 do své směrovací tabulky.



Obr. 2.7: Druhý krok objevu trasy.

Třetí krok: Obr.2.8.

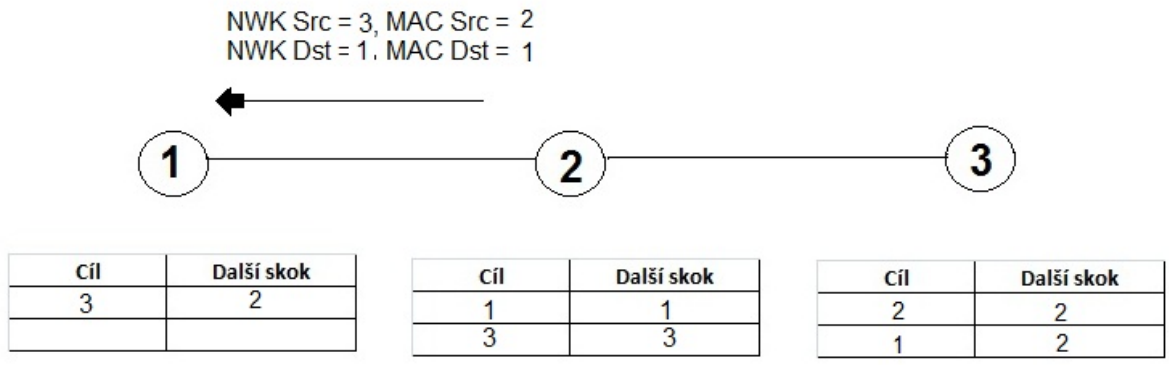
5. Uzel 3 zpracuje rámec a odešle potvrzovací ráme, i když nebyl žádnán. Uzel 3 nyní zná cestu k uzlu 1. (unicast rámec)



Obr. 2.8: Třetí krok zahrnující objev trasy.

Poslední krok a konečná konfigurace sítě: Obr.2.9.:

6. Uzel 2 přijme rámec a přidá cestu k uzlu 3 do jeho směrovací tabulky.
7. Uzel 2 má záznam o uzlu 1, takže směruje rámec do cíle.
8. Uzel 1 obdrží rámec a přidá cestu k uzlu 3 do jeho směrovací tabulky.



Obr.2.9.: Konečná konfigurace sítě, po objevu trasy.

AODV směrování

Je standardní algoritmus, používá další příkazy k provádění zjišťování trasy a objev trasy může trvat delší dobu. Tento algoritmus vybírá optimální trasy a může být využit k nalezení trasy doskupin.Směrovací algoritmus znamená speciální objev trasy postupu. Tento postup je stavový, což znamená, že každý uzel se účastní objevu trasy a musí sledovat pokrok a aktualizovat informace o trasách a byl objeven na základě nově přijatých rámců. Informace o kandidátech trasy jsou uloženy v směrovací tabulce. Potom co tentoproces objevucesty skončí, je tato informace přenesena do směrovací tabulky. Velikost Route Discovery tabulky (tabulka objevu trasy) určuje, kolik procesů objevutrasy,může uzel současně účastnit. Pole tabulky Route Discovery jsou popsány v tab.2.2.

Tab. 2.2:Záznamy tabulky objevu trasy.

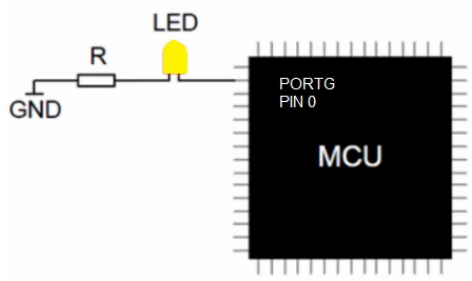
Název	Velikost, bitů	Popis
srcAddr	16	Síťová adresa vyžádání trasy uzlu
dstAddr	16	Síťová adresa cílového uzlu nebo ID skupiny, jak je uvedeno *multicast*oblasti
multicast	8	Označuje multicast objev trasy. Pokud je toto pole nastaveno na 1 a pak*dstAddr* pole obsahuje ID skupiny
senderAddr	16	Síťová adresa uzlu, ze kterého byl poslední zvolený požadavek na směrování přijat. Jedná se o další skok adresy k zdrojového síťového uzlu a na trase
forwardLinkQuality	8	Nahromaděné Link Quality ze zdrojového uzlu
reverseLinkQuality	8	Nahromaděné Link Quality od cílového uzlu
timeout	16	Množství zbývajícího času procesu zjišťování trasy

3 PRVNÍ LABORATORNÍ ÚLOHA – ZÁKLADNÍ PERIFERIE

Vlastní realizace proběhla pomocí stacku Lightweight Mesh, který Atmel nabízí volně na svých stránkách [2]. Který obsahuje všechny dokumenty a hlavně vzorové programy pro práci s LWM, ze kterých bylo vycházeno pro úpravu požadovaného zadání.

3.1 NÁVOD PRO PRÁCI S LED DIODAMI

API protokolu Lightweigh Mesh obsahuje některé speciální příkazy pro základní operace s LED diodami. V našem případě jsou diody zapojeny na portu G a pinech 0 – 2. Připojení diody je znázorněno na Obr.3.1. Správné nastavení portů můžeme vidět v include souboru "halLed.h" viz.výpis kódu 3.1. [3] Kde také můžeme vidět funkce pro práci s LED diodami.



Obr. 3.1: Připojení LED diody k procesoru. (převzato z [3])

Výpis kódu 3.1: halLed.h.

```
    // "halLed.h"
    *- Definitions ------------------------
-----------------------------------*/
      HAL_GPIO_PIN(LED0, G, 0);
      HAL_GPIO_PIN(LED1, G, 1);
      HAL_GPIO_PIN(LED2, G, 2);

    void    HAL_LedInit(void)INLINE
void HAL_LedOn(uint8_t i)
    void HAL_LedOff(uint8_t i)
    void HAL_LedToggle(uint8_t i)
```

Ve složce *LWM\apps\LEDKY\astudio\ZigBit_ATmega1281_Rf230*spustíte projekt v Atmel Studiu a otevře se vám vzorový program pro práci s LED diodami, hlavičkový soubor se jmenuje „tepmlate.c“. Kde vyzkoušíte, co dělají příkazy: *HAL_LedOn(), HAL_Off(), HAL_LedToggle(),* (do prázdného místa v závorce uvěřte číslo pinu, v našem případě (0-2).Doporučuji dávat mezi jednotlivé příkazy zpoždění, aby byla vidět změna stavu lidským okem. Důležité je také v hlavním programu provést inicializaci, ta se provádíme příkazem HAL_LedInit a následně váš kód psát do části APP_TaskHandler. Viz. výpis kódu 3.2: template.c - hlavní program.

```
    static  void  APP_TaskHandler(void)      //  do
této funkce pište váš kód
    {
      HAL_LedToggle(0);    // příkaz pro diody
      _delay_ms(500);    // zpoždění 500ms
    }
    int main(void)
    {
      SYS_Init();        // systémová inicializace
     HAL_LedInit();     // inicializace LED diod
        while (1)
      {
        SYS_TaskHandler();
```

3.2 POUŽITÍ ČASOVAČE

Lightweight Mesh prostředí systému poskytuje podporu pro softwarové časovače. Softwarové časovače mají nízkou hardwarovou režie (všechny běží od jednoho hardwarového časovače) a aplikace může spustit libovolný počet softwarových časovačů.Softwarový časovač je popsán *SYS_Timer_t* strukturou. Pole této struktury je popsáno v tabulce 3.1. A jak by mělo vypadat použití časovače je na výpisu kódu 3.3.

Tabulka 3.1: Parametry softwarového časovače.

Název	Popis
interval	Časový interval v milisekundách
mód	SYS_TIMER_INTERVAL_MODE - handler bude volán jednou v nastaveném intervalu SYS_TIMER_PERIODIC_MODE - handler bude volán pořád v intervalu, dokudnebude zastaven aplikací
handler (obslužná rutina)	obslužná funkce, měla by vypadat: „ void handler(SYS_Timer_t *timer)”

Softwarový časovač API je zastoupen následujícími funkcemi:

•*SYS_TimerStart ()* - spustí časovač

•*SYS_TimerStop ()* - zastaví časovač

•*SYS_TimerStarted ()* - zkontrolujte, zda je spuštěn časovač

Vlastní úkol: Nyní je vaším úkolem za použití časovače, rozblikat LED diody každé 2s. Pozn. Nezapomeňte v hlavním programu na spuštění časovače!

Výpis kódu 3.3:Použití časovače.

```
static SYS_Timer_t appTimer;
static void appTimerHandler(SYS_Timer_t *timer)
{
  // handler časovače

}
static void startTimer(void)
{
  appTimer.interval = 1000;
  appTimer.mode = SYS_TIMER_PERIODIC_MODE;
  appTimer.handler = appTimerHandler;
  SYS_TimerStart(&appTimer);
}
```

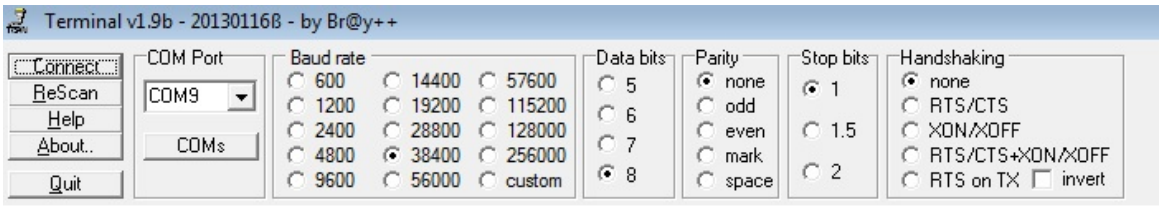
4 DRUHÁ LABORATORNÍ ÚLOHA - POUŽITÍ USART

Pro používání USART (Universal Synchronous / Asynchronous Receiver and Transmitter)sběrnice v API LWM je uživatelsky velmi přívětivé a stačí jen pár příkazů pro nastavení, která slouží pro komunikaci mezi mikroprocesorem a PC. Otevřete si projekt v adresáři:*LWM\apps\USART_KOMUNIKACE\astudio\ZigBit_ATmega1281_Rf230.*Vše důležité je nastaveno include souboru *"halUart.h"* a uživatel pouze nastavuje přenosovou rychlost (Baud rate), to se provádí v hlavní části programu příkazem*HAL_UartInit()* . K posílání bajtu přes USART slouží přikaz*HAL_UartWriteByte().*Vyz. výpis kódu 2.4:

Výpis kódu 4.1: USART_Komunikace.c

```
    int main(void)
    {
      SYS_Init();// systémová inicializace
      HAL_UartInit(38400);//     inicializace     UART
(přenosová rychlost)
      HAL_UartWriteByte('k');     //  poslání  jednoho
bajtu
      while (1)
      {
        SYS_TaskHandler();
       HAL_UartTaskHandler();      // obslužná rutina
pro UART
       APP_TaskHandler();
      }
    }
```

Následně musíme provést stejné nastavení na terminálu viz. Obrázek 4.1.

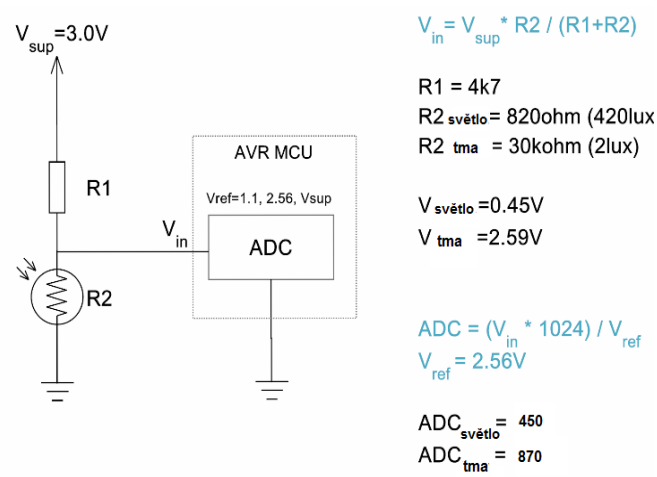


Obrázek 4.1: Nastavení parametrů přenosu v terminálu.

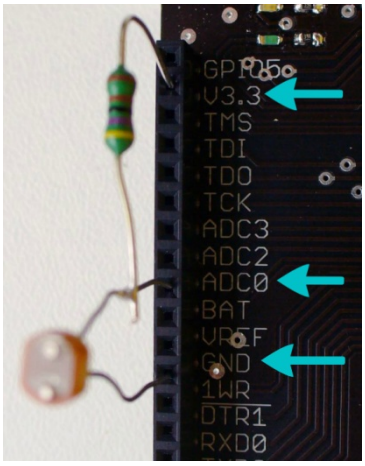
Vlastní úkol: Nyní vyzkoušejte posílání znaku do PC, následně pak program rozšiřte o funkci která se bude jmenovat *appSendMessage()* a která bude sloužit k přenosu řetězce přes USART. Např. pokud zavoláte*appSendMessage("Ahoj");*na terminálu bude stejný řetězec. PS: Realizujte pomocí cyklu *for*a délku řetězce zjistíte pomocí funkce *strlen().*

4.1 POUŽITÍ AD PŘEVODNÍKU

Procesor ATmega1281 disponuje až 8 ADC (analog-digital converter) kanály s rozlišením 10 bitů, bohužel API LWM nedisponuje žádnou speciální knihovnou pro práci s ADC. Takže musíte použít stejnou konfiguraci jako v tutoriálu 3: Viz.Obrázek 4.2.(převzato z[3]). Musíte nastavit všechny potřebné registry a to jsou: *ADMUX, ADCSRB, ADCSRA, ADCL a ADCH.*Zapojení přípravku s fotrezistorem je na obrázku 4.3.



Obrázek 4.2: Zapojení fotorezistoru. (převzato z [3])



Obrázek 4.3: Zapojení fotorezistoru. [3]

Vlastní úkol: Otevřete si projekt v adresáři: *LWM\apps\ADC_USART\ astudio\ZigBit_ATmega1281_Rf230.* Kde již máte potřebné nastavení ADC, Vaším úkole, je nyní s použitím předchozích znalostí upravit program tak, aby posílal každé 2s hodnotu osvětlení přes USART do PC.

POZN. Jelikož je výsledná hodnota 10 – bitová a je to číslo a my potřebujeme přenést řetězec. Je nutné převést tuto hodnotu pomocí funkce *itoa(cislo,retezec*znaku,10)*poslední číslo určuje zda chce dekadické číslo (10) nebo binární (2).

5 TŘETÍ LABORATORNÍ ÚLOHA - BEZDRÁTOVÝ PŘENOS

Pro funkční odesílání a příjem dat je potřeba nastavit několik parametrů, většina se provádí v konfiguračním souboru *confih.h.*

- Síťová adresa - uzlu, která se nastavuje pomocí *NWK_SetAddr ()* – parametr*addr*nemůže mít hodnotu*0xfff,* která je vyhrazena pro broadcast (př: *NWK_SetAddr(0x0001)*).

- Identifikátor sítě- (PAN ID) Příkazem*NWKSetPanId()*, opět nemůže být použita hodnota *0xffff.*

- Frekvenční kanál–uzlu se nastavuje příkazem *PHY_SetChannel(),*rozsah hodnot kanálů pro pásmo 2,4 GHz je 11 – 26 (*0x0b-0x1a*). Pro sub-GHz radia je tento parametr kanálu z rozsahu 0 - 10 (*0x00 – 0x0a*).

- Vysílací výkon – nastavení vysílacího výkonu procesoru. *PHY_SetTxPower(0x00);*Výkonový rozsah:(-17,2dBm) -3 dBm, odpovídá hodotám 0 - 15.

- Bezpečnostní klíč – se nastavuje příkazem *NWK_SetSecurityKey()*. Velikost bezpečnostního klíče je 16 bajtů. Př. nastavení (*NWK_SetSecurityKey ((uint8_t *) "Security12345678");*)

- Aplikační koncové body tzv. endpointy–v případě příjmu dat a následné potřeby zpětné komunikace s uzlem, se registruje jeho end-point. End-poity umožňují na jednom modulu provozovat více služeb. (End-point identifikátor by neměl být větší než 16).

Př. registrace end-pointu: Výpis kódu 5.1.

```
static bool appDataInd (NWK_DataInd_t * ind)
{
/ / přiajatý rámec
return true;
}
NWK_OpenEndpoint (1, appDataInd); // uložení end-pointu
```

5.1 ODESÍLÁNÍ DAT

Aby bylo možné provést přenos dat, aplikace musí nejprve vytvořit požadavek *NWK_DataReq_t* typu, který specifikuje přenášená data, jejich velikost a nastaví parametry přenosu a definuje callbackfunkci (volána pokaždé), která bude informovat o výsledku přenosu. Funkce *NWK_DataReq()* se používá pro odesílání dat a může být volána pouze jednou pro jednu *NWK_DataReq_t*strukturu, dokud není potvrzen její callback (př. potvrzení odeslání dat). Seznam parametrů strukturyjev tab.5.1

Tabulka 5.1: NWK_DataReq() struktura.

Název	Popis
dstAddr	Síťová adresa cílového zařízení
dstEndpoint	Endpoint číslo na cílovém zařízení
srcEndpoint	Lokální end-point
options	Požadavek možností dat, jakákoliv kombinace konstant uvedených v tabulce:5.2.
data	Ukazatel na přenášená data
size	Velikost přenášených dat
confirm	Ukazatel na potvrzení zpětného volání. Měl by mít následující formát:"Void potvrzení (NWK_DataReq_t * req)"
status	Toto pole je vyplněno stackem (doplní LWM automaticky)
control	Toto pole je vyplněnostackem (doplní LWM automaticky) a přístupné z potvrzovacího callbacku.

Tabulka 5.2:Požadavek options.

Název	Popis
NWK_OPT_ACK_REQUEST	Žádost o potvrzení
NWK_OPT_ENABLE_SECURITY	Šifrování přenášených dat
NWK_OPT_BROADCAST_PAN_ID	Nastavení cílové PAN ID pro broadcast 0xffff
NWK_OPT_LINK_LOCAL	Nastavit Link v rámci kontrolního polele na 1
NWK_OPT_MULTICAST	Nastavit Multicast pole a poslat zprávu do skupiny označeného dstAddr

Př. výpisu kódu 5.2. pro odeslání dat.

```
static uint8_t message;
static NWK_DataReq_t nwkDataReq;
static void appDataConf(NWK_DataReq_t *req)
{
if (NWK_SUCCESS_STATUS == req->status)
// frame was sent successfully else
// some error happened
```

```
}
static void sendFrame(void)
{
nwkDataReq.dstAddr = 0;
nwkDataReq.dstEndpoint = 1;
nwkDataReq.srcEndpoint = 5;
nwkDataReq.options = NWK_OPT_ACK_REQUEST | NWK_OPT_ENABLE_SECURITY;
nwkDataReq.data = &message;
nwkDataReq.size = sizeof(message);
nwkDataReq.confirm = appDataConf;
NWK_DataReq(&nwkDataReq);
}
```

5.2 PŘÍJÍMÁNÍ DAT

Pokud aplikace zaznamenáindikacicallback funkce, bude moci přijímat data.Je-li rámec přijímán v zásobníku, následující funkce je:*"boolappDataInd (NWK_DataInd_t * ind)"*. Struktura *NWK_DataInd_t je* popsána v tab. 5.3. Callback musí vracet Boolean hodnotu, která ukáže, zda stack LWM můžeodeslat potvrzovací rámec. Tato funkce umožňuje aplikace kontrolu toku dat (flow kontrol).

Tabulka 5.3: NWK_DataInt_t struktura.

Název	Popis
dstAddr	Síťová adresa cílového zařízení
dstEndpoint	Endpoint číslo na cílovém zařízení
srcEndpoint	Zdrojový endpoit
options	Nastavení možností (př. bezpečnostní klíč, potvrzování…)
data	Ukazatel na přenášená data
size	Velikost přenášených dat
lqi	LQI přijatého rámce
rssi	RSSI přijatého rámce

Výpis kódu 5.3. pro příjem dat.

```
static bool appDataInd(NWK_DataInd_t *ind)
{
if (!appReadyToReceive)
return false;
// process ind->size bytes of the data pointed by ind->data
NWK_SetAckControl(APP_DO_NOT_SLEEP);
return true;
}
```

5.3 NÁVOD PRO SAMOSTATNOU PRÁCI

Vaším úkolem bude napsat program, který po zmáčknutí tlačítka na modulu č.1. rozsvítí LED dioda na modulu č. 2. a obráceně. Na modulech slouží tlačítko pouze pro restart procesoru, tak postupujtenásledovně: Otevřete nový projekt: *LWM\apps\template\astudio\ZigBit_ATmega1281_Rf230* vložte níže uvedený obsah do souboru config.h. Prostudujte významzadaných parametrů a zamyslete se nad nimi, upozorňuji, že pro každýmodul musí být ODLIŠNÁ adresa (pole*: #defineAPP_ADD),* Aby byla možná obousměrná komunikace, nemůžete posílat na stejnou adresu, z kterévysíláte.

Viz výpis kódu 5.4.

```
#ifndef _CONFIG_H_
#define _CONFIG_H_

#define APP_ADDR      1 // pro každý modul musí byt jina adresa !!
#define APP_CHANNEL   0x0f
#define APP_PANID     0x4567
#define APP_ENDPOINT 1

#define NWK_BUFFERS_AMOUNT  3
#define NWK_MAX_ENDPOINTS_AMOUNT   3
#define NWK_DUPLICATE_REJECTION_TABLE_SIZE     10
#define NWK_DUPLICATE_REJECTION_TTL      3000 //ms
#define NWK_ROUTE_TABLE_SIZE100
#define NWK_ROUTE_DEFAULT_SCORE    3
#define NWK_ACK_WAIT_TIME   1000  //ms
#define NWK_ENABLE_ROUTING

#endif // _CONFIG_H_
```

Program vyviňte tak, že bude testovat virtuální tlačítko př: port E,pin 0 pro lepší pochopení co by dělalo normální tlačítko na modulu. Nyní je potřeba nadefinovat potřebné proměnné a struktury: Výpis kódu 5.5.

```
HAL_GPIO_PIN (LED,G,0);     //naše zvolená dioda
HAL_GPIO_PIN (BUTTON,E,0);  //virtuální tlačítko
// Definice zde:
typedef enum AppState_t
{
  APP_STATE_INITIAL,
  APP_STATE_IDLE,
  APP_STATE_WAIT_CONF,
} AppState_t;

typedef struct AppMessage_t
{
  uint8_t buttonState;
} AppMessage_t;

static AppState_t appState = APP_STATE_INITIAL;
static AppMessage_t AppMessage;
static NWK_DataReq_t appNwkDataReg;
```

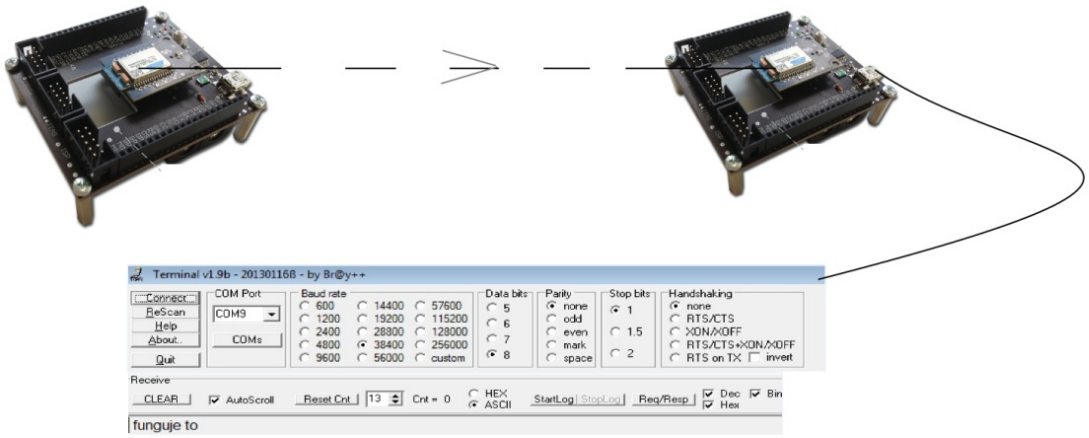
```
    static bool appButtonState = false;

    static void appDataConf (NWK_DataReq_t*req) // funkce pro potvrzeni prenosu
    {
      appState = APP_STATE_IDLE;
      (void) req;
    }
```

Nyní je potřeba vytvořit funkci *appSendMessage(uint8_t state)*která bude obsahovat všechny potřebné informace pro přenos, s kterými jste byli seznámeni v úvodu.Program vytvořte aby se dal použít na obou modulech beze změn (kromě adresy) usnadníte si práci.

Přenos dat

Dalším krokem bude rozšířit program o časovač a USART, tak že každých 5s budou přenese data z modulu č.1. do modulu č.2 a zobrazí se na terminálu v PC. Pro lepší představu třeba řetětec "funguje to" a u něj zobrazte hodnotu vysílacího výkonu RSSI, která je indikována u každého přijatého rámce.



Obrázek 5.1:Schéma zapojení

Použití USARTu jste se naučili v předchozích úlohách a časovače také. Tentokrát udělejte dva nezávislé programy, protože vysílací modul kde musí běžet časovač, nepotřebuje znát třeba USART komunikaci, což přijímací modul ano, práce bude přehlednější. Na vysílacímmodulu musítejen vytvořit správně strukturu dat pro jejich odeslání, příklad můžete vidět na výpisu z kódu 5.6.

```
    static uint8_t* priraddata (uint8_t*msg) // funkce vytvoří místo v paměti pro
zadaný řetězec
    {
      uint8_t*dat=malloc(strlen(msg)*sizeof(uint8_t));//podlevelikosti řetězce
      for (int i=0; i<strlen(msg);i++)
      {
            dat[i]=msg[i];
```

```
    }
    return dat;  // vrací proměnou dat
}
static void appSendMessage(uint8_t state)
{
    uint8_t*dat=priraddata("FUNGUJE TO ");
    AppMessage.buttonState = state;
    appNwkDataReg.dstAddr = 1 - APP_ADDR;
    appNwkDataReg.dstEndpoint = APP_ENDPOINT;
    appNwkDataReg.srcEndpoint =  APP_ENDPOINT;
    appNwkDataReg.options = NWK_OPT_ACK_REQUEST;
    appNwkDataReg.data = dat;
    appNwkDataReg.size = (strlen(dat)-1);
    appNwkDataReg.confirm = appDataConf;

    NWK_DataReq(&appNwkDataReg);
    appState = APP_STATE_WAIT_CONF;
    }
```

Funkce *priradata*na-alokuje (rezervace paměti programu za jeho běhu)místo pro data které chceme poslat, podle jejich délky.V calbacku časovače (TimerHandler) jen stačí pak volat: *appSendMessage(appButtonState);*A vše by mělo fungovat. Pokud by případně žádná data neposílala, problém může být že se neprovede inicializace před samotným odesíláním. Problém vyřešíte nastavením inicializace při zapnutí procesoru. (*APP_STATE_INITIAL).* U každého přijatého rámce je možné zobrazit vysílací výkon RSSI a indikátor kvalitu LQI. Jelikož RSSI nabývat hodnot 0-28 které jsou v dolních pět bitech registru a my čteme celý bajt, je potřeba tuto hodnotu vynásobit hodnotou dekadicky 31 (binárně 00011111) aby byl výsledek správný.

Na přijímači je nutné nastavit akorát komunikaci s PC a v momentě přijetí dat, je poslat do PC. Výpis kódu 5.7.

```
static bool appDataInd (NWK_DataInd_t*ind)
{ char str[2];
    AppMessage_t*msg= (AppMessage_t*)ind->data;

    if (msg->buttonState)
    {
    HAL_GPIO_LED_set();
    appSendMess(msg);  //funkce pro poslání přijaté zprávy do pc (msg)
    itoa(ind->rssi & (31),str,10); //převední výkonu na požadovanou hodnotu
    appSendMess(str);  //odesláni RSSI do pc
    }
          else
    {
    HAL_GPIO_LED_clr();

    }
    return true;
}
```

Opět pozor aby byla rozdílná adresa, jinak soubor *config.h*může zůstat stejný jako v předešlé úloze. Při každé změně, musíte projekt zkompilovat, aby se projevila.

5.4 SENZOR OSVĚTLENÍ

Pokud jste zvládli předešlé body, poslední částí je rozšíření stávajícího programu o přenos dat z a fotorezistoru. S použitím AD převodníku, který už také umíte používat. Modul č.1 bude přenášet každé 4s hodnotu osvětlení, bezdrátově do druhéhomodulu a do PC. Plus na přijímači bude svítit dioda při nezakrytém senzoru a když senzor zakryjeme a hodnota se změní pod Vámi nastavenou mez, dioda zhasne. Na vysílači musíte provést kompletní inicializaci AD převodníku. Nezapomeňte nadeklarovat globální proměnou *adcvalue*typu *uint16_t*. V odesílací části pak už jenom zavoláte změření hodnoty a po převedení ji pošlete. Odesílání budete volat z handleru časovače. Měření hodnotu můžete volat přímo v odesílací části, nebo lépe ve zvláštní funkci. Zde je ukázka první verze.

Odesílání hodnoty fotorezistoru do další desky. Výpis kódu 5.8.

```
static void appSendMessage(uint8_t state)
{
  char str[10];
  adcvalue =Read_ADC(0);    //načtení hodnoty
  itoa(adcvalue,str,10);    //převod na požadovaná tvar
  uint8_t*dat=priraddata(str);
  AppMessage.buttonState = state;

  appNwkDataReg.dstAddr = 1 - APP_ADDR;
  appNwkDataReg.dstEndpoint = APP_ENDPOINT;
  appNwkDataReg.srcEndpoint =  APP_ENDPOINT;
  appNwkDataReg.options = NWK_OPT_ACK_REQUEST;
  appNwkDataReg.data = dat;
  appNwkDataReg.size = (strlen(dat)-1);
  appNwkDataReg.confirm = appDataConf;

  NWK_DataReq(&appNwkDataReg);
  appState = APP_STATE_WAIT_CONF;

}
```

Na modulu č.2. musíte akorát nastavit podmínku podle které bude svítit, nebo nesvítit LED dioda. Výpis kódu 5.9.

```
static bool appDataInd (NWK_DataInd_t*ind)
{
  AppMessage_t*msg= (AppMessage_t*)ind->data;
  if (msg->buttonState)
  {
  HAL_GPIO_LED_set();
  appSendMess(msg);
  if (strcmp("600", msg) > 0) //porovnání dvou řetězců
  {
        appSendMess("svetlo");
        HAL_LedOff(2);
        HAL_LedToggle(0);
  }
```

```
      else {
            appSendMess("TMA");
            HAL_LedOn(2);
            }
      }
      else
      {
      HAL_GPIO_LED_clr();
      }
      return true;
    }
```

6 ČTVRTÁ LABORATORNÍ ÚLOHA - AUTOMATIZOVANÉ MĚŘENÍ

Další náplní práce bude vývoj firmware pro automatizované měření propustnosti a ztrátovosti mezi koordinátorem a bezdrátovými senzory. Vytvořte firmware jak pro koordinátora, tak pro koncové zařízení, pomocí kterého bude možno změřit propustnost a ztrátovost. Princip měření následující:

Koordinátor vyšle START paket pomocí, kterého spustí vysílání na druhém uzlu End Device 1 (ED1). ED1 zapne časovač a začne vysílat v pravidelném intervalu pakety o definované velikosti. Počet paketů, která má ED1 odeslat stanovme zatím na 1000.

Měření propustnosti bude probíhat následovně: Bude se měřit za jak dlouho je ED1 schopno odeslat 1000 paketů. Tj. spustí časovač, pošle první paket, a jakmile mu je potvrzen (Confirmation) posílá další paket. Jakmile ED1 dostal 1000 potvrzení je časovač zastaven a provede se výpočet doby trvání měření. Ze znalosti doby trvání a počtu odeslaných bitů můžeme stanovit propustnost. Tento test by se dal dělat i tak, že se nastaví časovač na konstantní dobu cca 30 sekund a po jeho spuštění se začnou posílat pakety. Propustnost se potom vypočítá jako počet správně přenesených paketů v bitech/ 30.

Měření ztrátovosti bude probíhat stejně. Dle nastaveného intervalu bude určen počet odeslaných paketů a počet přijatých potvrzení. Na základě poměru ACK/Počet Paketů víme, kolik přenosů se muselo opakovat a tak známe ztrátovost. ED vypočítá jak propustnost, tak ztrátovost a pošle výsledek zpět koordinátorovi, který ho zobrazí v konzoli.

Z každého přijatého paketu se bude navíc na koordinátorovi indikovat RSSI a LQI a na konci testu se určí na koordinátorovi průměr RSSI a LQI za celou dobu měření pro toto ED

RSSI - (Received Signal Strength Indication) hodnota vysílacího výkonu je 5-bitová hodnota indikující přijatý výkon na kanále v krocích 3 dB. Hodnota RSSI, která nabývá rozsahu 0-28. Síla přijatého signálu je vypočítána na základě hodnoty RSSI dle následujícího vztahu.

$$PRF = RSSI\_BASE\_VAL + 3 \times (RSSI - 1) \quad (1)$$

Parametr RSSI_BASE_VAL je roven -91 dBm, hodnota RSSI = 0 odpovídá síle signálu -91 dBm, hodnota RSSI = 28 odpovídá síle signálu > -10 dBm.

LQI- (Link Quality Indication) indikátor kvality služeb. Parametr LQI nabývá hodnota v rozsahu 0-255, přičemž hodnota 0 odpovídá nejmenší a hodnota 255 nejvyšší kvalitě dat.

6.1 POSTUP PRO SAMOSTATNOU PRÁCI

Pro požadované zadání můžete postupovat více způsoby k dosáhnutí výsledků. Další popis slouží jako jedno z možných řešení. Vytvořte si dva odlišné projekty jeden pro ED a druhý pro koordinátora, každý modul musí mít samozřejmě odlišnou adresu. Nejprve je potřeba přenést data z terminálu do koordinátora, že má vyslat na zadané adresy START paket. Pro detekování přijatých dat je v LWM funkce *UartBytesReceived()*. Která je volána pokaždé když přijdou do procesoru data po USARTu. Viz výpis kódu 6.1:

```
static uint8_t appDataReqBuffer[APP_BUFFER_SIZE];
static uint8_t appUartBuffer[APP_BUFFER_SIZE];
static uint8_t appUartBufferPtr = 0;

void HAL_UartBytesReceived(uint16_t bytes)     //funkce pro příjem dat
{
  for (uint16_t i = 0; i < bytes; i++)   //čtení přiajtých dat bajt po bajtu
  {
    uint8_t byte = HAL_UartReadByte();

    if (appUartBufferPtr == sizeof(appUartBuffer))
      appSendData();        //volání funkce pro přenos dat
   if (appUartBufferPtr < sizeof(appUartBuffer))
      appUartBuffer[appUartBufferPtr++] = byte;       //naplnění dat
  }
}
```

Přijatá data pak odešleme na požadovanou adresu ED1 funkcí appSendData().

Viz výpis kódu 6.2:

```
static void appSendData(void)
{
  if (appDataReqBusy || 0 == appUartBufferPtr)
    return;

  memcpy(appDataReqBuffer, appUartBuffer, appUartBufferPtr); //zkopírování dat
```

```
  appDataReq.dstAddr = 1-APP_ADDR;              //nastavení parametrů přenosu
  appDataReq.dstEndpoint = APP_ENDPOINT;
  appDataReq.srcEndpoint = APP_ENDPOINT;
  appDataReq.options = NWK_OPT_ACK_REQUEST;
  appDataReq.data = appDataReqBuffer;
  appDataReq.size = appUartBufferPtr;
  appDataReq.confirm = appDataConf;
  NWK_DataReq(&appDataReq);
  appUartBufferPtr = 0;
  appDataReqBusy = true;
}
```

Program modifikujte, aby přijatá data z terminálu, př. řetězec "start1" ("start123"). Aby uvedená čísla značili adresy koncových zařízení, s kterými bude probíhat měření, ušetříte si tím práci dále.

Na ED1 je potřeba, po přijetí konkrétního znaku, řetězce, čísla, čehokoliv co zvolíte. Nastavit potřebné proměné. Nyní je více možností postupu jak začít měřit propustnost (dva časovače, měření času a počtupaketů).

Př.přijetí START paketu. Výpis kódu 6.3.

```
static bool appDataInd(NWK_DataInd_t *ind)
{
  if (ind->data[0]=='s' && !measure)    //porovnání zda přišel znak ‘s‘
        {
              startTimer();             //spuštění časovače
              measure=true;             //nastavení logických proměných
              stop=true;
              timercounter=0;           //vynulování proměných
              ack=0;
              appSendData("s");         //poslání zpet koordinatorovo ‘s‘
        }
  return true;
}
```

Po přijetí START paketu se spustí časovač na Vámi nastavenou hodnotu, v tomto případě na 100 ms a nastaví logickou proměnou measure na true. Dokud je tato proměna *true* budou se odesílat data. Každých 100 ms se inkrementuje proměná *timercounter* která se bude inkrementovat, dokud nebude hodnota rovna 50. Padesátkrát za 100 ms, to je celková doba měření propustnosti po 5s. Délka datové části u standardu 802.15.4 je 127 bajtů. Po odečtení všech povinných částí LWM (viz obrázek 1. 3) dostaneme maximální hodnotu payloadu (přenesených dat) 103 bajtů. Výpočet propustnosti provedete na základě počtu přenesených paketů, velikosti přenesených dat a celkového času. V našem případě je velikost paketu dle následujícího vzorce.

$$\text{počet přenesených paketů x (90B x8)/5 =X bps} \qquad (2)$$

Výpis kódu 6.4:

```
static void appDataConf(NWK_DataReq_t *req) //funkce volána potvrzení přenosu
{
  static bool sendresult=false;
    if (measure)
  {
    citac++;
    appSendData(tdata); //tdata90B pole
    }
  else  if (stop)
  {
    stop=false;             //ukončení měření propustnosti
    sendresult=true;
    appSendData("e");       //odeslán příznak pro ukončení měření
   }
    else if (sendresult)
    {          sendresult=false;
          propustnost=(((citac*90*8)/5)); //vypocet propustnosti
          itoa(propustnost,str,10);
          appSendData(str);
          citac=0;               //vynulovani proměných
          ack=0;
          lost=0;
         }
    if (measure2)ack++ ;
 }
```

Po provedení měření propustnosti nastavte proměnou *measure* na *false* a proměnou *measure2* na true. Dál proveďte měření ztrátovosti obdobným způsobem. Výsledná hodnota je poměr potvrzených paketů a odeslaných paketů, pro výsledek v procentech vynásobte hodnotou 100, viz. vztah:

$$\frac{\text{Počet potvrzených paketů}}{\text{Počet odeslaných paketú}} \times 100 = X\ \% \qquad (3)$$

Výpis kódu obslužná rutina časovače 6.5:

```
static void appTimerHandler(SYS_Timer_t *timer)
{
  if (measure)
  {
        timercounter++;
        if (timercounter == 50)
        {
              measure=false;
              measure2=true;
              timercounter=0;
        }
  }
```

```
      else if (measure2)
      {
            if (timercounter<50)            //měření ztrátovosti
            {     appSendData("");
                  timercounter++;
            }
            else if (sedresult2)

            {     sedresult2=false;
                  measure2=false;
                  lost=((ack)*100)/timercounter;  //vypocet ztrátovosti
                  itoa(lost,str,10);
                  appSendData(str);               //odeslani vysledku
            }
            else
            {     sedresult2=true;
                  appSendData("c");               //konec měření
            }
      }
    }
```

Na koordinátorovy musíte počítat RSSI a LQI, které se indikuje u LWM u každého přijatého paketu. Proto každou přijatou hodnotu RSSI a LQI uložte do zvolené proměné, a výsledná hodnota už je aritmetický průměr hodnot. Všechny přijaté výsledky měření vypište do konzole. Výpis kódu 6.6.

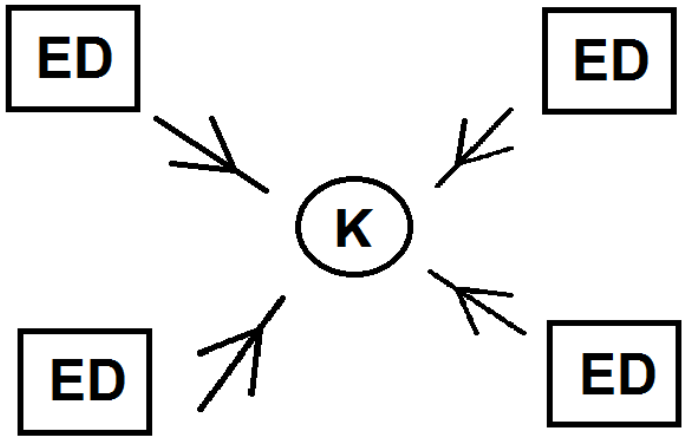
```
    static bool appDataInd(NWK_DataInd_t *ind)
    { citac++;
      rssisum+=(ind->rssi & (31));
      lqisum+=ind->lqi;
      rssiavg=rssisum/citac;
      lqiavg=lqisum/citac;

      if (ind->data[0]=='e' )
            { appSendMess("stop");
                  result=true;
            }
            else if (result)          //výpis všech přijatých hodnot
            {     result=false;
                  citac=0;
                  appSendMess("Propustnost: ");
                  for (uint8_t i = 0; i < ind->size; i++)
                  HAL_UartWriteByte(ind->data[i]);
                  appSendMess(" bps");
                  appSendMess("RSSI-avg: ");
                  itoa(rssiavg,str,10);
                  appSendMess(str);
                  rssiavg=(-91+3*(rssiavg-1));
                  itoa(rssiavg,str,10);
                  appSendMess(str);
                  appSendMess("LQI-avg: ");
                  itoa(lqiavg,str,10);
                  appSendMess(str);
                  lqisum=0;
                  rssisum=0;
```

```
                citac=0;
            }
            else if (ind->data[0]=='c' )
            {     result2=true;
            }
            else if (result2)
            {     result2=false;
                  appSendMess("Uspesne prenesnych paketu:");
                  for (uint8_t i = 0; i < ind->size; i++)
                  HAL_UartWriteByte(ind->data[i]);
                  appSendMess(" %");
            }
            if (ind->data[0]=='s' ) appSendMess("start");
      return true;
```

7 MĚŘENÍ PARAMETRŮ BEZDRÁTOVÉHO PŘENOSU

Na základě vytvořených návodů bylo provedeno měření parametrů bezdrátového přenosu v LWM, které bylo zaměřeno na komunikaci mezi koordinátorem a více koncovými zařízeními. Viz obrázek 7.1. Ztrátovosti přenosu a hodnoty RSSI a LQI na vzájemné vzdálenosti modulů. Pro každé měření bylo naměřeno 10 hodnot a z nich vytvořen aritmetický průměr kvůli přesnějším výsledkům.



Obrázek 7.1: Schéma vysílání více koncových zařízení na jednoho koordinátora.

7.1 MĚŘENÍ PROPUSTNOSTI

Moduly použité v laboratořích obsahují rádiový chip AT86RF230 od firmy Atmel, který pracuje v pásmu 2,4 GHz, maximální teoretická propustnost je 250kb/s. Této hodnoty však v reálných podmínkách zdaleka nedosahuje. V praxi je rychlost o něco nižši z důvodů zapouzdřovani dat do struktur nižších vrstev. Při měření propustnosti mezi dvěma moduly nebylo dosaženo vetší hodnoty jak 35 kb/s.Měření probíhalo při 90% z maximální teoretické velikosti (103 bajtů) přenesených dat pro použitou metriku výpočtů. Velikost přenášených dat byla 90 bajtů, vetší hodnotu se v tomto měření nepodařilo přenést. Důvodem takto nízké hodnoty oproti teoretické, mohl být zapnutý režim potvrzování paketů, který zpomaluje přenos. Výsledky měření ukázaly, že u LWM je nízká přenosová rychlost kompenzována vysokou spolehlivostí, v tomto případě vždy bylo 100% paketů doručeno do cíle.

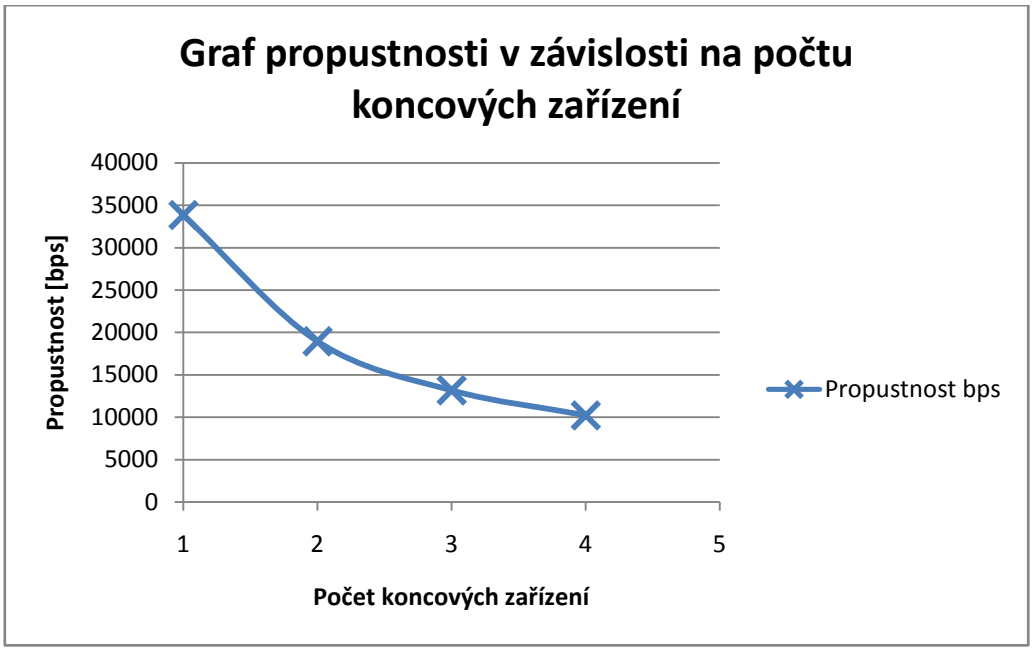
Při tomto měření nebyl zapnut zabezpečený přenos, po zapnutí zabezpečeného přenosu a následném provedeném měření, bylo dosahováno stejných výsledků. Na fyzické vrstvě jsou k rámci MPDU přidány pole SHR (Start of Header), které se skládá z poli Preamble a Start of Frame Delimiter a pole PHR udávající délku rámce. Při uvažovaní přenosové rychlosti 250 kbps lze dobu přenosu ramce s datovou časti MSDU o velikosti 90 bajtů vypočítat jako [4]:

$$\frac{(\text{MPDU} + \text{SHR} + \text{PHR}) \times 8}{250 \times 10^3} = \frac{(127+5+1)\times 8}{250 \times 10^3} = 4{,}256\ ms \tag{4}$$

Velikost potvrzovacích ACK rámců na PHY vrstvě je 11 B (5 bajtů je MPDU). Doba přenosu ACK rámce je pak 0,352 ms. Mezi přijetím rámce a zasláním potvrzeni se vloží časový interval 192 μs z důvodu prodlevy v přepnuti z přijímacího (Rx) do vysílacího modu (Tx). Z vypočtených hodnot lze určit celkovou dobu přenosu jednoho rámce o velikosti 90 bajtů:

$$\begin{aligned}(\text{CSMA-CA}) + (\text{přřenos dat}) + (\text{RX/Tx prodleva}) + (\text{ACK}) = \\ 2{,}368+4{,}256 +0{,}192+0{,}352= 7{,}168\ \text{ms}.\end{aligned} \tag{5}$$

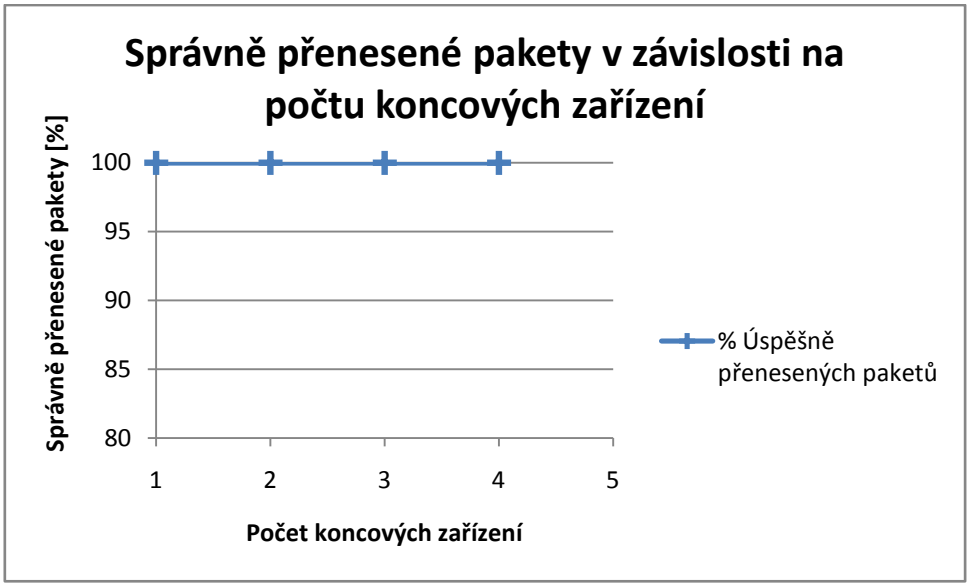
Po výpočtu doby přenosu jednoho paketu z měření, vychází hodnota 20 ms. Což je o 13 ms déle než teoretická vypočtená hodnota, důvod prodlevy je popsán výše. V dalším Obrázku 7.1. je na grafu znázorněna propustnost mezi koordinátorem a více koncovými zařízeními, během současné komunikace. Výsledky závislosti propustnosti na více zařízeních odpovídají očekávání, jelikož propustnost klesá s přibývajícím počtem zařízení komunikujících ve stejný okamžik.



Obrázek 7.1: Závislost propustnosti na počtu koncových zařízení.

7.2 Měření ztrátovosti

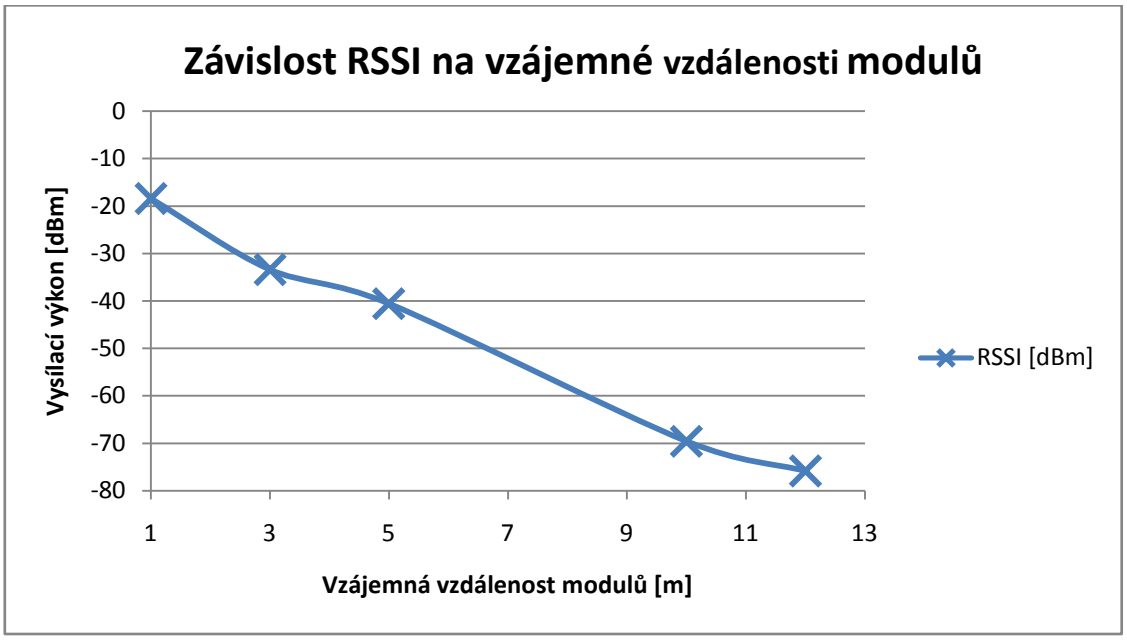
Při měření ztrátovosti nedocházelo k žádným ztrátám paketů během přenosu. U LWM se ukázal přenos paketů jako velmi spolehlivý, když i při maximálním vytížení nedocházelo k žádné ztrátovosti, při menší propustnosti stále byla úspěšnost přenesených paketů 100%. Na obrázku 7. 2. je znázorněna procentuální úspěšnost přenesených paketů.



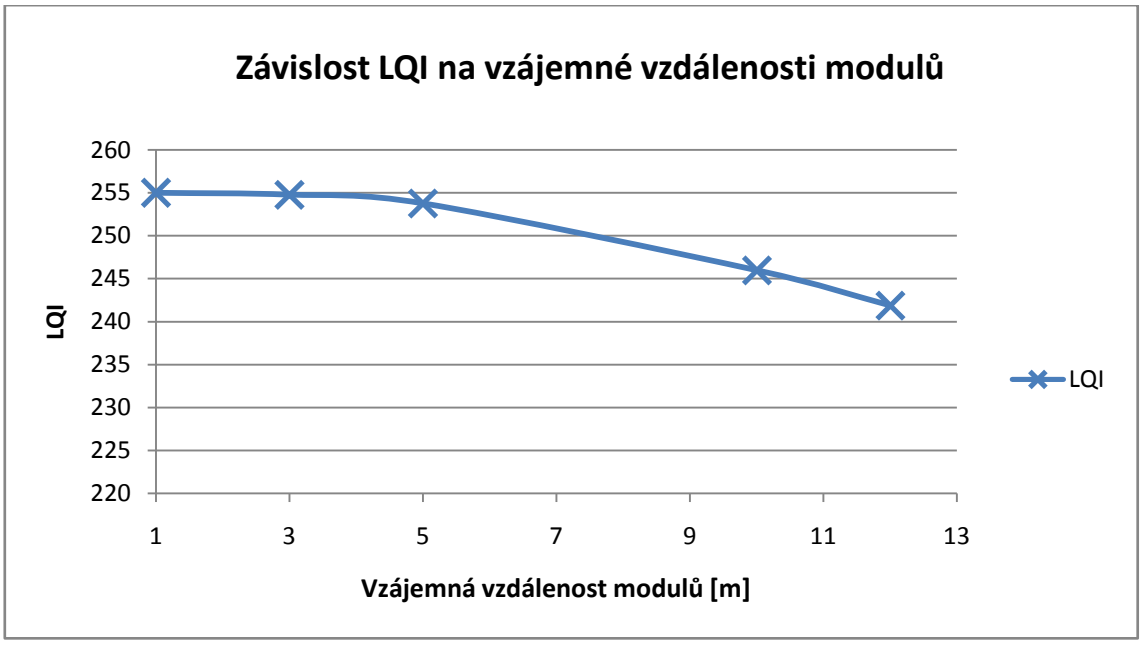
Obrázek 7.2: Závislost správně přenesených paketů na počtu koncových zařízení

7.3 Měření RSSI a LQI

Co je RSSI a LQI už bylo popsáno výše, měření těchto parametrů probíhalo během testování propustnosti vždy mezi dvěma moduly (koordinátor a koncové zařízení). Měření probíhalo v rodinném domě uvnitř budovy. Maximální vzdálenost kdy moduly komunikovaly, byla 12m. Během testování byl přenos maximálně vytížen, z toho důvodu hodnoty RSSI nejsou optimální a odpovídají naměřeným hodnotám a přenos nedosahoval větší vzdálenosti. Nastaven byl maximální vysílací výkon 3,2 dBm. Hodnoty LQI nikdy neklesly pod hodnoty 237, což odpovídá prostředí bez většího rušení. Na Obrázcích 7.3. a 7.4. jsou grafy pro RSSI a LQI.



Obrázek 7.3: Závislost RSSI na vzájemné vzdálenosti modulů



Obrázek 7.4: Závislost LQI na vzájemné vzdálenosti modulů.

8 ZÁVĚR

Cílem bakalářské práce byla implementace nového protokolu Lightweight Mesh na stávající výukovou platformu s procesorem ZigBee a vytvoření laboratorních úloh na konkrétní zadání. Úvod práce je zaměřen na teoretické poznatky o LWM a shrnuje jeho hlavní výhody a principy. Samotná realizace probíhala v Atmel studiu 6.2 a největší zdroj informací byl oficiální vývojový dokument [1], ze kterého bylo hlavně čerpáno. Nevýhodou je málo informací o LWM, ale vývojáři umístili do stacku (balíček) tzv. demo programy aby uživatelé mohli vidět názorné ukázky. Díky těmto programům bylo možné vidět jak LWM funguje a dostat určitou představu o práci sním a ve výsledku bylo z těchto programů vycházeno.

Po vytvoření samotných programů, byla vytvořena dokumentace, která bude sloužit jako návod pro studenty. Z provedených měření byly určeny skutečné parametry přenosu, hodnota propustnost byla mnohem nižší, oproti teoretické hodnotě 250 kb/s. Maximální naměřená hodnota propustnost 35 kb/s byla daleko za očekáváním, důvody takto nízké hodnoty byli popsány výše. Pozitivní výsledky ukázalo měření ztrátovosti, kdy při všech měřeních nedocházelo k žádnému zahazování paketů, ani při zvětšující se vzájemné vzdálenosti modulů. Toto byla také jedna z příčin nižší propustnosti. Měření hodnot vysílacího výkonu a indikátoru kvality služeb (RSSI, LQI) splnilo očekávání, když hodnoty vzájemně korelovaly a se vzrůstající vzdáleností tyto hodnoty dosahovaly horších parametrů. Tímto byly splněny všechny cíle bakalářské práce.

9 SEZNAM POUŽITÉ LITERATURY

[1] Atmel Lightweight Mesh: Developer Guide [online]. 2013 [cit. 2014–40–12]. Dostupné z: http://www.atmel.com/Images/Atmel-42028-Lightweight-Mesh-Developer-Guide_Application-Note_AVR2130.pdf

[2]ATMEL. *Lightweight Mesh* [online]. [cit. 2014-04-15]. Dostupné z: http://www.atmel.com/tools/lightweight_mesh.aspx

[3] Wislab. ŠIMEK, Milan. *TRAINING: HOW TO BUILD "WIRELESS SENSOR NETWORK"*[online]. [cit. 2014-04-15]. Dostupné z: http://wislab.cz/training-how-to-build-wireless-sensor-network

[4] ŠIMEK, Milan. *Bezdrátové senzorové sítě*. BRNO, 2013. 163 s. Skripta. VUT Brno.

10 SEZNAM POUŽITÝCH ZKRATEK A SYMBOLŮ

WSN Wireless Sensor Networks

API Application programming interface

MAC Media Access control

CRC Cyclic redundancy check

SRC Source

DST Destination

LED Light emitting diode

ADC Analog digital converter

LWM Lightweight mesh

11 PŘÍLOHA

Příloha obsahuje elektronickou verzi tohoto dokumentu a všechny vytvořené programy v Atmel Studiu, které budou sloužit studentům jako materiál k práci v laboratoři.