Studijní text

Arduino WSN

Lukáš Němec

Obsah

1 Úvod **2**

2 Arduino všeobecně **2**
2.1 Arduino IDE 2
2.2 Základy syntaxe 3

3 WSN uzel, JeeLib **4**
3.1 JeeLabs hardware 5
3.2 JeeLib 5

4 WSN síť **10**
4.1 Ukázkové aplikace 10
4.1.1 Alive 11
4.1.2 Sniffer 12

1 Úvod

Tento text slouží k úvodnímu seznámení s Arduinem a aplikacemi, které je možné pomocí Arduina vytvořit. V druhé a třetí části je následně zaměření na bezdrátovou komunikaci a vytváření senzorových sítí pomocí Uzlů založených na Arduinu. Veškeré zdrojové kódy lze nalézt buď v repozitáři projektu Edu-hoc `https://github.com/crocs-muni/Edu-hoc`, na stránkách Arduina `www.arduino.cc` nebo na stránkách knihovny JeeLib `http://jeelabs.net/projects/jeelib/wiki`.

2 Arduino všeobecně

Arduino je open-source platforma a zároveň fenomén několika posledních let. Poskytuje jednak samotný hardware a jednak software pro programování mikrokontrolerů ATMega. V současné době už existuje nejenom velké množství oficiálních Arduino desek, ale zároveň se objevuje velké množství jejich klonů. Tyto jsou buď motivovány snahou vytvořit vlastní a levnější variantu svého Arduina, nebo naopak některou z oficiálních desek rozšířit o funkcionalitu navíc. Mezi první kategorii typicky patří čínské klony typu Funduino a další, zatímco druhá kategorie obsahuje například specializované desky, jako jsou námi používané JeeLink a JeeNode USB.

Jelikož se budeme zabývat ad-hoc sítěmi a senzorovými sítěmi, tak z celé platformy využijeme pouze vývojové prostředí, zatímco hardware použijeme specializovaný pro naše potřeby, především už bude obsahovat rádiový modul. Začněme však od úplných základů práce s Arduinem.

2.1 Arduino IDE

Arduino IDE je velmi jednoduché prostředí pro vyvíjení programů, z praktického hlediska se jedná spíše o upravený textový editor s podporou pro Arduino. Výhodou je podpora pro většinu běžných operačních systémů, tedy je možné vyvíjet jak pod linuxem, tak windows či OS X. Může fungovat buď bez instalace, nicméně tato varianta je vhodná pouze pro samotné programovaní v případě, že pouze potřebujete programovat a komunikace s Arduinem samotným Vás nezajímá. Pokud Vás však například zajímá komunikace s některou z desek přes sériové rozhraní, vyplatí se prostředí nainstalovat.

Prostředí můžete využít i pokud se rozhodnete programovat ve svém oblíbeném textovém editoru a v Arduino IDE pouze kompilovat, či používat jiné nástroje které poskytuje.

Instalace Instalace samotná není složitý proces, spíše naopak. Spousta linuxových distribucí Vám práci usnadní, jelikož Arduino IDE se nachází v repositářích a jediné, co je potřeba pohlídat, je verze, která by měla být ideálně nejnovější dostupná, ideálně verze 1.5 a novější.

V případě OS Windows je nejednodušším řešením stáhnout instalační soubor přímo z oficiálních stránek `http://arduino.cc/en/Main/Software`, kde naleznete jak instalátor,

tak archiv pro případ, když nemáte administrátorské oprávnění na počítači, kde plánujete pracovat.

Pokud z nejrůznějších důvodů potřebujete instalovat jiným způsobem (například kompilovat ze zdrojového kódu), pak veškeré potřebné informace opět naleznete na oficiálních stránkách `http://arduino.cc/en/Main/Software`.

Nástroje

Verifikátor Z nástrojů, které prostředí nabízí je tím nejdůležitějším verifikátor, který slouží k ověření syntaxe napsaného kódu. Tuto možnost bohužel prostředí nenabízí průběžně, tedy je vhodné relativně pravidelně syntaxi kontrolovat.

Kompilátor Druhým podstatným nástrojem je kompilátor spojený s nahráváním na desku. Nejdříve je potřeba zvolit konkrétní model Arduina, případně model nejvíce podobný pro veškeré odvozené modely a klony. Dále je třeba zkontrolovat nastavení portu, na který je konkrétní deska připojena. V případě, že je připojena pouze jedna většinou prostředí port vybere správně, nicméně kontrola je vhodná. Zároveň je možné mít připojeno více desek a poté je potřeba zvolit jednu konkrétní.

V případě problémů s nalezením desky je vhodné restartovat celé prostředí, případně desku připojovat před spuštěním prostředí.

Serial monitor Serial monitor je posledním důležitým nástrojem, který je vhodné znát. Jeho funkcionalitu sice lze nahradit pomocí jakéhokoliv programu, který umožňuje komunikovat přes sériový port (například PySerial), na druhou stranu je Serial Monitor tou nejednoduší variantou.

Při spuštění je třeba nastavit frekvenci komunikace (na stejnou hodnotu, jako ve zdrojovém kódu programu) a následně je možné číst informace zaslané z desky, nebo případně desce posílat instrukce.

Pokud ze zobrazují znaky, které však nedávají smysl, bude pravděpodobně problém s různě nastavenou frekvencí desky a Serial Monitoru.

2.2 Základy syntaxe

Arduino je možné brát jako speciální případ jazyka C. Zůstává téměř vše a mění se pouze podoba funkce main, která je rozdělená na dvě části, `setup` a `loop`. Tedy je možné psát for cykly jako v jazyce C, stejně tak podmínky, funkce a téměř cokoliv dalšího, co budete potřebovat. Navíc Arduino přidává některé vlastní funkce a velké množství knihoven, které především slouží k příjemnější manipulaci s veškerým možným příslušenstvím. Nicméně můžeme nalézt i různé jiné knihovny, které především usnadňují psaní kódu v jazyce C.

`setup` Funkce sloužící k úvodnímu nastavení proměnných, počátečnímu spuštění a všem dalším úvodním nastavením. Při samotném spuštění se tato funkce zavolá jako první a právě jednou. Ideální je například ke spuštění komunikace po sériovém portu, nastavení pinů a inicializaci knihoven.

`loop` Funkce která provádí samotný běh programu, jak napovídá její název, tuto funkci můžeme vnímat jako nekonečný cyklus. Zavolá se až po funkci `setup`, tedy již můžeme počítat s inicializovaným prostředím a bude se volat neustále až do restartu desky nebo ztráty napájení. Zdrojový kód v této funkci je vhodné psát s ohledem na neustálé opakování a zároveň v případě, že potřebujeme něco pravidelně kontrolovat (stisknutí tlačítka) je vhodné mít buď celý kód dostatečně rychlý tak, aby kontrola probíhala v krátkých intervalech, nebo kontrolu provést několikrát za jeden průběh cyklu.[1]

Serial Jedna z nejdůležitějších knihoven slouží k ovládání sériového portu na straně desky. Na začátku je třeba ve funkci `setup` inicializovat společně s nastavením frekvence komunikace. Následně je možné jej začít používat, klidně již ve funkci `setup`. Všechny funkce této knihovny je třeba volat vždy jako `Serial.read()` nikoliv pouze `read()`.

`print` Slouží k výpisu hodnot a ladících informací přes sériový port. Je možné tisknout hodnotu samotnou, případně ji i formátovat (vhodné například pro hexadecimální hodnoty). Taktéž je možné použít funkci `println`, která navíc pošle i ukončení řádku. Při používání je třeba dát pozor, jelikož funkce je asynchronní, tedy vrátí návratovou hodnotu dříve, než začne odesílat znaky.

Pro obsluhu výstupu na sériovém portu je možné využít i funkce `write` (zápis binárních dat) nebo `flush` (počká na dokončení zápisu).

`read` Slouží ke čtení ze sériového portu, přečtenou hodnotu vrací jako návratovou hodnotu. Vhodné je její spojení s funkcí `available`, která vrací počet bytů dostupných k přečtení. Ideální použití je buď k ovládání desky pomocí příkazů z počítače, nebo k počátečnímu nastavení, které není možné nastavit v kódu napevno.

3 WSN uzel, JeeLib

Pokud odhlédneme od obecně zaměřeného Arduina a zaměříme se na bezdrátovou komunikaci, pak můžeme buď k běžné desce připojit ten správný modul a tím vytvořit jeden uzel bezdrátové sítě, nebo můžeme vybrat klon arduina, který je přímo tímto směrem zaměřený. Zajímá nás především přítomnost rádia a oproti ostatním aplikacím nepotřebujeme velké množství pinů pro další přídavné komponenty.

[1] některé desky umožňují využít speciální piny k přerušení funkce `loop` a okamžité reakci na událost. K přerušení je možné připojit k speciální vlastní funkci, která vykoná jeho obsluhu

Samotný uzel nemá příliš velké uplatnění, jeho síla přichází až při větších počtech, nicméně vyplatí se nejdříve seznámit se s aplikacemi pro jeden či dva uzly a pokročile varianty s výrazně větším množstvím uzlů nechat na později. S jedním uzlem je možné vyzkoušet veškerou komunikaci s počítačem a využití funkcí z běžných Arduino knihoven, jako například komunikace po sériovém portu pro počáteční nastavení proměnných.

S dvěma uzly je poté již možné začít komunikovat přes rádiové spojení, vyzkoušet jednoduché odeslání, přijetí zprávy a další základní funkce, které nabízí knihovna JeeLib.

3.1 JeeLabs hardware

JeeNode USB a JeeLib jsou klonem Arduina, navíc však obsahují rádiový modul. Oproti nejvíce rozšířeným Arduino deskám nemají napětí 5V ale pouze 3.3V a komponenty jsou na desce rozloženy odlišně, tedy většina běžného příslušenství k Arduinu není s těmito deskami kompatibilní. JeeLink je navíc chráněn plastovým krytem, tedy není možné k němu cokoliv připojit.

Druhou variantou je JeeLink, který obsahuje prakticky totožný hardware, nicméně je celý uzavřený v plastovém krytu a neumožňuje přístup k pinům. Taktéž se liší použitým typem USB konektoru, jelikož JeeNode USB obsahuje mini USB, zatímco JeeLink má standardní USB A konektor, který můžeme připojit přímo do počítače.

Obě varianty, jak JeeLink, tak JeeNode USB jsou založené na Arduino mini s procesorem ATmega328P.[2] JeeLink navíc obsahuje 16 Mbit flash paměti, kterou lze využít pro uložení dat aplikace.

Rádio, které je na desce přítomné dokáže komunikovat na třech různých frekvencích, konkrétně 433, 868, nebo 915 MHz. Frekvence je především důležitá pro správnou kalibraci antény, v případě běžného využití se nejedná o podstatný problém, ale při snaze získat co největší vysílací výkon a dosah je třeba mít délku antény a frekvenci rádia nastavenou ve vzájemné shodě.JeeLink má frekvenci nastavenou obvykle již z výroby a je označena barvou na čipu rádia. Žlutá barva znamená 868 MHz, zatímco zelená 433 MHz. [3]

Piny Jsou kontakty, které umožňují připojení dalších komponent, ať už se jedná o obyčejnou LED diodu, nebo světelný, či gravitační senzor. Možnost připojení komponent nabízí pouze JeeNode USB.

3.2 JeeLib

JeeLib je Arduino knihovna napsaná přímo pro uzly JeeLink, JeeNode USB a další kompatibilní zařízení. Umožňuje ovládání rádia a některých přídavných modulů. Více informací o této knihovně je možné zjistit přímo ze stránek projektu `http://jeelabs.net/projects/jeelib/wiki` nebo přímo z dokumentace knihovny, kterou je možné získat více

[2] tuto desku je třeba nastavit v Arduino IDE při programování

[3] HW použitý při stavbě testbedu má frekvenci nastavenou na 868 MHz

způsoby, buď vygenerovat vlastní verzi přímo pro Vámi používanou verzi knihovny pomocí nástroje Doxygen, nebo použít oficiální dokumentaci na stránkách JeeLabs `http://jeelabs.net/pub/docs/jeelib/`.

Z obsahu JeeLib je pro nás nejdůležitější komunikace s rádiem, kterou nalezneme pod ovladačem RF12 `http://jeelabs.net/pub/docs/jeelib/md_intro_rf12.html`. Tento obsahuje vše potřebné, od počátečního nastavení uzlu, přes odesílání a příjem zpráv, až po velmi jednoduchou variantu šifrování.

Přidání do IDE JeeLib vyžaduje Arduino IDE verze 1.5 a novější, se staršími verzemi není kompatibilní. Pokud je tento předpoklad splněn, pak je přidání knihovny otázka několika málo kliknutí. Nejdříve je však nutné získat knihovnu samotnou, ideálně přímo ze stránek JeeLabs `http://jeelabs.net/projects/jeelib/wiki`.

V záložce **Sketch**, nalezneme část **Import Library** a následně zvolíme buď novější variantu **Manage Libraries**, nebo starší verzi **Add Library**. Poté se již jedná o nalezení složky obsahující kód knihovny a její přidání. Knihovnu je taktéž možné přidat přímo ve formě ZIP archivu.

Formát hlavičky Hlavička je prvních 8 bitů každé zprávy. Určuje typ zprávy a jejího příjemce či odesílatele. První tři bity určují typ, tedy požadavek na ověření doručení zprávy a rozlišení unicast - broadcast. Zbylých 5 bitů určuje ID uzlu, buď se jedná o adresáta, pokud uzel tuto zprávu odesílá, nebo o odesílatele, pokud uzel tuto zprávu přijal (ke změně dochází při odeslání zprávy).

Jelikož máme na ID uzlu pouze 5 bitů, tak máme pouze 32 různých adres. Přidělujeme je v rozsahu 0-31 a krajní hodnoty jsou vyhrazeny pro speciální použití. ID 0 slouží jako univerzální ID pro broadcast při odesílání a uzel s ID 31 bude přijímat všechny zprávy v síti bez ohledu na adresáta.

Poslání zprávy Aby bylo možné poslat zprávu, je před samotným odesláním zprávy třeba zajistit, že nevysílá žádný jiný uzel, aby nedošlo k rušení. Knihovna JeeLib nám tuto starost poměrně usnadňuje, protože nemusíme řešit přímo přístup k médiu, ale pouze se dotážeme, zdali můžeme odesílat pomocí volání funkce `rf12_canSend()`, která nám odpoví, zdali můžeme poslat zprávu.

Zprávu následně odesíláme pomocí zavolání funkce `rf12_sendStart()`, kde jako parametry použijeme, hlavičku, ukazatel na odesílaná data a délku odesílaných dat. Podrobnější informace k oběma funkcím naleznete v dokumentaci. V případě že se nejedná o náročnou aplikaci a nepředpokládá se velká frekvence posílaných zpráv, je možné použít volání funkce `rf12_canSend()` v cyklu, dokud nevrátí pravdivou hodnotu. Celý tento kód je ještě možné zjednodušit pomocí funkce `rf12_sendNow()` která již nekonečný čekací cyklus obsahuje uvnitř a parametry má stejné jako `rf12_sendStart()`.

Přijetí zprávy Pro přijetí zprávy je třeba se pravidelně v krátkých intervalech dotazovat pomocí funkce `rf12_recvDone()`, která vrátí pravdivou hodnotu, pokud uzel přijal paket.

V případě, že v časovém intervalu mezi dvěma dotazy uzel přijme více paketů, pak je možné zpracovat pouze ten poslední a obsah paketu nalezneme v globálních proměnných.

Po přijetí zprávy je vhodné ověřit, zda-li odesílatel vyžaduje potvrzení o doručení, ideální řešení je pomocí následující podmínky:

Ukázka podmínky pro odeslání ACK zprávy

```
1  if(RF12_WANTS_ACK){
2      f12_sendStart(RF12_ACK_REPLY,0,0);
3  }
```

Síť ze dvou uzlů Při spojení mezi dvěma uzly musíme řešit periodické ověřování přijetí zprávy a pravidelné odesílání zprávy, případně odesílání zprávy na vnější příkaz (uzel může například umožnit odeslání zprávy v reakci na příkaz zaslaný přes sériový port). Jako ukázková aplikace pro tento účel nám poslouží ukázková aplikace z knihovny jeelib, konkrétně RF12demo, která umožňuje pomocí příkazů zaslaných přes sériový port ovládat většinu funkcí rádia, včetně odesílání zpráv.

Stačí tedy připojit dva uzly, na oba nahrát aplikaci RF12demo a například pomocí Serial Monitoru předávat příkazy uzlům. Ideální variantou je mít otevřené vše dvakrát tak, aby bylo možné pozorovat výstup obou uzlů zároveň.

Nastavení uzlu pomocí funkce setup v RF12demo

```
537  void setup () {
538      ...
539
540      Serial.begin(SERIAL_BAUD);
541      Serial.println();
542      displayVersion();
543
544      if (rf12_configSilent()) {
545          loadConfig();
546      } else {
547          memset(&config, 0, sizeof config);
548          config.nodeId = 0x81;       // 868 MHz, node 1
549          config.group = 0xD4;        // default group 212
550          config.frequency_offset = 1600;
551          config.quiet_mode = true;   // Default flags, quiet on
552          saveConfig();
553          rf12_configSilent();
554      }
555
556      rf12_configDump();
557      df_initialize();
558
559      ...
560  }
```

V ukázce funkce `setup` můžeme vidět počáteční nastavení sériového portu, kde nastavíme frekvenci na předem definovanou konstantu. Následně poté je podmínka, která

ověřuje uložení nastavení hodnot pro rádiovou komunikaci v paměti, kde pokud jsou hodnoty již uloženy, pak je aplikace pouze obnoví, zatímco pokud ještě uloženy nebyly, pak dojde k jejich nastavení a následnému uložení do paměti. tento přístup není nutný, v případě malého množství uzlů je rychlejší nastavovat každý individuálně při spuštění, nicméně při stavbě větší sítě již je potřeba hodnoty ukládat a díky tomu výrazně urychlit jakoukoliv změnu v síti.

Zjednodušená ukázka přijetí zprávy v RF12demo

```
582  if (rf12_recvDone()) {
583      byte n = rf12_len;
584      if (rf12_crc == 0)
585          showString(PSTR("OK"));
586      else {
587          if (config.quiet_mode)
588              return;
589          showString(PSTR(" ?"));
590          if (n > 20) // print at most 20 bytes if crc is wrong
591              n = 20;
592      }
593      ...
594      printOneChar(' ');
595      showByte(rf12_hdr);
596      for (byte i = 0; i < n; ++i) {
597          printOneChar(' ');
598          showByte(rf12_data[i]);
599      }
600
601      Serial.println();
602      ...
603  }
```

V ukázce kódu pro přijetí zprávy můžeme vidět, jak v aplikaci probíhá přijetí zprávy. Nejdříve je třeba splnit podmínku, že uzel přijal zprávu, následně dojde k ověření cyklického součtu, za účelem odhalení chyb při přenosu. Pokud při přenosu došlo k chybě, pak aplikace vypíše pouze prvních 20 znaků zprávy a zároveň ohlásí chybu. V případě správného doručení pak aplikace vypíše zprávy celou. Z ukázky byl pro přehlednost vynechán kód na hexadecimální výpis obsahu zprávy a signalizace činnosti uzlu pomocí led diody. V případě zájmu lze kompletní verzi nalézt přímo v kódu aplikace, počáteční číslo řádku je ponecháno shodné za účelem snadnějšího nalezení.

Zjednodušená ukázka odeslání zprávy v RF12demo

```
641  if (cmd && rf12_canSend()) {
642      activityLed(1);
643
644      showString(PSTR(" -> "));
645      Serial.print((word) sendLen);
646      showString(PSTR(" b\n"));
647      byte header = cmd == 'a' ? RF12_HDR_ACK : 0;
648      if (dest)
649          header |= RF12_HDR_DST | dest;
650      rf12_sendStart(header, stack, sendLen);
651      cmd = 0;
652
653      activityLed(0);
654  }
```

Na ukázce kódu pro odeslání zprávy můžeme vidět počáteční podmínku, kdy odesílat je možné pouze tehdy pokud nevysílá jiný uzel a zároveň tento uzel zprávu aktuálně nepřijímá. Následuje indikace pomocí led diody a samotné odeslání zprávy včetně vypsání informací o odesílání zprávy na sériový port.

Tyto dvě ukázky pokrývají prakticky celou základní funkcionalitu aplikace (obsah `loop` funkce), bez komentáře zůstávají pomocné funkce, které obstarávají zpracování příkazů přijatých přes sériový port, jelikož se jedná o použití velmi podobných principů jako při psaní běžného programu v jazyce C. taktéž bez komentáře zůstávají funkce sloužící k pokročilému ovládání rádia, jelikož jejich obsah přesahuje tento krátký manuál. V případě zájmu je možné více zjistit v oficiální dokumentaci ke knihovně JeeLib `http://jeelabs.net/pub/docs/jeelib/`.

4 WSN síť

Bezdrátová sensorová síť je složená z velkého množství malých autonomních zařízení, které spolu navzájem komunikují přes rádiové spojení. Každé zařízení, nazývané sensorový uzel, je napájené pomocí baterií a je vybavené potřebnými sensory pro monitorování okolí. Typický scénář nasazení sensorové sítě je náhodné rozmístění stovek až tisíců uzlů a následný sběr dat pomocí jednotlivých sensorů. Jako příklady oborů, kde mohou bezdrátové sensorové sítě nalézt uplatnění, lze uvést například armádu, zdravotnictví, zemědělství a další.

4.1 Ukázkové aplikace

Pro provoz jednoduché sítě zde jsou dvě základní aplikace. První periodicky odesílá data (čítač) z každého uzlu, zatímco druhá neslouží ke komunikaci, ale umožňuje jedinému uzlu zachytávat veškerý provoz v síti, což je vhodně jednak na ladění vlastních aplikací, tak na počáteční zjištění informací o síti, kterou neznáme.

4.1.1 Alive

Aplikace se statickým nastavením topologie sítě, každý uzel má pevně přidělené své ID a ID svého rodiče, ke kterému posílá vlastní zprávy a přeposílá veškeré zprávy přijaté. Takto je pevně ustanovený routovací strom a každý uzel periodicky zasílá zprávu svému rodiči. Zároveň každý uzel přeposílá všechny přijaté zprávy svému rodiči, tedy postupně všechny zprávy doputují k jednomu centrálnímu uzlu.

Pro správný provoz je třeba definovat cesty (routovací strom) tak, aby každý uzel měl svého rodiče v dosahu rádia a byl schopen jej nalézt. Pokud by tomu tak nebylo, tak by došlo k úplnému odříznutí jedné z větví stromu. Zároveň je třeba strom navrhnout tak, aby byla zátěž s přeposíláním zpráv rovnoměrně rozložená mezi všechny uzly, v ideálním případě by měl být routovací strom vyvážený[4].

Aplikace samotná funguje velmi jednoduše, podstatné části jsou popsány na následující ukázce funkce `loop`.

Funkce `loop` v aplikaci Alive

```
26  void loop () {
27    //if incoming message received
28
29    if(rf12_recvDone()){
30      if(RF12_WANTS_ACK){
31        rf12_sendStart(RF12_ACK_REPLY,0,0);
32      }
33
34      if(rf12_crc == 0){ //packet checksum is correct
35        //propagate to parent
36        byte header = B00000000;
37        //fill header using radioUtils
38        ru.resetAck(&header);
39        ru.setID(&header, parent);
40        rf12_sendNow(header, (const void*)rf12_data, rf12_len);
41      }
42    }
```

[4]toto není vždy možné, ale skutečný strom by se měl tomu ideálnímu co nejvíce přiblížit

```
43    delay(10);
44    counter++;
45
46    if(counter%100 == 0){
47      msgCounter++;
48      //send still alive msg
49      byte header;
50      //fill header using radioUtils
51      ru.resetAck(&header);
52      ru.setID(&header, parent);
53      rf12_sendNow(header, (const void*) &msgCounter, sizeof(msgCounter));
54      counter = 0;
55    }
56
57  }
```

V kódu funkce `loop` můžeme nejdříve vidět ověření, jestli uzel přijal zprávu. V případě kladného ověření probíhá ihned reakce na přijatou zprávu, tedy případné odeslání potvrzení přijetí a přeposlání zprávy dále, konkrétně uzlu, který je nastavený jako rodič.

Následuje krátká prodleva (aby uzel nebyl neustále vytížený) a poté zvýšení počítadla. Počítadlo je zde, aby uzel odesílal zprávy ve výrazně delších intervalech, než zprávy přijímá. Ideální stav je, když uzel přijme zprávu kdykoliv, zatímco odesílání musíme zpomalit, tedy provést jej pouze jednou za určitý počet cyklů.

Odesílání zprávy je poté již jednoduchý úkon, kde je využita knihovna radioUtils pro nastavení příjemce a všech dalších potřebných parametrů. Následně je zpráva odeslána pomocí funkce `rf12_sendNow`. V případě této aplikace si můžeme dovolit odesílání s aktivním čekáním, protože nepředpokládáme velký provoz na síti a neočekáváme velké vytížení uzlu. V opačném případě by bylo potřeba čekání zajistit vlastní pomocí, ideálně jej ještě zkombinovat s přijímáním zpráv, právě z důvodu, aby uzel při snaze odeslat zprávu se na delší dobu nezastavil čekáním na jednu funkci, když by zatím mohl alespoň přijímat zprávy, provádět měření pomocí sensorů či jinak interagovat s okolím.

4.1.2 Sniffer

Druhá aplikace slouží k zachycení veškerého provozu v síti. velmi dobře nám poslouží, když potřebujeme ladit vlastní aplikaci a pozorovat, co který uzel zasílá za zprávy. Zároveň ji taktéž můžeme použít v roli útočníka a velmi jednoduše pasivně odposlouchávat veškerou komunikaci v síti. Jediným omezením je dosah rádia. Aplikace používá trochu pokročilejší koncepty oproti Alive aplikaci, protože se předpokládá její nasazení i při testování sítě s velkým množstvím provozu. Ve funkci `setup` nastavíme ID rádia na hodnotu 31, tedy uzel, který přijímá všechny zprávy v síti. Toto je klíčová část této aplikace, ostatní je pouze již zpracování a výpis zpráv. Sniffer jako takový je speciální aplikací, protože jako jeden z mála neodesílá žádné zprávy. Opět se podíváme na obsah funkce `loop`.

Funkce `loop` v aplikaci Sniffer

```
32  void loop () {
33
34      if (rf12_recvDone()) {
35          // quickly save a copy of all volatile data saveLen = rf12_len;
36          saveCrc = rf12_crc;
37          saveHdr = rf12_hdr;
38          memcpy(saveData, (const void*) rf12_data, sizeof saveData);
39          rf12_recvDone();
40          // release lock on info for next reception
41          if (saveCrc != 0) {
42              su.print("CRC error #", output);
43              su.println(saveLen, DEC, output);
44          }
45          else {
46              su.print("OK (", output);
47              String header = "";
48              for (byte i = 0; i < 8; ++i) {
49                  header.concat(bitRead(saveHdr, 7-i)); // read bytes of header
50              }
51              su.print(header, output);
52              su.print("b) ", output);
53              su.print(saveHdr & RF12_HDR_ACK ? "REQ " : " ", output);
54              su.print(saveHdr & RF12_HDR_CTL ? "ACK " : " ", output);
55              su.print(saveHdr & RF12_HDR_DST ? "DST:" : "SRC:", output);
56              su.print(saveHdr & RF12_HDR_MASK, output);
57              su.print(" #", output);
58              su.println(saveLen,DEC, output);
59              // print out all data bytes, wrapping long lines byte pos = 0;
60              int pos = 0;
61              for (byte i = 0; i < saveLen; ++i) {
62                  su.print(' ', output);
63                  su.print(saveData[i],HEX, output);
64                  pos += 2;
65                  if (saveData[i] >= 16) ++pos;
66                  if (pos > 75) {
67                      su.println();
68                      pos = 0;
69                  }
70              }
71              if (pos > 0) su.println();
72          }
73      }
74  }
```

Můžeme opět vidět ověření přijetí zprávy, nicméně protože zpráva nebyla určena tomuto uzlu, tak již neodesíláme potvrzení o doručení. Kvůli požadavku na rychlost nejdříve zkopírujeme všechny informace o přijaté zprávě do vlastních proměnných a umožníme příjem další zprávy co nejdříve. Následuje ověření správnosti doručení (CRC - cyclic redundancy check) a poté samotné zpracování zprávy.

Můžeme vidět, že hlavička je zpracována bit po bitu tak, aby bylo vidět všech 8 bitů jednotlivě. Následně jsou zpracovány všechny další položky zprávy. můžeme vidět, že na výstup není použita běžná knihovní funkce, ale knihovna serialUtils, která nám umožňuje jednotlivým výpisům přiřadit prioritu a tak například v pozdějších fázích vynechat ladící výpisy.