Radio communicatie met Arduino

Patrick van Looy & Bram Leenders

May 6, 2014

1 Inleiding

Met behulp van radiocommunicatie kunnen apparaten, zoals computers, met elkaar communiceren zonder een fysieke verbinding daarvoor nodig te hebben. Dit leent zich voor het makkelijk opzetten van (grote) netwerken, omdat de verbindingen zonder planning vooraf kunnen worden opgezet. Bij bijvoorbeeld Smart Dust kunnen de agents na de verspreiding zelf connecties opzetten en hier gebruik van maken.

Een nadeel van draadloze communicatie, is dat er vaak meer last is van storing dan wanneer er een fysieke verbinding (i.e. een kabel) aanwezig is. Doordat de communicatie niet "afgesloten" van de buitenwereld plaats vind, kunnen er externe storingszenders zijn. Voorbeelden van storingen zijn bijvoorbeeld andere agents die communiceren, obstakels die een signaal blokkeren of weerkaatsing van eerder gestuurde berichten.

2 Probleemstelling

In dit onderzoek wordt gekeken naar verschillende modi waarop radiocommunicatie met Arduino's gedaan kan worden. Het doel is om erachter te komen welke modus het minst last heeft van storing en (dus) de laagste error rate heeft.

In de tests wordt het effect van drie verschillende factoren onderzocht:

- Het frequentiekanaal
- De outputpower van verzonden pakketten
- Datatransmissiesnelheid

De tests moeten uitslag geven welke instellingen zorgen voor de beste communicatie.

3 Methodologie

Om "beste" communcatie kwantificeerbaar te maken gebruiken we de error rate. We definiëren de error rate als het aantal niet of incorrect ontvangen berichten gedeeld door het aantal verstuurde berichten;

error rate $= \frac{\text{niet ontvangen}}{\text{verstuurd}} = 1 - \frac{\text{ontvangen}}{\text{verstuurd}}$

Voor goede communicatie is het van belang dat deze zo laag mogelijk, idealiter nul, is. In dit onderzoek is de error rate de enige eigenschap waarop we

de instellingen beoordelen, en laten we andere factoren zoals bandbreedte of opgenomen vermogen achterwege.

In de testopstelling is gebruik gemaakt van twee Arduino Uno's, de Nordic nrf24l01+ radio en de RF42 library. Tenzij expliciet anders vermeldt gebruiken de radio's frequentiekanaal 0, een transmissionspeed van 250kbps en de hoogste outputpower (0 dBm). De afstand tussen beide radio's is vijf meter en er is sprake van een line of sight (geen blokkerende objecten). De tests zijn uitgevoerd in een ruimte met andere elektrische aparatuur die ook van radiocommunicatie gebruik maakte.

Tijdens de test zendt een Arduino duizend maal een pakket; wanneer de andere Arduino het pakket ontvangt stuurt deze hem terug. Wanneer het pakket voor de tweede maal ontvangen wordt, telt dat als één succesvol ontvangen pakket. Er wordt dus naar een volledige roundtrip gekeken. Beide radio zenders gebruiken telkens dezelfde instellingen, en de timeout tijd is zeer ruim gekozen om dit geen beperking te laten zijn.

De gebruikte code is te zien in Appendix A.

4 Resultaten&Analyse

Deze sectie geeft de resultaten van de uitgevoerde tests en toelichting daarbij. Er zijn drie factoren onderzocht; de outputpower, het frequentiekanaal en de datatransmissiesnelheid.

4.1 Outputpower

In de eerste serie tests is gekeken naar het effect van verandering van de output power op de error rate. Hierbij is de hypothese dat een sterker output signaal bij de zender een sterker input signaal bij de ontvanger geeft. Dus, dat een sterker input signaal een lagere error rate geeft.

Outputpower	Error rate
0 dBm	0.4%
-6 dBm	4.8%
-12 dBm	11.1%
-18 dBm	21.9%

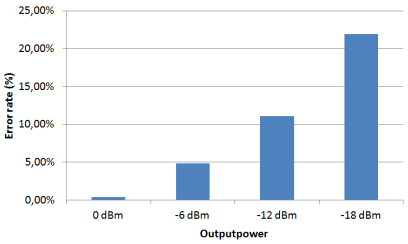

Figure 1: Error rate bij verschillende output sterktes.

In de resultaten is duidelijk terug te zien dat deze hypothese klopt: een sterker output signaal geeft een lagere error rate. Er is een zeer significant verschil; een toename van factor 64 in vermogen geeft een error rate die ongeveer een factor 55 lager is.

4.2 Datatransmissiesnelheid

Hierbij zien we dat de error rate slechts weinig verandert; hoewel het relatieve verschil vrij groot is (factor twee) blijft de error rate erg laag. Afgaande op deze

Datatransmissiesnelheid	Error rate
250 kbps	0.4%
1 mbps	0.6%
2 mbps	0.8%

Figure 2: Error rate bij verschillende datatransmissie snelheden.

resultaten kunnen we dus stellen dat een lage datatransmissiesnelheid de error rate positief beïnvloed.

Echter, omdat de error rate in alle gevallen erg laag was, raden we aan om eerst uitvoeriger te testen binnen opstellingen die een hogere error rate hebben.

4.3 Frequentiekanaal

Zoals in de introductie al kort genoemd is, kunnen radiozenders elkaar storen omdat ze hetzelfde medium gebruiken. Om dit te voorkomen kunnen zenders verschillende frequenties gebruiken, waardoor ze elkaar niet of minder storen. In de gebruikte testruimte zijn mogelijke stoorzenders onder andere omringende Arduino's en laptops die WiFi gebruiken. Omdat de frequentieband van de Arduino's (2.4 GHz) gedeeld wordt met WiFi, verwachten we dat de frequenties rond WiFi erg veel last hiervan hebben.

De testresultaten ondersteunen deze hypothese: er is duidelijk zichtbaar dat frequenties dicht bij de 2.4 GHz erg veel storing (bijna 25%) hebben, terwijl frequenties die daar verder vanaf liggen vrijwel geen storing (1.3% error rate) hebben. Wat verder opvalt, is dat niet alleen het precieze WiFi frequentiekanaal gestoord worden maar de frequentiekanalen ernaast ook. Het is dus zaak om een frequentiekanaal te zoeken dat zo ver mogelijk van gebruikte kanalen af ligt.

Tabel 3 toont een overzicht van de meetresultaten van de geteste frequentiekanalen.

Frequentiekanaal	Error rate
0	4.7%
15	2.4%
30	10.6%
45	21.1%
60	24.6%
75	13.6%
90	1.6%
105	1.3%

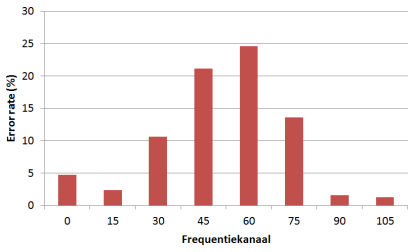

Figure 3: Error rate bij verschillende frequentiekanalen.

5 Conclusie

Uit de tests is gebleken dat de laagste error rate behaald wordt bij een sterk outputsignaal met lage datatransmissiesnelheid, op een frequentiekanaal dat niet door andere zenders gebruikt wordt.

Natuurlijk is voor het opzetten van een sensornetwerk meer nodig dan alleen een lage error rate. Zo kan een hogere datatransmissiesnelheid de voorkeur hebben boven een lage, ookal heeft deze een hogere error rate. Dit omdat de bandbreedte van 2mbps acht keer hoger is dan 500kbps, dus voor grote hoeveelheden data is het energiezuiniger en sneller om op hoge snelheid te zenden.

Voor de outputpower geldt hetzelfde; hoewel de error rate lager is, verbruikt de zender ook veel meer stroom (11,3 mA bij 0dBm vs. 7 mA bij -18dBm). Ook hierbij kan dus gekozen worden voor een setting met hogere error rate om stroom te bezuinigen.

A Bijlage 1 - Code

```
/* Copyright (C) 2011 J. Coliz <maniacbug@ymail.com> */

#include <SPI.h>
#include "nRF24L01.h"
#include "RF24.h"
#include "printf.h"

// Hardware configuration
RF24 radio(3, 9);
const int role_pin = 7;
const uint64_t pipes[2] = { 0x123456789aLL, 0x987654321bLL };
const int sendValue = 170; // binary; 10101010
const int numberOfPackets = 1000;
const int RESETVAL = 42;

// Role of the Arduino; sender or pong-backer
typedef enum { role_ping_out = 1, role_pong_back } role_e;
const char* role_friendly_name[] = { "invalid", "Ping out", "Pong
    back"};
const rf24_pa_dbm_e outputPowerLevel[] = {RF24_PA_MAX,
    RF24_PA_HIGH, RF24_PA_LOW, RF24_PA_MIN};
const rf24_datarate_e datarateLevel[] = {RF24_250KBPS, RF24_1MBPS,
    RF24_2MBPS};

// The role of the current running sketch
role_e role;
void setup(void) {
    pinMode(role_pin, INPUT);
    digitalWrite(role_pin,HIGH);
    delay(20); // Just to get a solid reading on the role pin

    // read the address pin, establish our role
    if ( ! digitalRead(role_pin) )
        role = role_ping_out;
    else
        role = role_pong_back;

    Serial.begin(57600);
    printf_begin();
    printf("\n\rRF24/examples/pingpair/\n\r");
    printf("ROLE: %s\n\r",role_friendly_name[role]);

    // Setup and configure rf radio
    radio.begin();
    radio.setRetries(0,0);
    radio.setDataRate(datarateLevel[0]);
    radio.setPALevel(outputPowerLevel[0]);
    radio.setChannel(0);
    radio.setPayloadSize(8);

    if ( role == role_ping_out ) {
        radio.openWritingPipe(pipes[0]);
        radio.openReadingPipe(1,pipes[1]);
    } else {
        radio.openWritingPipe(pipes[1]);
        radio.openReadingPipe(1,pipes[0]);
    }

    radio.startListening();
    radio.printDetails();
```

```
}


// Number of succesfully received packages; 0 <= success <= rounds
int success = 0;
// Rounds of communication so far; 0 <= rounds <= numberOfPackets
int rounds = 0;
int test = 0; int test2 = 0; int testChannel = 0;

void loop(void) {
    if (role == role_ping_out) {
        rounds++;

        radio.stopListening();
        bool ok = radio.write( &sendValue, sizeof(int) );
        radio.startListening();

        // Wait here until we get a response, or timeout (250ms)
        unsigned long started_waiting_at = millis();
        bool timeout = false;
        while ( ! radio.available() && ! timeout )
            if (millis() - started_waiting_at > 250 )
                timeout = true;

        // Describe the results
        if (!timeout) {
            int receivedValue;
            radio.read( &receivedValue, sizeof(int) );

            // Successfull round trip of our value! Increase our
                success counter.
            if(receivedValue == sendValue) {
                success++;
            }
        }

        if (rounds == numberOfPackets) {
            printf("\n--------\n");
           // printf("Power level: %i (0=MAX, 3=MIN)\n", test);
           // printf("Data rate: %i (0=250kbps, 1=1mbps
               2=2mbps)\n", test);
            printf("Channel: %i\n", testChannel);
            printf("# packets sent:                %i\n",
                numberOfPackets);
            printf("# packets correctly received: %i\n", success);
            printf("--------\n");
            // Reset counters
            success = 0;
            rounds = 0;

            radio.stopListening();
            radio.setPALevel(outputPowerLevel[0]); // Max power;
                increase chance of succesfully receiving it
            bool ok = radio.write( &RESETVAL, sizeof(int) ); //
                Pray this will be received
            radio.startListening();

            //test = (test+1)%3;
            test2 = (test2+1)%8;
            //radio.setDataRate(datarateLevel[test]);
            // radio.setPALevel(outputPowerLevel[test]);
```

```
            testChannel = (15*test2);
            radio.setChannel(testChannel);
        }

        delay(10);
    }

    // Pong back role.  Receive each packet and send it back
    if ( role == role_pong_back ) {
        if ( radio.available() ) {
            int v;
            bool done = false;
            while (!done) {
                // Read the sent value
                done = radio.read( &v, sizeof(int) );
                delay(10);
            }

            if (v == RESETVAL) {
                //test = (test+1)%3;
                test2 = (test2+1)%8;
                //radio.setDataRate(datarateLevel[test]);
                // radio.setPALevel(outputPowerLevel[test]);
                radio.setChannel(15*test2);
                printf("Finished test; moving to next!\n");
            }

            radio.stopListening();
            radio.write( &v, sizeof(int) );
            radio.startListening();
        }
    }
}
```