

Practicum 2 Betrouwbare communicatie

Peter van Dijk & Elizabeth Schermerhorn

9 mei 2014

Inhoudsopgave

1	Packet Error Rate-metingen	3
1.1	Inleiding	3
1.2	Probleemstelling	3
1.3	Methodologie	3
1.3.1	De hardware	3
1.3.2	De software	4
1.3.3	De instellingen/vastgestelde constanten	4
1.4	Resultaten en analyse	5
1.4.1	Metingen en resultaten	5
1.5	Conclusie	7
2	Betrouwbare end-to-end-communicatie	8
2.1	Inleiding	8
2.2	Probleemstelling	8
2.3	Methodologie	8
2.4	Resultaten en analyse	8
A	De radio en de RF24-library	9
B	Code Sender	9
C	Code Receiver	10

1 Packet Error Rate-metingen

1.1 Inleiding

De Nordic RF24 is een radiozender en -ontvanger voor de 2.4GHz band. De RF24 heeft echter een aantal verschillende instellingen voor de kanalen, output-power en datasnelheid. Het doel van dit onderzoek is bepalen wat de optimale instellingen zijn om de Packet Error Rate (PER) zo laag mogelijk te krijgen. Eerst zal de probleemstelling besproken worden samen met vastgestelde hypothesen, vervolgens zal de methodologie aan bod komen en als laatste zullen de resultaten besproken en geanalyseerd worden.

1.2 Probleemstelling

De onderzoeksvraag die hier beantwoord zal worden is: *"Voor welke waarden voor de outputpower, transmissiesnelheid en frequentiekanaal is de PER het laagst."* Hierbij horen de volgende hypothesen:

1. Op het moment dat er meerdere mensen op hetzelfde frequentiekanaal zitten zal dit voor veel pakket verlies zorgen.
2. Bij een hogere outputpower zal het pakketverlies kleiner zijn.
3. Bij een hogere transmissiesnelheid zal er meer pakketverlies optreden.

In het volgende onderzoek zullen deze hypothesen getoetst worden.

1.3 Methodologie

Om deze hypothesen te toetsen moeten er metingen gedaan worden. Hier zijn verschillende zaken van belang.

- De gebruikte hardware
- De gebruikte software
- De instellingen/vastgestelde constanten
- De onderzoeksopstelling

1.3.1 De hardware

De Hardware die gebruikt is bij de metingen is een Arduino UNO en een Nordic nRF24L01 draadloze transceiver.

1.3.2 De software

Om de hardware te gebruiken wordt gebruik gemaakt van de opensourcesoftwarebibliotheek voor de Arduino, die te vinden is op:

<http://maniacbug.github.io/RF24/>

Met deze software kan de radio aangestuurd worden en kunnen pakketjes worden verzonden en ontvangen. De radio luistert en verstuurt pakketten over een bepaald kanaal. Het is van belang dat de radio's hetzelfde kanaal gebruiken, zodat ze met elkaar kunnen communiceren. De kanalen hebben een nummer van 0 tot en met 125.

Om de metingen uit te voeren en onze hypothesen te testen is er gebruik gemaakt van één zender en één ontvanger. De zender gebruikt `sender.ino` en de ontvanger gebruikt `receiver.ino`. Deze programma's zijn te vinden in de appendix. De zender zendt pakketjes met daarin het pakketnummer. De ontvanger begint met luisteren en onthoudt het nummer van het eerste pakketje dat hij heeft ontvangen. Vervolgens zal de ontvanger blijven luisteren totdat deze 1000 pakketten heeft ontvangen. Uit de inhoud van het laatste pakket kan worden opgemaakt hoeveel pakketten er verloren zijn gegaan voor de duizend die er ontvangen zijn. De PER wordt berekend met de formule:

$$(((\text{laatste pakketnummer}) - [\text{eerste pakketnummer}]) / [\text{aantal ontvangen pakketten}])$$

In de software van de Sender en Receiver is gebruik gemaakt van de volgende drie methoden om de verschillende waarden aan te kunnen passen.

- `setChannel(uint8 i)`
met $0 \leq i \leq 125$
- `setPALevel(PALevel level)`
met $\text{level} \in \{ \text{RF24_PA_MAX}, \text{RF24_PA_MIN}, \text{RF24_PA_LOW} \}$
- `setDataRate(Rate rate)`
met $\text{rate} \in \{ \text{RF24_1MBPS}, \text{RF24_2MBPS} \}$

Door met deze drie methoden de waarden aan te passen verkrijgen we de gewilde meetresultaten. Deze zullen in sectie 1.4 besproken worden.

1.3.3 De instellingen/vastgestelde constanten

Om deze metingen bruikbaar te houden moeten er een paar constanten gedefinieerd worden.

- De grootte van de payload
- Het aantal te versturen berichten

Het aantal keer dat er een pakket opnieuw verstuurd wordt is niet van belang in deze implementatie omdat het zo geïmplementeerd is dat het aantal ontvangen berichten wordt bijgehouden en het aantal verzonden berichten wordt bijgehouden. Voor de grootte van de payload is er gekozen voor de grootste mogelijke waarde die deze kan aannemen, namelijk 255 bytes. Deze waarde is gekozen omdat de onderzoeksvraag is wanneer is de radio communicatie het meest betrouwbaar en dit moet getest worden voor reële waarden. Het aantal te versturen berichten is op 1000 gezet. Hier is voor gekozen vanwege het idee omdat meer waarden betekent nauwkeurigere resultaten.

De meetopstelling bestaat uit twee radio's welke 270cm uit elkaar geplaatst zijn. Om de resultaten consistent te houden moet bij elke meting deze afstand aangehouden worden. Verder worden berichten niet opnieuw verzonden als er geen acknowledgement volgt, want dit betekent pakket verlies en dat moet gemeten worden.

1.4 Resultaten en analyse

Hier zullen de resultaten weergegeven en geanalyseerd worden.

De constanten zoals ze standaard gebruikt zullen worden. Als ze gewijzigd worden voor de test dan wordt dit aangegeven.

- afstand tussen radio's: 270 cm
- Geen hertransmissie van pakketten
- payload: 255 bytes
- channel: 125

1.4.1 Metingen en resultaten

Tijdens de metingen waren er geen andere radio's aan het sturen, waardoor de radio's geen last van interferentie hadden.

Instellingen	PER	gemiddelde PER
outputpower: MAX	1/1000	2/3000
transmissiesnelheid: 2MBps	1/1000	
delay: 25 ms	0/1000	

Instellingen	PER	gemiddelde PER
outputpower: MAX	0/1000	0/3000
transmissiesnelheid: 2MBps	0/1000	
delay: 50 ms	0/1000	

Instellingen	PER	gemiddelde PER
outputpower: MAX	0/1000	0/3000
transmissiesnelheid: 1MBps	0/1000	
delay: 50 ms	0/1000	

Instellingen	PER	gemiddelde PER
outputpower: MIN	85/1000	1273/3000
transmissiesnelheid: 2MBps	669/1000	
delay: 50 ms	519/1000	

Instellingen	PER	gemiddelde PER
outputpower: LOW	0/1000	8/3000
transmissiesnelheid: 2MBps	2/1000	
delay: 50 ms	6/1000	

Instellingen	PER	gemiddelde PER
outputpower: MAX	0/1000	0/3000
transmissiesnelheid: 2MBps	0/1000	
delay: 50 ms	0/1000	
channel: 0		

Instellingen	PER	gemiddelde PER
outputpower: MAX	0/1000	1/3000
transmissiesnelheid: 2MBps	1/1000	
delay: 50 ms	0/1000	
channel: 64		

Verder zijn er nog enkele kleinere tests uitgevoerd op een netwerk met interferentie. Dit leverde de volgende resultaten:

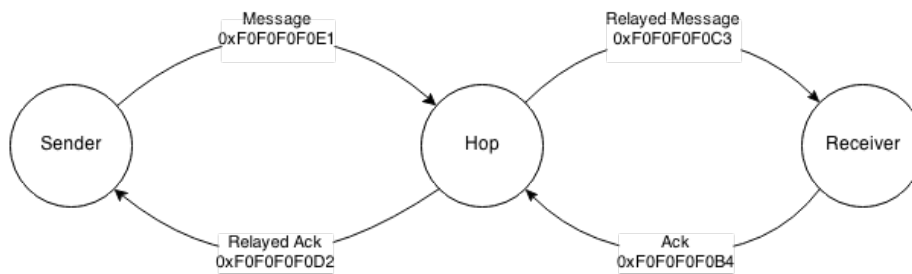
Instellingen	PER	gemiddelde PER
outputpower: MAX	47/300	433/3000
transmissiesnelheid: 2MBps	77/300	
delay: 50 ms	9/300	

Om het resultaat nauwkeuriger en accurater te kunnen weergeven is ervoor gekozen om alle tests in drievoud uit te voeren zodat vreemde gebeurtenissen opgemerkt worden. Een voorbeeld is een error rate > 1 . Tussen deze resultaten is de instelling HIGH voor outputpower niet weergegeven, aangezien deze voor de gebruikte versie van de radio niet een mogelijke waarde was.

1.5 Conclusie

Als er naar de gevonden waarden gekeken wordt, is de PER onacceptabel wanneer de outputpower op MIN staat. Een gemiddelde PER van 1273/3000 pakketten zorgt op den duur voor veel hertransmissies dus veel vertraging op het netwerk. Dit is een situatie die niet gewenst is. Wanneer naar de overige instellingen gekeken wordt en de daarbij behorende gemiddelde PER dan zijn ze perfect of een $PER < 0.01$. Hier moet bij vermeld worden dat op het moment dat er meer interferentie is op het netwerk de PER omhoog schiet op het netwerk. Op dat moment is het kiezen van een ongebruikt frequentiekanaal van belang. Dit experiment heeft plaats gevonden in een interferentievrije ruimte.

De onderzoeksvraag is: *"Voor welke waarden voor de outputpower, transmissiesnelheid en frequentiekanaal is de PER het laagst."* Het antwoord dat hierbij hoort is dat voor elke waarde van de transmissiesnelheid en het frequentiekanaal de PER niet boven de 0.01 uit komt. Dit geldt echter niet voor de outputpower; op het moment dat de outputpower op minimaal gezet wordt bedraagt de gemiddelde PER 1273/3000. Hieruit kan geconcludeerd worden dat de outputpower tenminste op laag of maximaal gezet moet worden om de PER tot een minimum te beperken.



Figuur 1: Opstelling van het netwerk

2 Betrouwbare end-to-end-communicatie

2.1 Inleiding

2.2 Probleemstelling

2.3 Methodologie

2.4 Resultaten en analyse

A De radio en de RF24-library

In dit onderzoek wordt gebruik gemaakt van de rf24 radio. Deze radio heeft de volgende eigenschappen:

- **Frequentieband:** 2.4000-2.4835 GHz
- **Datasnelheid:** 1 of 2 Mb/s
- **Aantal kanalen:** 126 RF-kanalen
- **Modulatietechniek:** Gaussian Frequency Shift Key(GFSK)

Energiemodus	Energieverbruik in Ampère
Standby-I	22 μ A
Standby-II	320 μ A
Power down	900 nA

Zendmodus	Energieverbruik in Ampère
0 dBm	11.3 mA
-6 dBm	9 mA
-12 dBm	7.5 mA
-8 dBm	7 mA

Ontvangstmodus	Energieverbruik in Ampère
2 Mbps	12.3 mA
1Mbps	11.8 mA

B Code Sender

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

RF24 radio(3, 9);
int messages_sent = 0;

const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

void setup(void)
{
  Serial.begin(57600);
  printf_begin();
  printf("\n\rRF24/examples/pingpair/\n\r");

  //
```

```

    // Setup and configure rf radio
    //
    radio.begin();
    radio.setPALevel(RF24_PA_MAX);
    radio.setDataRate(RF24_2MBPS);
    radio.setRetries(0,0);
    radio.setPayloadSize(255);
    radio.setChannel(125);
    radio.openWritingPipe(pipes[0]);
    radio.startListening();
    radio.printDetails();
}

void loop(void){
    // First, stop listening so we can talk.
    radio.stopListening();

    // Take the time, and send it. This will block until complete
    unsigned long time = millis();
    printf("Now sending %lu...",time);
    bool ok = radio.write( &messages_sent, sizeof(unsigned int) );
    messages_sent++;
    printf("number of messages sent: %u\n\r",messages_sent);

    // Now, continue listening
    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 > 200 )
            timeout = true;

    // Try again 1s later
    delay(25);
}

```

C Code Receiver

```

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

RF24 radio(3, 9);

```

```

const uint64_t pipes[2] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL };

unsigned long received = 0;
unsigned long sent = -1;
unsigned long initVal = -1;
bool started = false;
bool stopped = false;

void setup(void)
{
    Serial.begin(57600);
    printf_begin();
    received = 0;
    printf("\n\rRF24/examples/pingpair/\n\r");

    //
    // Setup and configure rf radio
    //
    radio.begin();
    radio.setChannel(125);
    radio.setRetries(0,0);
    radio.setPayloadSize(255);
    radio.setDataRate(RF24_2MBPS);
    radio.openReadingPipe(1,pipes[0]);
    radio.startListening();
    radio.printDetails();
}

void loop(void)
{
    // if there is data ready
    if ( radio.available() )
    {
        // Dump the payloads until we've gotten everything
        bool done = false;
        while (!done)
        {
            // Fetch the payload, and see if this was the last one.
            done = radio.read( &sent, sizeof(unsigned long) );
            stopped = received >= 1000;
            // Spew it
            if(started && !stopped){
                received = received+1;
            }
        }
    }
}

```

```

        printf("Got packet: %u", received);
        printf(" payload: %u", (sent-initVal));
        printf(" lost: %u\r\n", ((sent-initVal)-received));
    }else if(!started && sent != -1 ){
        started = true;
        initVal = sent;
    }
    // Delay just a little bit to let the other unit
    // make the transition to receiver
    delay(20);
}

    // First, stop listening so we can talk
    radio.stopListening();

    // Now, resume listening so we catch the next packets.
    radio.startListening();
}
}

```