Practicum 2 Betrouwbare communicatie

Peter van Dijk & Elizabeth Schermerhorn

9 mei 2014

Inhoudsopgave

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, outputpower 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 hypotheses 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 zend 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:

(([laatste pakketnummer]-[eerste pakketnummer])/[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(uint˙8 i)
  met $0 \leq i \leq 125$
- setPALevel(PALevel level)
  met level $\in$ { RF24_PA_MAX, RF24_PA_MIN, RF24_PA_LOW }
- setDataRate(Rate rate)
  met rate $\in$ { RF24_1MBPS, 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 grootst 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 bericht 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 transmissiesnelheid: 2MBps delay: 25 ms	1/1000 1/1000 0/1000	2/3000

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

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.

2 Betrouwbare end-to-end-communicatie

2.1 Inleiding

In het vorige experiment is de betrouwbaarheid van een singlehop-netwerk onderzocht. Nu kan de betrouwbaarheid van een multihop-netwerk getoetst worden en kunnen de waarden vergeleken worden om te bepalen welk netwerk betrouwbaarder is.

2.2 Probleemstelling

De probleemstelling waarvan uit zal worden gegaan is *"Is het mogelijk een betrouwbaar end-to-end multihop-connectie op te zetten?"*. De hypothese is dat dit mogelijk is in combinatie met :

- MIN zal het slechtst presteren en MAX het beste(transmissie sterkten)
- alternating bit protocol(ABP) Met deze drie aspecten verwerkt in de implementatie zou het multihop-netwerk moeten werken.

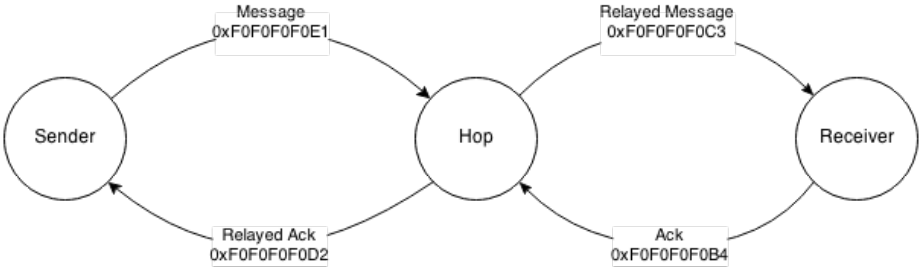
2.3 Protocol

Het protocol wat veelal gebruikt wordt in deze implementatie is het ABP. Dit protocol houdt twee zaken in:

- hertransmissies sturen
- kijken of er iets binnenkomt

Het ABP zorgt ervoor dat wanneer er geen acknowledgement ontvangen wordt er een hertransmissie gestuurd wordt. Door een bit op 0 of 1 te zetten kan er gekeken worden of er nieuwe data ontvangen wordt of niet. Dit is de globale werking van het protocol. Dit protocol is gebruikt in de implementatie van het multihop-netwerk, veelal op dezelfde manier maar dit zal toegelicht worden.
In de implementatie komt dit protocol terug in alle nodes. Alle nodes moeten weten of er berichten binnen komen. Om dit te registreren is er gekozen voor het ABP.

De zender verstuurt de berichten en wacht totdat er een acknowledgement terugkomt voordat een nieuw bericht wordt verstuurd. Op het moment dat er een time-out plaatsvind zal er een hertransmissie plaatsvinden. Op het moment dat er een bericht binnenkomt dan zal dit geregistreerd worden en zal dit bericht verder worden afgehandeld.



Figuur 1: Opstelling van het netwerk

2.4 Methodologie

Om de onderzoeksvraag te kunnen beantwoorden is er een meetopstelling nodig. De meetopstelling is weergegeven in Figuur 1 met een Sendernode, een Hopnode en een Receivernode. De namen geven het al aan, de Sendernode stuurt de berichten (en wacht op bevestiging), de Hopnode stuurt de berichten door en de Receivernode ontvangt de berichten (en bevestigd de ontvangst).

2.4.1 De software

De software die gebruikt is voor dit experiment is grotendeels hetzelfde als het voorbeeld pingpair dat wordt meegeleverd bij de RF24 library voor de Arduino. In deze implementatie zijn slechts de adressen van de pipes gewijzigd naar de vier die te zien zijn in Figuur 1 en is de pakketgrootte veranderd naar 255 bytes.

De Hopnode luistert naar de Sendernode en Receivernode en stuurt berichten van de Sendernode door naar de Receivernode en andersom. In de implementatie is ervoor gezorgd dat er hertransmissies gestuurd kunnen worden als er een time-out optreed. Er is gebruik gemaakt van een aantal constanten, namelijk:

- payload = 255 bytes
- hertransmissies : 15
- transmissiesnelheid: 2MBps

Deze constanten zullen tijdens het testen niet worden veranderd.

2.4.2 De hardware

Deze bestaat uit drie identieke nodes. Deze nodes zijn allemaal voorzien van een radio waarmee ze de berichten kunnen ontvangen en versturen. De nodes zijn in een lijn geplaatst met een meter afstand tussen Sender en Hop en tussen Hop en Receiver. De experimenten zijn uitgevoerd in een netwerk waar nog meer radioś aan het sturen zijn.

2.5 Resultaten en analyse

Om de resultaten te verkrijgen die gewenst zijn worden er telkens andere waarden gebruikt voor de transmissie sterkte. De transmissiesterkte geldt voor alle drie de nodes(zender, hop en ontvanger). De drie waarden die hiervoor gebruikt zijn:

- RF24_PA_MIN
- RF24_PA_LOW
- RF24_PA_MAX

De waarden die gevonden zijn bij de volgende instellingen volgen hieronder.

testnummer	**PER(RF_PA_MIN)**	**gemiddelde**
1.	1/1000	25/3000
2.	17/1000	
3.	7/1000	

testnummer	**PER(RF_PA_LOW)**	**gemiddelde**
1.	0/1000	2/3000
2.	1/1000	
3.	1/1000	

testnummer	**PER(RF_PA_MAX)**	**gemiddelde**
1.	0/1000	1/3000
2.	0/1000	
3.	1/1000	

In de bovenstaande drie grafieken staan de PER samen met de transmissie sterkte waarmee de pakketjes verstuurd zijn. Er moet rekening gehouden worden met de omgeving waarin deze experimenten zijn uitgevoerd.

2.6 Conclusie

De hypothese die gesteld werd: *"Is het mogelijk een betrouwbaar end-to-end multihop-connectie op te zetten?"*. Deze hypothese wordt nu gestaafd door de resultaten die verkregen zijn door de experimenten uit te voeren met veranderende transmissie sterktes. Uit de grafieken bij 2.5 kan opgemaakt worden dat een multi-hop verbinding, welke op deze manier geïmplementeerd is, een redelijke implementatie geeft met een lage error rate. Zoals verwacht in de hypothese is zenden op minimum de slechtst scorende, maar zolang je hertransmissies hebt is dit een degelijke implementatie. De hertransmissies komen voort uit het geïmplementeerde alternating bit protocol. Door het ABP en de beste transmissie sterkte te kiezen is er een stabiel netwerk opgezet wat in het dagelijks leven toegepast kan worden.

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 μA
Standby-II	320 μA
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

Het volgende aspect van de RF24 radio is Enhanced ShockBurst. Dit is een pakket gerichte data link layer protocol. Belangrijke functies van Enhanced ShockBurst zijn:

- Automatische pakket samenvoeging
- 6 data pipe multiceivers
- pakket afhandeling

Het formaat van het datapakket bestaat uit verschillende delen, het adres, pakket identificatie, geen acknowledgement bits en CRC. De velden hebben de volgende functies:

Data veld	Functie
Adres	Het adres van de ontvanger
Pakket identificatie	Om te achterhalen of het pakket een hertransmissie is.
geen acknowledgement bit	Als deze bit gezet is mag het pakket niet automatisch erkend worden.
CRC	Error detectie mechanisme voor het pakket

Enhanced Shockburst heeft als feature automatisch pakket validatie. In ontvangst modus zoekt de RF24 constant naar valide adressen welke gegeven zijn in het RX_ADDR registers. Op het moment dat een valide adres gedetecteerd

wordt, zal Enhanced Shockburst het pakket gelijk valideren. Enhanced Shockburst is voorzien van een ART(auto retransmission) functie. Door deze functie is Enhanced Shockburst in staat een pakket uit zichzelf opnieuw te versturen als er geen acknowledgement wordt ontvangen. Er zijn drie redenen waarom de ART functie stopt met het luisteren na het versturen van een pakket. De ART stopt met luisteren na:

- Auto retransmit delay(ARD) is verlopen
- Geen gematcht adres gevonden binnen 250μ s
- Nadat er een pakket ontvangen is

De radio is niet in staat om te broadcasten maar om deze functionaliteit wil te krijgen moet er gebruik gemaakt worden van een multiceiver.
De adressen van een multiceiver moeten altijd sterk op elkaar lijken, alleen de least significant byte moet anders zijn. De volgende 4 adressen volgen uit dit principe:

- F0F0A
- F0F0B
- F0F0C
- F0F0D

Doordat de Least significant byte anders is zijn dit vier geldige adressen.

Met de volgende functies worden waarden van de RF24 radio ingesteld:

Functie	Methode
Kanaal instellen	void RF24::setChannel (uint8_t channel)
Zendvermogen	void setPALevel (rf24_pa_dbm_e level)
Aantal pogingen van ART instellen	void RF24::setRetries (uint8_t delay, uint8_t count)
Adres van de zender instellen	void openWritingPipe (uint64_t address)
Adres van de ontvanger instellen	void openReadingPipe (uint8_t number, uint64_t address)

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();
  }
}
```

D Code Hop

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

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

const uint64_t pipes[4] = { 0xF0F0F0F0E1LL, 0xF0F0F0F0D2LL, 0xF0F0F0F0C3LL, 0xF0F0F0F0B4LL};
const uint8_t readingPipeSender = 1;
const uint8_t readingPipeReceiver = 2;
uint8_t receivedFrom;

bool listen_receiver = true;
```

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

  radio.begin();
  radio.setRetries(15,15);
  radio.setChannel(48);
  radio.setPayloadSize(255);
  radio.setPALevel(RF24_PA_MIN);
  radio.setDataRate(RF24_2MBPS);

  radio.openReadingPipe(readingPipeSender,pipes[0]);
  radio.openReadingPipe(readingPipeReceiver,pipes[3]);

  //
  // Start listening
  //
  radio.startListening();
  radio.printDetails();
}
void loop(void){
     //
  // Pong back role.  Receive each packet, dump it out, and send it back
  //
  // Message from sender
  if ( radio.available( &receivedFrom ) )
  {
    if(receivedFrom == readingPipeSender){
      // Dump the payloads until we've gotten everything
      unsigned long message;
      bool done = false;
      while (!done)
      {
        // Fetch the payload, and see if this was the last one.
        done = radio.read( &message, sizeof(unsigned long) );

        // Spew it
        printf("Got payload %lu...",message);

        // Delay just a little bit to let the other unit
        // make the transition to receiver
        delay(5);
      }
```

```
      // First, stop listening so we can talk
      radio.stopListening();
      radio.openWritingPipe(pipes[2]);
      // Send the final one back.
      radio.write( &message, sizeof(unsigned long) );
      printf("Relayed message to receiver.\n\r");
      // Now, resume listening so we catch the next packets.
      radio.startListening();
    }else if(receivedFrom == readingPipeReceiver){

      // Dump the payloads until we've gotten everything
      unsigned long message;
      bool done = false;
      while (!done)
      {
        // Fetch the payload, and see if this was the last one.
        done = radio.read( &message, sizeof(unsigned long) );

        // Spew it
        printf("Got payload %lu...",message);

        // Delay just a little bit to let the other unit
        // make the transition to receiver
        delay(5);
      }

      // First, stop listening so we can talk
      radio.stopListening();
      radio.openWritingPipe(pipes[1]);

      // Send the final one back.
      radio.write( &message, sizeof(unsigned long) );
      printf("Relayed message to sender.\n\r");
      // Now, resume listening so we catch the next packets.
      radio.startListening();
    }
  }
}
```