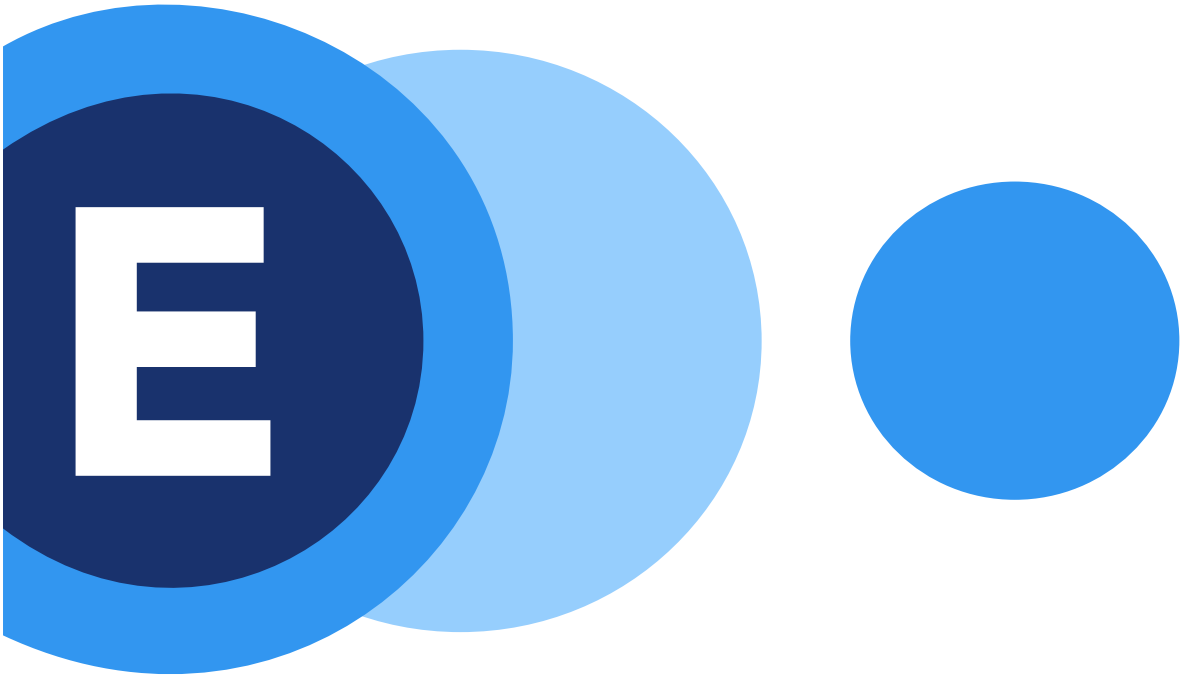
SallandElectronics

WAVENED WSN FOR SMART HOME EXPERIMENTS

SPECIFICATION



Document information

Title	WaveNed WSN for smart home experiments
Subject	Specification
Current version	0.2
Status	preliminary release
Author	Joost van Velzen
Project id	N/A
Customer	WoTS 2016
Filename	WaveNed WSN for smart home experiments.doc
Document Id	WOTS2016-WSN1
Template	SE_EN_basic.dot
Last updated	28-Sep-16 08:07:00 by Joost van Velzen.
Number of pages	13, including front page and this information page.

Revision history

Date	Version	Status	Author	Description	Distribution
06-06-2016	0.1	Preliminary	JvV	First release	External
16-09-2016	0.2	Draft	JvV	Removed x,y,z from long message	External

CONTENTS

1 INTRODUCTION

Dear hacker, let me introduce myself: I am Joost van Velzen. I run the innovation department at SallandElectronics, which basically means that my job is one long hackathon. I often do large scientific experiments with cutting edge technology together with universities. Last December we did a very exciting experiment with 1000 wireless bracelets. You get to win 16 of these bracelets pre-programmed with custom firmware, specially designed for hacking together a cool smart home solution.

I cannot open up the source of the firmware (at this moment), so I tried to give you the most low level interface I could come up with, which means you get maximum freedom to do whatever you like with these.

Let me be very clear about this: The sensornetwork is experimental hardware and software and quite unique. You can't buy this: the only way to get these is to win the hackathon…

Good luck!

Joost van Velzen

2 OVERVIEW

The kit is a wireless sensornetwork running the WaveNed fusion MAC. The kit consists of:

- 4 Sniffers (to get the sensor data to a computer)
- 8 Anchors labeled 1 to 8 (for indoor localization, point of interest tagging or presence detection)
- 4 Wristbands labeled A, B, C and D (for wearing)
- 4 Micro usb cables for charging / connecting the sniffer.

All the devices are wristbands with different firmwares. Only A, B, C and D are intended to be worn.

The wristband is a slapband type. There is no battery, but instead the devices have a supercap that charges to full capacity in roughly 90 seconds.

Installation

I will not be pre-installing the anchors: you get to choose where to put them and how you want to use them. My recommendation would be to first just have one wristband, one sniffer and two anchors per team to play around with and do some desk testing, but if you keep things to yourself and don't share the hardware during experiments, then you are going to be limiting what you can do with the sensornetwork. So, I highly recommend you share with the other teams for bigger experiments (so you can win their hardware in the end ☺).

Charging, connecting and leds

Please take care not to bend the USB cable when it is plugged into the device: It is an SMT micro-usb port and yes you can break it if you are not careful. No leds indicate an empty supercap. The anchors have a red LED only. The wearable nodes use red and orange and the sniffers also use green. The sniffers don't have a supercap so they don't charge. The other devices need to charge 90 seconds to reach full capacity.

3 INTERFACE

I have kept the interface deliberately simple:

Each wristband will transmit roughly 11 messages per second:

- 10 of these messages contain accelero and button data
- One message also includes RSSI measurements of the other wristbands and the anchors

The sniffer will also add an RSSI measurement of the received message.

3.1 Serial port settings

Set the serialport to 460800, 8N1, no flow control.

3.2 Serial data format.

The line starts with LF and ends with CR. Thus CR is actually the start character.

The lines look as follows:

```
Cz>2FF7F00
```

```
Cz:A1B2D3E0F1G2H3I0J1K2L3
```

The difference between the two lines is the length, but it is also identified by the third character: a short line has a greater than sign '>', and a long line has a colon ':'. Other than that the first four characters have the same meaning in both formats.

Each wristband sends the short line roughly 10 times per second and the long line just once per second.

The first character identifies the wristband. There are four and they are identified by A, B, C or D. The line in the example is sent by wristband C.

Each line has signal strengths that are sent as base64 chars. They are coloured red and green below:

```
Cz:A1B2D3E0F1G2H3I0J1K2L3
```

The green char indicates the signal strength with which the sniffer received the message. Value 0 (base 64 'A') is the strongest and value 63 (base64 '/') is the weakest signal, this is because the value is actually negative and has an offset about 30 that was removed for "compression".

The red chars are the signal strengths that the wristband measured from the anchors and other wristbands. In this example I put in a base64 char that corresponds to the id of the received device. So, the signal strength for wristband A is in position 5. The anchors follow the wristbands and I put E to L in as signal strength to identify them. Now as this is a message from wristband C it is obviously never going to measure any signal from itself and therefore it is omitted in the data. Have a guess what this implies for the formats send by the other three wristbands.

Following the signal strength is a 2bit number (0,1,2,3), this indicates how many seconds ago the signal strength was measured. 0 is this round, 1 is one second ago, 2 is two seconds ago and 3 is "please ignore me" as the data is simply old or even never measured.

The character in position 4 of the short message (it is a 2 in this example) indicates the state of the button. The value is 0 if the button is currently pressed. Any higher value indicates that the button is not currently pressed, but it will indicate how many messages ago the button was last pressed. Thus 1 indicates the button was pressed during the last. Two indicates it was two messages ago etc. the value is 3 bits, thus, if the value is 7, then the button was either pressed 7 messages ago or not at all.

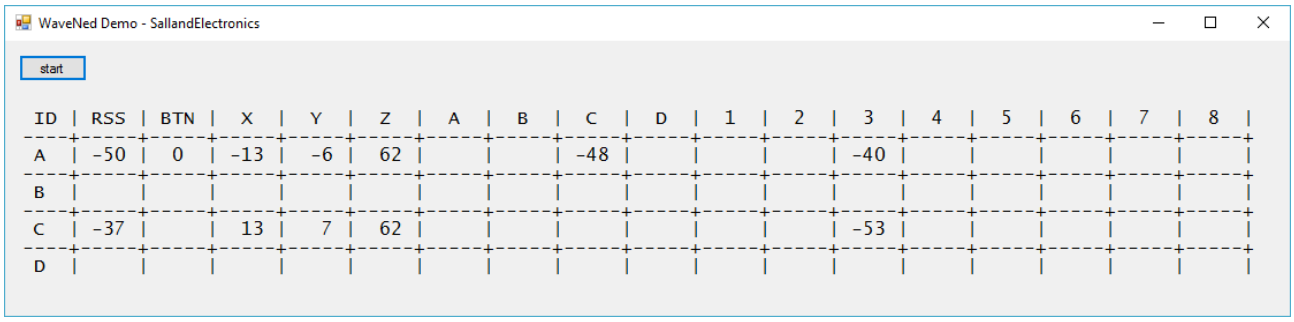
The last 6 characters of the short message are accelerometer data. The values correspond to X,Y and Z and use two hexadecimal characters per value to create a single two's complement byte value.

```
Cz>2FF7F00
```

So in the example above the values coloured Red-Green-Blue are X-Y-Z in that order and they correspond to -1, 127, 0. To be able to detect the orientation in 3D it is easiest to have a range of 2G. So 2G corresponds to 127. As we always have 1G around to keep us from being flung off this spinning planet, this means that you can quite easily detect the orientation of the device and you may even be able to pull off some slow motion gesture-based interfacing.

Now this is 2.4GHZ RF technology and at a venue like the WoTS there's going to be a lot of WiFi, Bluetooth etc. sending interfering messages. This isn't a problem, because the WSN was designed to cope with that and the mac can handle a lot of missed messages. However you need to implement some robustness into your application to cope with this: The sniffer may miss a lot of messages, especially if you use just one sniffer.

4 DEMO APPLICATION



The demo application connects to the sniffer via a serialport over USB and shows the data that is received.

In this case wristbands A and C are active and in range and we can see the data.

The button on wristband A is currently pressed.

Both wristbands show roughly 1G on the Z axis. Which isn't surprising as they are currently lying on the table.

Wristband C is very close to the sniffer, wristband A is a bit further away but still within one meter. The RSS is in dBm, which is a logarithmic scale. The wristbands are optimized for indoor localization and therefore the range is deliberately short so we get more accuracy out of the RSSI value. As a result you get to -80 at about 10 meters.

Wristband A has received a message from wristband C. Wristband C is next to the sniffer which is why the RSS for wristband A is almost identical to the strength of the message from wristband C.

Wristband C has not detected wristband A yet. This can happen, though typically the detection of wristband A will happen within a few seconds. However this illustrates that you should not assume that a measurement is present in the data of both nodes. Your code should work fine when just one wristband has detected the other.

Note that only the wristbands with ids A, B, C and D send data to the sniffer. The anchors with id 1-8 do not send any data to the sniffer.

4.1 Demo source

The code below can be copy-pasted into a new windows forms application created with visual studio and targeted at the .net framework 2.0.

You can also use it on linux or macOs using a different editor such as monoDevelop. You will need mono to run the executable on a platform other than windows.

You can also download this code or the executable from the hackathon github.

```
// Hi there!
//
// This is a simple demo to show you how to interface the sensornetwork sniffer.
// I have kept things extremely simple and hackable.
// You will not find any exception handling and you may find some ugly constructions that will however be very easy to read.
// I have set the target framework to .Net 2.0 so this code will work on pretty much any MONO version, including old buggy versions.
// So Yes, this will run on Linux, Raspberry Pi, Mac etc.. (oh, and Windows)
//
// Happy hacking!
//
// Joost van Velzen / SallandElectronics


using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;

// added
using System.IO.Ports;
```

```
namespace WaveNedDemo
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        SerialPort  com = null;
        DateTime    lastReport;
        Wsn         wsn;

        List<Button> ports = new List<Button>();

        private void button1_Click(object sender, EventArgs e)
        {
            // close com
            if ( com != null )
            {
                if (com.IsOpen)
                {
                    bwReadCom.CancelAsync();
                }

                com.Dispose();
                com = null;
            }

            // remove any buttons we created

            foreach (Button btn in ports)
            {
                Controls.Remove(btn);
            }

            ports = new List<Button>();

            int lastX = button1.Right;

            String[] portNames = SerialPort.GetPortNames();

            // add a button for each port, a bit cumbersome but we avoid using some of the controls that don't work on older verisons of mono
            if (portNames.Length > 1)
            {
                foreach (string portName in portNames)
                {
                    Button btn = new Button();
                    btn.Text = portName;
                    btn.Top = button1.Top;
                    btn.Left = lastX + 5;
                    btn.Width = button1.Width;
                    lastX = btn.Right;
                    btn.Click += portButton_Click;
                    btn.Font = button1.Font;

                    ports.Add(btn);
                    Controls.Add(btn);
                }

                label1.Text = "click on a button to select a com port";
            }
            else
            if ( portNames.Length == 1 )
            {
                com = new SerialPort(portNames[0], 921600, Parity.None, 8, StopBits.One);

                // set end of line to be \r only
                com.NewLine = "\r";

                com.Open();

                // this way we wait at least a tenth of a second so it is likely we will have sufficient data to show
                lastReport = DateTime.Now;

                wsn = new Wsn();

                bwReadCom.RunWorkerAsync();

            }
            else
            {
                label1.Text = "No COM port found...";
            }

        }

        private void portButton_Click( object sender, EventArgs e)
        {
            // alternatively you can download the FTDI C# wrapper for their driver, which works better if you are using windows
            com = new SerialPort( ((Button)sender).Text, 921600, Parity.None, 8, StopBits.One);

            // set end of line to be \r only
            com.NewLine = "\r";

            com.Open();
```

```
    // this way we wait at least a tenth of a second so it is likely we will have sufficient data to show
    lastReport = DateTime.Now;

    wsn = new Wsn();

    bwReadCom.RunWorkerAsync();

    // remove any buttons we created
    foreach (Button btn in ports)
    {
        Controls.Remove(btn);
    }

    ports = new List<Button>();
}

// Yeah yeah, I know.. This is a very very bad and highly inefficient example... but it works :-)
private int Base64(char c) { return "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/=".IndexOf(c); }
private int Hex(   char c) { return "0123456789ABCDEF".IndexOf(c); }
private int WbId(  char c) { return "ABCD".IndexOf(c); }
private int WbBtn( char c) { return "0123456".IndexOf(c); } // note: 7 is a valid value, but it means the button is not pressed
private int WbTtl( char c) { return "012".IndexOf(c); }     // note: 3 is a valid value, but it means that the rssi is invalid, so...

private int TwosComplement( int uint8 ) { return (uint8 >= 128) ? (-256 + uint8) : uint8; }

private void bwReadCom_DoWork(object sender, DoWorkEventArgs e)
{
    while ( !bwReadCom.CancellationPending )
    {
        // read a string
        string data = com.ReadLine(); // not handling exceptions...

        // start of line is \n, remove anything before it
        int startPos = data.IndexOf('\n');
        if (startPos != -1) data = data.Substring(startPos + 1);

        // get the time
        DateTime now = DateTime.Now;

        // parse
        if ( ( data.Length == 10 ) && ( data[2] == '>' ) )
        {
            // probably a correctly formatted short line
            int id = WbId(data[0]);
            if (id != -1)
            {
                WristBand wb = wsn.wristband[id];
                wb.lastShort = now;
                wb.strength = Base64(data[1]);
                wb.button = WbBtn(data[3]);
                wb.x = TwosComplement((Hex(data[4]) * 16) + (Hex(data[5])));
                wb.y = TwosComplement((Hex(data[6]) * 16) + (Hex(data[7])));
                wb.z = TwosComplement((Hex(data[8]) * 16) + (Hex(data[9])));
            }
        }
        else
        if ( ( data.Length == 25 ) && ( data[2] == ':') )
        {
            // probably a correctly formatted long line
            int id = WbId(data[0]);
            if (id != -1)
            {
                WristBand wb = wsn.wristband[id];
                wb.lastLong = now;
                wb.strength = Base64(data[1]);

                // read wristbands with a lower id
                for (int i = 0; i < id; i++)
                {
                    wb.rssi[i] = Base64(data[3 + (2 * i)]);
                    wb.ttl[i] = WbTtl(data[4 + (2 * i)]);
                }

                // clear the id of this wristband: it has no data for itself
                wb.rssi[id] = -1;
                wb.ttl[id] = -1;

                // read wristbands with a higher id and then all the anchors
                for (int i = id; i < 11; i++)
                {
                    wb.rssi[i + 1] = Base64(data[3 + (2 * i)]);
                    wb.ttl[i + 1] = WbTtl(data[4 + (2 * i)]);
                }
            }
        }

        // check if we need to update the screen
        if ( ( now - lastReport )  >= TimeSpan.FromMilliseconds( 100 ) )
        {
            // last report was more than 100 ms ago!
            lastReport = now;

            // create a snapshot of the wsn state and throw it to the UI thread...
            Wsn wsnCopy = wsn.CreateSnapshot( now );
            bwReadCom.ReportProgress(1, wsnCopy);
        }
    }
```

```
        com.Close();

        if (bwReadCom.CancellationPending) e.Cancel = true;
    }

    private void bwReadCom_ProgressChanged(object sender, ProgressChangedEventArgs e)
    {
        // This is happening on the UI thread, we can update the form here!
        Wsn wsn = (Wsn)(e.UserState);

        label1.Text = " ID | RSS | BTN |  X  |  Y  |  Z  |  A  |  B  |  C  |  D  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |\r\n" +
                      "----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+\r\n" +
                      " A  |" + wsn.wristband[0].ToString() + "\r\n" +
                      "----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+\r\n" +
                      " B  |" + wsn.wristband[1].ToString() + "\r\n" +
                      "----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+\r\n" +
                      " C  |" + wsn.wristband[2].ToString() + "\r\n" +
                      "----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+\r\n" +
                      " D  |" + wsn.wristband[3].ToString() + "\r\n";
    }
}

// Class to store the data in (also used as a snapshot)
public class Wsn
{
    public WristBand[] wristband = { new WristBand(), new WristBand(), new WristBand(), new WristBand() };

    public Wsn CreateSnapshot( DateTime now )
    {
        Wsn newWsn = new Wsn();

        for (int i = 0; i < 4; i++) wristband[i].SaveSnapshot(now, newWsn.wristband[i]);

        return newWsn;
    }
}

public class WristBand
{
    public int button;

    // first the 4 wristbands, then the 8 anchors. wristbands include the wristband itself with ttl 3 and strength 0
    public int[] rssi = new int[ 12 ];
    public int[] ttl = new int[12];

    public int x;
    public int y;
    public int z;
    public int strength;
    public DateTime lastShort;
    public DateTime lastLong;

    public void SaveSnapshot( DateTime now, WristBand copy )
    {
        copy.lastShort = lastShort;
        copy.lastLong = lastLong;

        copy.strength = -1;

        if ( ( now - lastLong ) <= TimeSpan.FromSeconds( 3 ) )
        {
            // the last long message is still valid
            copy.strength = strength;
            copy.rssi = (int[])(rssi.Clone());
            copy.ttl = (int[])(ttl.Clone());
        }
        else
        {
            // the last long message is no longer valid
            copy.ttl = new int[] { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 };
        }

        if ((now - lastShort) <= TimeSpan.FromSeconds(3))
        {
            // the last short message is still valid
            copy.button = button;
            if ( lastShort > lastLong ) copy.strength = strength;
            copy.x = x;
            copy.y = y;
            copy.z = z;
        }
        else
        {
            // the last short message is no longer valid
            copy.button = -1;
            copy.x = int.MinValue;
            copy.y = int.MinValue;
            copy.z = int.MinValue;

        }
    }

    public override string ToString()
    {
        // " RSS | BTN |  X  |  Y  |  Z  |  A  |  B  |  C  |  D  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |"

        string txt = " " + ( ( strength != -1 ) ? ( - 1 * ( strength + 30 ) ).ToString() : "   " ) + " |" +
                     "  " + ( ( button != -1 ) ?  button.ToString() : " " ) +                      "  |" +
```

```
                    ( ( x != int.MinValue ) ? string.Format( "{0,4}", x.ToString() ) : "    " ) +   " |" +
                    ( ( y != int.MinValue ) ? string.Format( "{0,4}", y.ToString() ) : "    " ) +   " |" +
                    ( ( z != int.MinValue ) ? string.Format( "{0,4}", z.ToString() ) : "    " ) +   " |";

            for ( int i = 0; i < 12; i++ )
            {
                txt += " " + ((ttl[i] != -1) ? (-1 * (rssi[i] + 30)).ToString() : "   ") + " |";
            }

            return txt;
        }
    }
}
```

5 THAT'S ALL FOLKS…

So that's all the technical details you need to know, as I said it is nicely low level. There is no way to control the leds on the wristband, but if you need lights, then head over to my description of this year's WoTS gadget. You'll find that you have plenty of lights to play with ☺.

Happy hacking and good luck!

Joost van Velzen

P.s. This specification is subject to change without prior notice (you'll probably notice when you start testing).