UNIVERSITÄT DUISBURG-ESSEN
FAKULTÄT FÜR WIRTSCHAFTSWISSENSCHAFTEN
INSTITUT FÜR INFORMATIK UND WIRTSCHAFTSINFORMATIK
LEHRSTUHL FÜR PERVASIVE COMPUTING

Bachelorarbeit

Evaluation of a lightweight debugging platform for mobile sensor networks

Michael Krane
Matrikelnummer: 2233018

Networked Embedded Systems Group
Institut für Informatik und Wirtschaftsinformatik
Fakultät für Wirtschaftswissenschaften
Universität Duisburg-Essen

22. September 2015

Erstprüfer: Prof. Dr. Pedro José Marrón
Zweitprüfer: Prof. Dr. Gregor Schiele
Zeitraum: 1.Juli 2010 - 28.September 2010

Inhaltsverzeichnis

Abbildungsverzeichnis

Abstract

In this thesis we evaluate SHAMPU, a test bed for WSNs developed at the University of Duisburg-Essen. SHAMPU is a platform, which allows the debugging and reprogramming of an attached sensor node. Our focus for this evaluation is the wireless communication between SHAMPU nodes, for which the ANT protocol is used. We design and run several different experiments in order to determine the capabilities of the ANT-chip used.

Kapitel 1.

Introduction

In the current years the number of Cyber-physical systems (CPSs) has drastically increased. This is mostly due to the emergence of the Internet of Things (IoT), as well as the existence of several user friendly kits, like the RaspberryPi or the Arduino. With this new trend new challenges are created in the monitoring and debugging of these devices. This is especially difficult for applications, where the sensors are deployed in a hard to reach area. In addition to this, the deployed device can often fulfil a multi-purpose role and might need to be reconfigured for different tasks.

In order to address both these problems SHAMPU(Single chip Host for Autonomous Mote Programming over USB)[SSMM14] was developed at the University Duisburg-Essen. SHAMPU was designed with the goal of being as small, lightweight and energy efficient as possible. In order to use SHAMPU as a test bed, it needs to be attached to an existing node. SHAMPU is then capable to not only monitor and debug the attached node, but also to program it. These reprogramming capabilities make it possible to reprogram an attached sensor, without having to physically be near it. To achieve this each SHAMPU node is equipped with a wireless interface. In the current configuration an ANT[Dyn13] radio chip. All SHAMPU nodes form a Wireless sensor network (WSN) of their on, which allows not only the communication between the nodes, but also a connection to a base station. This base station not only acts as a central data sink for the whole network, but is also able to push commands and data to the SHAMPU nodes.

In this thesis we evaluate the wireless capabilities of the SHAMPU framework. For this purpose we develop use-cases for the different tasks that the SHAMPU framework can perform: Program a device, collect data during operation and interact with the system itself during runtime. For each of the test cases we design and run different experiments with the goal to assess if the chosen wireless technology is able to fulfil all the use cases.

1.1. Related work

There are already several different WSN test beds available. Each of these test beds were designed to fulfil a different role.

Some test beds like FlockLab [LFZ13] allow to attach a wide array of different sensors and use a JTAG interface to be able to precisely capture debugging and timing information.

However, these types of test beds rely on a wired connection to interface with the test bed itself. That makes it exceedingly difficult to test and debug already deployed WSN, since it might not be possible to easily get to the sensor in the planted location.

Other available solutions address this problem by attaching the test bed directly to the node and use a wireless connection to communicate with the test bed. An example for this is Sensei-UU [RHF+09] which uses a wireless 802.11 network. This allows the WSN to be tested and debugged while it is deployed. One drawback of this solution is the power draw of 802.11 devices, which is huge, compared to other technologies. If the node itself is run of an external power source that isn't a problem, but the use of a battery to power the node will hardly be feasible.

To address this power issue, it is possible to use a different technology which offers a lower power mode. Two examples for this are BTNodes [MK] or Smart-Its [KL00], which use a Bluetooth connection. The structure of the network itself is similar to Sensei-UU: each node in the network has its own test bed attached and the test beds communicate with a central base station.

The main problem with Bluetooth is the limited size of the network, which makes it difficult to set up and use more complex network topologies. Also Bluetooth communication is always 1:1 and does not allow broadcasts. The newest version of Bluetooth addresses the network size with the introduction of ScatterNets, but the setup and maintenance of the network still remains challenging.

Kapitel 2.

Technical Background

2.1. SHAMPU

SHAMPU (Single chip Host for Autonomous Mote Programming over USB) [SSMM14] is a WSN testbed which allows remote debugging and reprogramming of sensor nodes. Its main advantages over other testbeds (see section 1.1) are portability, low cost, small size and low energy consumption. SHAMPU is used as an extension to an already existing sensor node. The only requirement is that the node provides an USB-Interface which is connected to SHAMPU. This single connection to the attached node makes SHAMPU completely OS independent. The SHAMPU framework (see Figure 2.1) itself is split

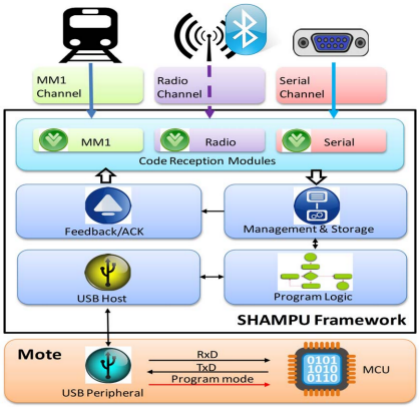

Abbildung 2.1.: Overview of the SHAMPU Framework [SSMM14]

into multiple modules. The most important part for this thesis is the Code Reception Module, which allows the use of different protocols to connect to and communicate with the SHAMPU device. One option for the wireless communications with the SHAMPU device is the ANT protocol, on which we will focus in this evaluation.

2.2. ANTAP1MxIB RF

In order to use the ANT protocol each SHAMPU-mote is equipped with an ANTAP1MxIB RF Transceiver Module. The module was chosen because of its small form factor (20mm x 20mm) and its very low power draw. The ANTAP1M can handle up to 4 different ANT channels with a combined message rate which ranges between 0.5Hz and 200 Hz.

Abbildung 2.2.: SHAMPU base station

In order to more easily test the capabilities of the ANT chip, we use a SHAMPU base station for all the experiments. This base station (see Figure 2.2) contains the ANTAP1MxIB and a serial interface, which allows the ANT to asynchronously communicate with a PC over RS-232.

2.3. ANT

ANT [Dyn13] is a wireless protocol which operates in the 2.4 GHz ISM Band. It was originally developed in 2003 by Dynastream Innovations Inc. for the use in wireless sensors. The ANT protocol is designed for the use in low power WSNs focusing on scalability and ease of use.

One of the advantages ANT has over other protocols, such as Bluetooth or ZigBee, is the high level of abstraction the ANT Protocol provides.

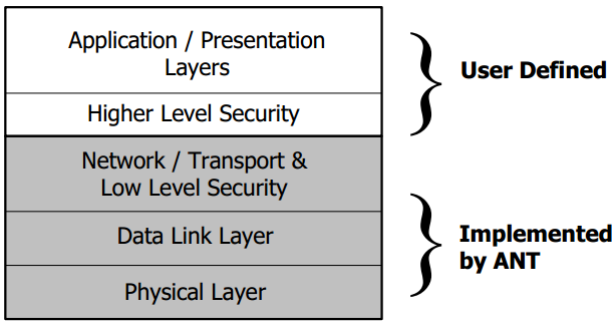

Abbildung 2.3.: OSI-Layer vs. ANT Protocol[Dync]

This level of abstraction is achieved by incorporating the first 4 OSI-layers (see Figure 2.3) into the ANT protocol, thus allowing even low-cost MCU to set up and maintain complex wireless networks, since all the details of the communication are handled by the ANT-chip.

2.3.1. ANT Topology

In order for the ANT protocol to work each mote needs to be part of a network. As shown in figure 2.4 the ANT protocol can be used to create simple or considerably more complex networks. Each mote inside a network is called an ANT node. In order for two nodes to communicate with each other they need to be connected via a channel.

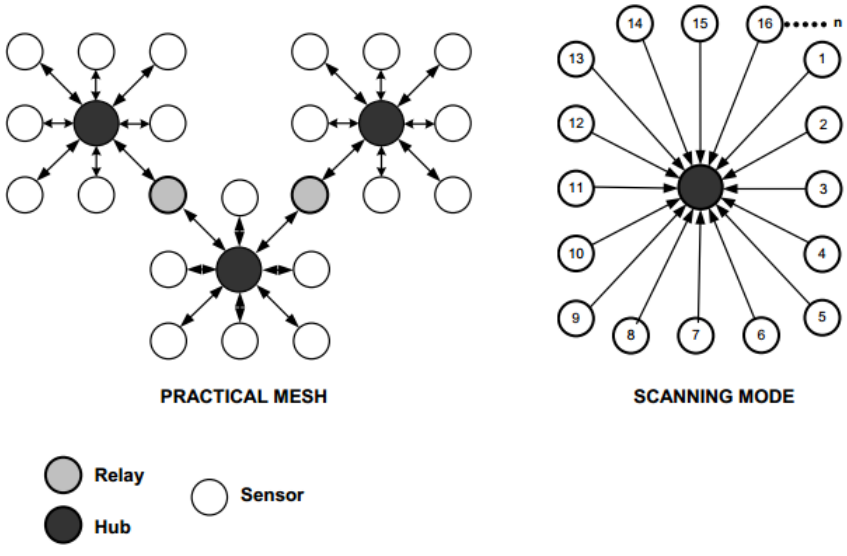

Abbildung 2.4.: Example ANT Topologies[Dyn13]

2.3.2. ANT Channels

Most of the available channel types are bidirectional. The ANT protocol, however, still differentiates between master and slaves nodes. While a master node mostly sends data and a slave node mostly receives data, the slave retains the ability to respond to the incoming data.

The ANT protocol knows 125 different channels, each of them 1 MHz wide. Each Channel can support a data rate of 1 Mbps and up to 65533 nodes. To avoid interference between the channels isochronous self adjusting TDMA technology is used, which allows the ANT nodes to change the transmit timings as well as the frequency that is being used for the current channel. The use of the TDMA is completely handled by the ANT protocol, which means the user has no control over the process.

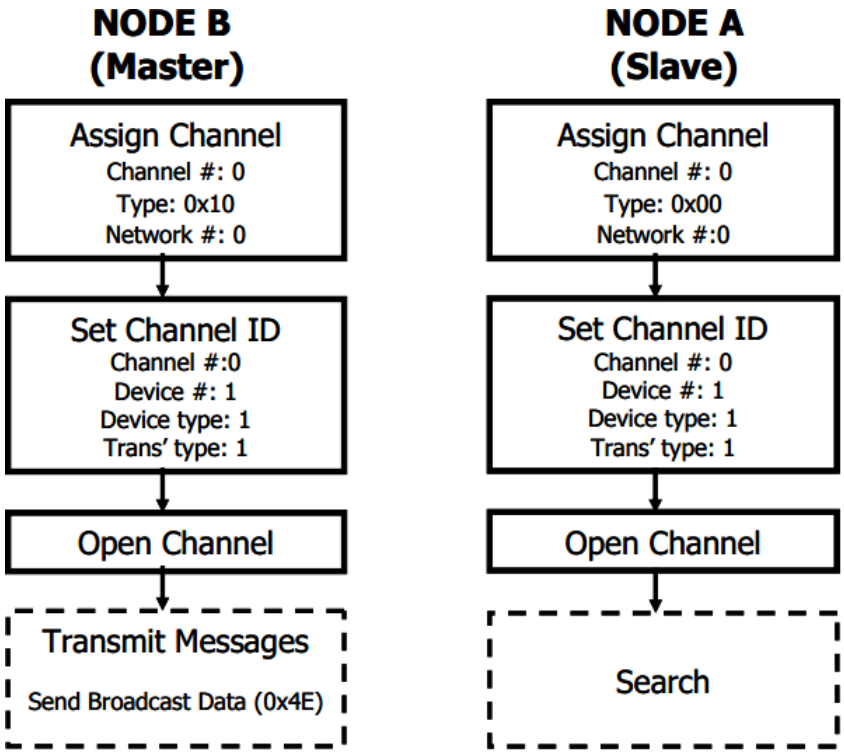

Abbildung 2.5.: ANT Channel Establishment[Dyn13]

Figure 2.5 shows the procedure of opening up a channel between two ANT nodes. In the first step the Channel configuration is set, most important in this context being the channel type: 0x00 signifies that the node is a slave while 0x10 means that the node acts as a master. The next step is to set the Channel ID. Here the slave node needs to set the values of the master device in order to correctly connect. It is possible, however,

to set wildcard values, which allows the slave to connect to any device sending on the same frequency. Before the final step, it is optionally possible to set the frequency of the channel and the message period. These steps are not mandatory though, since ANT defaults to fixed values for these: 2466 Hz transmission frequency and a message period of 8192. The last step constitutes of the opening of the channel itself, where care should be taken that the master opens the channel before it is opened by the slave device.

The ANT protocol distinguishes between two different Channel types:

Independent Channels
Independent Channels are used if there is only one node which transmits data. There is no limit to the amount of slave devices which receive the messages that are being sent out. Furthermore, the message being sent out is broadcast to all nodes, it is not possible to address only specific nodes.

Shared Channels
Shared Channels are used if there is more than one node which sends data. This type of channel is facilitated by the use of a Shared Channel Address, which in turn reduces the amount of data that can be transmitted at a time. All ANT nodes still receive every message, but only forward messages which have a matching address. The Channel master can decide to either use one or two bytes as the address, which allows for either 255 or 65535 slave devices in the same channel.

2.3.3. ANT Communication

The ANT protocol supports three different data types: broadcast, acknowledge and burst. The data type is not part of the channel configuration, thus channels are able to use any combination of data types. The only exceptions are unidirectional channels, which can exclusively send broadcast data. The various data types differ in the way data is handled and transmitted.

- Broadcast data
  Broadcast data represents the most basic data type and, at the same time, the default. To start a broadcast transmission, the command needs to be issued just once, since the last sent packet is periodically resent as a broadcast. It does not matter if the last packet was part of a burst or acknowledge transmission, if no new data is available this packet is resent. Figure (MISSINGPIC) shows, how each broadcast transmission is aligned to the channel message period. Since there is no answer from the receiving node, it is not possible to determine if the packet was transmitted correctly.
- Acknowledge data
  Acknowledge data can be used to make sure a node has received a transmitted packet. After receiving an acknowledge packet the node will send a message back

to the sender. Acknowledge data can only be used with bidirectional channels, yet both the master and the slave can use it. Figure (MISSINGPIC) shows that, just like broadcast data, acknowledge data is always aligned to a time slot, yet the answer from the receiver is sent immediately back to the sender.

- Burst data
  Burst data provides a method to quickly transmit large amounts of data. This is achieved by ignoring the normal channel time slots and sending the packets immediately one after the other. This allows for a transmission rate of up to 20 kbps, which is much higher than other data transmission types. Similar to acknowledge data at the end of the transmission the sender is informed if the transfer failed or succeeded. The drawback of this method is, that burst data is prioritized over all other transmissions and will interrupt other transmissions over the same channel.

2.3.4. ANT messages

In the ANT protocol each message has the basic format as specified in Figure 2.6. Each

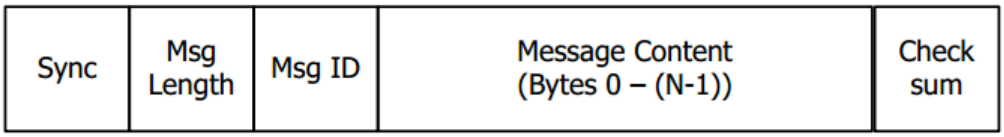

Abbildung 2.6.: Ant message structure[Dyn13]

message starts with a special Sync-Byte and ends with a checksum, which is calculated by xoring all previous bytes. The Msg Length byte is the number of Message Content bytes. The Msg ID byte specifies which kind of data is contained in the message. The ANT protocol also provides an extended message format, which allows to attach further information to each message. The maximum length of the message content is 8 for each of the three data types.

Kapitel 3.

Evaluation of SHAMPU

In order to assess the capabilities of SHAMPU, we plan and run experiments designed to evaluate the SHAMPU framework according to the following use-cases:

- **Scheduled data-transmission**
  In order for SHAMPU to work as a debugging and logging platform, the base station needs to periodically receive data from all the nodes in the network. ANT provides two different data types which can be used to transport data in this case: Broadcast data and Acknowledge data. The experiments designed for this use case try to determine the maximum throughput for each of the two transfer modes.
- **Unscheduled data-transmission**
  There are several possible cases, where it is not feasible to use a scheduled data-transmission. Instead, the burst mode must be used, which allows to transmit data at a much higher rate. This use case covers several different scenarios, e.g.:
  - Reprogramming of a node: SHAMPU is able to reprogram the attached node. For this process, SHAMPU needs to receive a new firmware, which can amount to several hundred kB.
  - SHAMPU RAM-Dumps: SHAMPU has 128kB of RAM, which can be used to save collected data during an experiment. At the end of the experiment the complete memory needs to be transmitted back to the base station.
- **Communication range**
  Since SHAMPU is architecture independent it can be used in different situations. Therefore it is important to know, how far away from the base station the nodes can be placed. As SHAMPU tries to be energy efficient, it might also be viable to reduce the range for smaller set ups to save even more energy.

To cover all the mentioned test cases we designed different experiments, which test one or more of the described categories. The following section describes the experiments according to the following template:

Description
A description of the experiment and the category being evaluated.

Use-Case
The use-case which the experiment tries to test.

Network Topology and Configuration
A diagram of the network topology in which the experiment is run and pseudo code which describes the program being run on the master and the slave. Default values are not listed.

Testing methodology
A description how the experiment is performed.

Result
The results of the experiment and any additional data collected during the experiment.

3.1. Common experiment parameters

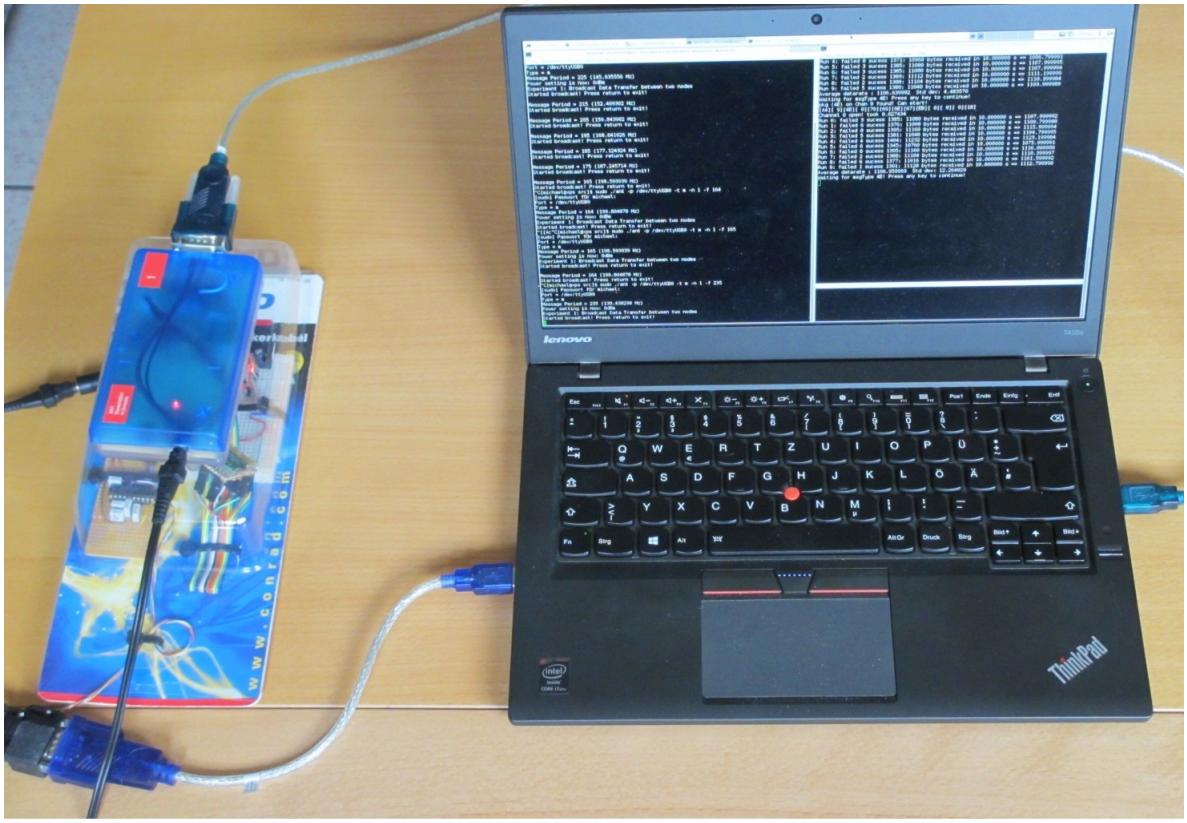

Abbildung 3.1.: Experiment set up

If not otherwise noted in the experiment description each experiment was run in the Mobility Lab of the Networked Embedded Systems group at the University of Duisburg-Essen (SA 327), with the two base stations in the configuration which can be seen in figure 3.1. The following table describes the parameters which are used in the experiments.

Device Number	33
Device Type	1
Transmission Type	1
ID_CHAN1	0
FREQ_CHAN1	2466 Hz
ID_CHAN2	1
FREQ_CHAN2	2477 Hz
Message Period	8192
min_Channel_Period	0x00A5
max_Channel_Period	0xFFFF

Tabelle 3.1.: ANT default configuration

Also, there is an important difference between the message period used in the pseudo code and the frequency used in the figures which display the results. The period describes the size of the gap between two messages, the frequency describes how many messages are sent in 1 second. The channel period p can be used to calculate the frequency of the time slots $f_t = \frac{32678}{p}$.

3.2. Experiment 1: Broadcast Data Transfer between two nodes

Description

Broadcasting is one way of periodically transmitting data between two or more ANT nodes. Since all broadcast packets are synchronized to a fixed time-slot, the data throughput can be increased by decreasing the channel period. The experiment itself is split into two parts. In the first part we try to determine the highest possible data throughput. Also we try to determine if the channel period has an effect on the time it takes for a slave node to find and join an existing channel. The second part is a test of the highest detected data throughput. Here we try to evaluate if there are any variations of the data throughput over a much longer interval.

Use-Case

Scheduled Data-transmission

Network Topology and Configuration

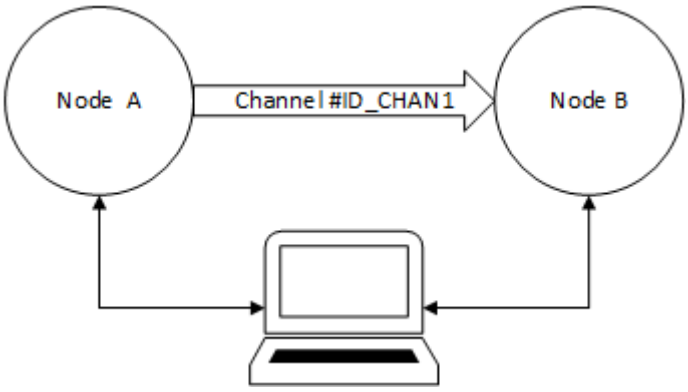

Abbildung 3.2.: Toplogy experiment 1

Network Topology and Configuration

The 2 nodes are placed right next to each other.

Testing methodology

Node A acts as the master and node B as the slave. For both nodes the channel period is set to the highest value and the channel is opened. Node B records how long it takes to join the channel and how many bytes it received over a 100s interval. The measurement is repeated 10 times and the average values are saved. Then the channel period is decreased and the process is repeated. For the second

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod)
ANT_OpenChannel(ID_CHAN1, ANT_Bidirectional_Master)
ANT_SendBroadcastData(ID_CHAN1, [0x01, 0x02, 0x03, 0x04])
wait_for_user_input()
ANT_CloseChannel(ID_CHAN1)
channelPeriod = decreaseChannelPeriod()
}
```

Pseudo code 1: Broadcast data single channel (Master)

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
for (i in 0..9) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod)
ANT_OpenChannel(ID_CHAN1, ANT_Bidirectional_Slave)
count = 0
for (100 seconds)
if (receivedPacket() == ANT_BROADCAST_DATA)
count++;
print (count * 8 / 100) + " Bytes per second"
wait_for_user_input()
ANT_CloseChannel(ID_CHAN1)
}
channelPeriod = decreaseChannelPeriod()
}
```

Pseudo code 2: Broadcast data single channel (Slave)

part, the channel period is set to the value which achieved the highest speed. The experiment is then left running for a period of 10 hours in one continuous run.

Result

Figure 3.3 shows the transmission speeds achieved for the different frequencies. Up to a frequency of 128 Hz, the measured data rate matches the expected theoretical maximum rate. For the highest supported frequency (199.8 Hz), the measured data rate is much lower than the maximum rate. Even if we account for transmission errors the difference is significant. As seen in figure 3.4 the measured data rates for frequencies above 140 Hz all fall short of the expected values. However the average data throughput of these frequencies remains consistently at around 1100 Bps. The experiment was repeated multiple times, running it at different times

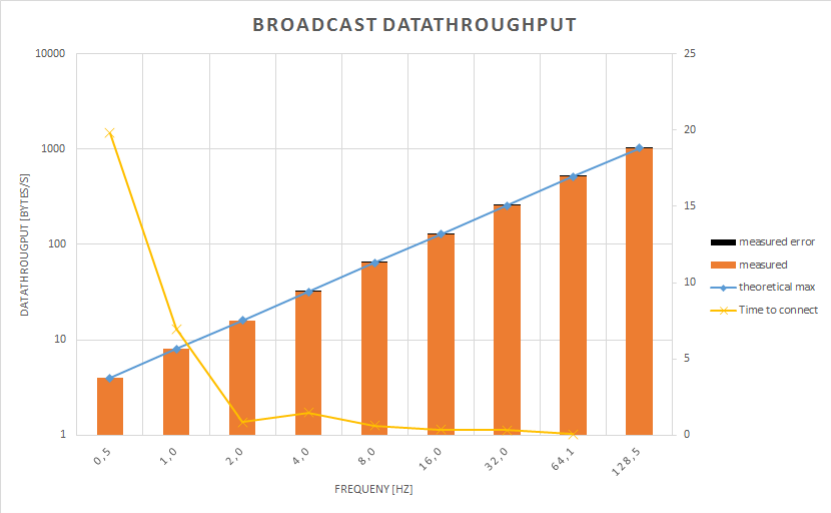

Abbildung 3.3.: Broadcast data rate (0.5Hz - 128Hz)

and places, thus an environmental factor can be excluded. That means, the reason for the upper limit has to be found with the hardware itself. See section 4.1 for a discussion about possible reasons for this upper limit.

The time it takes for a Node to join a channel falls rapidly once the frequency is above 1 Hz. For frequencies above 64 Hz, the time it takes to join a channel becomes negligible, with times around 75 ms (see Figure 3.3). All the measured values are below the specified worst case channel acquisition times of the ANT protocol [Dyna].

Figure 3.5 shows the transmission speed for the highest supported frequency 199.8 Hz over time. The results show that the data throughput stays fairly consistent with a standard deviation of only 30 Bps, or 2.7%. Interesting to note is the time period from 1:00PM to 2:50 PM, where the deviation from the average is much smaller. The exact reason for this phenomenon in unknown, since during this time period the environment of the test set up did not change in any way.

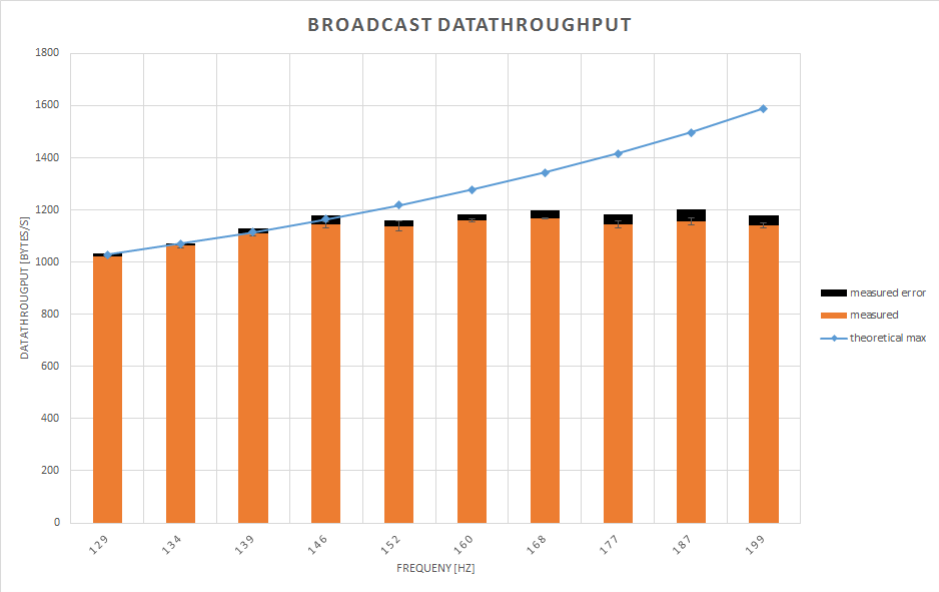

Abbildung 3.4.: Broadcast data rate (129Hz - 199Hz)

3.3. Experiment 2: Broadcast Data Transfer between multiple nodes

Description

In experiment 1 we determined the channel period, which allows for the maximum throughput. In this experiment we try to determine, how the maximum throughput is affected by the amount of channels in the network. SHAMPU needs two channels, so it can work correctly: One channel which sends data from the base station to the nodes in order to control them and another channel by which the nodes can send debugging and other information.

Use-Case

Scheduled Data-transmission

Network Topology and Configuration

Network Topology and Configuration

The two nodes are placed right next to each other.

Testing methodology

In this experiment each node acts as a master for a different channel. Node A is the master for Channel 0 and Node B is the master for Channel 1. The measurements itself are identical to the ones in experiment 1, except that the data is recorded on

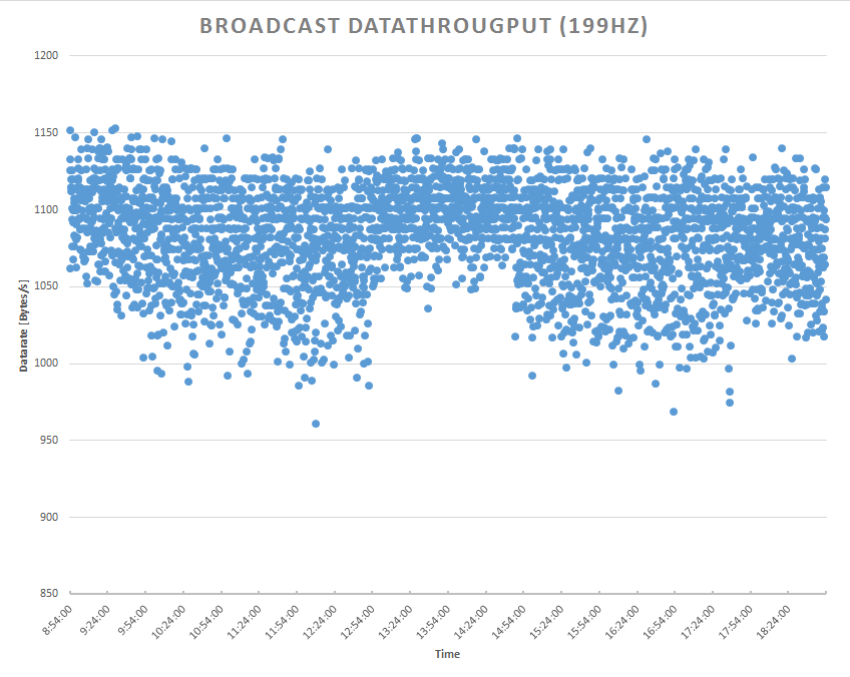

Abbildung 3.5.: Broadcast data rate over time (200Hz)

both nodes. The channel period is decreased until the data throughput decreases, or the connection can no longer be established.

Result

Figures 3.7 shows the results of the measurements, the left bar displaying the throughput for channel 0, the right bar displaying the throughput for channel 1. The blue line shows the theoretical maximum of the data throughput for the given frequency. Up to a frequency of 64 Hz the data throughput increases with the frequency for both channels. However there is a complete drop at 128 Hz. Figure 3.8 shows that somewhere between 72 Hz and 76 Hz the data throughput begins to drop off. The maximum combined data rate for two channels adds up to around 1160 Bps, which is in line with the result from experiment 1. From this data we can assume, that the entire available data rate of 1160 Bps is simply split among all existing channels.

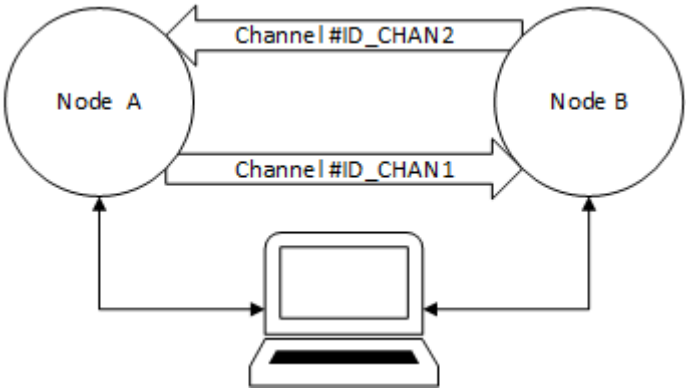

Abbildung 3.6.: Toplogy experiment 2

3.4. Experiment 3: Acknowledge Transfer delay

Description

In this experiment we try to determine the time it takes for a node to receive and acknowledge a transmitted packet. The value is important in order to be able to determine the reaction time of SHAMPU to commands sent by the base station, for example to set up a separate channel for a burst transmission.

Use-Case

Scheduled Data-transmission

Network Topology and Configuration

Testing methodology

Node A acts as the master and node B as the slave. For both nodes the channel period is set to the highest value and the channel is opened. Node A then sends a total of 10 acknowledge messages and measures how long it takes until it receives the acknowledge signal. The channel period is then decreased and the experiment repeated.

Result

Figure 3.10 and 3.11 show the delays for the tested frequencies. The blue line shows the size of the gap between two timeslots and can act as a resolution for each measurement. For the lower frequencies the results are skewed, since each acknowledge packet is aligned to a time-slot. Thus the biggest part of the delay is waiting for the next time-slot. For frequencies greater than 64 Hz the data is considerably more useful, since the resolution is lower than the measured values. The delay for those frequencies is around 18 ms, with no notable difference between

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod)
openChannel(ID_CHAN1, ANT_Bidirectional_Master)
ANT_SendBroadcastData(ID_CHAN1, [0x01, 0x02, 0x03, 0x04])
ANT_SetChannelPeriod(ID_CHAN2, channelPeriod)
openChannel(ID_CHAN2, ANT_Bidirectional_Slave)
count = 0
for (10 seconds)
if (receivedPacket() == ANT_BROADCAST_DATA)
count++
print (count * 8 / 10) + " Bytes per second"
wait_for_user_input()
ANT_CloseChannel(ID_CHAN1)
ANT_CloseChannel(ID_CHAN2)
channelPeriod = decreaseChannelPeriod()
}
```

Pseudo code 3: Broadcast data transfer two channels (Master)

128 Hz and 200 Hz. Since the procedure is the same for the higher channel periods, it can be assumed that the delay is the same as the one for the lower channel periods.

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod)
openChannel(ID_CHAN1, ANT_Bidirectional_Slave)
ANT_SetChannelPeriod(ID_CHAN2, channelPeriod)
openChannel(ID_CHAN2, ANT_Bidirectional_Master)
ANT_SendBroadcastData(ID_CHAN2, [0x01, 0x02, 0x03, 0x04])
count = 0
for (10 seconds)
if (receivedPacket() == ANT_BROADCAST_DATA)
count++
print (count * 8 / 10) + " Bytes per second"
wait_for_user_input()
ANT_CloseChannel(ID_CHAN1)
ANT_CloseChannel(ID_CHAN2)
channelPeriod = decreaseChannelPeriod()
}
```

Pseudo code 4: Broadcast data transfer two channels (Slave)

3.5. Experiment 4: Acknowledge Data Transfer between two nodes

Description

This experiment is almost identical with experiment 1. The main difference is that we use acknowledge data instead of broadcast. It is also important to note that the master records how many successful packets are transmitted. The main goal of the experiment is to see if the data throughput is decreased compared to broadcast data, especially for smaller channel periods.

Use-Case

Scheduled Data-transmission

Network Topology and Configuration

Testing methodology

The testing methodology is the same as experiment 1a, except that the master sends acknowledge messages and waits for the slave to confirm the successful transmission before sending the next packet.

Result

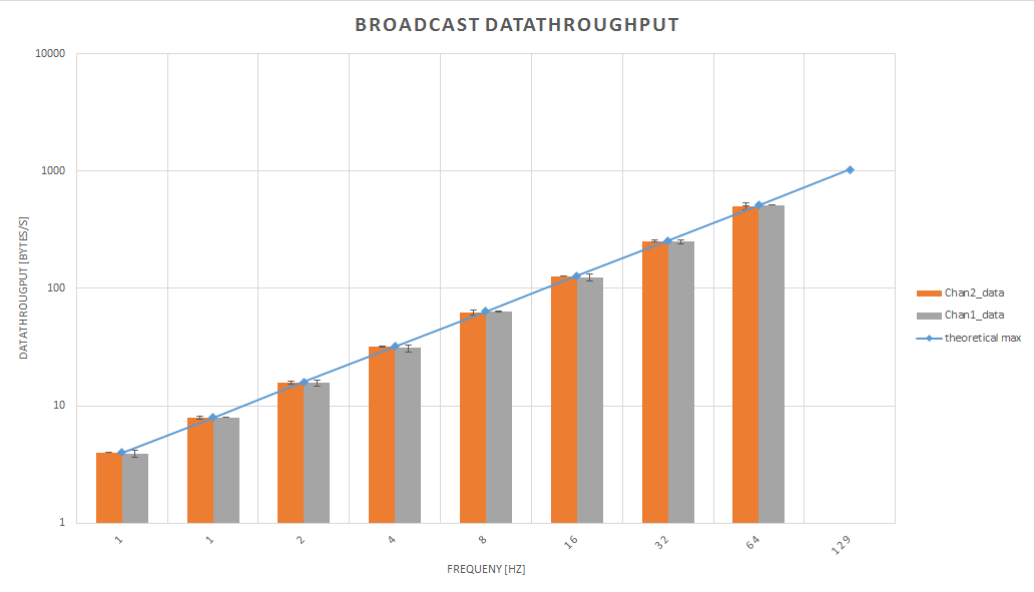

Abbildung 3.7.: Broadcast data through - 2 channels (0.5Hz - 129Hz)

Figure 3.13 shows the measured transmission speeds for the different tested frequencies. For the lower frequencies, the values align very well with the maximum data rate. Notable however is the sharp drop off at 129 Hz. Figure 3.14 shows that the drop off in the data throughput rate starts around 69 Hz. This result can be explained with the results of experiment 3. The delay of an acknowledge message is roughly 18 ms. The problem is that ANT retransmits the last sent packet as a broadcast packet. If the frequency for this retransmission is too high it can interfere with the acknowledge message that the slave sends back to the master. For example at 200 Hz the message is broadcast every 5 ms, resulting in too much data for the channel to handle. The transmission is thus interrupted. Since there is no increase of the data throughput between 64 Hz and 128 Hz, but a huge drop off at 69 Hz it can be assumed that somewhere around 69 Hz the channel is at the maximum data throughput of around 1100 Bps. Again see section 4.1 for a discussion about possible reasons for this upper limit.

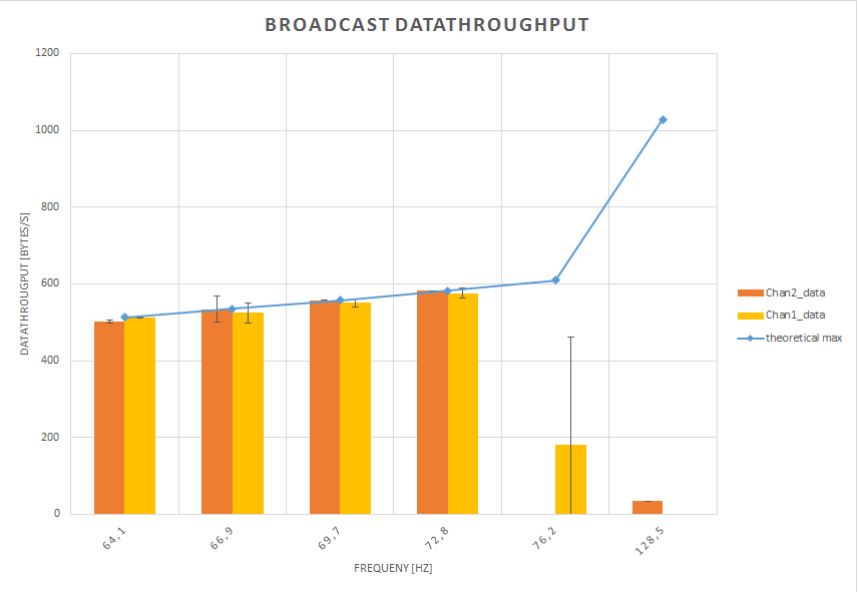

Abbildung 3.8.: Broadcast data through - 2 channels (64Hz - 129Hz)

3.6. Experiment 5: Burst Data Transfer between two nodes

Description

Burst data transmissions make it possible to drastically increase the throughput rate. This allows a SHAMPU base station to quickly transmit a new firmware to a node or a node to dump its RAM back to the base station. According to the specification rates of up to 20 kbps can be achieved. To fully utilize this speed a baud rate of 50000 is needed. Since we are using 19200 baud the maximum speed is expected to be less than 20 kbps. Furthermore we try to determine if the size of the burst transfer has an impact on the speed, since longer bursts will disrupt communications on other channels.

Use-Case

Unscheduled Data-transmission

Network Topology and Configuration

Testing methodology

Since the burst transmission mode of our ANT-library is not working correctly (see section 4.3) we use an ANTUSB-m Stick, which acts as a master. Node B is a normal base station and receives the burst transfers. On the master side, the program ANTWareII [Dynd] is used to create the channel and then send bursts with

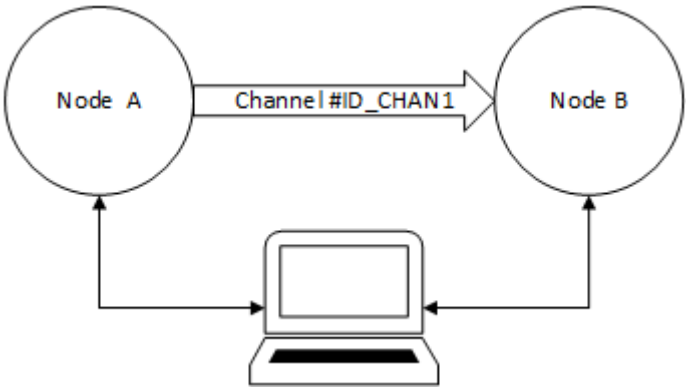

Abbildung 3.9.: Toplogy experiment 3

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod);
ANT_OpenChannel(ID_CHAN1, ANT_Bidirectional_Master);
duration = 0.0
for (i in 0..10) {
ANT_SendAcknowledgedData(ID_CHAN1, [0x01, 0x02, 0x03, 0x04])
start = getTime()
wait_for_ack()
print (getTime() - start) + " s"
}
ANT_CloseChannel(ID_CHAN1)
channelPeriod = decreaseChannelPeriod()
}
```

Pseudo code 5: Acknowledge data delay (Master)

different sizes. Each size is sent 10 times and the values are recorded and averaged. The size is then doubled and the experiment repeated. The first burst packet cannot be counted directly since the start of the burst can only be determined after the ANT chip has already received the first packet.

Result

Figure 3.16 shows the achieved data rates for the different packet sizes and also the length of the transfer. As seen the values all hover around the same value of 1165 Bps. This is approximately the same value as the one which can be achieved with the other two types of data. Again see section 4.1 for a discussion about possible reasons for this upper limit.

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod);
ANT_OpenChannel(ID_CHAN1, ANT_Bidirectional_Slave);
wait_for_user_input();
ANT_CloseChannel(ID_CHAN1);
channelPeriod = channelPeriod >> 1;
}
```

Pseudo code 6: Acknowledge data delay (Slave)

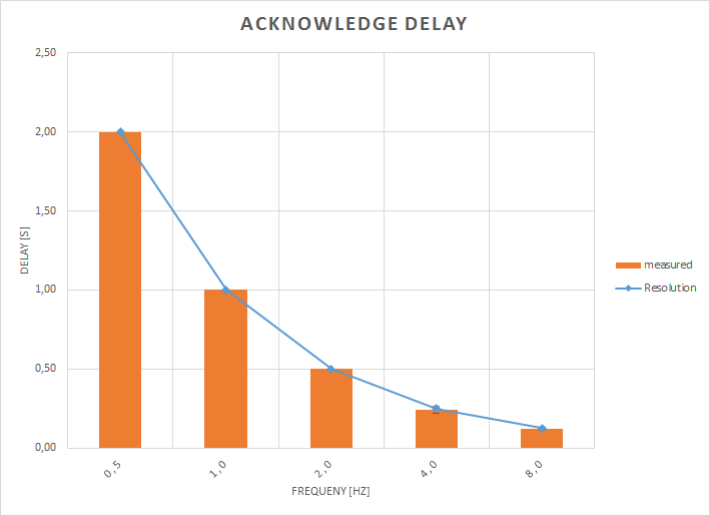

Abbildung 3.10.: Acknowledge delay (0.5Hz - 8Hz)

3.7. Experiment 6: Maximum communication Range

Description

In this experiment we try to determine the correlation between the maximum range and the power setting of the ANT radio. One of SHAMPU's advantages is the low power consumption, so it might be possible to further reduce the power consumption by decreasing the power level of the ANT radio, especially in smaller environments where there is no need for such a long range. According to the datasheet the maximum range for communication is 30m. However the ANT documentation does not specify for which power setting this range can be achieved.

Use-Case

Communication Range

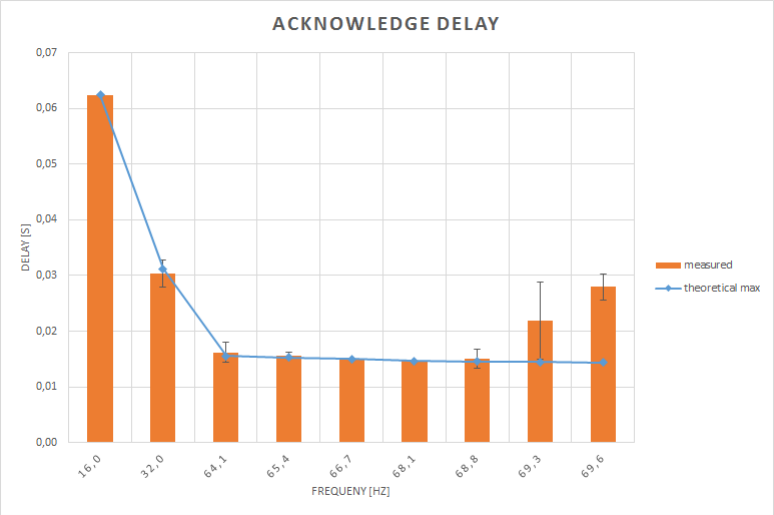

Abbildung 3.11.: Acknowledge delay (16Hz - 70Hz)

Network Topology and Configuration

Testing methodology

At the beginning of the experiment Node A and B are placed right next to each other. Node A acts as a master and keeps broadcasting the same message. Node B is the slave and tries to connect to the channel. If the connection is successful, the distance between the two nodes is increased by 0.4 meters. This process is repeated until Node B can no longer connect to the channel and the connection times out. This happens after 30s of searching. At this point the distance is no longer increased, but rather decreased until Node B is able to successfully connect to the channel. The whole experiment is then repeated for each available power setting.

Result

Figure 3.18 shows the transmission range for each power setting. As expected the maximum distance goes up with the higher power settings. But the result at the highest power setting is disappointing, since we did not get close to the claimed maximum range of 30m. The ANT documentation states that the maximum range of 30m can only be achieved in öptimal conditions"[Dyn13]. It is possible for several different unfavourable factors, such as multiple 802.11 networks in the vicinity, or even the plastic case of the base station, to interfere with the ANT signal.

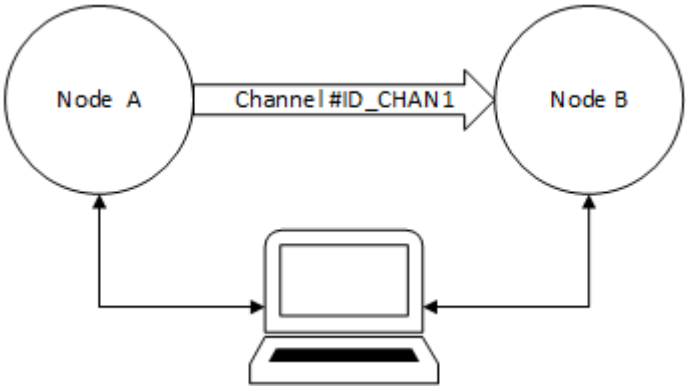

Abbildung 3.12.: Toplogy experiment 4

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod)
ANT_OpenChannel(ID_CHAN1, ANT_Bidirectional_Master)
count = 0
for (10 seconds) {
ANT_SendAcknowledgedData(ID_CHAN1, [0x01, 0x02, 0x03, 0x04])
wait_for_ack()
count++
}
print (count * 8 / 10) + " Bytes per second"
ANT_CloseChannel(ID_CHAN1)
channelPeriod = decreaseChannelPeriod()
}
```

Pseudo code 7: Acknowledge data transfer (Master)

```
channelPeriod = max_Channel_Period
while (channelPeriod >= min_Channel_Period) {
ANT_SetChannelPeriod(ID_CHAN1, channelPeriod)
ANT_OpenChannel(ID_CHAN1, ANT_Bidirectional_Slave)
wait_for_user_input()
ANT_CloseChannel(ID_CHAN1)
channelPeriod = channelPeriod >> 1
}
```

Pseudo code 8: Acknowledge data transfer (Slave)

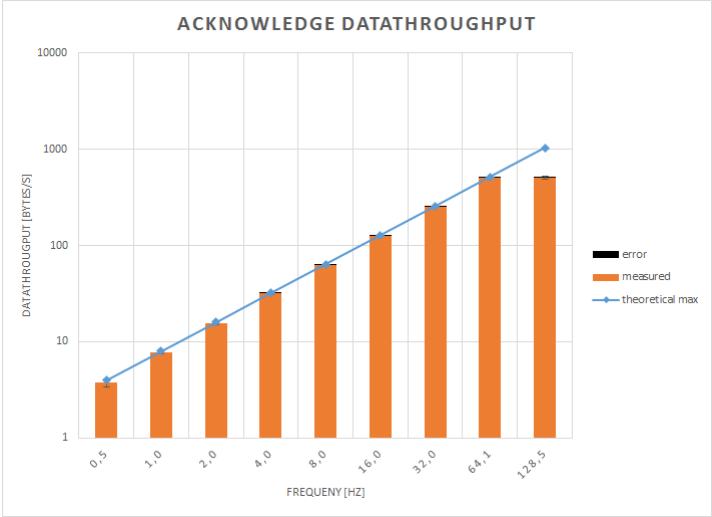

Abbildung 3.13.: Acknowledge data rate (0.5Hz - 129Hz)

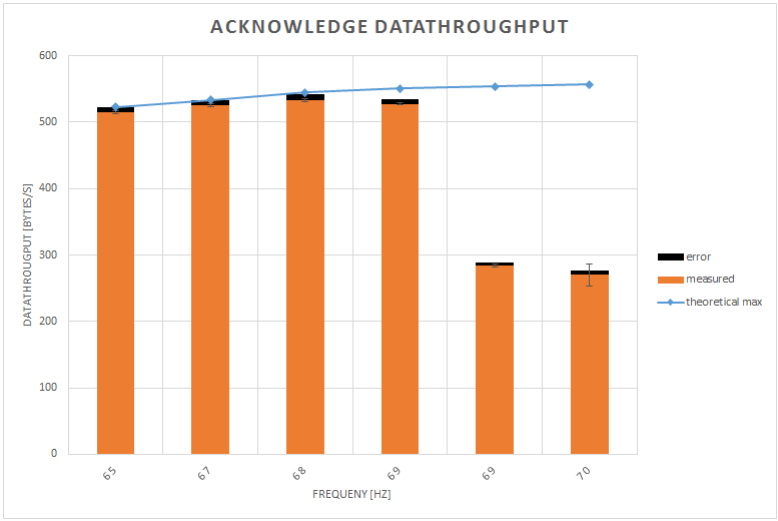

Abbildung 3.14.: Acknowledge data rate (65Hz - 70Hz)

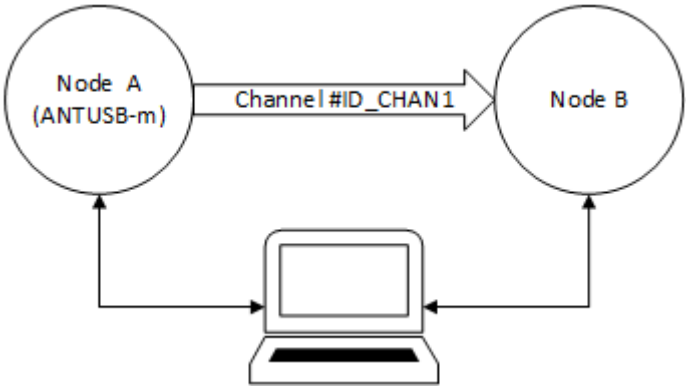

Abbildung 3.15.: Toplogy experiment 5

```
size = START_SIZE
ANT_OpenChannel(ID_CHAN1, ANT_Bidirectional_Slave);
while (size >= END_SIZE) {
for (i in 0..10) {
wait_for_first_burst_packet()
start = getTime()
wait_for_last_burst_packet()
print (getTime() - start) / (size - 1) " Bytes per second"
}
size = 2 * size
}
```

Pseudo code 9: Burst data transfer (Slave)

```
for (pSetting in Available_PowerSettings) {
ANT_SetTransmitPower(pSetting)
openChannel(ID_CHAN1, ANT_Bidirectional_Master)
ANT_SendBroadcastData(ID_CHAN1, [0x01, 0x02, 0x03, 0x04])
wait_for_user_input();
closeChannel(ID_CHAN1);
}
```

Pseudo code 10: maximum communication range (Master)

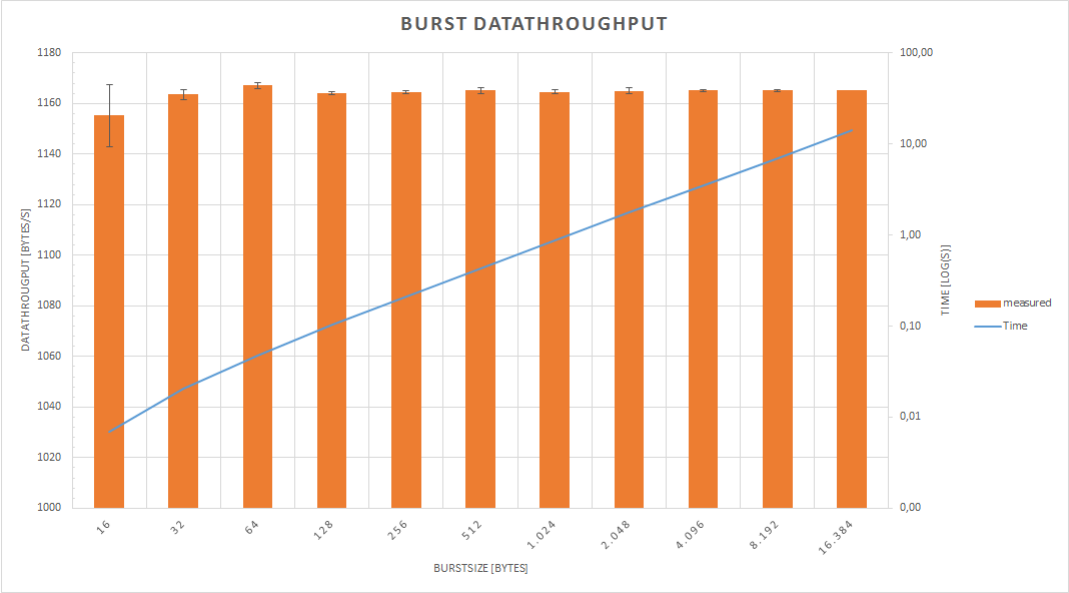

Abbildung 3.16.: Burst data rate

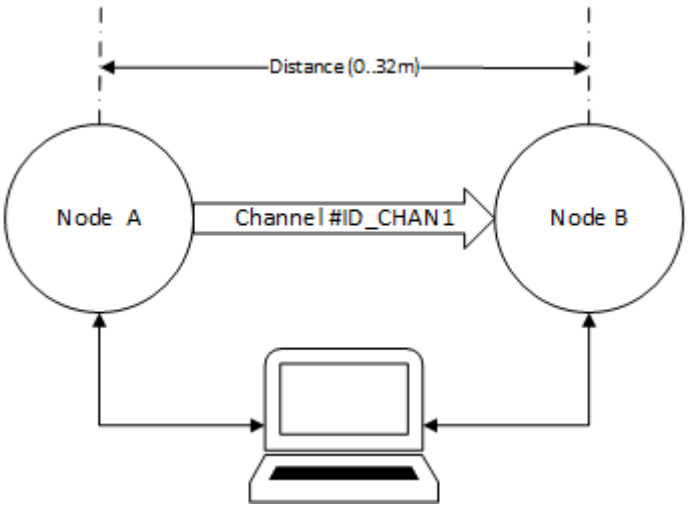

Abbildung 3.17.: Topology experiment 6

```
distance = 0.0
stopInc = false
loop {
openChannel(ID_CHAN1, ANT_Bidirectional_Slave)
wait_until(received_Packet == ANT_BROADCAST_DATA ||
received_Packet == ANT_MESSAGE_EVENT_RX_SEARCH_TIMEOUT) {
if (stopInc) {
if (wasTimeout) distance -= .4
else print "Connection found : " + distance
} else {
if (wasTimeout) {
stopInc = true; distance -= .54
print "Connections lost : " + distance
} else distance += .4
}
}
closeChannel(ID_CHAN1);
}
```

Pseudo code 11: maximum communication range (Slave)

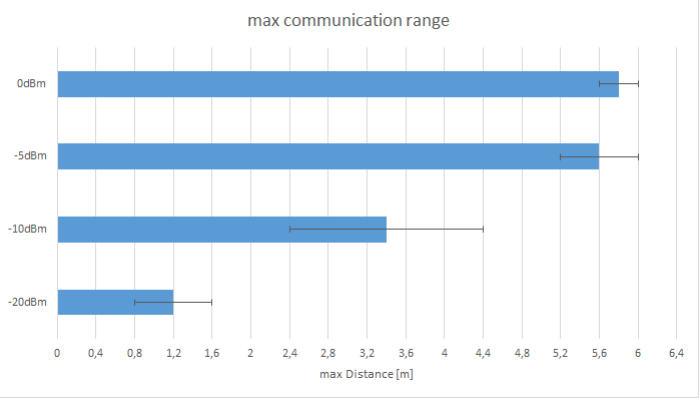

Abbildung 3.18.: maximum comunication range

Kapitel 4.

Conclusion

4.1. Maximum Data Throughput

All our experiments seem to confirm that there is a hard limit for the maximum data throughput which can be achieved with our current set up. The ANT AP1MxIB supports a maximum frequency of 200 Hz, so we would expect to see a maximum transmission rate of 1600 Bps. However, we missed that limit by around 500 Bps or 30%. For a burst transmission there should be up to 20 kbps of throughput, but again we only achieved around 1100 Bps, a loss of 1400 Bps or 56%.
To eliminate environmental influences, the experiments were repeated in different locations and times of day and night. Since the throughput did not change noticeably, it can be assumed that there are no easily eliminated environmental factors which affect the maximum rate. We conclude that the root cause for the lower than expected throughput must be attributed to the hardware or software.

If we exclude environmental factors, we are left with three possible reasons:

- **RS-232 Connection**
  The base station is connected to a PC over a serial connection. For the speed we chose 19200 baud, which should be more than enough for a maximum message rate of 200 Hz. For the burst mode however, this is one limiting factor. With 19200 baud, the theoretical maximum is 1920 Bps. This explains some, but not all of the measured difference.
- **ANT API**
  The software we use is not officially supported by ANT. It is thus possible that there are some bugs which have an impact on the performance. Except for the missing burst modus (see 4.3), there were no unusual measured values during the experiments. If there is a bug, it might be hard to find without rewriting all the experiments and running them with the official ANT library [Dynb]

- **ANT AP1MxIB**
  Due to the black box nature of ANT, it is very hard to exactly determine what causes the problem. The inner workings of the chip are not documented in any way, and the error messages of the protocol are not very specific. The chip itself is from 2007 and the manufacturer no longer recommends the use of the chip[Dyne]. There are alternatives available, like the newer ANTAP281M5IB chip [Dynf].

4.2. Summary

In conclusion of our experiments, it seems af if the ANT chip falls short of its full potential:

- **Scheduled data-transmission**
  The achieved data throughput of around 1160 Bps is valid for a single chip. That means the main bottleneck of the network is the base station, since it acts as a command and control server for each node in the network. The easiest way to fix the problem is to use more than one base station. This is possible, since we do not need a fully meshed network for the SHAMPU nodes to communicate with each other. They solely need to talk to the base station.
- **Unscheduled data-transmission**
  The achievable burst data throughput is especially problematic. SHAMPU is equipped with 128 kBytes of RAM. With the current set up it takes about 2 minutes to dump the entire memory to a base station, where is can be further analysed. If the full potential of the burst mode could be made available, the duration can be shortened to 70s. The long burst duration poses a problem, as long bursts disrupt the communication on other channels. For this reason the transmission of these memory-dumps has to be very carefully scheduled to avoid network congestion.
- **Communication Range**
  The maximum achieved range of 6m is probably the most disturbing result of this thesis. The short range signifies, that the physical location of the base station is critical for a successful deployment of SHAMPU. It also means that a larger network requires more than one such base station to be able to reach all nodes.

With all these limitations of the data throughput, the network setup has to be chosen very carefully. One option is to use additional SHAMPU nodes as relays, which serve a dual purpose. They allow to extend the range of the network, while increasing the data throughput of the whole network.
Another possibility to counteract the limitation is to leverage the numerous advantages of the SHAMPU framework, such as its low weight, form factor and power draw. A mobile base station, e.g. TrainSense [SSZ+13], which can easily be moved around in the location where the SHAMPU nodes are deployed, makes it possible to address all above mentioned problems. The range no longer represents a problem, since the base station is simply moved towards the node until a connection can be achieved. At the same time, the limited range provides a solution for the limited data throughput. Since the amount of nodes, which are able to connect to the base station can be controlled its possible to only ever have a small number of nodes transmitting data to the base station.

In the end, while ANT is not the most powerful wireless solution available, its design choices (low power consumption and ease-of-use) coincide almost completely with the

design goals of SHAMPU.

4.3. Future Work

Due to lack of time and the limited hardware availability we were not able to fully explore and evaluate the ossibilities of ANT in the SHAMPU network. Especially the following four areas should be revisited:

- **Power consumptions**
  Each experiment tries to find the maximum of either data throughput or the communication range. We did not measure how the power consumption changes if, for example, the message period is reduced. Since SHAMPU tries to be very low-powered, this is an important measurement for the decision whether or not SHAMPU can be used for a specific application. The data sheet of the ANT AP1MxIB module provides interesting data [Dync]: The maximum current draw seems to be around 5 mA for a continuous burst transmission and around 40 μA for a normal broadcast operation.
- **Burst mode**
  We use a custom library, which provides an API to interface with the ANT-Chip. For this thesis an attempt was made to add the missing burst transfer mode. However, due to time constraints we were unable to get the mode working correctly. Burst packets need to have a precise timing, but the black box nature of the ANT protocol makes it a challenge to debug the code.
- **Shared channels**
  Due to missing hardware, we were only able to test the communication between two nodes. Shared channels could not be tested, yet we expect the available data throughput to be in line with the results from our experiments. ANT uses up to 2 bytes of the 8 byte payload to specify the address of the receiver. Therefore we expect to lose approximately 12.5% to 25% of throughput if shared channels are used. It would be important to confirm this to fully assess the usefulness of the ANT chip.
- **New ANT-chip**
  As mentioned before the chip currently in use is old and no longer recommended for use. The successor of the current chip, the ANTAP281M4IB has roughly the same specifications as the current chip. Therefore the experiments should be rerun with the newer model in order to determine whether the ANT-chip is the limiting factor for data throughput.

Anhang A.

Source Code

```
#include <unistd.h>
#include <stdio.h>
#include <getopt.h>
#include "experiment.h"

char* ProgName; /* error.c */
extern uint16_t period;
extern FILE * fp;

int main(int argc, char** argv) {
int option = -1;
char deviceType = 0;
int experimentNum = -1;
char* port = "";

ProgName = fileName(argv[0]); /* error.c */
if (argc == 1) {
error("Usage: %s -p /dev/ttyUSB[Number] -t [m(aster) or s(lave)] -n experiment [-f start_
}

while ((option = getopt(argc, argv, "p:t:n:f:")) != -1) {
switch (option) {
case 'p' :
port = optarg;
break;
case 't' :
deviceType = optarg[0];
break;
case 'n' :
experimentNum = atoi(optarg);
break;
```

```
case 'f' :
period = atoi(optarg);
break;
default:
exit(EXIT_FAILURE);
}
}

if (deviceType != 's' && deviceType != 'm') {
error("Wrong deviceType! Must be 'm' or 's' is: %c\n", deviceType);
}

if (strncmp(port, "/dev/ttyUSB", 11)) {
error("Port information wrong!");
}

fp = fopen("result.txt", "a");
printf("Port = %s\nType = %c\nMessage Period = %d (%f Hz)\n", port, deviceType, period,
initANT(port);
setTransmitPower(ANT_TRANSMIT_POWER_0DBM);

switch (experimentNum) {
case 1:
printf("Experiment 1: Broadcast Data Transfer between two nodes\n");
doExperiment1(deviceType);
break;

case 2:
printf("Experiment 2: Broadcast Data Transfer between multiple nodes\n");
doExperiment2(deviceType);
break;

case 3:
printf("Experiment 3: Ackowledge Data Delay\n");
doExperiment3(deviceType);
break;

case 4:
printf("Experiment 4: Ackowledge Data Transfer between two nodes\n");
doExperiment4(deviceType);
break;
```

```
case 5:
printf("Experiment 5: Communication Distance\n");
doExperiment5(deviceType);
break;

case 6:
printf("Experiment 6: Burst Data Transfer between two nodes\n");
doExperiment6(deviceType);
break;

default:
printf("Available Experiments:\n");
printf("Experiment 1: Broadcast Data Transfer between two nodes\n");
printf("Experiment 2: Broadcast Data Transfer between multiple nodes\n");
printf("Experiment 3: Ackowledge Data Delay\n");
printf("Experiment 4: Ackowledge Data Transfer between two nodes\n");
printf("Experiment 5: Communication Distance\n");
printf("Experiment 6: Burst Data Transfer between two nodes\n");
error("%d is not a valid experiment! \n", experimentNum);

break;
}
flushBuffer();
fclose(fp);
return EXIT_SUCCESS;
}
```

Literaturverzeichnis

[Dyna] Dynastream Innovations Inc.: *ANT Channel Search and Background Scanning Channel*. Seiten 1–8.

[Dynb] Dynastream Innovations Inc.: *ANT Windows Library Package with source code*.

[Dync] Dynastream Innovations Inc.: *ANTAP1MxIB RF Transceiver Module*. Seiten 1–17.

[Dynd] Dynastream Innovations Inc.: *ANTware II*.

[Dyne] Dynastream Innovations Inc.: *AP1 Module product page*.

[Dynf] Dynastream Innovations Inc.: *AP2 RF Transceiver Module*. Seiten 1–22.

[Dyn13] Dynastream Innovations Inc.: *ANT Message Protocol and Usage*. Seiten 1–134, 2013.

[KL00] Kasten, Oliver und Marc Langheinrich: *First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network The Smart-Its Prototype*. System, (00), 2000.

[LFZ13] Lim, Roman, F Ferrari und Marco Zimmerling: *FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems*. Proceedings of the 12th ..., Seiten 153–165, 2013.

[MK] Moser, Thomas und Lukas Karrer: *The EventCollector Concept*. Topology.

[RHF+09] Rensfelt, Olof, Frederik Hermans, Christofer Ferm, Per Gunningberg und L. Larzon: *Sensei-UU: A Nomadic Sensor Network Testbed Supporting Mobile Nodes*. 2009.

[SSMM14] Smeets, Hugues, Chia-Yen Shih, Tim Meurer und Pedro José Marrón: *Demonstration Abstract: A Lightweight, Portable Device with Integrated USB-host Support for Reprogramming Wireless Sensor Nodes*. In: *Proceedings of the 13th International Symposium on Information Processing in Sensor Networks*, IPSN '14, Seiten 333–334, Piscataway, NJ, USA, 2014. IEEE Press.

[SSZ+13] Smeets, Hugues, Chia-Yen Shih, Marco Zuniga, Tobias Hagemeier und Pedro Josè Marrón: *TrainSense: A Novel Infrastructure to Support Mobility in Wireless Sensor Networks.* In: *Proceedings of the 10th European Conference on Wireless Sensor Networks*, EWSN'13, Seiten 18–33, Berlin, Heidelberg, 2013. Springer-Verlag.

Erklärung

German

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

English

I hereby declare that I have written this Bachelor thesis independently, using no other than the specified sources and resources, and that all quotations have been indicated.

Essen, 22. September 2015
(Place, Date)

Michael Krane