Anti Bicycle Theft

Documentation

Kevin Freeman (TODO1)
Martin Schwarzmaier (TODO2)
Georg-August-Universität Göttingen

January 25, 2016

Practical Course on Wireless Sensor Networks

Lab Advisor: Dr. Omar Alfandi
Lab Assistants: Arne Bochem, M.Sc.

Contents

1 Introduction

... bicycle theft, current problem (goettingen e.g.), pro contra to GSM/Bluetooth e.g.
Neues Vorgehen: also 1. intro, 2. used motes + sensorboard + mongodb + nodejs etc und 3 walkthrough =¿ wie man es benutzt und 4 dann alle bestandteile und protokolle etc erklären

2 Environment

Before presenting the project we firstly show what the whole environment looks like. Firstly, IRIS motes[1] are used (see Figure 1). Programs for the motes are written in nesC[2], compiled and flashed with tinyOS. Additionally, the computer used provides packages mentionend in Section 3 and is connected to a base station via USB Gateway, which can be seen in Figure 2. Some of the motes are connected to a MTS420CC sensorboard (see Figure 3). A GPS antenna is attached to the sensorboards.[3]

Figure 1: Standard IRIS mote without sensorboard.

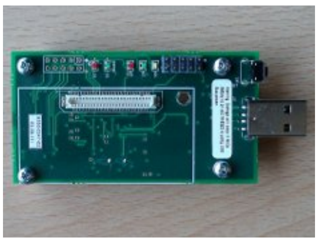

Figure 2: USB Gateway, used to forward packets via USB.

Figure 3: MTS420CC sensorboard, able to process GPS signals.

[1] A full description can be found at `http://www.memsic.com/userfiles/files/Datasheets/WSN/IRIS_Datasheet.pdf`

[2] `http://www.tinyos.net/api/nesc/doc/ref.pdf`

[3] Figure 1, 2 and 3 have been take from `https://user.informatik.uni-goettingen.de/~sensorlab/Hardware.php`

3 Walkthrough

This section demonstrates the project without showing technical details, as they will be explained in Section 4.

3.1 Flashing the motes

In order to make this project work, at least one IRIS mote as a base station as well as one iris mote equipped with a MTS420CC board including a gps antenna are needed. Each additional bicycle will need the same sensorboard and gps antenna. Additionally, it is possible to extend the network range with extra motes. This brings us to the following amount of needed motes:

a) one base station `[./nesC/base_station]`

b) one MTS420CC sensorboard with gps antenna per bicycle `[./nesC/bike]`

c) any amount of network node motes `[./nesC/nodes]`

3.2 Starting MongoDB, NodeJS and SerialForwarder

As the whole project is supposed to be user friendly, a webserver (Node.js) is implemented including a database (MongoDB). The webserver can be started by running the following command and will be available under `localhost:8080` afterwards:

```
$ node ./webapp/app.js
```

Additionally, the MongoDB daemon has to be started:

```
# mongod
```

Now the user is able to register to the webservice and mark his bicycles as stolen them already as shown in Figure 4 and 5. Once gps data is collected, he is able to review this on the same webpage, as seen in Figure 6.
On the other side, the base station establishes a connection to the database via python, which is calling the java classes `net.tinyos.tools.Send` and `net.tinyos.tools.Listen`. These are available after starting the `SerialForwarder` with:

```
$ java net.tinyos.sf.SerialForwader
                                    --comm serial@/dev/ttyUSB1:iris
```

Afterwards, only the `listen.py` has to be run, which imports our `./python-api/env/bikeDB/bikeDB.py` script. This will enable automatic gathering of stolen bicycle IDs from the database and push them into the network, but will also collect gps coordinates from bicycles and save them in the database.

```
$ python2 ./python-api/env/bikeDB/listen.py
```

Our test environment used the following Linux packages:

1. nodejs (version 5.5.0)

2. python2 (version 2.7.11)

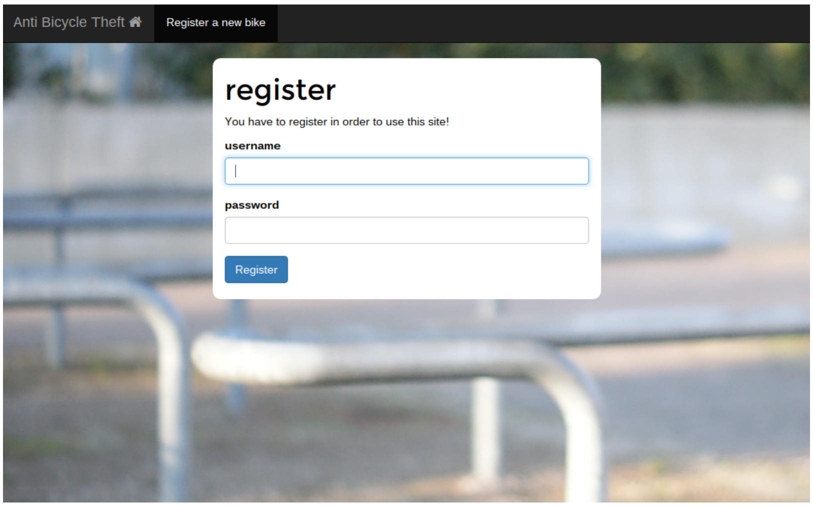

Figure 4: Registration website

3. mongodb (version 3.2.0)
4. jdk7-openjdk (version 7.u95_2.6.4)
5. python2-pymongo (version 3.2)

%TODO: brauch man npm? Packages needed node, mongo, python2, java, pymongo

3.3 Registration and Tracking

%TODO Screenshot of Views
is this section still necessary?

4 network protocols and mote insight %todo: clever sectionname here

As the walkthrough only showed how to use the project, this section will show the programs and protocols used for each mote in detail. As all motes are supposed to talk with each other, one superior header file is implemented. It provides easy changeable variables for the maximal amount of bicycles, that can be stolen at once as well as how many gps coordinates are gathered in a single packet as seen in Listing 1. The gathering process uses the Collection %TODO quelle protocol and the propagating of stolen bicycle `IDs` is done via the Dissemination protocol.

```
//...
#define MAXBIKES 10
#define COORDS_PER_PACKET 2
typedef nx_struct EasyDisseminationMsg
```

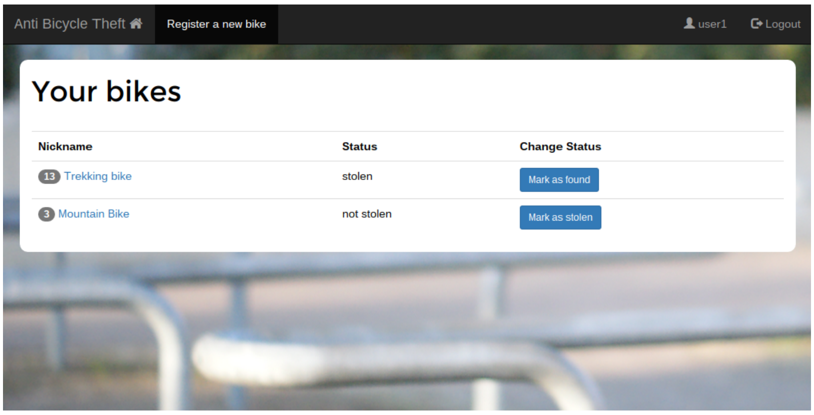

Figure 5: Viewing registered bikes

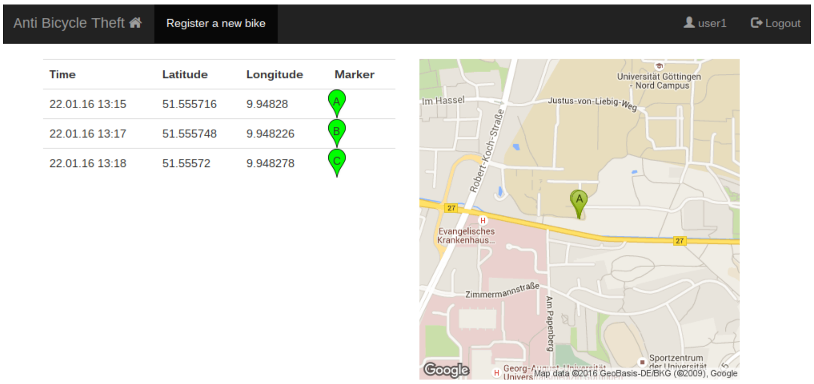

Figure 6: Viewing gathered gps coordinates

```
{
    nx_uint16_t bikes[MAXBIKES];
} EasyDisseminationMsg;
//...
typedef nx_struct EasyCollectionMsg
{
    nx_uint16_t nodeid;
    nx_uint32_t current_time;
    nx_uint32_t time[COORDS_PER_PACKET];
    nx_uint32_t lat[COORDS_PER_PACKET];
    nx_uint32_t lon[COORDS_PER_PACKET];
} EasyCollectionMsg;
```

Listing 1: DataMsg.h, content of packets

Obviously, the content and variable types can be changed easily, too.

4.1 Base Station

Our base station `[./nesC/base_station]` is connected to a computer via USB. On this computer, a `SerialForwarder` is run in order to establish a possibility to send and receive data to and from the base station. For this project, on the one hand, we have to push `ID`s of stolen bikes to the base station in order to disseminate them through the network. On the other hand, the collection protocol is implemented to gather information from stolen bikes, like coordinates. This is done via a python script, which can be seen in Listing 2. In line %TODO it imports our bikeDB class, which is simply providing methods to read and write to the database.

```
todo: push the new one -.-
```

Listing 2: listen.py, ...

%TODO several picutres here (CLI, serialforwarder)

4.2 Network Node

The network nodes are completely omittable as they only enlarge the network. More network nodes are needed if a great availability of the network is wanted. These are connected with other network nodes and disseminate and collect the data mentioned already to and from the base station and bicycle motes. Currently, they are doing nothing but disseminating and collecting. However, functionality can be easily added in the `./nesC/nodes/NodeC.nc`.

4.3 Bicycle Mote

Bicycle motes are attached to each bicycle and equipped with a MTS420CC sensorboard including GPS-antenna.
%TODO picture of stuff
Each of them has a unique identifier (`ID`), which is used in order to link a mote to a bicycle. On our webview the user is then able to mark his bicycle as stolen.

Afterwards, the `ID` is disseminated through the whole network. If the bicycle approaches a network node, it receives a dissemination packet. These are called "pings". As the bicycle now knows that the network is in range and available, it will check, if its own `ID` is marked as stolen. If so, the GPS antenna is powered on and starts approximate 90 seconds later to save data (current runtime, latitude, longitude).
After recording data successfully and receiving another ping, the bicycle mote will dump the data into the network using the collection protocol.
%TODO: kann weg? The amount of data per packet can be easily changed within the `./nesC/DataMsg.h`, as seen in Listing 1.
Each mote has a RAM of 8kB. Therefore, it is possible to store 600 coordinate-tuples on the bicycle mote RAM. As we are approximately storing one tuple every 3 minutes, it is possible to store the coordinates of the last 30h. This can be extended by saving onto the measurement flash itself, which we left out for future work, as 30h is enough for all our testcases as well as the battery time is limited due to usage of GPS as well. %TODO batterien haben 3.4k mAh oder so, gps brauch 70 mAh

4.4 Topology

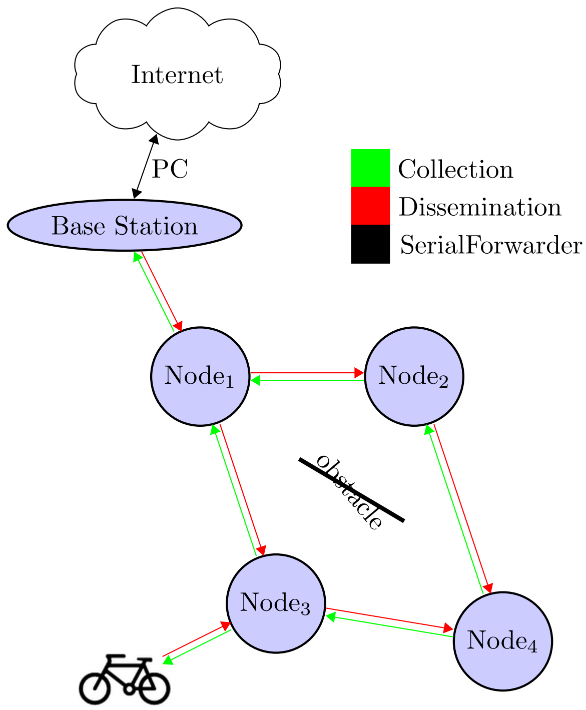

Figure 7: Example network showing the protocols used

- extensible by "nodes"

5 Conclusion

6 Future Aspects

The projects objective is to point out the possibility of a new tracking system using IRIS motes. Therefore, some additions have been left out, as they would not change the procedure, but increase the security, for example. All of these points are explained in this section.

1. **Privacy and authentication**. Obviously, admins are able to review data of users and mark specific bicycles as stolen, to start the gatherin process. Therefore, the gps data gathered could be encrypted using asymmetric cryptography. The user will receive encrypted data and decrypt it using his private key.

2. **Encrypted traffic**. Relating to the previous point, the whole traffic of the network, including dissemination of stolen bicycle IDs, should be encrypted to ensure privacy and integrity of packets.

3. **Further dissemination of stolen IDs**. Bicycle motes, regardless of their state, can save and disseminate the `IDs` of stolen bicycles. This way, a bicycle is able to start gathering data without connecting to the network, but to another bicycle. Obviously, one has to take care of synchronicity.

4. **Measurement flash**. Instead of just keeping the gathered gps data in the memory, one could save them to the flash of the mote and load them again when receiving a ping of the network. This provides the possibility to store a huge amount of coordinates.

5. **Battery life**. When implementing the previous mentioned point, the battery life has to be extended as well as GPS needs a lot of battery. However, one could use two approaches:

   (a) A stolen bicycle is going to be used. Therefore, one could power the mote using the energy of a dynamo attached to the bicycle.

   (b) At some places, for example a cellar, the gps antenna is not able to maintain a connection. Hence, the gps antenna could be powered off after e.g. 1 minute without connection. Afterwards, start the gps again after a small amount of time and check for connection again.

6. **Hiding of the mote**. The thief may not see and be able to remove the mote from the bicycle. So the mote has to be hidden somewhere in the frame of the bicycle without losing signal.

7. **GPS exchangeability**. As GPS drains a lot of battery, one could think of implementing a wifi scanner instead of GPS. As wifi is widely used, nearly everywhere wifi networks can be found. Using the `SSIDs` only (without connecting to them) one is able to specify the current location. This obviously consumes less power than gps and will even work, when a bicycle is e.g. in a garage, where a gps antenna will not be able to establish a connection.

%TODO gps VS GPS

7 Relevant codes passages... just a dump

7.1 base_station

Receiving stolen bicycle IDs from PC

```
//...
for (i=0;i<MAXBIKES;i++)
{
    pkt.bikes[i]=msg->data[i*2]*256+msg->data[i*2+1];
}
//...
```

Listing 3: DataMsg.h, content of packets

7.2 base_station

Receiving stolen bicycle IDs from PC

```
#define MAXPOSITIONS 100
//...
uint32_t lons[MAXPOSITIONS];
uint32_t lats[MAXPOSITIONS];
uint32_t times[MAXPOSITIONS];
//reading
atomic
{
    for (i=current_reading_pos;i!=current_writing_pos;i++)
    {
        msg->time[j] = times[i];
        msg->lat[j] = lats[i];
        msg->lon[j] = lons[i];
        times[i]=0;
        lats[i]=0;
        lons[i]=0;
        current_reading_pos++; //we read the value
        if (current_reading_pos==MAXPOSITIONS)
            current_reading_pos=0;

        j++;
        if (j==COORDS_PER_PACKET)
            break;
    }
}
//writing
atomic
{
    lats[current_writing_pos]=(uint32_t)(lat*1000000);
    lons[current_writing_pos]=(uint32_t)(lon*1000000);
    times[current_writing_pos]=(uint32_t)((call LocalTimeMicro.get())/1000); //
    current_writing_pos++;
```

```
    if (current_writing_pos==MAXPOSITIONS)
        current_writing_pos=0;
    call Leds.led0Toggle();
}
//receiving IDs and starting gps if stolen
event void Value.changed()
{
uint8_t i;
const EasyDisseminationMsg* newVal = call Value.get();
bool found=FALSE;
pkt = *newVal;
for (i=0;i<MAXBIKES;i++)
{
    if (pkt.bikes[i]==secret())
    {
        stolen=0x01;
        found=TRUE;
        call Leds.led1On();
        if (gps_started==0)
        {
            gps_started=1;
            call Timer.startOneShot(90000); //wait X/1000 secs
            call GpsControl.start();
        }
        else if (gps_started==2) //startDone for gps
        {
            call Leds.led2Toggle();
            //it is stolen AND received a broadcast
            //=> DUMP ONE PACKET
            sendMessage();
        }
    }
}
if (found==FALSE)
{
    call Leds.led1Off();
    stolen=0x00;
    if (gps_started>1)
    {
        call GpsControl.stop();
        gps_started=0;
    }
}
```

Listing 4: DataMsg.h, content of packets

8 References

Dissemination and collection protocols for TinyOS: http://tinyos.stanford.edu/tinyos-wiki/index.php/Network_Protocols