veritas
iustitia
libertas

Freie Universität Berlin

Hands-on course „Mobile Communications“

Summer 2008

Material and Assignments in the areas of:

Medium Access in Wireless Networks

Freie Universität Berlin

Institute of Computer Systems & Telematics

A. Liers, H. Ritter, M. Baar, J. Schiller

http://cst.mi.fu-berlin.de/

Content

1 Introduction

To give you deeper insights into some of the aspects of the wide area of mobile communications, the hands-on course “mobile communications” was initially set up in the year 2002. Based on the network-related lectures “Telematics” and “Mobile Communications”, the course is meant to give a better understanding of central concepts of mobile networks and future mobile services.

The course is divided into three parts that cover the different layers of the classical communication layer model:

- Medium Access in wireless networks using the license-free 868 MHz band
- Routing in wireless ad-hoc networks
- Application development and service usage with mobile devices

As it is the second time that we offer this hands-on course, we have improved on some aspects of the content and documentation of this course. Nevertheless, hands-on courses are always a moving target, as technology advances and interesting new things come up. So feel free to help us to improve even more. And finally, we hope that you will have some fun with testing and programming mobile communications stuff!

Achim Liers, Hartmut Ritter, Min Tian, Michael Baar, Jochen Schiller

2 868 MHz Radio Communication

This part of the course deals with problems of medium access and protocols for data transfer; based on the license-free 868 MHz frequency band. In the next sections you will find technical references, programming hints, a proposal for a common frame format and practical assignments.

2.1 Technical References

The hardware used in this course provides consists of an USB stick with an onboard 868 MHz radio transceiver (= transmitter and receiver). Depending on the used driver, the USB stick provides you with a virtual serial port connection from the PC to the transceiver. A picture of the USB stick is given in Figure 1.

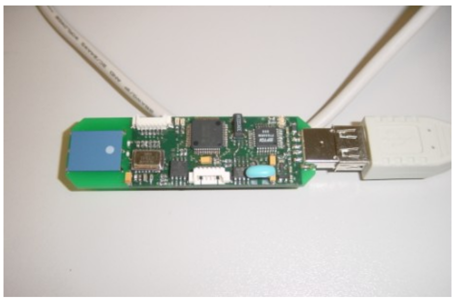


Figure 1: The USB stick provided for the course

It consists of four parts: A chip for the USB interface (from FTDI), a microcontroller (TI MSP430) mainly for purposes of flow control, and the transceiver module with attached antenna. This is sketched in Figure 2.

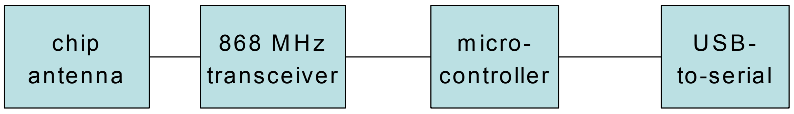


Figure 2: The parts of the USB stick

Though you don’t need to understand the software running on the microcontroller for this practical course, in the following the parts of the USB stick will be explained in short.

2.1.1 USB-to-Serial Converter

The USB-to-Serial converter, realised with a chip from FTDI (cf. www.ftdichip.com), transports packets from the USB interface to the microcontroller and vice versa. The Windows driver handles initialization of the converter and provides a virtual serial port on the PC. The USB-to-Serial converter itself is connected on board of the USB stick with the microcontroller again over a serial connection.

2.1.2 Microcontroller

The microcontroller, a Texas Instruments MSP430 (cf. www.ti.com), performs only simple tasks in this setup. It has to carry out flow control, as it acts as a gateway between the stream-oriented 868 MHz transceiver interface and the block-oriented USB-to-Serial converter. An incoming byte stream from the transceiver interface has to be buffered by the MSP, until the USB-to-serial converter encapsulates the bytes in one packet of limited maximum size after another.

2.1.3 868 MHz Transceiver

The 868 MHz transceiver module (TR1001 of RFM, cf. www.rfm.com) provides a very simple serial bitstream interface for sending and receiving bytes in the license-free 868 MHz frequency band. The transceiver module acts completely transparent: It takes the bytes that are provided at the input and sends them at 868 MHz, using On-Off-Keying. If it receives a signal above a certain threshold, it demodulates it and sends the bytes to the microcontroller. Note, that the transceiver implicitly acts in a half-duplex way: Either it transmits or it receives at any given moment. The default state is receiving.

2.2 Software Interface

The software interface at the PC is a serial port (“COM port”). Which port number is assigned, depends on the further installed hardware at the PC. You can find out using the Windows Device Manager (Start->Settings->Control Panel->System->Hardware), looking for the FU Funkboard at the COM and LPT port tab.

2.3 Communication using the 868 MHz module

In this section, some problems related to the special kind of radio module that is used in the course are discussed.

2.3.1 Features and requirements of the 868 MHz transceiver

The setting you need for access to the COM port are:

- 19200 bit/s
- 8 bits, no parity, 1 stop bit
- Hardware flow control on

2.3.2 Features and requirements of the 868 MHz transceiver

There are two modes of the 868 MHz module

- Receive mode:

  This is the default mode of the module. In receive mode, data from the air interface is passed as a bit stream to the serial interface. This is valid as long as a carrier is detected (as indicated by the carrier detect LED). If the receiver just receives noise, it is not passed to the serial interface.
- Send mode

  If the module receives data from the serial port, it immediately switches to the send mode and sends the data. It does not wait until no carrier is detected on the air interface. Therefore, collision avoidance has to be implemented by software.

The radio transceiver requires a bit stream, that

- contains – on average – as many high as low bits (DC balance, “Gleichstromfreiheit”)
- begins with a preamble enabling synchronization of the oscillator on board of the radio module.

There are different possibilities to fulfil these hardware requirements. The next section presents the solution that is to be deployed in this course.

2.3.3 Lowest Level Framing

A frame format that fulfils the hardware requirements is presented in Figure 3. It starts with the preamble five times AAh (10101010b) for synchronization of the radio part and twice FFh for the synchronization of the UART (universal asynchronous receiver and transmitter, basic chip of nearly any serial interface).

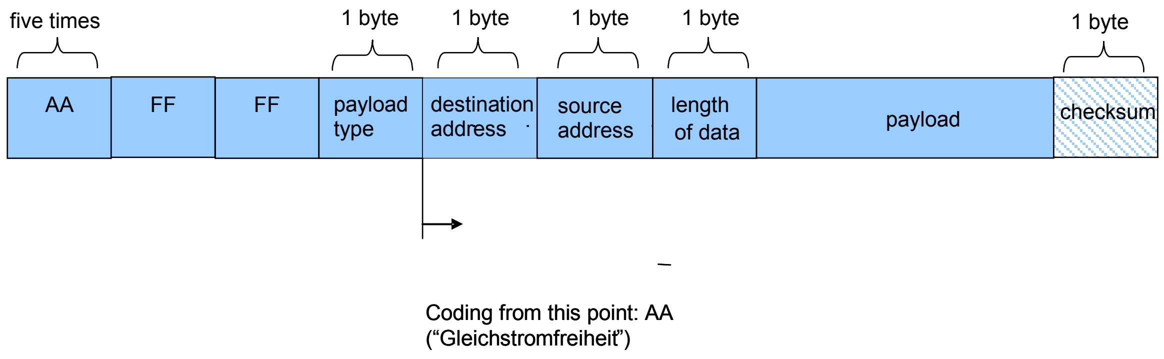


Figure 3 Format of lowest level frame

The remainder of the frame contains data that is structured by the higher level: The payload type denotes the type of data that is contained in the payload field, destination address and source address have to be interpreted by higher layers as well, and finally the data length field gives the size of the payload.

Beginning with the destination address, all data bytes are sent twice: Immediately after the data byte, the inverted data byte is sent. Example: If the destination address is 0x01, the data that is really sent is 0x01h 0xFEh.

The checksum is to be calculated as Check(i) = Modulo(NOT Check(i – 1) + Byte_i,255) with Check(0) = 0xAA and i between 1 and #Bytes on the complete packet (starting from payload type).

Protocol Type	**Name**	**Description**	**CRC**
0x01	RTS	Ready to send primitive	no
0x02	CTS	Clear to send primitive	no
0x03	PING	Pong request	no
0x04	PONG	Pong reply	no
0x10	SAbP Protocol	Payload is SAbP data	yes
0x11	ASCII	Payload is Ascii Text	yes
0x12	UNICODE	Payload is Unicode Text	yes
0x40 .. 0xFF		User defined	

Table 1: Protocol Type

Address	**User**	**Description**
0x00	Lecturer	Listener Node
0x10 … 0xEF	Students	Use 0x10 + PC-Number
0xFF	Broadcast	Use with care!

Table 2: Protocol Address

Note 1: Please follow these format rules as otherwise a communication between different modules of different groups will obviously not be possible!

Note 2: In your own interest, keep the implementation of these lowest-level framing simple and transparent and write separate methods for encapsulation and decapsulation. You will need these functions later on when running IP over the 868 MHz transceivers.

2.3.4 Higher level protocols

868 MHz communication is typically used for wireless sensor devices. Examples are remote sensors for home automation, like temperature sensors in the garden that can be read while sitting on the couch. Another example is remote controls for cars built into the key. In all these cases, the communication pattern is a typical master/slave pattern.

Therefore, we propose a simple, partly ASCII-based protocol called SAbP shown in Figure 4. Any node that wants to start communication, acts as a master. There are two primitives: A simple “PING” message to find out whether a node at a specific destination address is reachable. And a “DATA_REQ” primitive that allows to request data from another node.

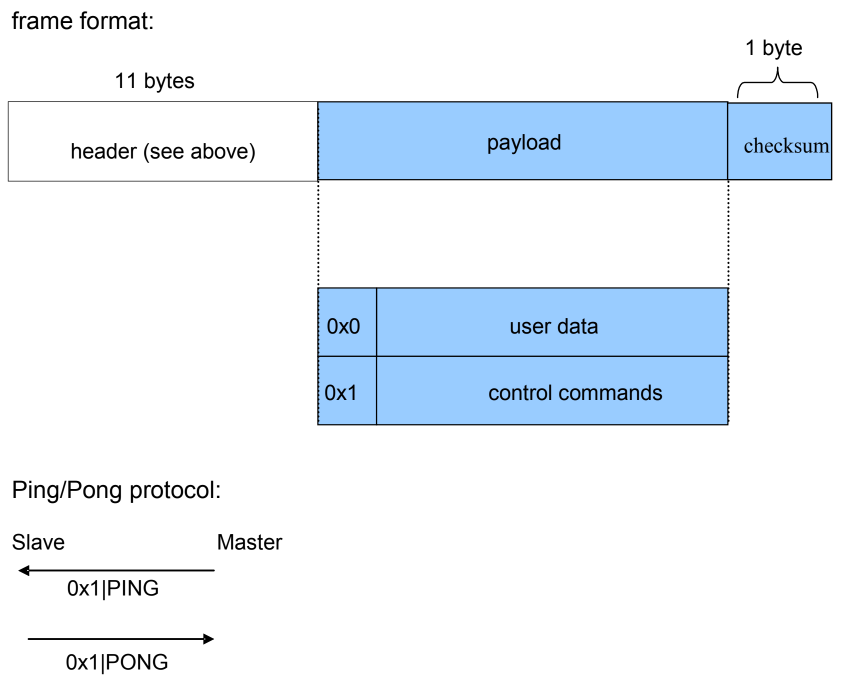
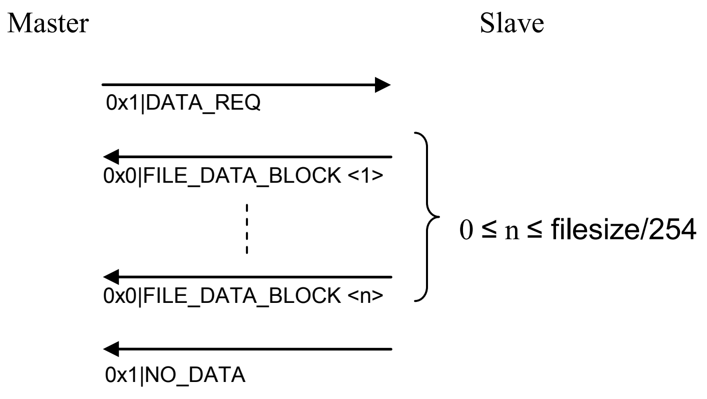


Figure 4:Simple ASCII-based protocol SAbP for 868 MHz communication

Note: The proposed protocol again builds the basis for any communication between nodes of different groups, so you should stick to it. Nevertheless, feel free to augment it, as long as you are backward-compatible – and your software can deal with unknown protocol primitives as well!

SAbP File Transfer protocol:



If currently no data is available the slave shall reply with the NO_DATA command.

Command	**Value**	**Description**
PING	0x01	Pong request
PONG	0x02	Ping reply
DATA_REQ	0x03	
NO_DATA	0x04	

Table 3: SAbP Control Commands

Field	**Position [Byte]**	**Description**
Offset	0..3	Byte offset of data following
File Data	4..254	Part of the file starting at <off-set>

Table 4: File Data Block