Semester Thesis

Traffic Monitoring in Sensor Networks

Raphael Schmid

Departments of Computer Science and Information Technology and Electrical Engineering,
ETH Zurich

Summer Term 2006

Supervisors:
Nicolas Burri and Pascal von Rickenback, Distributed Computing Group
Prof. Dr. Roger Wattenhofer, Distributed Computing Group

Table of Contents

1. Introduction and Task Description

The subject of this semester thesis was *traffic monitoring in sensor networks*.

Up to now, the user had to use a simple command window or a text editor for investigating sensor network data. Furthermore, if he wanted to realize a distributed investigation using more than one sniffing sensor node, there was no possibility to merge the gathered data. Instead, the user had to look at several command windows or text editors. All this procedures are not really convenient.

The goal of this semester thesis was to develop an application which gathers, merges and displays sensor network data. The application should allow the user to work not just with byte streams. Instead, the displaying should help to reconstruct the behavior of the sniffed sensor network in a better way. Therefore the user interface should base on well defined sensor network packet formats, and a possibility for the user to easily define new sensor network packet formats and sensor node platform architecture formats is needed.

In the next sections, I will discuss my developed application, which consists of five sub-applications. I focused my work on the complex displaying of the gathered data.

2. High-level Software Architecture

The whole application consists of the following five sub applications (see Figure 1): The network packet listener *TOSBase* (running on a sensor node), the application *SerialForwarder* receiving sniffed network packets from the *TOSBase*, the application *TinyWireshark Client* receiving the sniffed sensor network packets from the *SerialForwarder*, and finally the application *TinyWireshark Server* receiving the sniffed sensor network packets from the *TinyWireshark Client*. The open source application *NetTime*[1], which can either act as a server or a client, is responsible for the synchronization of the system clocks of the computers running the *TinyWireshark Server* and *Clients*.

TOSBase and *SerialForwarder* are freely available open source programs from the *TinyOS-Cygwin-Environment 1.0*[2].

The *TinyWireshark Client* and *TinyWireshark Server* are custom Java applications.

When the *TOSBase* gathers a sensor network packet, it immediately forwards it to the *SerialForwarder* using, for instance, a serial connection. The *SerialForwarder* then uses a local network connection to the *TinyWireshark Client* for forwarding the sensor network packet right away. When the *TinyWireshark Client* receives a sensor network packet, it measures the current system time and stores this timestamp together with the received sensor network packet in an overall packet. Afterwards, it either sends the overall packet immediately to the *TinyWireshark Server* (using a network connection) or stores the overall packet in local memory, depending on the user preferences. On receiving a packet the *TinyWireshark Server* displays the new data in the graphical user interface.

NetTime runs completely independent from the *TinyWireshark Server* and *Client*. The user is responsible for running *NetTime.*

In general, the application is split into one or several clients and one server. The client notebooks or PCs are responsible for gathering the sensor network packets. The server notebook or PC is responsible for merging, storing and showing the gathered packets of the clients.

1 http://sourceforge.net/projects/nettime
2 http://www.tinyos.net/, http://dcg.ethz.ch/projects/tos_ide/

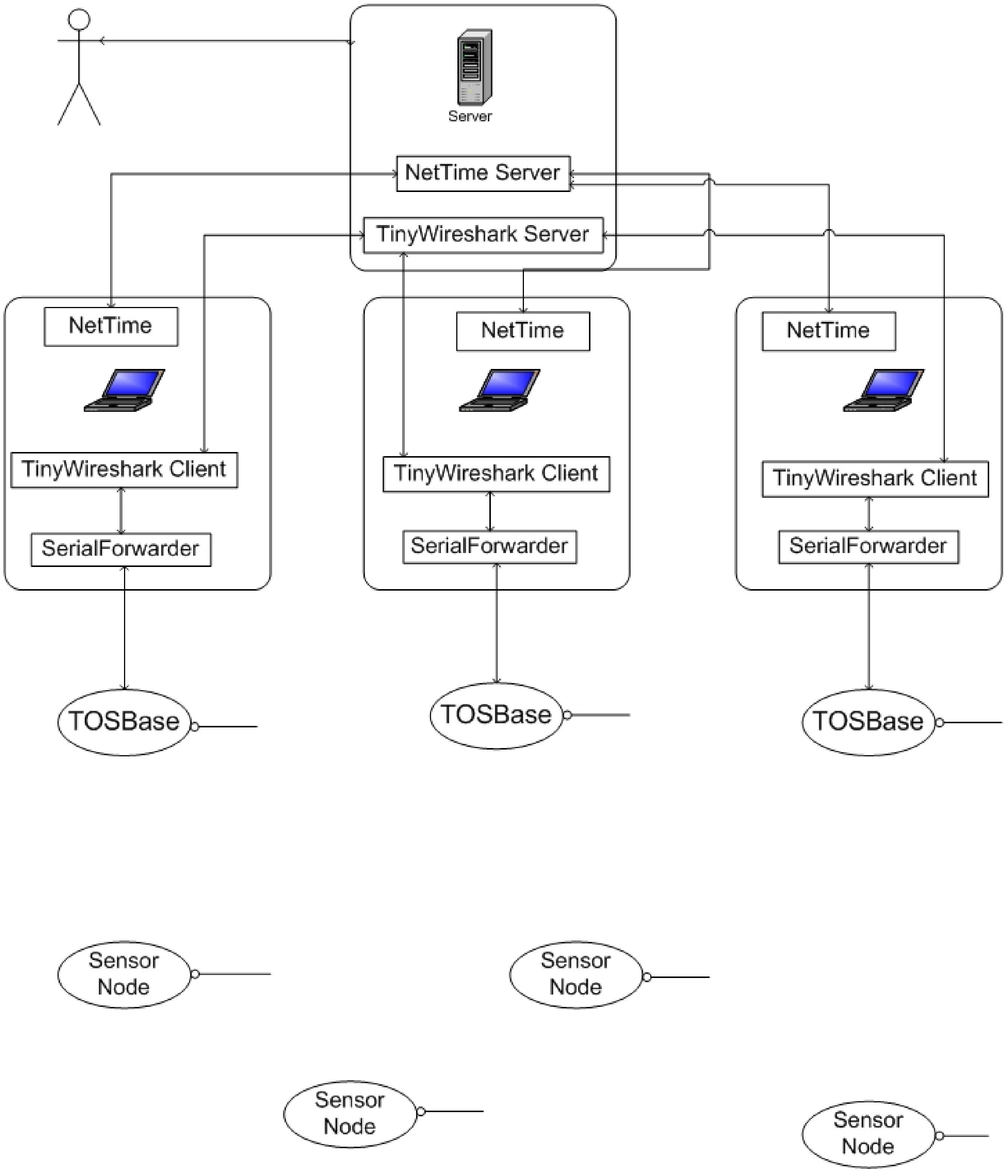
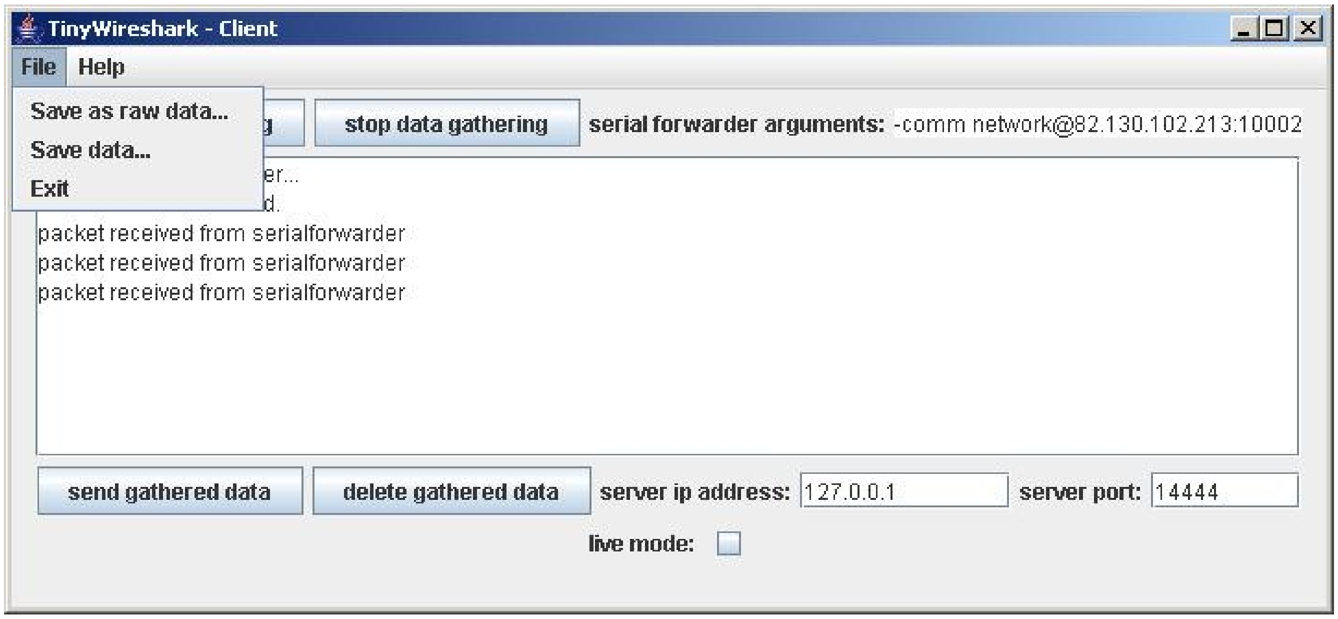


Figure 1: Architecture of the application

3. TinyWireshark Client

The architecture of the *TinyWireshark Client* is quite simple. It is divided into the two packages *gui* and *client.* The package *gui* is responsible for the graphical user interface. The package *client* is responsible for receiving data from the *SerialForwarder*, for sending data to the *TinyWireshark Server*, and for storing data to hard disk.

The user can change the arguments for *SerialForwarder* using the GUI (see Figure 2). The arguments determine how the *SerialForwarder* has to connect to the data gathering sensor node running the *TOSBase.* The user can also change the network settings (i.e., the IP address and the IP port) for the connection to the *TinyWireshark Server*.



serial forwarder arguments: -comm network@82.130.102.213:10002

```
er...
d.
packet received from serialforwarder
packet received from serialforwarder
packet received from serialforwarder
```

server ip address: 127.0.0.1 server port: 14444

live mode: ☐

Figure 2: The *TinyWireshark Client* graphical user interface

The button *start data gathering* is used to enable the *TinyWireshark Client* to receive incoming data from the *SerialForwarder*. Each arriving sensor network packet is time stamped with the current system time and stored in main memory. Note that the accuracy depends on the underlying operating system and hardware clock, but in general the error should not be worse than a few milliseconds. If higher precision is required, native system calls become necessary, for example using the Java Native Interface[3]. Further inaccuracies are added while the packet is transmitted from the data gathering sensor node to the *TinyWireshark Client*. If the sensor node is connected to the *TinyWireshark Client* using wireless LAN, an error of several milliseconds may be added!

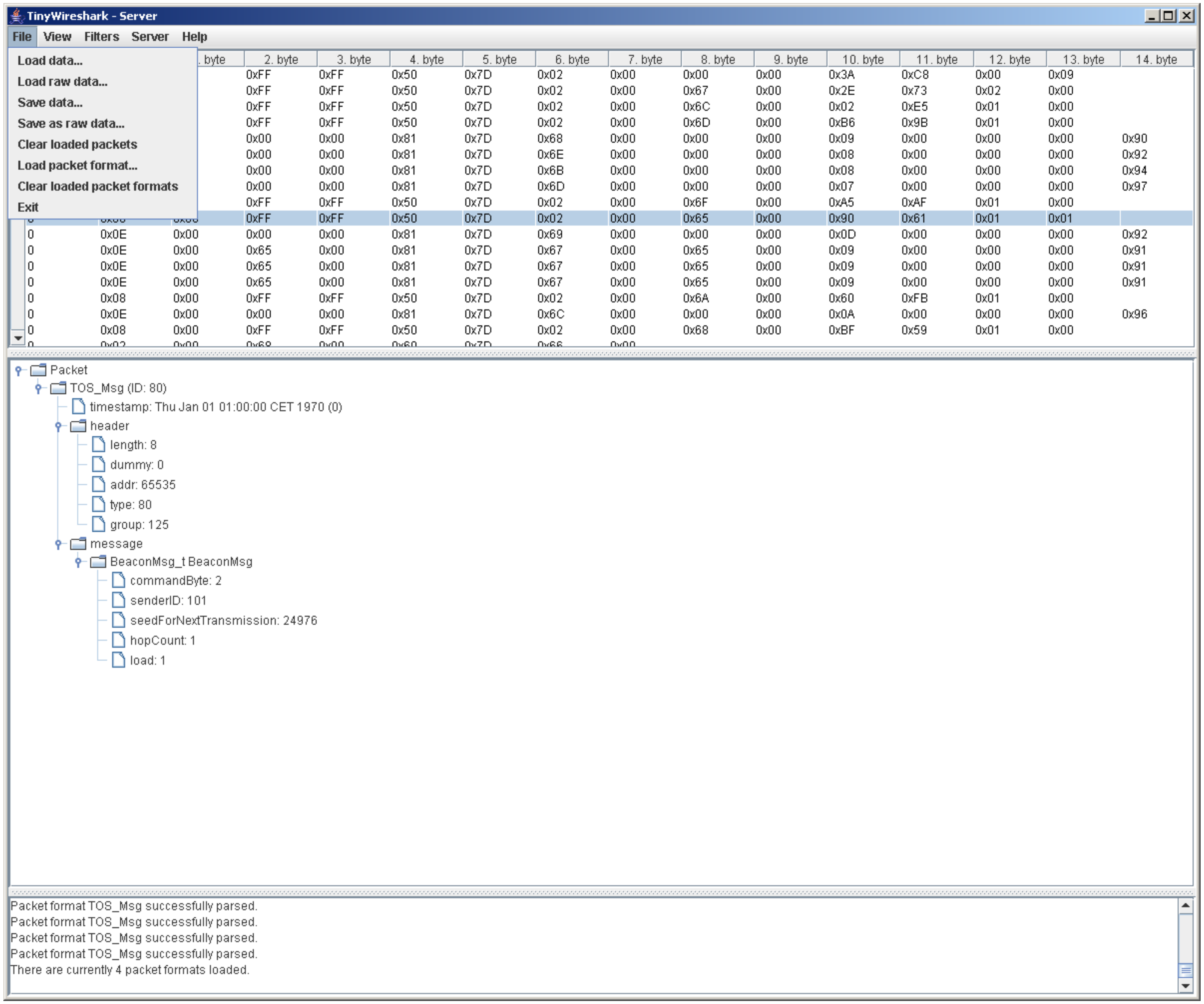
3 http://java.sun.com/j2se/1.5.0/docs/guide/jni/index.html

The button *stop data gathering* is used to prevent the *TinyWireshark Client* from receiving incoming sensor network packets from the *SerialForwarder.*

If the button *send gathered data* is clicked, the *TinyWireshark Client* tries to send the locally stored data to *TinyWireshark Server* according the specified IP parameters. If no *TinyWireshark Server* is available or if the network connection is interrupted, the user can save the locally stored data to hard disk. This can be done either in a text file (raw data format) or as serialized Java objects.

Pressing the button *delete gathered data*, the *TinyWireshark Client* leads to the deletion of all stored data.

If the *live mode* check box is activated, the *TinyWireshark Client* immediately sends every sensor network packet received from the *SerialForwarder* to the *TinyWireshark Server* (together with the timestamp) and does not store the data locally.



TinyWireshark - Server

File View Filters Server Help

Load data...
Load raw data...
Save data...
Save as raw data...
Clear loaded packets
Load packet format...
Clear loaded packet formats
Exit

		. byte	2. byte	3. byte	4. byte	5. byte	6. byte	7. byte	8. byte	9. byte	10. byte	11. byte	12. byte	13. byte	14. byte
			0xFF	0xFF	0x50	0x7D	0x02	0x00	0x00	0x00	0x3A	0xC8	0x00	0x09	
			0xFF	0xFF	0x50	0x7D	0x02	0x00	0x67	0x00	0x2E	0x73	0x02	0x00	
			0xFF	0xFF	0x50	0x7D	0x02	0x00	0x6C	0x00	0x02	0xE5	0x01	0x00	
			0xFF	0xFF	0x50	0x7D	0x02	0x00	0x6D	0x00	0xB6	0x9B	0x01	0x00	
			0x00	0x00	0x81	0x7D	0x68	0x00	0x00	0x00	0x09	0x00	0x00	0x00	0x90
			0x00	0x00	0x81	0x7D	0x6E	0x00	0x00	0x00	0x08	0x00	0x00	0x00	0x92
			0x00	0x00	0x81	0x7D	0x6B	0x00	0x00	0x00	0x08	0x00	0x00	0x00	0x94
			0x00	0x00	0x81	0x7D	0x6D	0x00	0x00	0x00	0x07	0x00	0x00	0x00	0x97
			0xFF	0xFF	0x50	0x7D	0x02	0x00	0x6F	0x00	0xA5	0xAF	0x01	0x00	
			0xFF	0xFF	0x50	0x7D	0x02	0x00	0x65	0x00	0x90	0x61	0x01	0x01	
0	0x0E	0x00	0x00	0x00	0x81	0x7D	0x69	0x00	0x00	0x00	0x0D	0x00	0x00	0x00	0x92
0	0x0E	0x00	0x65	0x00	0x81	0x7D	0x67	0x00	0x65	0x00	0x09	0x00	0x00	0x00	0x91
0	0x0E	0x00	0x65	0x00	0x81	0x7D	0x67	0x00	0x65	0x00	0x09	0x00	0x00	0x00	0x91
0	0x0E	0x00	0x65	0x00	0x81	0x7D	0x67	0x00	0x65	0x00	0x09	0x00	0x00	0x00	0x91
0	0x08	0x00	0xFF	0xFF	0x50	0x7D	0x02	0x00	0x6A	0x00	0x60	0xFB	0x01	0x00	
0	0x0E	0x00	0x00	0x00	0x81	0x7D	0x6C	0x00	0x00	0x00	0x0A	0x00	0x00	0x00	0x96
0	0x08	0x00	0xFF	0xFF	0x50	0x7D	0x02	0x00	0x68	0x00	0xBF	0x59	0x01	0x00	

```
Packet
  TOS_Msg (ID: 80)
    timestamp: Thu Jan 01 01:00:00 CET 1970 (0)
    header
      length: 8
      dummy: 0
      addr: 65535
      type: 80
      group: 125
    message
      BeaconMsg_t BeaconMsg
        commandByte: 2
        senderID: 101
        seedForNextTransmission: 24976
        hopCount: 1
        load: 1
```

```
Packet format TOS_Msg successfully parsed.
Packet format TOS_Msg successfully parsed.
Packet format TOS_Msg successfully parsed.
Packet format TOS_Msg successfully parsed.
There are currently 4 packet formats loaded.
```

Figure 3: The *TinyWireshark Server* graphical user interface

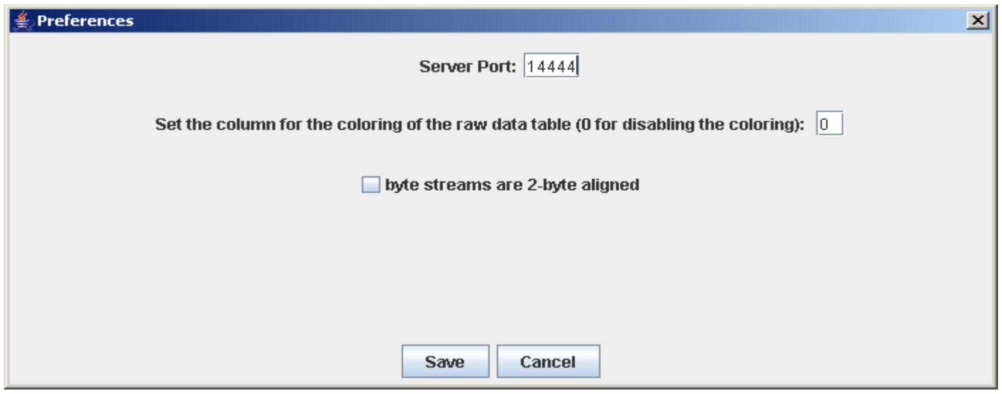
4. TinyWireshark Server

The *TinyWireshark Server* displays the gathered sensor network packets in a user-friendly way. The source code is divided into the four packages *server, gui, filtersetLoading* and *packetFormatloading*.

The package *server* is the main package. It is responsible for the data model, for receiving data from the *TinyWireshark Clients,* for loading and storing data from and to hard disk, and for the coordination of all other packages. The data model is a vector of packets. Each packet consists of a timestamp and the actual network data packet.

There are two possibilities how data may be added to the *TinyWireshark Server*. The first way is to receive data directly from the *TinyWireshark Clients.* The user can start the server using the *Server menu* (see Figure 3). The port the server will listen to can be specified in the *Preferences window* (see Figure 4). For each incoming data packet, the *TinyWireshark Server* will create a new thread for handling the packet. The second way is to load data from hard disk. Data packets from hard disk are either stored as serialized Java objects or as raw data. Raw data means each byte is stored as a hexadecimal number consisting of two digits. Raw data files are readable by every text editor. They were stored either by a *TinyWireshark Server* or by a *SerialForwarder,* for example if the user just used the *SerialForwarder* for gathering sensor network data. Serialized Java objects were stored from the *TinyWireshark Client* or *TinyWireshark Server.*

There is also a possibility to save data to hard disk, either as serialized Java objects or as raw data.


Server Port: 14444

Set the column for the coloring of the raw data table (0 for disabling the coloring): 0

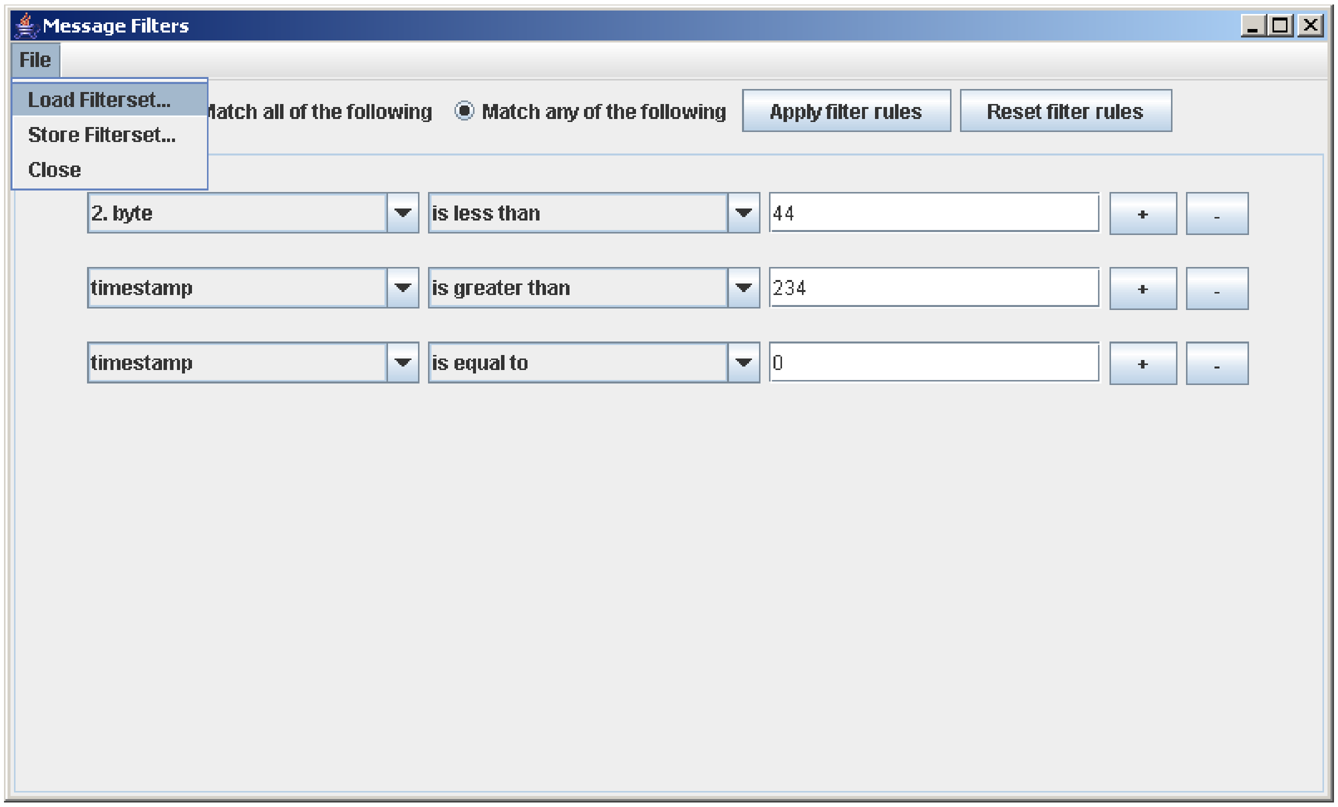
☐ byte streams are 2-byte aligned

Figure 4: The Preferences window of the *TinyWireshark Server*

The package *gui* is responsible for the whole graphical user interface, including the visible data representation (see Figure 3). Data is presented in two ways.

In the upper part of the GUI, data packets are represented in a table as byte streams together with the timestamp (if available) in the first column. Byte values are displayed either as hexadecimal or decimal strings. The user can choose the format using the *view* menu. Further, it is possible to color table rows according to a specified column. The column can be set in the *Preferences window* (see Figure 4). And last but not least, there exists a filter mechanism where the user can define a filter set (see Figure 5). Data packets have to pass the filter set before they are displayed in the table. It was a requirement to allow loading and displaying hundreds of thousands of packets. If the program always rendered all packets in the visible table, the memory usage would be several hundreds of mega bytes, which of course does not make any sense. I got two ideas to solve the problem: either to reduce the memory usage of the actual cells of the table or to render the table dynamically. I decided to implement the dynamic rendering. Therefore, the table renders and displays only the necessary rows, which are currently visible to the user. If the user scrolls up or down, unneeded rows are deleted and new necessary rows are added.

In the lower part of the GUI, the current selected data packet (a row) of the table is displayed as a tree according to a corresponding sensor network packet format. If no fitting sensor network packet format is found an empty tree is displayed. XML-defined sensor network packet formats can be loaded using the *File* menu. The *Preferences window* (see Figure 4) allows specifying whether the actual data of the sensor network packets (which is nothing more than a byte stream) is two byte aligned or not. This possibility is provided because some sensor node architectures require that each address of all two and four byte values is two byte aligned.



2. byte	is less than	44
timestamp	is greater than	234
timestamp	is equal to	0

Figure 5: The Filter window of the *TinyWireshark Server*

The package *packetFormatLoading* provides the functionality for loading XML-defined sensor network packet formats (see XML schema Figure 8). It was desirable to implement a possibility to define and load such packet formats in an easy way even after the application was compiled and rolled out. I decided to implement this interface using XML. With XML, the user can define a new format in a few minutes and he does not need programming skills. Figure 9 shows an example of an XML defined sensor network packet format. Another argument is the availability of fast and easy-to-use XML parsers in J2EE[4]. If the user has loaded one or several sensor network packet formats and clicks a row in the packet table, the *TinyWireshark Server* searches for a fitting format. If a definition is found, the packet is parsed according to the corresponding sensor network packet format and is displayed as a tree in the lower part of the GUI.

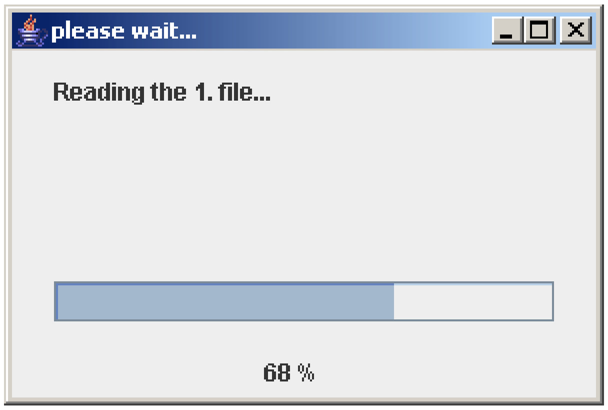


Figure 6: A Progress window of the *TinyWireshark Server*

The package *filtersetLoading* provides the functionality for defining and applying a set of filters (see Figure 5) and for loading and storing XML-defined filter rules to and from hard disk. The rule set is stored in an XML format (see Figure 7). I chose XML again because of the availability of the fast and easy-to-use XML parsers in J2EE[5]. An additional advantage is the possibility to modify a stored rule set in an easy way. Therefore, also an XML file written by a text editor can be loaded. Of course, the file has to be valid according to the defined XML schema (see Figure 7).

4 http://java.sun.com/webservices/docs/1.5/api/index.html
5 http://java.sun.com/webservices/docs/1.5/api/index.html

5. Conclusion and Future Work

The biggest challenges were performance issues, for example the updating of the table and the GUI updating procedure in general (especially if there are several *TinyWireshark Clients* running in *live mode).* It is now still possible that the GUI updating procedure is too slow, if the arrival rate of data packets is too high. But nevertheless the actual functionality of the *TinyWireshark Server* should work properly unless there are not enough resources for handling all incoming data packets.

Another challenge was handling the variety of sensor node platform architectures and sensor network packet formats. Some parts of the *TinyWireshark Server* had to be rewritten and new code had to be added to some Java classes used by the *SerialForwarder* to support additional sensor node platform architectures. A detailed description of how to add a new architecture is available in the readme.txt file, in the package *Semester Thesis Traffic Monitoring in Sensor Networks*.

As this is the first released version of *TinyWireshark*, there is a lot of potential for improvements and additional features. Some ideas are:

a) at the *TinyWireshark Server*

- Recognition of duplicated packets (if a single sensor network packet is gathered by two *TinyWireshark Clients*)

- Allow filter rules according to loaded packet formats

- Allow sorting of loaded sensor network packets according to a selectable column

- Allow importing of sensor network packet formats given as data structures in C header files

- Make the *TinyWireshark Server* scalable according to incoming network load

- Improve the data model to reduce memory usage

- Reduce the memory usage during the conversion of raw data to the internal data model

- Improve the performance of the GUI updating procedures

b) at the *TinyWireshark Client*

- Improve the data model to reduce memory usage

- Improve the accuracy of the timestamp added to gathered data packets

- Improve the performance of the GUI updating procedures

Although the project *traffic monitoring in sensor networks* was quite labor intensive (135 KB Java Bytecode without imports), I enjoyed working on it. I learned a lot about software designing, working with embedded platforms and improved my knowledge in many other parts of computer science.

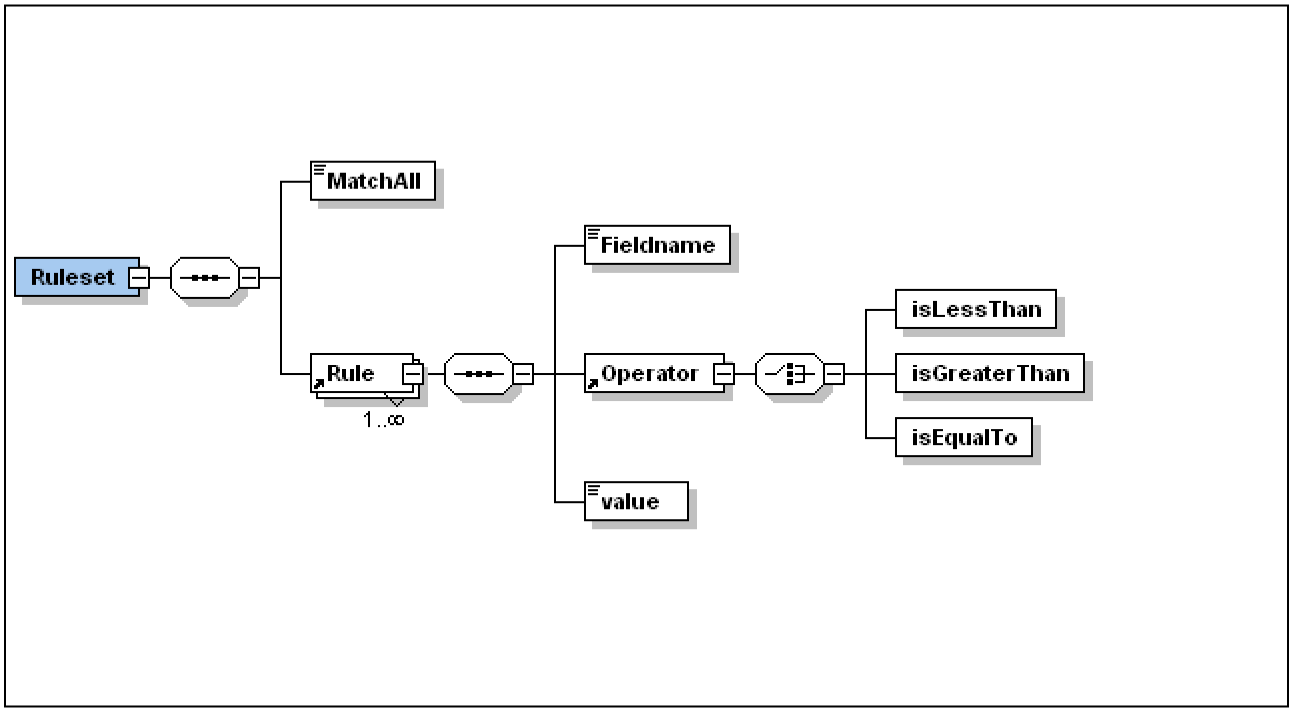


Figure 7: XML schema for the XML rule sets

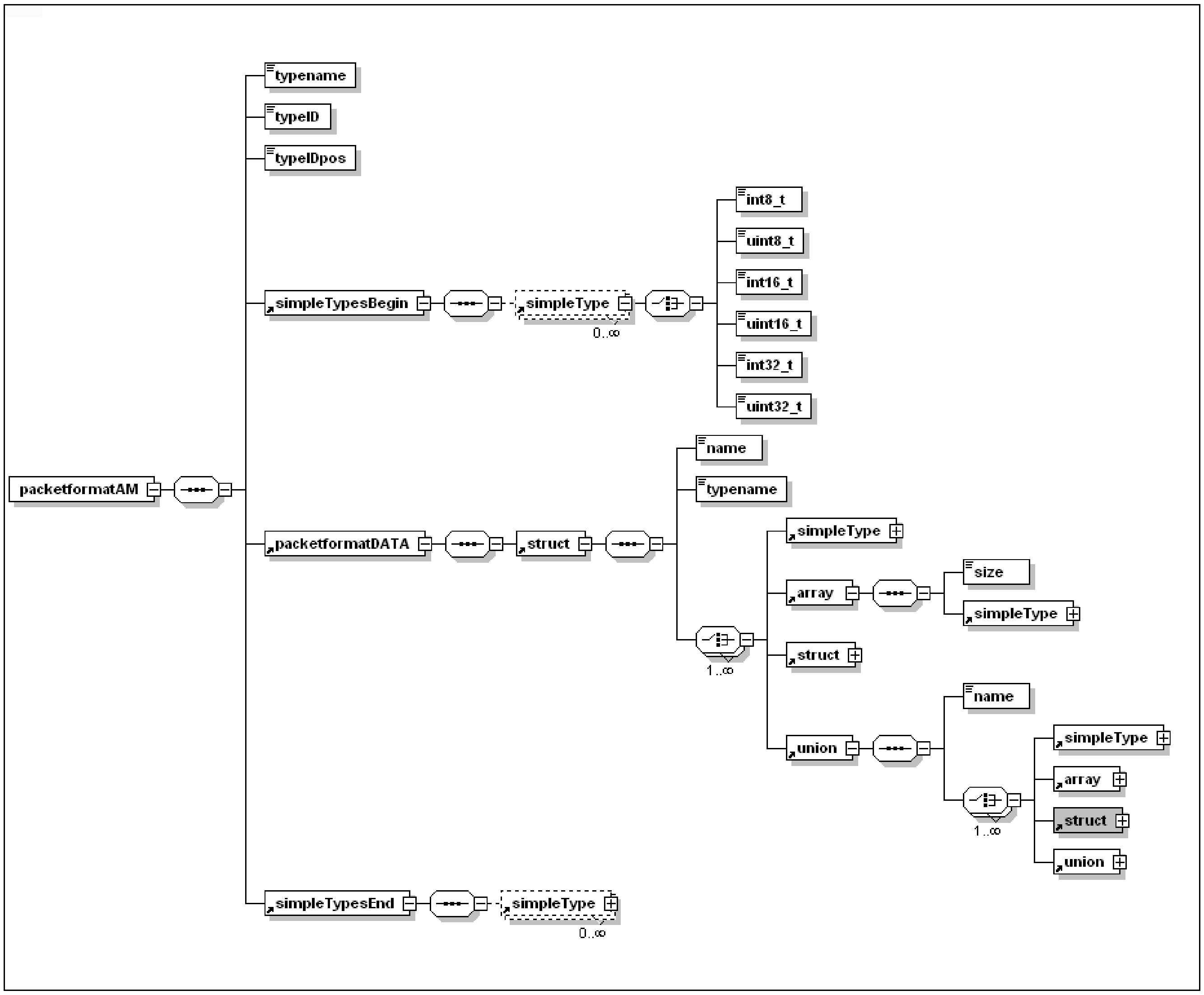
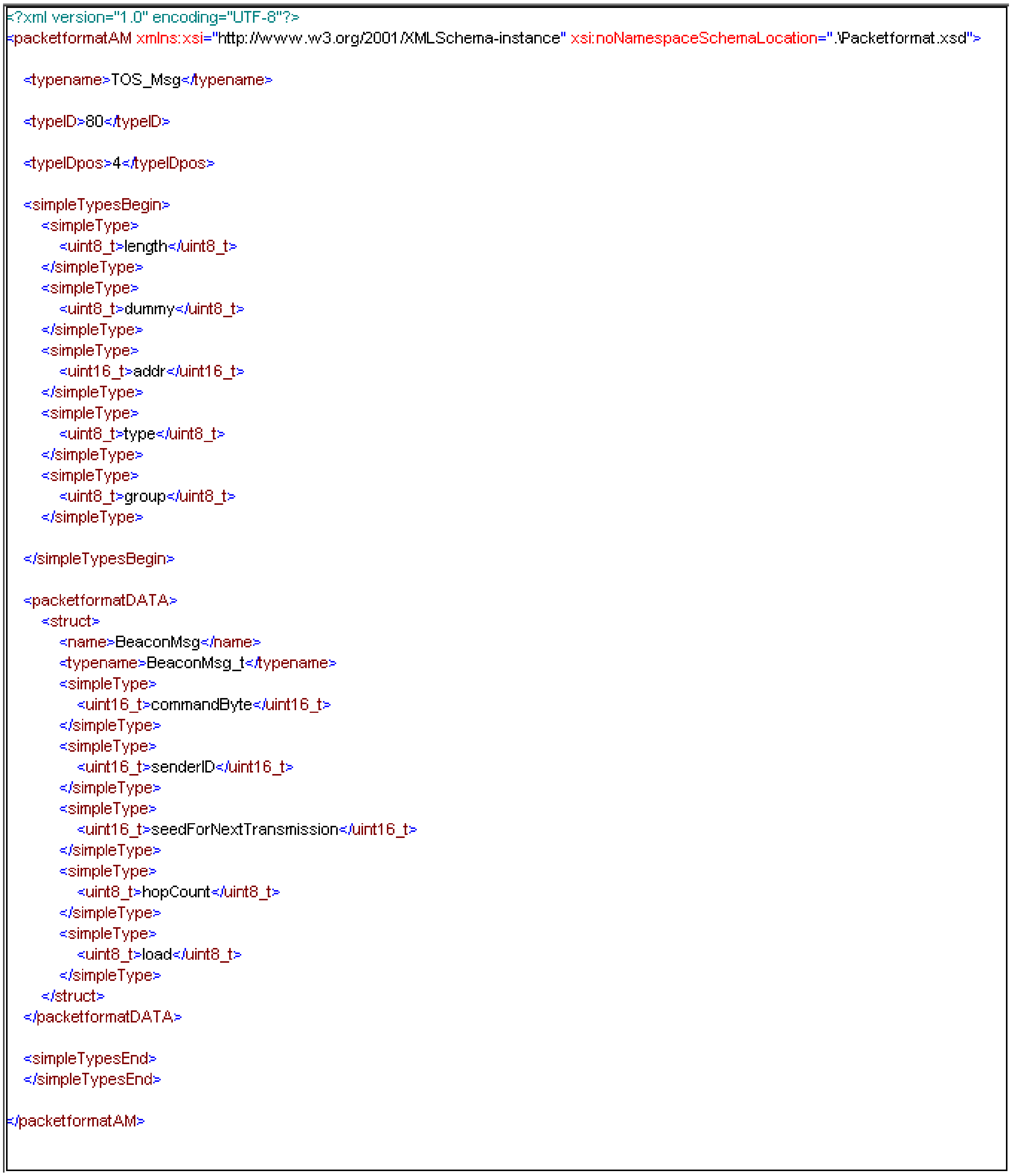


Figure 8: XSD schema for the XML defined packet formats



```xml
<?xml version="1.0" encoding="UTF-8"?>
<packetformatAM xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation=".\Packetformat.xsd">

  <typename>TOS_Msg</typename>

  <typeID>80</typeID>

  <typeIDpos>4</typeIDpos>

  <simpleTypesBegin>
    <simpleType>
      <uint8_t>length</uint8_t>
    </simpleType>
    <simpleType>
      <uint8_t>dummy</uint8_t>
    </simpleType>
    <simpleType>
      <uint16_t>addr</uint16_t>
    </simpleType>
    <simpleType>
      <uint8_t>type</uint8_t>
    </simpleType>
    <simpleType>
      <uint8_t>group</uint8_t>
    </simpleType>

  </simpleTypesBegin>

  <packetformatDATA>
    <struct>
      <name>BeaconMsg</name>
      <typename>BeaconMsg_t</typename>
      <simpleType>
        <uint16_t>commandByte</uint16_t>
      </simpleType>
      <simpleType>
        <uint16_t>senderID</uint16_t>
      </simpleType>
      <simpleType>
        <uint16_t>seedForNextTransmission</uint16_t>
      </simpleType>
      <simpleType>
        <uint8_t>hopCount</uint8_t>
      </simpleType>
      <simpleType>
        <uint8_t>load</uint8_t>
      </simpleType>
    </struct>
  </packetformatDATA>

  <simpleTypesEnd>
  </simpleTypesEnd>

</packetformatAM>
```

Figure 9: An XML defined packet format