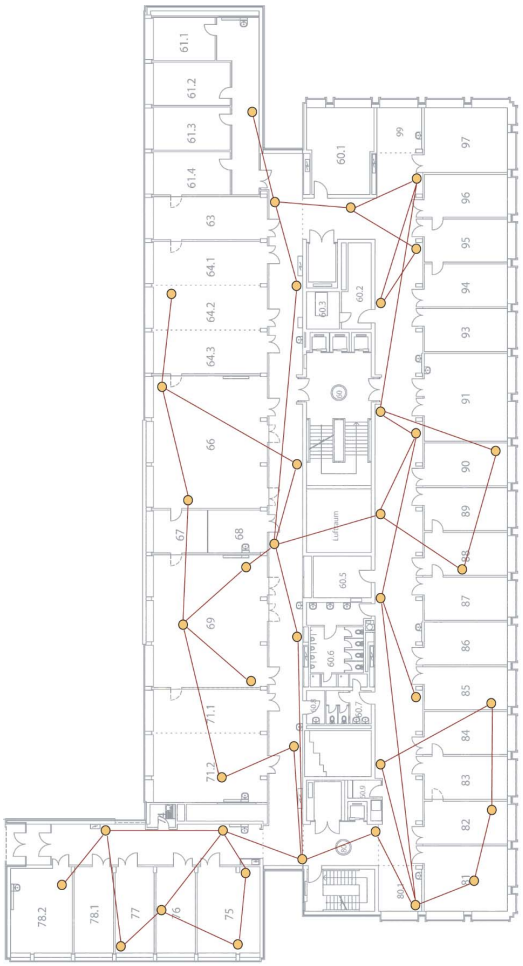
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TUHH
Technische Universität Hamburg-Harburg

Online Sensor-Network Monitoring



Sven Christian Zimmermann

Studienarbeit SA-2005-26
Sommersemester 2005

Betreuer:
Dipl-Ing. Jan Beutel
Dipl-Ing. Matthias Dyer

Professoren:
Prof. Dr. rer. nat. Ulrich Killat
(Technische Universität Hamburg-Harburg)

Prof. Dr.-Ing. habil. Lothar Thiele
(Eidgenössische Technische Hochschule Zürich)

25.07.2005

TIK
Institut für Technische Informatik
und Kommunikationsnetze
Computer Engineering and
Networks Laboratory

Vorwort

Diese Arbeit wurde in der Zeit vom 25. April bis 25. Juli 2005 an der Eidgenössischen Technischen Hochschule Zürich im Institut für Technische Informatik und Kommunikationsnetze unter der Leitung von Herrn Prof. Dr.-Ing.habil. Lothar Thiele erstellt.

Von Seiten der Technischen Universität Hamburg-Harburg fand die Betreuung durch Herrn Prof. Dr.rer.nat. Ulrich Killat aus dem Arbeitsbereich Kommunikationsnetze statt.

Mit meiner Unterschrift versichere ich, dass ich diese Arbeit eigenhändig erstellt habe.

Zürich, 25. Juli 2005

Sven Christian Zimmermann
Ohlstedter Str. 30
22397 Hamburg
Deutschland

Matrikel Nr. 15628

Danksagung

Ich möchte mich bei allen bedanken, die mir diese Arbeit ermöglicht haben:

Herrn Prof. Ulrich Killat danke ich ganz herzlich für die Betreuung meiner Arbeit von Hamburg aus, was eine Zusammenarbeit mit der ETHZ überhaupt erst ermöglicht hat.

Herrn Prof. Lothar Thiele danke ich für die freundliche Betreuung vor Ort.

Meinen Betreuern Jan Beutel und Matthias Dyer danke ich für die freundliche und qualifizierte Unterstützung, die mir jederzeit zur Verfügung stand, sowie für die hilfreichen Anregungen und die unbürokratische Organisation im Vorwege.

Lukas Winterhalter und Daniel Hobi möchte ich für die nette Zusammenarbeit, die interessanten Diskussionen, sowie die Hilfestellungen aller Art danken.

Dem gesamten Institut TIK möchte ich herzlich für die freundliche und hilfsbereite Arbeitsatmosphäre danken.

Inhalt

1

Einleitung

1.1 Einführung

Eine bekannte Vision für ad hoc Netzwerke [16] geht davon aus, dass unendlich viele kleinste „Sensorknoten" kollaborativ ein Netzwerk und somit eine Applikation bilden. In anderen Visionen [14, 5] wird davon ausgegangen, dass solche Systeme weite Anwendungsbereiche abdecken können und dass die einzelnen Komponenten unterschiedliche Ressourcen aufweisen.

Die BTnodes [4] bestehen aus einem Atmel AVR Mikrokontroller, einem Bluetooth Modul und einem Low-Power Radio. Zusammen mit der im NCCR-MICS [32] entwickelten BTnut System Software bilden sie eine sehr kompakte programmierbare Plattform für die Entwicklung mobiler ad hoc- und Sensornetze. An diese Knoten können diverse Peripheriegeräte (z.B. Sensoren) angehängt werden. Mit der geeigneten Software bauen viele Sensorknoten selbstständig ein Sensornetzwerk auf, über das die Sensordaten transportiert werden können.

Die Mica Motes und ihr Betriebssystem TinyOS sind ein ähnliches System, das an der UC Berkeley entwickelt [13, 19, 18] und von Crossbow [31] kommerzialisiert wurde. Dieses System arbeitet jedoch mit einem proprietärem Funkprotokoll auf Basis eines Chipcon CC1000 [7] Radios. TinyOS ist heute der de-facto Industriestandard für Sensorplattformen.

Heute werden Applikationen für Sensornetze meist explorativ entwickelt. Hierzu ist relativ viel Aufwand an Personal, Know-How und entsprechenden iterativen Designzyklen notwendig. Erfahrungsberichte von solchen Experimenten gibt es von Szewcyk [25, 26, 24], Cerpa [6], Hemingway [12], Mainwaring [20] und anderen. Erste Ansätze, die koordinierte Methoden und Verfahren eines ganzheitlichen Entwicklungsprozesses zum Ziel haben, gibt es bereits. Insbesondere sind in den Teilbereichen der Simulation [18, 21, 17], Emulation [10],

Entwicklung [9, 4], Inbetriebnahme [15], Test [30], Validierung und Verifikation [3] Lösungen vorhanden. Einen besonderen Ansatz stellt hier das so genannte Deployment-Support Netzwerk (DSN) [2, 1, 3] dar, welches als temporäres Werkzeug während des Entwicklungs- und Inbetriebnahmeprozesses sowie zur Überwachung angewendet werden kann.

1.2 Aufgabenstellung

In dieser Arbeit sollen die bestehenden Monitoring Werkzeuge in eine Web-fähige Lösung (z.B. Java Applet) portiert werden. Dazu wird es notwendig sein, eine serverseitige Koppelung mit dem DSN-Netzwerk herzustellen sowie einen Client zu implementieren, der die Darstellung im Web-Browser übernimmt.

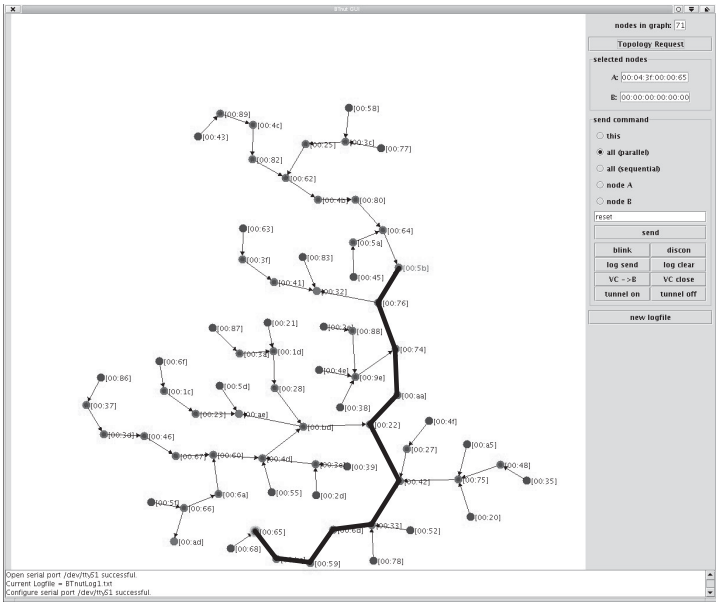


Abbildung 1-1 JAWS: Deployment Support-Network Monitoring GUI.

Neben der Entwicklung von geeigneten Werkzeugen wird es in dieser Arbeit wichtig sein, sich mit aktuellen Trends und Forschungsergebnissen auseinanderzusetzen. Die Arbeit ist in das NCCR-MICS Projekt eingebunden.

Das vorhandene Jaws[1] GUI[2] (siehe Abb. 1-1) kann für eine Monitoring- und Kontrollstation verwendet werden. Wünschenswert wäre eine Online-Monitoring Funktion, z.B. mittels eines Web Applets (Browser), um einen remote Zugriff zu ermöglichen.

Optional: Logging, erweitern der Funktionalität

[1] Jaws ist kein Akronym sondern ein frei erfundener Name zur Bezeichnung des Netzwerks. Jaw ist die englische Übersetzung von Kiefer. Jaws bezeichnet das aus vielen Knoten über Bluetooth (Zähne) verbundene Netzwerk. Viele Zähne (Bluetooth) ergeben einen Kiefer. [2] GUI = Graphical User Interface

2

Rahmenbedingungen

2.1 Allgemeine Anforderungen

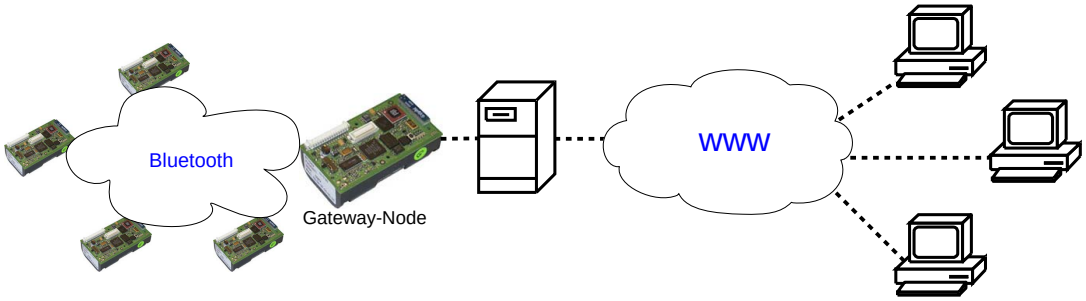


Abbildung 2-1
Grundlegendes Schema des zu entwickelnden Gesamtsystems

Hauptziel war der Zugriff auf das Sensor-Netzwerk via Intranet und Internet (siehe Abb. 2-1), um eine Festinstallation des Sensornetzwerkes allen am Projekt beteiligten Mitarbeitern zugänglich zu machen und gleichzeitig die Möglichkeit zu schaffen, von außerhalb auf das System zugreifen zu können.

Außerdem galt es, das GUI um neue Funktionen zu erweitern und bestehende zu optimieren.

Im Anschluss daran sollten die Möglichkeiten zum Testen und Evaluieren des Netzes verbessert und die von den einzelnen Knoten gesammelten Informationen zentral in einer Datenbank gespeichert werden.

Des Weiteren war erwünscht, die gesamte Konfiguration aus dem Quelltext in eine Konfigurationsdatei auszulagern.

Im Gegensatz zu anderen Arbeiten beschäftigt sich diese nicht mit der Programmierung der BTnodes selber. Somit hatte diese Studienarbeit auch keinerlei Ein-

fluß auf die Netzwerktopologie oder andere bezüglich des Bluetooth-Netzwerks getroffene Entscheidungen. Vielmehr wurde das Interface zu diesem, also die Ein- und Ausgabe auf der seriellen Schnittstelle, genau definiert. Eine detaillierte Beschreibung der Schnittstelle ist im Anhang C zu finden.

2.2 Technische Rahmenbedingungen

Neben den allgemeinen Anforderungen an die Lösung sollten insbesondere folgende Technologien evaluiert und in die Arbeit eingebunden werden:

- die Lösung soll auf modernen Webtechnologien (Java-Applet, HTML, XML) aufbauen
- die Benutzersicht soll webbasiert und browserunabhänig sein (ein HTML-Client mit installierter JRE wird vorausgesetzt)

Zur Lösung des Problems konnte jede geeignete Technologie genutzt werden. Aufgrund der Implementierung des bereits vorhandenen GUIs in Java war es jedoch sinnvoll, auf dem bereits vorhandenen Code aufzubauen und das System in dieser Sprache zu erweitern.

3

Vorgehensweise

Grundlage der Arbeit war eine von Matthias Dyer entwickelte grafische Benutzeroberfläche (siehe Abb. 3-1).

Diese Applikation ermöglicht auf einem direkt mit dem Sensor-Netzwerk verbundenen PC das Abfragen und Darstellen der Netzwerktopologie, das Senden von Befehlen und die Ausgabe der Antworten auf einer Konsole sowie die Simulation eines manuell erstellten Graphens.

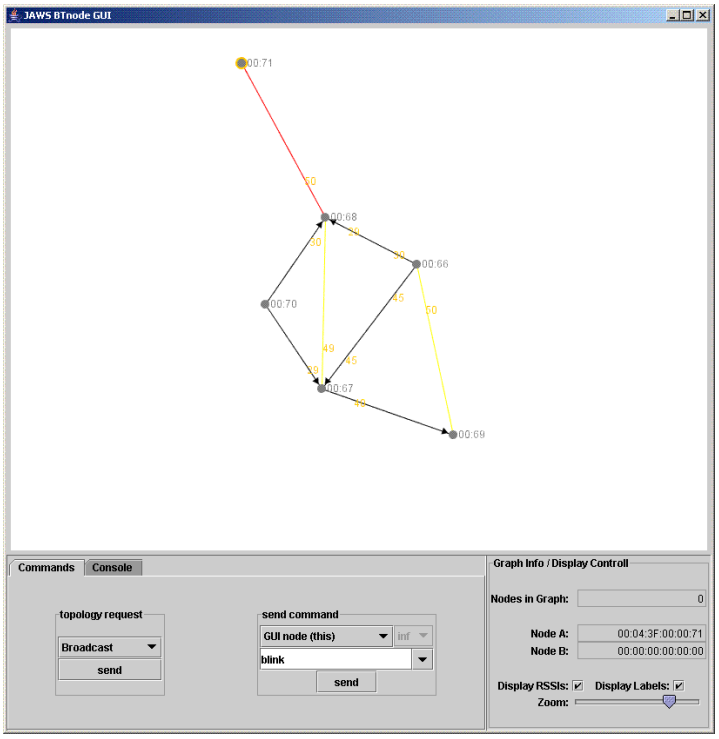


Abbildung 3-1
Die von Matthias Dyer entwickelte graphische Benutzeroberfäche im Originalzustand

Um eine Web-fähige Lösung zu schaffen, wurde das bestehende GUI in ein Applet und ein Servlet aufgetrennt.

Java [23] bietet mit der Remote Method Invocation (RMI) die Möglichkeit, ganze Objekte über das Netzwerk zu verschicken.

Der Vorteil in der Anwendung dieses Verfahrens besteht darin, dass der Graph nur einmal im Server erstellt werden muss und daraufhin an alle Clients übertragen werden kann.

Nachteile ergeben sich jedoch daraus, dass ein Großteil der Logik im Server implementiert ist, was den Kommunikationsaufwand erhöht und besonders bei einer größeren Anzahl von Clients zu Performanceproblemen führen kann.

Das größte Problem besteht jedoch darin, dass die Implementierung des erzeugten Graphens nicht serialisierbar ist, was jedoch eine Grundvoraussetzung zur Nutzung von RMI darstellt.

Deshalb wurde letztendlich ein anderer Ansatz gewählt und die Nachrichten des Gateway-Knotens werden über das Servlet an das Applet übertragen. Erst dort findet das Parsen der Informationen und somit das Erstellen des Graphens statt.

Somit befindet sich ein Großteil der Logik im Applet und der Server ist bezüglich der Anzahl mit ihm verbundener Clients besser skalierbar.

Nach erfolgreicher Implementierung der Client-Server-Kommunikation wurde die Funktionalität des GUIs erweitert, die Datenbankanbindung geschaffen und die Konfiguration des Servers ausgelagert.

4

Architektur

Wie in (Abb. 4-1 oben) dargestellt, besteht das System aus den drei Komponenten BTnode, Server und Client.

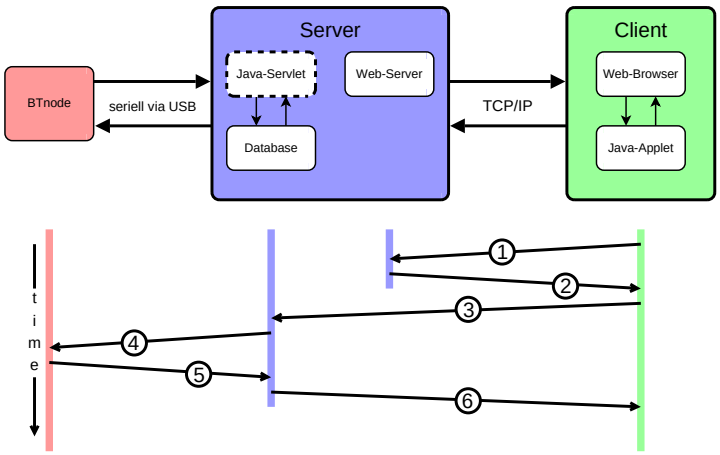


Abbildung 4-1
Oben: *Übersicht über die Client-Server-Architektur*

Unten: *Vereinfachte Darstellung der Kommunikationsreihenfolge beim Starten des Applets im Web-Browser*

Der Server nimmt hierbei die zentrale Vermittlerrolle ein und ermöglicht so die Kommunikation zwischen dem direkt mit ihm verbundenen BTnode und den über TCP/IP kommunizierenden entfernten Clients.

Als Kommunikationsbeispiel innerhalb des Systems soll der Start des Applets näher erläutert werden (vergleiche Abb. 4-1 unten).

1. Der Web-Browser greift auf den Web-Server zu.
2. Dieser sendet das in eine HTML-Seite eingebettete Applet zurück.
3. Nach dem Start des Applets erfragt dieses nun Daten vom Servlet.
4. Das Servlet reicht die Anfrage an den BTnode weiter,
5. erhält die Antwort von diesem zurück

6. und sendet sie an das Applet weiter.

Die Schritte 3 bis 6 wiederholen sich bei jeder Anfrage des GUIs.

4.1 Client

Der Client besteht aus einem Java-Applet (siehe Abbildung 4-2), welches in eine HTML-Seite eingebettet im Web-Browser dargestellt wird.

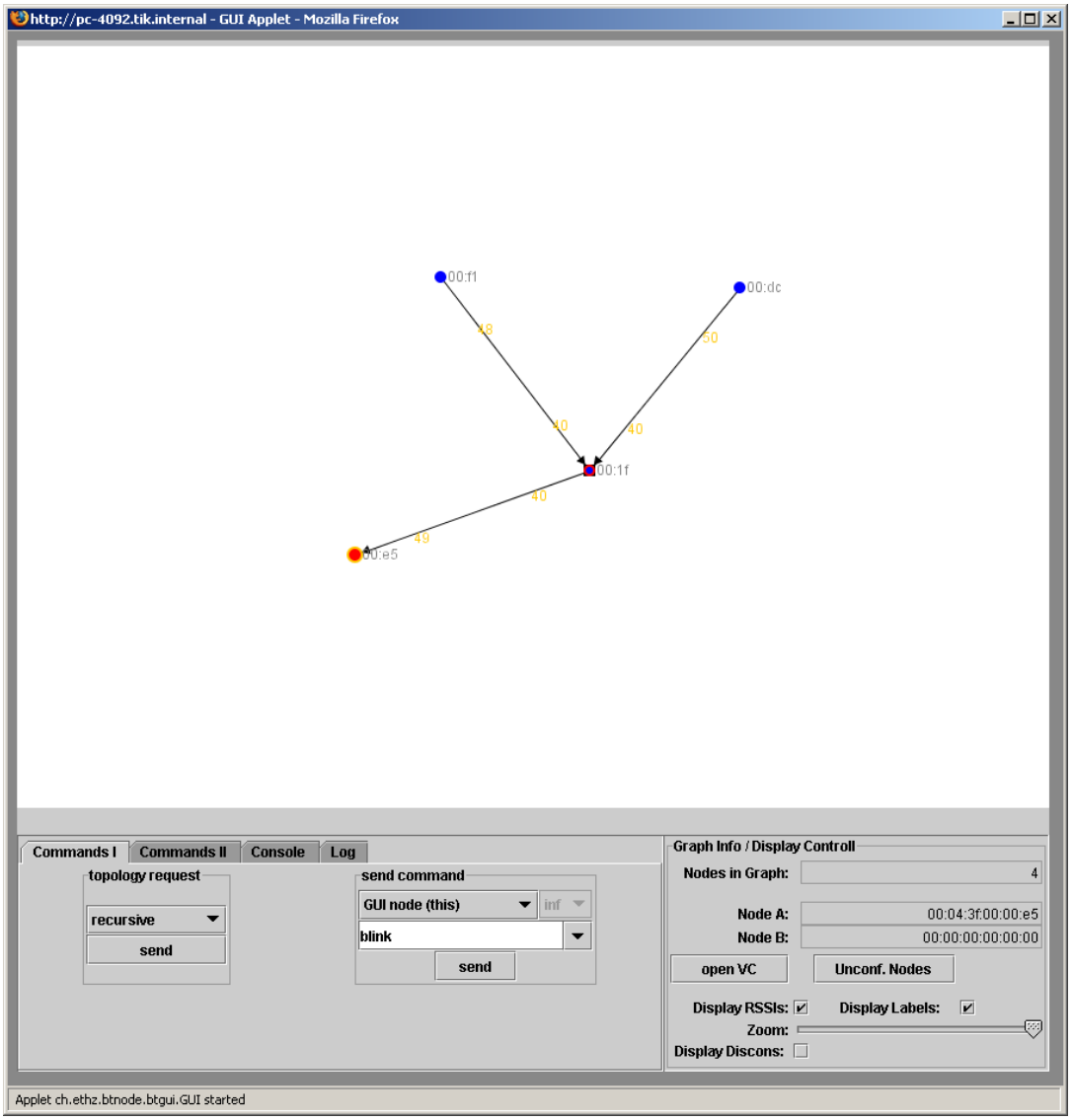


Abbildung 4-2
Java-Applet gestartet im Web-Browser

4.1.1 HTML

Die HTML-Seite [28] fordert den Web-Browser dazu auf, den vom Java-Applet benötigten Code vom Server zu laden und anschließend zu starten.

4.1.2 Java-Applet

Das Java-Applet kann logisch in die folgenden vier Komponenten unterteilt werden:

1. **Kommunikation:** Das Applet stellt eine TCP/IP-Verbindung zu genau dem Server her, von dem es geladen wurde. Eine Kommunikation des Applets mit jedem anderen Rechner widerspricht den Java Sicherheitsrichtlinien und wird vom Applet automatisch unterbunden.
2. **Parser:** Die vom Server erhaltenen Daten werden zeilenweise interpretiert. Die Ankunft von Daten kann den Parser zu Methodenaufrufen veranlassen.
3. **Renderer:** Die vom Parser gesammelten Informationen werden im Graphen optisch aufbereitet.
4. **Darstellung:** Der Graph sowie der Kontroll- und Anzeigebereich werden im Applet angezeigt und ermöglichen eine Interaktion des Benutzers mit dem Sensor-Netzwerk.

4.2 Server

Der Server besteht, wie in Abb. 4-1 oben dargestellt, aus den drei Komponenten Webserver, Java-Servlet und Datenbank.

4.2.1 Webserver

Der Webserver hat in dieser Architektur primär zwei Aufgaben:

1. Bereitstellen von statischen Daten (HTML-Seiten und Code des Applets)
2. Empfangen neuer Dateien zum späteren Verteilen im Sensor-Netzwerk via PHP (Upload eines hex-Files auf den Server)

4.2.2 Java-Servlet

Das Java-Servlet stellt im Normalfall die Kommunikation zwischen dem BTnode und den Clients her. Zusätzlich besteht die Möglichkeit, die Daten der Kommunikation in einer Datenbank zu speichern.

Im Wiedergabemodus (siehe Kapitel 6.3.3, Seite 24) wird ein in der Datenbank gespeicherter Datensatz an den Client gesendet. Eine Verbindung zum BTnode kommt in diesem Fall nicht zustande.

4.2.3 Datenbank

Die Anbindung an eine MySQL-Datenbank ermöglicht einerseits das Speichern der Kommunikation zwischen BTnode und Server zur späteren Analyse sowie andererseits das Abspielen einer zuvor aufgezeichneten oder manuell kreierten Kommunikation.

Diese Aufgaben hätten auch mit Hilfe einer Textdatei gelöst werden können, jedoch bietet eine Datenbank entscheidende Vorteile:

- Eine persistente Verwaltung langfristig zu erhaltender Daten, die einzelne Läufe des Servlets überstehen müssen, wird ermöglicht.
- Eine effiziente Verwaltung dieser Daten wird gewährleistet, so dass ein Zugriff auf einen bestimmten Datensatz möglichst schnell erfolgen kann.
- Die Unterstützung des Datenschutzes durch Benutzerrollen, Rechte, etc. verhindert eine unbeabsichtigte Manipulation der Daten durch Dritte.
- Auf MySQL aufbauende Standardtools zur späteren Analyse können leicht eingebunden werden.

Im Gegensatz dazu bietet eine Textdatei natürlich den Vorteil, die Daten direkter bearbeiten zu können. Dies kann jedoch auch durch ein Web-Frontend[1] der Datenbank ermöglicht werden, was zusätzlich bequem über den Web-Browser zu erreichen ist.

Warum wurde gerade MySQL als Datenbank gewählt?

Der Datenbankserver MySQL ist laut eigenen Angaben die populärste Open-Source-Datenbank der Welt. Er ist plattformunabhängig und hat sich für kleine bis mittelgroße Projekte als Quasi-Standard durchsetzen können. Damit stellt MySQL eine attraktive Alternative zu hochpreisigen, komplexeren Datenbank-Technologien dar und hat sich für diese Anwendung angeboten.

4.3 BTnode

Der BTnode wird mit Hilfe eines speziellen USB-Adapters mit dem Server verbunden und kann dort nach Installation eines Treibers[2] über die serielle Schnittstelle angesprochen werden.

Um mit Java auf die serielle Schnittstelle zugreifen zu können, werden zusätzliche Pakete und Dynamic Link Librarys (DLLs) benötigt.

[1] Das auf PHP basierende PHPMyAdmin ist z.B. ein solches Web-Frontend. [2] BTnode CP2101 USB to UART Bridge Controller

5

Implementation

Das System ist ein Zusammenspiel aus mehreren Komponenten, die in unterschiedlichen Sprachen implementiert wurden. Eine genaue Übersicht über die verwendete Software, Sprachen und Versionen ist im Anhang D zu finden.

5.1 Java

Das Applet sowie auch das Servlet wurden in Java implementiert. Hauptgrund für die Wahl von Java war, dass das bereits bestehende GUI in Java implementiert wurde. Somit war es möglich, den ursprünglichen Code zu nutzen und zu erweitern.

5.1.1 Applet

Eine Vollständige Beschreibung der Implementation würde den Rahmen dieser Arbeit sprengen. Deshalb soll hier nur auf die wesentlichen Punkte näher eingegangen werden.

Die graphische Oberfläche des Applets wurde von Matthias Dyer mit Hilfe der Java-Swing-Bibliotheken erstellt. Swing ermöglicht es, einfach einzufügende GUI-Komponenten zu erstellen, die verschiedene Stile für Buttons sowie Dialoge enthalten, um damit eine moderne Oberfläche für Java-Programme zu erzeugen.

Nach intensiver Einarbeitung in das Thema war es möglich die bestehende Applikation in ein Applet zu portieren. Hierzu konnten die ursprünglich in `JFrames`[1] erzeugten Applikations-Komponenten in ein `JApplet` eingebettet werden.

Neben der Anpassung der graphischen Oberfläche war es auch notwendig in der Logik einige Änderungen vorzunehmen. Die wichtigsten Anpassungen bestan-

[1] JFrame und JApplet sind Klassen aus der Swing-Bibliothek

den in der Portierung der Methode `main()` in `init()` sowie im Entfernen des zuvor vorhandenen Konstruktors.

Eine Java-Applikation ruft beim Starten automatisch die Methode `main()` auf. Ein Java-Applet ignoriert hingegen diese Methode und ruft stattdessen zuerst die Methode `init()` auf. Anschließend wird die Methode `start()` ebenso wie bei jedem erneuten Darstellen des Applets im Vordergrund aufgerufen. Es existieren noch zwei weitere durch äußere Einwirkung aufgerufene Methoden des Applets. Wird ein Applet nicht mehr im Vordergrund dargestellt, so wird die Methode `stop()` aufgerufen. Vor dem Beenden des Applets wird automatisch die Methode `destroy()` ausgeführt.

Die Hauptklasse der Applikation (`GUI`) enthielt einen Konstruktor, der vor dem Aufruf der Methode `main()` bereits einige Variablen und Objekte initialisierte. Ein Applet ist jedoch nicht in der Lage vor Aufruf der Methode `init()` einen Konstruktor auszuführen. Somit musste dieser entfernt und die in ihm enthaltenen Aufgaben an andere Orte verteilt werden.

Des Weiteren soll die Implementierung der rekursiven Topologie-Abfrage genauer beleuchtet werden. Wünschenswert wäre eine Topologie-Anfrage an das gesamte Netz. Bei Versuchen mit einer größeren Anzahl von Knoten hat sich jedoch herausgestellt, dass die Abfrage der Daten via Broadcast zu Engpässen und Datenverlusten führen kann. Ursache ist die auf den Knoten eingesetzte Software, die zur Zeit weder eine ausreichende Pufferung der zu übertragenden Pakete sicherstellt, noch ein erneutes Senden verlorener Pakete initiiert.

Um dennoch bereits jetzt Versuche mit einer größeren Anzahl an Knoten ausführen zu können, war es erwünscht, dass das GUI die Möglichkeit bietet, Daten rekursiv von den Knoten im Netz abzufragen. Die Implementierung benutzt dazu primär zwei Schritte.

In einem ersten Schritt werden alle Knoten als „bisher keine Daten gesendet" markiert. Dazu durchläuft ein Iterator alle sich zu diesem Zeitpunkt im Graphen befindlichen Knoten.

In einem zweiten Schritt folgt die eigentliche rekursive Abfrage der Daten. Dazu wird eine Topologie-Anfrage an den Gateway-Knoten gestellt. Dieser liefert als Antwort die MAC-Adressen aller direkt mit ihm verbundener Knoten zurück.

Da der Gateway-Knoten erfolgreich seine Topologie-Informationen an das GUI übertragen hat, wird dieser als „bereits Daten gesendet" markiert. An alle mit dem zuvor befragten Knoten verbundenen Knoten, die als „bisher keine Daten gesendet" markiert sind, wird eine weitere Topologie-Anfrage gestellt. Somit ist sichergestellt, dass vom Gateway-Knoten ausgehend alle mit dem Netzwerk verbundenen Knoten genau einmal befragt werden.

Um jedoch hierbei das bereits beim Broadcast aufgetretene Problem der Engpässe zu umgehen, werden die Topologie-Anfragen nicht direkt gesendet, sondern

zuvor einer FIFO-Queue hinzugefügt. Wird eine wählbare Anzahl an gleichzeitigen Topologie-Anfragen nicht überschritten, so kann eine weitere an das Netz gesendet werden. Zur Kontrolle wird eine Variable geführt, die beim Senden einer neuen Anfrage erhöht und beim Parsen einer Antwort verringert wird. Sollte auf Grund eines Fehlers keine Antwort von dem befragten Knoten kommen, wird diese drohende Blockierung des Applets durch einen Timeout aufgelöst.

Gleichzeitig kontrolliert das Servlet, dass zu jedem Zeitpunkt nur maximal eine Anfrage pro Knoten gestellt werden kann. Auch hierbei werden mögliche Deadlocks durch einen Timeout abgefangen.

5.1.2 Servlet

In Abb. 4-1 oben (Seite 7) ist das System mit nur einem Client dargestellt. Ziel des Autors war jedoch, dass das System mehrere Clients zeitgleich mit dem Sensor-Netzwerk verbindet.

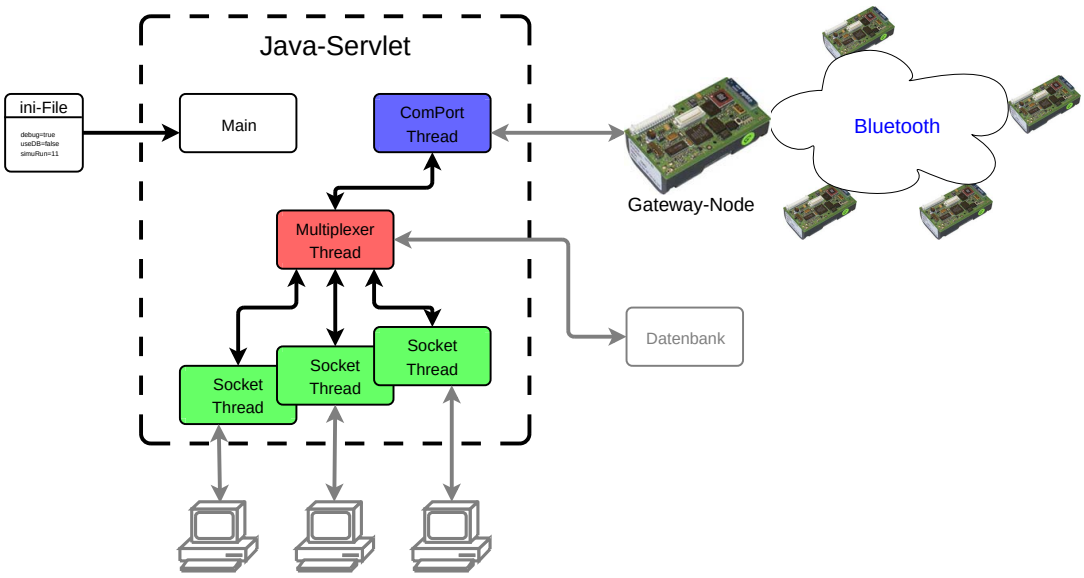


Abbildung 5-1
Kommunikation innerhalb des Java-Servlet

Zum besseren Verständnis soll das schon erwähnte Java-Servlet (siehe Abb. 5-1) näher erläutert werden. Kern des Servlets ist der Multiplexer, der bidirektional die Kommunikation zwischen dem Gateway-Knoten, den Clients und der Datenbank herstellt.

Dazu ist der Multiplexer-Thread zur Inter-Thread-Kommunikation über Pipes sowohl mit dem ComPort-Thread als auch mit den Socket-Threads verbunden.

Beim Start des Servlets existiert initial nur der Main-Thread. Dieser startet den ComPort-Thread sowie den Multiplexer-Thread. Im Anschluss daran wartet der

Main-Thread auf Port 3333 auf eine eingehende Verbindung. Ist eine solche Verbindung zustandegekommen, so wird ein neuer Socket-Thread erzeugt, der diese übernimmt. Da die Verbindung an einen höheren Port verwiesen wurde, ist der Port 3333 wieder frei und der Main-Thread kann erneut auf eine eingehende Verbindung warten.

5.2 Datenbankanbindung

Die MySQL-Datenbank ist über die Java Database Connectivity (JDBC) [22] Technologie in das Servlet eingebunden. Dieses bietet die Möglichkeit, aus Java heraus eine SQL-Anfrage an die Datenbank zu stellen sowie die Antworten zu erhalten und komfortabel weiter zu verarbeiten.

Jede in die Datenbank gespeicherte Nachricht wird mit einem Timestamp versehen. Dies ist auch die Voraussetzung für eine spätere Analyse sowie für die zeitgetreue Wiedergabe.

5.3 Build Tool (Apache Ant)

Ant [8] ist ein in Java geschriebenes Werkzeug zum automatisierten Erzeugen von Programmen aus Quelltext.

Damit erfüllt es den gleichen Zweck wie das weit verbreitete Programm `make`, nämlich die automatisierte Erstellung von installierbaren Software-Paketen aus existierendem Quelltext, Bibliotheken und sonstigen Dateien.

Gesteuert wird Ant durch eine XML-Datei [27], die so genannte Build-Datei (`build.xml`).

Um die Anzahl der zu ladenden Dateien und somit die Wartezeit zu minimieren, werden fast alle benötigten Dateien im JAR-Archiv `gui_applet.jar` zusammengefasst und komprimiert. Dieses beinhaltet neben den Java-Klassen auch das Manifest-File, welches Zusatzinformationen über das JAR-Archiv enthalten kann. In diesem Fall definiert es die Pfade zu drei weiteren vom Applet benötigten JAR-Archiven[2].

Diese drei Archive enthalten ausschließlich Klassen von externen Projekten. Um den modularen Aufbau nicht zu stören, wurde bewusst nicht ein einzelnes großes JAR-Archiv erstellt. Der Zeitgewinn durch das Laden einer einzelnen Datei im Vergleich zu vier Dateien mit gleicher Gesamtgröße ist zu vernachlässigen.

5.4 HTML-Seiten

Die HTML-Seiten [28] enthalten einen `<APPLET>`-Tag, der den Web-Browser dazu auffordert, den Code des Java-Applets vom Server zu laden und anschließend

[2] Folgende Archive werden zusätzlich benötigt: jung-1.5.2.jar, colt-1.2.0.jar und commons-collections-3.1.jar

zu starten. Eine JavaScript-Anwendung passt die Größe des Browser-Fensters und des Applets an die Monitorauflösung des Client-PCs an. Zusätzlich wird sichergestellt, dass pro Browserinstallation nur ein Applet gestartet wird, um weitere Probleme zu vermeiden.

5.5 PHP-Seite

Die PHP-Seite [11] `hex_upload.php` dient wie ihr Name bereits ankündigt, dem Upload einer HEX-Datei auf den Server. Dazu stellt sie im Browser einen Dialog zur Dateiauswahl bereit. Nach erfolgreichem Erhalt der Datei verschiebt das PHP-Skript die empfangene Datei in den gewünschten Ordner und bestätigt dies dem Anwender.

6

Funktionsbeschreibung

6.1 Graphische Benutzeroberfläche (GUI)

Optisch teilt sich das Java-Applet (siehe Abb. 4-2, Seite 8) in den Graphen, der die obere weiße Fläche einnimmt, und den Kontroll- und Anzeigebereich am unteren Ende des Applets.

6.1.1 Graph

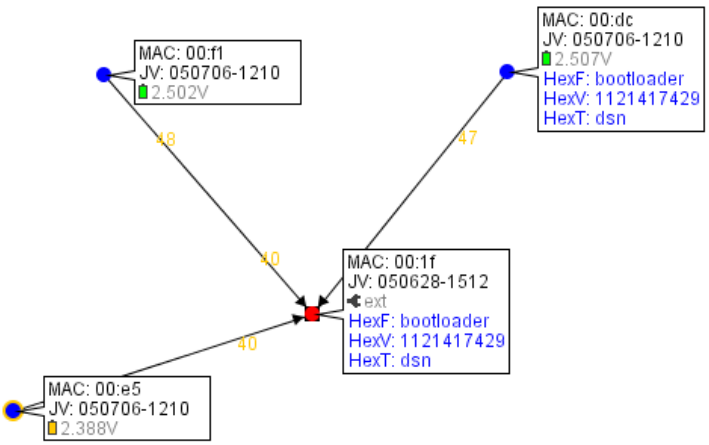


Abbildung 6-1
Graph des Applet mit Zusatzinformationen zu jedem Knoten.

Der Graph (siehe Abb. 6-1) dient sowohl der Darstellung der Netzwerktopologie als auch dem Anzeigen weiterer Informationen ausgewählter Knoten. Er besteht aus Knoten und Kanten. Die Knoten können sich in einem der folgenden Stati befinden:

- **Client:** Ein Client ist mit ein bis drei Servern verbunden und wird im Graphen als blauer Kreis dargestellt.

- **Server:** Ein Server ist mit null bis acht Clients verbunden und wird im Graphen als roter Kreis dargestellt.
- **Client und Server:** Ist ein Knoten gleichzeitig Client und Server, so wird er im Graphen als blauer Kreis mit rotem Rand dargestellt.
- **Unconfirmed:** Ein Knoten gilt als unconfirmed, wenn dieser bei der letzten Topologie-Abfrage keine Antwort gesendet hat oder er manuell auf diesen Status gesetzt wurde. Im Graphen ist er als grauer Kreis dargestellt.
- **Gateway:** Dieser Knoten ist direkt mit dem Server verbunden und dient als Gateway zwischen Server und Bluetooth-Netzwerk. Im Graphen ist er durch eine Umrandung mit einem schwarzen Quadrat erkennbar.

Die grauen Zeichen neben dem Knoten (siehe Abb. 4-2, Seite 8) enthalten die letzten vier Stellen seiner MAC[1]-Adresse.

Die Kanten sind gerichtet und zeigen vom Client zum Server. Wurde eine Verbindung erfolgreich zwischen zwei Knoten hergestellt, so wird dies durch eine schwarze Kante dargestellt. Besteht für zwei Knoten die Möglichkeit sich zu verbinden, sie befinden sich also in Reichweite, sind jedoch nicht verbunden, so wird die Kante gelb dargestellt. Eine Kante wird rot dargestellt, wenn die Verbindung zwischen zwei Knoten durch einen Fehler nicht hergestellt werden konnte.

Die Zahlen an jedem Ende einer Kante geben den von jedem Knoten gemessenen RSSI[2] an.

Innerhalb des Graphens haben die drei Maustasten eine Sonderfunktion und beziehen sich immer auf den zur Mausposition nächstgelegenen Knoten:

- Die linke Maustaste wählt den Knoten als Knoten A aus.
- Die rechte Maustaste wählt den Knoten als Knoten B aus.
- Die mittlere Maustaste erfragt vom Knoten Zusatzinformationen und stellt sie direkt neben diesem in einem Rechteck dar. Ein erneutes Betätigen der mittlere Maustaste entfernt die Zusatzinformationen aus dem Graphen und löscht sie ebenfalls aus dem Speicher des Applets.

Folgende Zusatzinformationen können dargestellt werden (siehe Abb. 6-1):

- MAC-Adresse des BTnode, wobei nur die letzten vier Zeichen angezeigt werden (Prefix MAC)
- Version der JAWS-Firmware im Datum-Zeit Format (Prefix JV)
- Spannungsversorgung: Bei einer externen Spannungsversorgung des Knotens erscheint ein schwarzes Stecker-Symbol, gefolgt von `ext`. Bei einer Batterieversorgung erscheint ein kleines Batteriesymbol, gefolgt von der Spannung als Zahlenwert. Das Batteriesymbol kann drei Farben annehmen. Dabei symbolisiert grün eine ausreichende, orange eine bedenkliche und rot eine kritische Spannung.

[1] MAC = Media Access Control [2] RSSI = Received Signal Strength Indication

- Befindet sich eine HEX-Datei auf dem Knoten, so werden zusätzlich noch der Name (Prefix HexF), die Version (Prefix HexV) sowie das Ziel (Prefix HexT) angegeben.

6.1.2 Kontroll- und Anzeigebereich

Der Kontroll- und Anzeigebereich teilt sich vertikal in die links angeordneten vier auswählbaren Tabs (`Commands I`, `Commands II`, `Console` und `Log`) und den rechts angeordneten statischen Bereich "Graph Info / Display Controll".

6.1.2.1 Der Tab Commands I

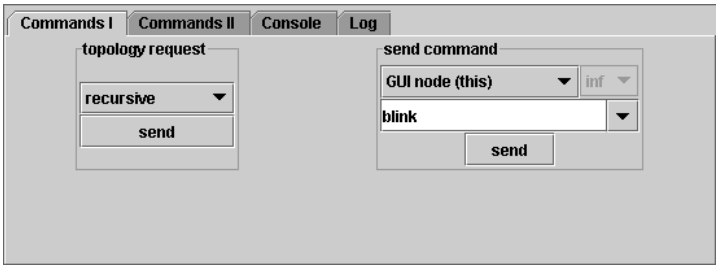


Abbildung 6-2
Tab Commands I beinhaltet die Funktionen „topology request" und „send command"

Der Tab `Commands I` (siehe Abb. 6-2) enthält die zwei Funktionen „topology request" und „send command".

„topology request" ermöglicht ein erneutes Abfragen der Topologie. Hierbei ist es möglich, über das Drop-Down-Menü zu wählen, ob die Abfrage

- als Broadcast zeitgleich an alle verbundenen Knoten,
- rekursiv an alle verbunden Knoten
- oder nur an einen zuvor ausgewählten Knoten A oder B

gesendet werden soll.

„send command" ermöglicht das Senden eines beliebigen Befehls. Über ein Drop-Down-Menü besteht die Möglichkeit zu wählen, ob der Befehl

- nur an den direkt verbundenen Gateway-Knoten,
- über einen Broadcast an alle Knoten
- oder an einen zuvor ausgewählten Knoten A oder B

gesendet werden soll. Zur Erleichterung werden bereits einige Befehle in einem weiteren editierbaren Drop-Down-Menü bereitgestellt. Wird ein noch nicht im Menü enthaltener Befehl gesendet, so wird er diesem hinzugefügt.

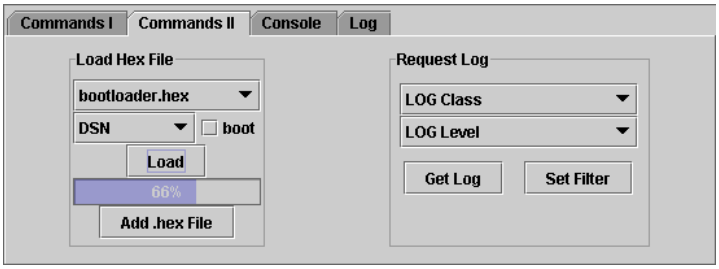


Abbildung 6-3
Tab Commands II beinhaltet die Funktionen „Load Hex File" und „Request Log"

6.1.2.2 Der Tab Commands II

Der Tab `Commands II` (siehe Abb. 6-3) enthält die Funktionen „Load Hex File" und „Request Log".

„Load Hex File" ermöglicht das Laden einer Datei auf den Gateway-Knoten. Das oberste Drop-Down-Menü stellt eine Auswahl aller auf dem Server vorhandener Dateien bereit. Befindet sich die gewünschte Datei noch nicht auf dem Server, so kann sie über den Button „Add .hex File" auf den Server geladen werden.

Das zweite Drop-Down-Menü ermöglicht eine Auswahl zwischen „DSN" und „target" und legt fest, ob die Datei für die Knoten selbst („DSN") oder ein angeschlossenes Target bestimmt ist.

Rechts neben dem beschriebenen Menü befindet sich die Checkbox „boot". Wird diese ausgewählt, so ist es möglich, in Kombination mit einer vorherigen Wahl von „DSN" einen Knoten durch einen „reset" mit einer neuen Firmware zu programmieren. Der Butten „Load" startet den Befehl und die Balkenanzeige informiert über den Fortschritt des Vorgangs.

Eine vollständig auf den Gateway-Knoten geladenen Datei wird sich selbstständig rekursiv im Netz verbreiten. Befinden sich mehrere Dateien im Netz, so werden alle durch die zuletzt auf den Gateway-Knoten geladene Datei ersetzt.

„Request Log" ermöglicht den Empfang von Log-Meldungen eines an den Knoten A angeschlossenen Targets. Über die beiden Drop-Down-Menüs ist es möglich, die Log-Klasse und das Log-Level zu wählen.

Der Button „Get Log" erfragt einmalig vom Knoten A alle gespeicherten Log-Meldungen, passend zur gewählten Log-Klasse und zum gewählten Log-Level.

Der Button „Set Filter" fordert den Knoten A dazu auf, ab sofort alle zur gewählten Log-Klasse und zum gewählten Log-Level passenden Meldungen an das GUI zu senden.

6.1.2.3 Der Tab Console

Der Tab `Console` (siehe Abb. 6-4) enthält eine Konsole, die alle vom und zum Gateway-Knoten gesendeten Nachrichten darstellt. Alle vom GUI ans Netz gesendeten Befehle werden in roter Schrift dargestellt.

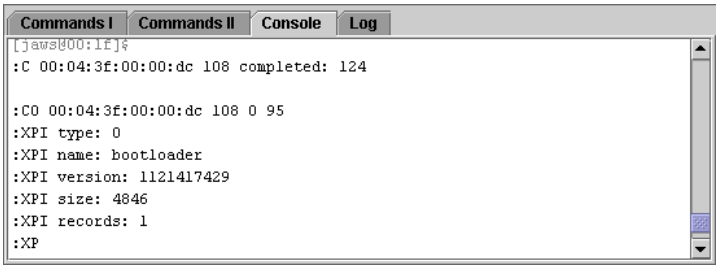


Abbildung 6-4
Der Tab Console beinhaltet eine Konsole zur Anzeige der Kommunikation mit dem Gateway-Knoten.

6.1.2.4 Der Tab Log

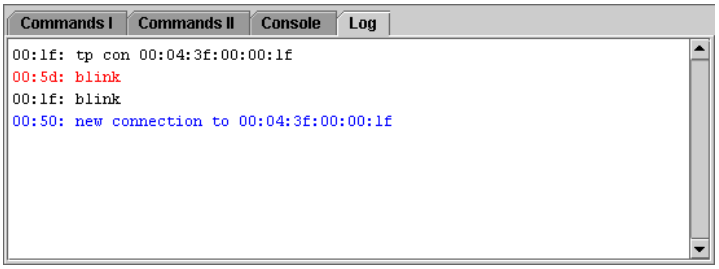


Abbildung 6-5
Tab Log beinhaltet eine Konsole zur Anzeige der von den Knoten empfangenen Log-Meldungen.

Der Tab `Log` (siehe Abb. 6-5) enthält eine Konsole zur Anzeige aller über die Funktion „Request Log" (siehe Kapitel 6.1.2.2) angeforderten Log-Meldungen. Jede Zeile beginnt mit der verkürzten MAC-Adresse der Quelle. Zusätzlich wird den ersten fünf Knoten, von denen eine Log-Meldung empfangen wurde, eine Schriftfarbe fest zugeordnet. Log-Meldungen aller weiterer Knoten werden in grauer Schrift dargestellt.

6.1.2.5 Graph Info / Display Controll

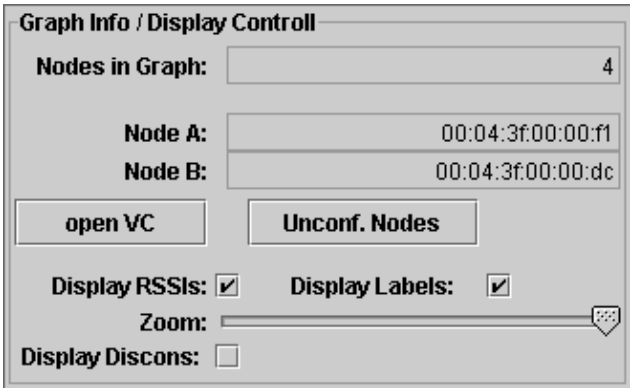


Abbildung 6-6
Der Bereich Graph Info / Display Controll enthält weitere Anzeigen, Optionen und Funktionen.

Der Bereich „Graph Info / Display Controll" enthält mehrere Anzeigen, Optionen sowie Funktionen.

Neben der Anzahl der sich im Graphen befindenden Knoten, werden ebenfalls die MAC-Adressen der als Knoten A und B ausgewählten Knoten angezeigt.

Über den Button „open VC" ist es möglich, eine virtuelle Verbindung vom Knoten A zum Knoten B aufzubauen. Der Button „Unconf. Nodes" setzt den Status aller Knoten im Graph auf „unconfirmed".

Eine virtuelle Verbindung vom Knoten A zum Knoten B wird netzintern folgendermaßen aufgebaut: Das Netzwerk wird von einem Knoten A mit einer route-request-Nachricht geflutet. Dabei speichert jeder Knoten die ID der Verbindung, von der diese Anfrage kam. Dadurch wird eine Route zurück zum Knoten A im Netz aufgebaut. Wenn der Zielknoten B seine Antwortnachricht entlang der Route zurücksendet, weisen die dazwischenliegenden Knoten der Verbindung, von der die Antwort kam, eine lokale Virutal-Circuit-ID zu. Dies ist die Route zum Zielknoten B.

Nachdem die Initialisierung abgeschlossen ist, können über die virtuelle Verbindung Pakete mit minimaler Headerlänge transportiert werden. Sollte eine Verbindung abreißen, so werden die Endpunkte der Verbindung benachrichtigt.

Eine virtuelle Verbindung wird zur Zeit nur zum Aufbau eines Tunnels verwendet. Dieser bietet die Möglichkeit, die serielle Schnittstelle eines BTnodes über das Bluetooth-Netzwerk zu tunneln.

Die Checkboxen „Display RSSIs" und „Display Labels" ermöglichen das Ein- und Ausblenden aller RSSIs bzw. Knotenbezeichnungen.

Der Schieberegler „Zoom" variiert den Abstand zwischen den Knoten.

Über die Checkbox „Display Discons" ist es möglich, die gelben, nicht verbundenen Kanten ein- und auszublenden.

6.2 Server

Es besteht die Möglichkeit den Server sowohl über eine Konfigurationsdatei als auch über ein kleines Server-GUI zu konfigurieren.

6.2.1 Konfiguration via Datei

Die Konfigurationsdatei `Server.ini` (siehe Anhang B.1, Seite 29) wird beim Starten des Java-Servlet eingelesen. In ihr werden alle veränderbaren Parameter festgelegt.

6.2.2 Konfiguration via GUI

Wird in der Konfigurationsdatei `Server.ini` die Benutzung des Server-GUIs aktiviert, so erscheint beim Servletstart automatisch die in Abb. 6-7 dargestellte graphische Benutzeroberfläche.

Das Server-GUI ermöglicht die bequeme Auswahl der Datenbankoptionen beim Serverstart. Der Benutzer muss sich zuerst entscheiden, in welchem Funktionsmodus (siehe Kapitel 6.3) der Server gestartet werden soll.

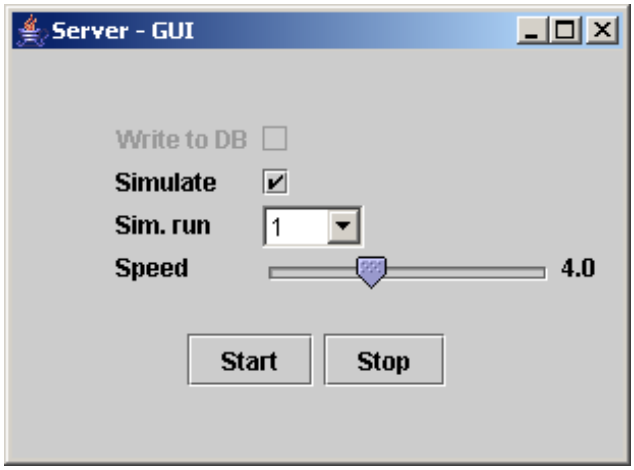


Abbildung 6-7
Server-GUI zur Auswahl der Datenbankoptionen des Java-Servlet

Nach dem Auswählen des Wiedergabemodus (Simulation) werden die weiteren Optionen zur Spezifikation der Wiedergabe aktiviert. Aus dem Drop-Down-Menü kann einer der verfügbaren und zuvor in der Datenbank gespeicherten Datensätze ausgewählt werden. Zusätzlich ermöglicht der Schieberegler das Verändern der Abspielgeschwindigkeit. Der Faktor Speed = 1.0 entspricht der Wiedergabe in Echtzeit. Wird Speed = 4.0 gewählt, so findet die Wiedergabe mit vierfacher Geschwindigkeit statt. Es können Werte zwischen 0,1 und 10 ausgewählt werden.

Der Button „Start" startet den Multiplexer mit den gewählten Einstellungen.

Der Button „Stop" beendet das Servlet und schließt das Server-GUI.

6.3 Funktionsmodi

Das System kann in verschiedenen Modi betrieben werden.

6.3.1 Standard

Im Normalfall kommuniziert das Applet über das Servlet mit dem BTnode (siehe Abb. 4-2, Seite 8).

Zusätzlich ist es möglich, ohne Beeinflussung des Systems alle zwischen Servlet und BTnode ausgetauschten Nachrichten in eine Datenbank (siehe Kapitel 4.2.3, Seite 10) zu speichern. Hierbei wird jede Nachricht mit einem Zeitstempel versehen. Des Weiteren wird bei jedem Start des Servers, bei dem das Speichern der Daten aktiviert ist, ein Index erhöht. Über diesen Index ist es im Wiedergabemodus möglich, genau diesen Datensatz zu wählen.

6.3.2 Lokale Simulation

Beim Start des Applets zur lokalen Simulation wird keine Kommunikation zum Servlet aufgebaut. Statt dessen werden nach dem Start einmalig Nachrichten lo-

kal geparsed. Die Nachrichten müssen dazu im Voraus manuell erstellt werden. Dieser Modus dient primär dem Test und der Entwicklung der Darstellung des Graphens im GUI.

6.3.3 Wiedergabe

Im Gegensatz zur lokalen Simulation ist im Wiedergabemodus eine Kommunikation des Applets mit dem Servlet notwendig. In diesem Modus wird eine zuvor in der Datenbank abgespeicherte Kommunikation erneut abgespielt. Da jede in die Datenbank geschriebene Nachricht mit einem Zeitstempel versehen wird, können Wartezeiten zwischen zwei aufeinanderfolgenden Nachrichten berücksichtigt werden. Dabei ist die Wiedergabe sowohl in Echtzeit als auch über einen Faktor skalierbar schneller oder langsamer möglich.

7

Zusammenfassung und Ausblick

Ziel dieser Studienarbeit war die Implementierung einer über HTTP zugreifbaren graphischen Benutzeroberfläche (GUI) zur Steuerung und Überwachung eines über Bluetooth kommunizierenden Sensornetzwerkes. Diese Aufgabe ist vollständig gelöst worden und zusätzlich wurde das GUI um neue Funktionen erweitert. Gleichzeitig sind auch folgende Ziele erreicht worden:

- Eine Testinstallation auf einem Linux-Server ist funktionsfähig und bildet die Grundlage für eine via Intranet und Internet erreichbare Festinstallation des Systems.
- Diese Studienarbeit bildet die Grundlage für weitere Arbeiten, wie z.B. die Masterarbeit von Daniel Hobi und Lukas Winterhalter mit denen eine enge Zusammenarbeit stattfand.
- Die Implementation wurde bewusst kompatibel zu der Topologie des XTC-Algorithmus [29] entwickelt, um keine Beschränkung des Graphens auf Baumstrukturen zu erzeugen.
- Das System konnte erfolgreich mit mehr als 50 Knoten getestet werden.

Ziel dieser Arbeit war einzig und allein die Programmierung des GUIs und nicht die Weiterentwicklung des zu nutzenden Sensornetzwerkes. Deshalb wird ausschließlich auf die Schnittstelle zu diesem Netzwerk näher eingegangen und nicht auf die dem Sensornetzwerk zugrunde liegenden Konzepte und Implementierungen.

Weitere Aufgaben:
Als weitere Aufgaben für spätere Arbeiten sollten Versuche gestartet werden und eine Analyse der gesammelten Daten stattfinden. Des weiteren können zusätzliche Dienste und Funktionen implementiert werden.

A

Installationsanleitung

Diese Anleitung soll eine Hilfestellung zur Installation des Systems bieten.

Als einfach zu installierende und konfigurierende Serverlösung wurde für die Testinstallation auf einem Linux-PC (Debian) das von Apachefriends[1] entwickelte Xampp eingesetzt. Xampp bietet eine einfache Installation aller für das System benötigter Serverdienste sowie deren Konfiguration über ein Web-Frontend.

Natürlich ist auch eine Installation auf anderen Plattformen möglich oder die manuelle Installation der einzelnen Komponenten unter Linux. Im folgenden wird jedoch von der Installation der Serverdienste via Xampp auf einem Linux-PC ausgegangen. Auf anderen Systemen müssen ggf. Pfade angepasst werden.

Benötigt werden folgende Serverdienste:

- Webserver - Apache 2.0 inkl. PHP
- MySQL-Datenbank
- FTP-Zugang zum automatischen Upload der Software via Ant-Skript

Zusätzlich wird vorausgesetzt, dass auf dem System eine Java Runtime Environment (JRE) inkl. Java Comm Serial Adapter (RXTX) bereits installiert[2] ist.

Xampp (unter Linux auch Lampp genannt) befindet sich nach erfolgreicher Installation im Verzeichnis `/opt/lampp/`. Der Befehl `/opt/lampp/lampp start` startet alle benötigten Serverdienste. Der Befehl `/opt/lampp/lampp stop` beendet sie wieder.

Das Verzeichnis `/opt/lampp/htdocs` enthält alle Dateien und Verzeichnisse auf die der Webserver Apache zugreifen kann. In der Testinstallation wurden

[1] http://www.apachefriends.org [2] Nähere Informationen zur Installation unter Windows siehe http://www.btnode.ethz.ch/projects/jaws/install-win.html

diesem Verzeichnis die zwei Unterverzeichnisse `btnode` und `javaServer` hinzugefügt. Dabei beinhaltet `btnode` alle vom Webserver benötigten Dateien, also die HTML- und PHP-Seiten sowie die JAR-Dateien zur Anzeige des Applets im Browser. Das Unterverzeichnis `javaServer` enthält im Gegensatz dazu alle vom Servlet benötigten Dateien. Dieser Ordner könnte theoretisch auch außerhalb von `/opt/lampp/htdocs` liegen, da ein Zugriff des Webservers nicht stattfindet. Um jedoch alle benötigten Dateien möglichst nahe beieinander zu halten wurde dieser Ort gewählt.

Der Sourcecode sowie alle benötigten Dateien sind sowohl auf der beiliegenden CD als auch auf folgendem CVS-Server zu finden:

Host: `cvs.tik.ee.ethz.ch`

Repository path: `/proj/teccvs/CVS_Tree`

Module: `jaws_gui2`

Zur Implementierung des Systems wurde Eclipse[3] 3.0 genutzt. Nach erfolgreichem Laden des Projekts in Eclipse müssen in der Datei `Server.ini` die Datenbank-Einstellungen sowie der Portname und in der Datei `build.xml` der Pfad `remoteServer` sowie der Benutzername und das Passwort in den Targets `ftp all to WWW-Server` und `ftp all to JAVA-Server` angepasst werden.

Nach erfolgreicher Konfiguration kann das Target `do it all for remote Server` gestartet werden, das wiederum eine Reihe von Ant-Skripten startet, die alle benötigten Dateien in das Verzeichnis `deploy` kopieren und diese im Anschluss via FTP auf den Server lädt. Hierzu ist es notwendig das Xampp bereits gestartet wurde.

Nach erfolgreichem Upload kann auf dem Server im Verzeichnis `/opt/lampp/htdocs/javaServer/` das Servlet gestartet werden. Hierzu ist folgender Aufruf notwendig: `java ch.ethz.btnode.btserver.Server`. **Tipp:** Zur Vereinfachung kann unter Linux der Befehl `screen` benutzt werden.

Im Webbrowser ist die Jumppage des Applets unter dem Pfad `http://{servername}/btnode/` erreichbar. Der Link „GUI-Applet mit Kommunikation" öffnet ein neues Fenster in dem das Applet geladen wird.

Um die Datenbankfunktionalität des Servlets nutzen zu können muss zusätzlich noch eine neue Datenbank in MySQL erstellt werden. Trägt diese den Namen `BTnode` so sind in der Datei `Server.ini` keine weiteren Änderungen nötig. In der neu erstellten Datenbank `BTnode` kann durch Ausführen der Datei `btnode.sql` die benötigte Tabelle `log` erstellt werden. Ein Beispieldatensatz ist bereits enthalten.

Zusätzlich muss in der MySQL-Datenbank ein neuer Benutzer erstellt werden. Benutzername und Passwort müssen mit den Angaben in der Datei `Server.ini` übereinstimmen.

[3] http://www.eclipse.org

B

Sourcecode

B.1 Server.ini

```
#Config file for jaws_gui2

################################################
#               General Settings               #
################################################

#Show debug output
debug=true #[true / false] Default: false

#Print Communication to standard output
printComm=true #[true / false] Default: false

useServerGui=false #[true / false] Default: true


################################################
#                 Connections                  #
################################################

#COM-Port to connect to btNode:
portName=/dev/ttyS9 #name of Com-Port
baudrate=19200 #speed of Com-Port

#TCP-Port to listen on:
serverPort=3333 #Default: 3333
```

```
################################################
#               My-SQL Database                #
################################################

#Enable to use any DB-function (DB must run!!!)
useDB=false #[true / false] Default: false

#Settings:
mySqlDriver=com.mysql.jdbc.Driver
mySqlUrl=jdbc:mysql://localhost:3306/BTnode
mySqlUser=java
mySqlPasswd=******

#Save communication with btNode to DB
writeToDB=false #[true / false] Default: false


################################################
#                 Simulation                   #
################################################

#Enable to simulate the run specified by 'simuRun'
simulate=false #[true / false] Default: false

#Select the simulation run
simuRun=11

#Adjust the simulation speed 1.0 = realtime, 10 = 10xrealtime
speed=1.0     #[0.1 ... 10.0]


################################################
#                   Paths                      #
################################################

#where to save to and load from .hex-files
hexPath=hex #Default: hex
```

C

Protokollspezifikation

C.1 DSN Commands

C.1.1 Topology Information

C.1.1.1 Trace

Trace: The target node sends a reply packet back to the host. On the way to the host each intermediate node appends its address to the packet.

command	`tp trace <addr>`

output (async)	`:TR 1 <1st hop-addr>`
	`:TR 2 <2nd hop-addr>`
	`...`
	`:TR N <addr>`

C.1.1.2 Topology

command	`tp con <addr>`

output (async)	`:T <source-addr> <num-entries>`
	`:TE <1st neighbor addr> <con-state> <NRSSI>`
	`:TE <2nd neighbor addr> <con-state> <NRSSI>`
	`...`

con-state 0 = unconnected, 1 = neighbor is my master, 2 = neighbor is my slave, 3 = denied connection

NRSSI Negative RSSI in dB. Range: [0..100] (0=perfect, 100=bad)

C.1.2 Logging

Definitions

log class: 0 = all classes

log level: 0 = silent, 1 = error, 2 = warning, 3 = info

C.1.2.1 Log Transfer

Output stored log entries to DSN.

command	`dsn sendlog <host-addr> <log-class> <log-level>`
output (async)	`:DGL <log-class> <log-level> failed: <reason>`

sends (async)	`:DL <source-addr> <log-class> <log-level>`
	`<length>`
	`<log-data>`

Permanently send new log entries to DSN according to log class and log levels specified. Deactivated by choosing loglevel zero. Omitting log class/level returns current setup.

command	`dsn logfilter <host-addr> <log-class> <log-level>`
output (async)	`:DLF <log-class> <log-level> ok` `:DLF <log-class> <log-level> failed: <reason>`

sends (async)	`:DL <source-addr> <log-class> <log-level>`
	`<length>`
	`<log-data>`

C.1.3 Remote Command Execution

command	`dsn cmd <trans-id> [<host-addr>]`
input	`<remote-cmd>`

output (async)	`:C <source-addr> <trans-id> failed: <reason>`
	`:C <source-addr> <trans-id> completed:`
	`<total-len>`

output (async)	`:CO <source-addr> <trans-id> <seq-nr> <len>` `<remote-cmd-output>`

trans-id Unique id used by GUI/user to map cmd result pkts to issued cmds.

seq-nr Sequence number always starting at zero.

total-len Length of the complete `<remote-cmd-output>` in bytes.

len Length of the following `<remote-cmd-output>` part in bytes.

C.2 Target Commands

C.2.1 SPI

command	tg flash
output	:TGF ok :TGF failed: <reason>

C.2.2 Logging

Output log buffer to terminal.

command	tg log show <log-class> <log-level>
output	?

Defines which messages are stored in log buffer from target node.

command	tg logfilter <log-class> <log-level>
output	:TGL ok :TGL failed: <reason>

C.2.3 Target Monitoring

command	tg get bat
output	Battery Voltage: <voltage> V

C.2.4 Target Control

command	tg set power on | off
output	:TGP ok :TGP failed: <reason>

C.3 Local Commands

command	get bat
output	Battery Voltage: <voltage> V

voltage format example: Battery Voltage: 3.3 V

C.3.1 Program management

Load program code to local SRAM bank.

command	`loadhex <version> [<type> <active-flag> <name>]`
output	`ready to receive hex data, press enter for quit`
input	program hex data
output	`:LH completed: <nl> lines read` `:LH failed: <reason>`

version 32-bit integer.

type Program type. dsn=0, target=1

active-flag Bootloader will copy the program to flash memory if active-flag=85 and type=0.

name Program name.

command	`xbank get proginfo`
output	`:XPI no program`

output	`:XPI type: <progtype>`
	`:XPI name: <progname>`
	`:XPI version: <version>`
	`:XPI records: <nr-records>`
	`:XPI size: <progsize>`
	`:XPI boot: <boot-addr>`
	`:XPI active: <active-flag>`

command	`xbank set progtype dsn [boot] | target`

command	`xbank set progver <version>`

D

Verwendete Software und Sprachen

Softwarename	Version
Eclipse	3.0.2
Apache Server	2.0.54
MySQL Server	4.1.12
PHP Server	5.0.4
ProFTPD	1.2.10

Tabelle D-1: Verwendete Software

Sprache	Version
Java	j2sdk1.4.2_06
Apache Ant	1.6.2
MySQL Connector	3.1.8
Jung	1.5.2
PHP	5.0.4

Tabelle D-2: Verwendete Sprachen und Pakete

Abbildungen

Literaturverzeichnis

[1] J. Beutel. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, chapter Location Management in Wireless Sensor Networks. CRC-Press, Boca Raton, FL, 2004.

[2] J. Beutel, M. Dyer, M. Hinz, L. Meier, and M. Ringwald. Next-generation prototyping of sensor networks. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 291–292. ACM Press, New York, November 2004.

[3] J. Beutel, M. Dyer, L. Meier, and L. Thiele. Scalable topology control for deployment-sensor networks. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 359–363. IEEE, Piscataway, NJ, April 2005.

[4] J. Beutel, O. Kasten, F. Mattern, K. Römer, F. Siegemund, and L. Thiele. Prototyping wireless sensor network applications with BTnodes. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 323–338. Springer, Berlin, January 2004.

[5] L. Blazevic, L. Buttyan, Capkun S., S. Giordano, J.P. Hubaux, and J.Y. Le Boudec. Self organization in mobile ad hoc networks: the approach of Terminodes. *IEEE Communications Magazine*, 39(6):166–174, June 2001.

[6] A. Cerpa, J.E. Elson, M. Hamilton, J. Zhao, D. Estrin, and L. Girod. Habitat monitoring: application driver for wireless communications technology. *ACM SIGCOMM Computer Communication Review*, 31(2):20–41, April 2001.

[7] Chipcon. *CC1000, Single Chip Very Low Power RF Transceiver*, April 2002.

[8] The Apache Software Foundation. The apache ant project, 07 2005. http://ant.apache.org/.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC language: A holistic approach to networked embedded systems. In *Proc.*

ACM SIGPLAN 2003 Conf. Programming Language Design and Implementation (PLDI 2003), pages 1–11. ACM Press, New York, June 2003.

[10] L. Girod, J. Elson, A. Cerpa, T. Stathapopoulos, N. Ramananthan, and D. Estrin. EmStar: A software environment for developing and deploying wireless sensor networks. In *Proc. USENIX 2004 Annual Tech. Conf.*, pages 283–296, June 2004.

[11] The PHP Group. Php, 07 2005. http://www.php.net/.

[12] B. Hemingway, W. Brunette, T. Anderl, and G. Borriello. The Flock: Mote sensors sing in undergraduate curriculum. *IEEE Computer*, 37(8):72–78, August 2004.

[13] J.L. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proc. 9th Int'l Conf. Architectural Support Programming Languages and Operating Systems (ASPLOS-IX)*, pages 93–104. ACM Press, New York, November 2000.

[14] J.P. Hubaux, T. Gross, J.Y. Le Boudec, and M. Vetterli. Toward self-organized mobile ad hoc networks: The Terminodes Project. *IEEE Communications Magazine*, 39(1):118–124, January 2001.

[15] J.W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 81–94. ACM Press, New York, November 2004.

[16] J.M. Kahn, R.H. Katz, and K.S.J. Pister. Next Century Challenges: Mobile Networking for Smart Dust. In *Proc. 5th ACM/IEEE Ann. Int'l Conf. Mobile Computing and Networking (MobiCom '99)*, pages 271–278. ACM Press, New York, August 1999.

[17] O. Landsiedel, K. Wehrle, and S. Götz. Accurate prediction of power consumption in sensor networks. In *Proc. 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS-II)*, page to appear. IEEE, Piscataway, NJ, May 2005.

[18] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In *Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003)*, pages 126–137. ACM Press, New York, November 2003.

[19] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, Brewer E., and D. Culler. The emergence of networking abstractions and techniques in TinyOS. In *Proc. First Symp. Networked Systems Design and Implementation (NSDI '04)*, pages 1–14. ACM Press, New York, March 2004.

[20] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *Proc. 1st ACM Int'l Workshop Wireless Sensor Networks and Applications (WSNA 2002)*, pages 88–97. ACM Press, New York, September 2002.

[21] V. Shnayder, M. Hempstead, B. Chen, G. Werner-Allen, and M. Welsh. Simulating the power consumption of large-scale sensor network applications. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 188–200. ACM Press, New York, November 2004.

[22] Inc. Sun Microsystems. Java database connectivity (jdbc), 07 2005. http://java.sun.com/products/jdbc/.

[23] Inc. Sun Microsystems. Javatm 2 platform, standard edition, v 1.4.2 api specification, 07 2005. http://java.sun.com/j2se/1.4.2/docs/api/.

[24] R. Szewczyk, A. Mainwaring, J. Polastre, J. Anderson, and D. Culler. An analysis of a large scale habitat monitoring application. In *Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004)*, pages 214–226. ACM Press, New York, November 2004.

[25] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin. Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40, June 2004.

[26] R. Szewczyk, J. Polastre, A. Mainwaring, and D. Culler. Lessons from a sensor network expedition. In *Proc. 1st European Workshop on Sensor Networks (EWSN 2004)*, volume 2920 of *Lecture Notes in Computer Science*, pages 307–322. Springer, Berlin, January 2004.

[27] W3C. Extensible markup language (xml), 07 2005. http://www.w3.org/XML/.

[28] W3C. Hypertext markup language (html), 07 2005. http://www.w3.org/MarkUp/.

[29] R. Wattenhofer and A. Zollinger. XTC: A practical topology control algorithm for ad-hoc networks. In *Proc. 18th Int'l Parallel and Distributed Processing Symposium (IPDPS '04)*, page 216. IEEE CS Press, Los Alamitos, CA, April 2004.

[30] G. Werner-Allen, P. Swieskowski, and M. Welsh. MoteLab: A wireless sensor network testbed. In *Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05)*, pages 483–488. IEEE, Piscataway, NJ, April 2005.

[31] Crossbow Technology Inc. http://www.xbow.com.

[32] NCCR-MICS: Swiss National Competence Center on Mobile Information and Communication Systems. http://www.mics.org.