BTnodes

A Distributed Environment for Prototyping Ad Hoc Networks

Jan Beutel, Oliver Kasten, Matthias Ringwald
BTnodes are about…

A new class of devices

A new application/OS paradigm

Quick and easy assembly into novel networked applications
Overview

Project goals and directions – The need for prototyping platforms
Bluetooth networking

Building a prototyping architecture
- BTnode hardware details
- Pro’s and con’s – competitor systems
- BTnode System Software details

Event driven OS/application integration

Example applications
Summary and outlook
Initial projects

Smart everyday objects
by attaching sensor nodes:
- self aware
- context sensitive
- cooperative
- integration into computing environment

Ad hoc networking scenarios
- integrated application protocols
- scalable multi-hop routing

Wearable Computing

Ubiquitous Computing
Consumer electronics integration

- Mobile Phone
- BTnode
- BTnode
- PDA
- Camera
- PC Peripherals
Backend connectivity

Connectivity to
- application servers
- other networks

Clusters of mobile networks
- using GSM
- using SMS services
- Wireless LAN
- interfacing to other sensor networks
A new project driver

NCCR MICS – The Terminodes Project

Strategic research thrust within Switzerland
- fundamental and applied questions for a new generation of mobile communication and information systems
- based on self-organization
- all disciplines from device technology to the economic models

Network of academic partners
- government funded, with industry cooperation
- 30+ faculty, 70+ PhDs, 4 years
- EPFL leading house (Martin Vetterli)

BTnodes as a fast prototyping environment for large networks
A new paradigm for networking...

- autonomous
- self-supportive
- cheap
- collaborative
- limited resources
- reusable
- adaptable
- failure
- different scales
- continuous
- large population
- quasistatic
- unreliable
- self-repairing
- mobile
- interoperable
- self-organizing
- self-configuring
- scalable
- reconfigurable
- heterogeneous
- low-power
- infrastructure less
- sporadic
- local
- self-powered
- maintenance
- disposable
- reliable
-十月 1, 2003
The need for a prototyping platform

No (large) ad hoc systems available today
- multihop only investigated in theory and simulation
- test the real thing with systems, protocols and services

Limited resources
- leading to a new application/OS integration paradigm
- borrowing from embedded and distributed systems

So why is nobody doing the real thing?
- bulky, cumbersome
- short standalone operating times
- hard to manage multitude of devices
- limitations on features and communication front-ends
- often only point to point
- SDK’s only for selected “customers”
Initial architecture requirements

A device magnitudes smaller than a PDA … equally flexible and programmable … … and supporting Bluetooth.

Communication  Computation  Sensing
Bluetooth Piconets

Communication organized in piconets
- controlled by one master
- up to 7 active slaves
- 255 inactive (parked) slaves

Master-Slave
- implements centralized control
- synchronization of all slaves
- only master-slave communication

Multiple piconets
- separate channels
- no coordination

Piconets can be combined in Scatternets
Bluetooth Host Controller Interface

**Standard interface for protocol software**
- providing access to lower levels of the protocol stack

```
HCI_COMMAND
HCI_EVENT
```

```
OGF  OCF  PARAMETERS
0     0     max 64 kBytes
```

Applications
- SDP
- RFCOMM
- L2CAP

Host Controller Interface

Host processor

Physical interface

Bluetooth module

Audio
- Link Manager
- Baseband
- RF

October 1, 2003
Bluetooth connections

Managed by the host controller

Statemachine for each connection

Link layer control & adaptation (L2CAP)
- connection-oriented
- connectionless data
- link multiplexing over a single “air interface”
- packet segmentation and reassembly
- channel abstraction
- encryption
- link key security
- …
Building a prototype architecture

Autonomous wireless communication and computing platform based on a Bluetooth radio module and a microcontroller.

Requirements
- small form factor, low component count
- standardized wireless interface
- flexible and cost effective deployment of large quantities of nodes
Hardware details

**Integrated PIFA antenna**

**Communication**
Ericsson Bluetooth Module

**LED, reset, clocks**

**Memory**
- 128 kB Flash
- 244 kB SRAM
- 4 kB EEPROM

**CPU**
Atmel ATmega128L
8-Bit RISC (max. 8 MHz ~8MIPS)

**Power management**

**Generic sensor interfaces**

**UART and I2C data interfaces**

**Dimensions**
- 61 mm
- 40 mm
Designing for power aware operation

- switchable power supply for Bluetooth
- MCU with 6 power down modes, low idle/sleep current
- frequency scaling: 7.3 MHz - 57 kHz

- single power supply (3.6 - 16 V), single internal voltage (3.3 V)
- battery charge indicator
- direct current access shunts for all components
- internal Vcc available at every connector for external sensor modules

<table>
<thead>
<tr>
<th>Current bill of material</th>
<th>50 parts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parts</td>
<td>60 USD</td>
</tr>
<tr>
<td>Assembly</td>
<td>5 USD</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>45 USD</td>
</tr>
</tbody>
</table>

Unit cost @ 200 units 110 USD
### Power consumption example

**Sensor network example: 10% duty cycle**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Power consumption [mW]</th>
<th>Lifetime [h]</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 sec sensing</td>
<td>12</td>
<td>252</td>
</tr>
<tr>
<td>2 sec communication</td>
<td>160</td>
<td>19</td>
</tr>
<tr>
<td>54 sec idle</td>
<td>0.5</td>
<td>6048</td>
</tr>
<tr>
<td><strong>Total duty cycle</strong></td>
<td>~ 6.5 mW</td>
<td>421</td>
</tr>
</tbody>
</table>

*on 840 mAh Li-ion
# A look at our competitors

<table>
<thead>
<tr>
<th>UC Berkeley motes (Culler et al.)</th>
<th>full custom radio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Motes and IPAQ’s (Estrin et al.)</td>
<td></td>
</tr>
<tr>
<td>Mote-on-a-chip “spec” (Hill)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bluetooth devices</th>
<th>proprietary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel research motes (Kling)</td>
<td>limited availability</td>
</tr>
<tr>
<td>Blue Wand (Zitterbart)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Testbeds</th>
<th>complex and low integration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft wake-on wireless (Bahl)</td>
<td></td>
</tr>
<tr>
<td>UCLA i-Badge (Srivastava)</td>
<td></td>
</tr>
<tr>
<td>UCLA/WINS/Rockwell nodes (Srivastava)</td>
<td></td>
</tr>
<tr>
<td>PicoRadio Testbed (Rabaey)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other</th>
<th>low power</th>
</tr>
</thead>
<tbody>
<tr>
<td>TeCo Smart It’s (Beigl)</td>
<td>reduced features</td>
</tr>
<tr>
<td>PicoBeacon (Rabaey)</td>
<td>full custom</td>
</tr>
<tr>
<td>WiseNet (Enz)</td>
<td></td>
</tr>
<tr>
<td>ParkTab (Weiser)</td>
<td></td>
</tr>
<tr>
<td>Active Bat (Hopper et al.)</td>
<td></td>
</tr>
</tbody>
</table>
UCB motes compared

Originated in operating system research, now commercialized

Less I/O and memory, same host MCU

Custom radio frontend (Chipcon CC1000)
- Baseband and MAC processing on host CPU
- less free resources
- much more fine grained RTOS
- no standard wireless interface

Mature system level know-how
- TinyOS operating system
- NesC language abstraction
- TOSSIM simulation environment
- TinyDB sensor query system
- many sensor peripherals available
Intel research mote

Enhanced building block for wireless sensor networks

Prototype based on Zeevo Bluetooth module
- uses embedded ARM core, proprietary
- based on TinyOS
- modular design

Bluetooth advantages
- link level reliability and performance
- interoperability

Future Intel motes
- commercialization in research and industry
- further integration as SOC and open software platform
BTnode System Software

Lightweight OS
- event-driven application model
- cooperative multithreading
- device drivers (UART, RTC, ADC, ...)
- static memory allocation
- minimum memory footprint

Programming
- standard C language
- high-level Bluetooth interface
- system software available as library
- emulation environment on Linux
Event driven OS/application integration

Approach common to embedded systems

Geared towards processing of external events
  – sensor values, data packets, state changes
  – only one handler active at a time

One application per system at a time
  – application resident in device
  – no dynamic process model
  – events triggered by OS/driver functions and applications

No real-time OS knowledge necessary for application design
void handler( /* ... */ ) {}
void main() {
    btn_disp_ev_reg( RECEIVE_EV, handler, 0 );
    btn_disp_run();
}
Distributed event and execution model

Supporting many distributed nodes
- event based communication
- spontaneous interaction
- disconnected clusters and failures

Different resources available
- per node instance
- at a certain time

Code/computational mobility
- collaborative signal processing
- outsourcing of tasks to nodes with free resources
Going lean on software

**OS/application configuration usually fits into 4 kB internal SRAM**
- applications use up to 64 kB SRAM
- 3 x 60 kB pages for additional user data
- EEPROM for nonvolatile user parameter storage

**Minimum application**
- 1710 bytes ROM
- 462 bytes RAM

**Limited resources**

**Power savings**
Developing for BTnodes today…

Standard embedded toolchain not scalable to many devices
  – cross compilation using AVR-GCC
  – serial port upload (programming)
  – serial console “printf-style” and JTAG debugging, LED patterns

Works to get the system going.
  – too clumsy for large networking applications in the long run
Developing – Linux emulation

Native compilation and execution on Linux
- using adapted drivers to match the host system
- with a serial Bluetooth device on a PC or iPAQ

Bluetooth PC
+ use unlimited resources of host
+ bridging networks
+ comfortable application debugging

BTnode
+ deployment platform
- slow upload necessary
Developing – Network reprogramming

Bootloader support of the MCU
- new bootcode in multihop payload
- Flash reprogramming from pre-cached bootcode data
- reprogramming of single node and by network flooding

Per-hop performance
- transmitting 10 kB: ~0.8 sec
- writing to Flash: ~0.2 sec

Overall performance
- ~ 10s to transmit, write and reboot with 80 kB bootcode

xhop(ABCDE)
program_flash()
flood_boot()
return_result()
xhop(reverse_route)
Example applications

Multitude of applications realized
– simple data acquisition
– interaction with consumer devices
– advanced networking concepts

Hands on training
– from tool installation to first application in less than a day

Student projects
– over 20 completed at ETHZ alone
Physical activity detection network

Many sensors reveal more context
- use multiple motion sensors for context awareness
- architecture required to combine those sensors
- map hierarchical topology to human body
Better avalanche rescue through sensors

Ubiquitous Computing scenario: data acquisition and aggregation

- oximeter
- oxygen sensor
- accelerometer, inclination
- heart rate, oxygen saturation
- air pocket detection
- orientation, mobility
Bluetooth enabled appliances

Communication with other Bluetooth enabled devices
- standard Bluetooth profiles for SMS, object push and RFCOMM

BTnode enabled egg carton
SMS from egg carton
Interactive dialog
WURM - Wearable unit with RC modules

WURM hardware architecture CPU
- legacy C-code, binary only code
- low-intensity, background tasks

Reconfigurable unit
- high-performance tasks
- low-power tasks

Case study Bluetooth/Ethernet-Bridge
- IP access point for WURM modules using BTnodes
- Soft CPU (LEON, 32bit SPARC)
- HW minimal TCP/IP stack
- HW Ethernet MAC
Undergrad projects with Lego Mindstorms

Constructing simple games using Lego Mindstorms robots
- Connect Four
- Minesweeper
- Nine Men's Morris

Augmented Lego RCX controller
- remote control
- data acquisition
- clustering
- offloading computation

Inter-robot communication
- simple command broadcast
- using a standard application on BTnodes
TreeNet topology discovery

Formation of large Scatternets
- robustness
- simplicity
- independent
- distributed

Every node executes algorithm
- until single tree is reached
- topology broadcast to root

Topologies are used for
- geometric routing
- positioning algorithms
- broadcast services

```java
loop {
    inquiry();
    while (nodes_found) {
        if not_max_degree {
            connect_to_tree();
        }
    }
}
```
XHOP multihop network

Bluetooth multihop source routing prototype

- integrated scalable application protocol
- based on Dynamic Source Routing (CMU)
- routing across piconet borders to support >8 nodes

Remote topology discovery

- script like command language in the payload

Performance

- 1-2 sec per hop, depending on inquiries
TinyBT using dual Bluetooth BTnodes

Dual frontend BTnodes

Bluetooth software stack for TinyOS
- scalability to multiple frontends
- good power performance due to high throughput of Bluetooth

Allowing formation of large, transparent Scatternets
- transparent routing
- end to end connections
- high performance

Power consumption profiling
- detailed in-situ measurements
After over 2 years of BTnodes

200 units with 16 research groups
- smart objects
- networking
- wearable computing
- perceptual computing
- operating systems

Good community interaction

Open Source
- BTnode System Software
- examples
- BTnode design data
- development tools

VTT, FI
DSG, ETH Zurich, CH
PCCV, ETH Zurich, CH
TecO, University of Karlsruhe, GE
PLAY, Interactive Institute, SE
TIK, ETH Zurich, CH
IFE Wearable Lab, ETH Zurich, CH
NTT DoCoMo, Munich, GE
Ptolemy Group, UC Berkeley, USA
Art of Technology, Zurich, CH
DistLab, Diku, Copenhagen, DK
LAP, EPF Lausanne, CH
CS Department, Lancaster University, UK
LSL, EPF Lausanne, CH
TinyOS Group, UC Berkeley, USA
University of Uppsala, SE
Lessons learned

Standard layered OS/application models do not hold

Was Bluetooth a good choice?
- not ultra-low power, but that wasn’t our goal
- available, versatile and functional at a very high level
- slowly becoming ubiquitous

Software engineering is no small task
- software system planning and strategy, lacking coordination
- stepwise implementation, testing, verification
- many parallel software projects have evolved

Scalability of systems not solved from development perspective
- emulation mode is a valuable tool
- remote deployment, debugging and monitoring in high quantity is unsolved
The lessons applied

Multiple, adaptable radio frontends adaptable to
- resource availability (power, computation, infrastructure access…)
- application requirements
- network topology
- node characteristics

Need for more devices
- reduction of BOM by 50%
- design shrink, physical integration
- lower power

Make BTnodes available as a platform kit for partners
- rethinking of BTnode System Software
To probe further...

To get started, visit the BTnodes website at [http://www.btnode.ethz.ch](http://www.btnode.ethz.ch)

**BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks**

Welcome to the BTnode Platform!

**Overview**

The BTnode is an autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller. It serves as a demonstration platform for research in mobile and ad hoc connected networks (MANETs) and distributed sensor networks. The BTnode has been jointly developed by the Computer Engineering and Networks Laboratory (TIK) and the Research Group for Distributed Systems at ETH Zurich. Currently the BTnode is primarily used in two major research projects: NCCE-MACS and Smart-ITS.

**BTnode features at a glance**

- Microcontroller: Atmel ATmega128L (8 MHz @ 8 MIPS)
- Memories: 64 kbyte RAM, 128 kbyte FLASH ROM, 4 kbyte EEPROM
- Bluetooth radio module, Ericsson ROK 101 007
- External Interfaces (I2P, UART, SPI, I2C, GPIO, ADC,...)
- 4 LEDs
- Standard C Programming

**Quickstart**

To get started is quite straightforward. Before you can start off developing applications for the BTnode you need to:

- Download and install the development tools (compilers). See the tools section.
- Download and install the BTnode System Software. See installing the BTnode System Software.
- Buy a hardware programmer. We recommend the Atmel STK 500 programmer.
- Build your own programming cable (one per programmer). See Hardware.
- Get BTnodes or serial Bluetooth devices that can be used in emulation mode.
- Compile and download your first example application.

Copyright (c) 2000-2003 BTnode Project