TinyOS on BTnodes

Attila Dogan-Kinali

Jan Beutel, Matthias Dyer, Prof. Lothar Thiele
Outline

• Introduction
  • Why?
  • BTnode
  • TinyOS

• TinyOS on BTnodes
  • Challenges and Problems
  • Hardware Issues

• Conclusions
Why?

- Measurements in large areas with no infrastructure
  - Fire detection in forests
  - Weather data collection in uninhabited areas
- Measurements on mobile objects
  - Animals
  - Good tracking in large warestores
Why TinyOS?

- TinyOS became a quasi standard
- Comparison of different platforms
- Not being isolated
BTnode
TinyOS Principle

- Component based
  - nesC - C based component language
  - Components are plugged together
- Cooperative Multitasking
  - Tasks are never interrupted by other tasks
  - Tasks can call commands
- Event driven
  - Interrupts are mapped to events
  - Events can call event handlers
  - Tasks can trigger events
Designflow

• An application consists of
  – self written components
  – instructions on how to connect them together
• These components are combined with the TinyOS components
• The combined components are converted to C code
• gcc is used to compile the resulting code
TinyOS on BTnode

- BTnodes are a superset of Mica2 motes
- Porting means adjustments to hardware differences
  - Different pin assignments
  - Latch between µC and LEDs
  - Power switches
- Porting means to find the one line that needs to be changed
TinyOS on BTnode – Problems

• Poorly documented
  – No design documents
  – No structural documents
  – No documentation about the inner working of nesC
  – Hardly any comments in the code

• nesC
  – Non-standard language
  – Not designed to be debugable
  – Deviates from its own design paradigm
BTnode - Latch Problems
Network Application – Surge
TinyOS - Any good?

• Good ideas
  – Component model
  – Event driven

• Design could be better
  – nesC paradigm sometimes impractical
  – not well extensible

• Documentation needs improvements