BTnodes

Topology Discovery and Multihop Networking

Jan Beutel

IP9 - Communicating Embedded Systems
Outline

Highlight the idiosyncrasies of multihop ad hoc networking on real devices

BTnode - Ad hoc networking prototyping platform

Constructing network topologies using Bluetooth

First implementation of a robust, self-healing tree topology
Large ad hoc network topologies

How to construct an ad hoc network topology with Bluetooth
- large network, many devices
- all devices connected, supporting transparent multihop transport

Understanding the limits and benefits of Bluetooth

- XHOP prototype
- TreeNet topology
BTnode architecture

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

Bluetooth has the advantage of
- availability today for experimentation
- compatibility to interface to consumer appliances
- an abstract, standardized high level digital interface
Bluetooth prototyping platforms

Integrated hardware features
- 8-Bit RISC, max. 8 MIPS, 128 kB Flash, 64 kB SRAM, 180 kB data cache
- operating from 3 cell batteries

Event-driven lightweight OS

Dual Bluetooth stack for TinyOS
- developed on the BTnodes
- scalability to multiple frontends
- good energy-per-bit ratio due to high throughput of Bluetooth
  DistLab, U Copenhagen [Leopold2003]

imote – the Intel research mote
- 1G prototype based on Zeevo Bluetooth module, integrated arm host
  [Kling2003]
Other BTnode applications

Many successful BTnode applications

- The Lighthouse location system [Roemer2003]
- Smart product monitoring [Siegemund2002]
- Bluetooth enabled appliances [Siegemund2003]
- Smart It’s friends [Siegemund2003]
- XHOP/R-DSR multihop prototype [Beutel2002]
- Distributed positioning – TERRAIN implementation [Frey2003]
- Physical activity detection network [Junker2003]
- Better avalanche rescue through sensors [Michahelles2002]
- Wearable unit with reconfigurable modules [Plessl2003]
- Undergrad projects with Lego Mindstorms [Blum2003]
- …

Mostly relying on simple point to point data links
Constructing network topologies

Scatternet formation algorithms
- many theoretical studies and simulations often far away from reality
- improvements in the current Bluetooth voting draft specification v1.2

BlueStars [Basagni2002/3], BlueRing [Lin2003] …
- make assumptions on physical prerequisites not available today
- assume “perfect” connection performance
- assume symmetric data availability on nodes

Ad hoc network topologies only in simulations
- usually all using the same underlying physical models
- often lacking realistic distributed system models for large networks
- limited access to appropriate hardware devices
# BTnode networking – definitions

<table>
<thead>
<tr>
<th>Four states</th>
<th>Useful operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>- IDLE</td>
<td>- inquiry() – find other nodes</td>
</tr>
<tr>
<td>- MASTER</td>
<td>- connect() – open connection</td>
</tr>
<tr>
<td>- SLAVE</td>
<td>- roleSwitch() – change MS relation</td>
</tr>
<tr>
<td>- MASTERSLAVE</td>
<td>- sendData() – data transport</td>
</tr>
</tbody>
</table>

## Hardware limitations on the BTnodes/Bluetooth
- max. 7 active slaves in one Piconet
- while in `inquiry()` and `connect()` a node is not visible
- while in SLAVE or MASTERSLAVE a node is not visible
- while in SLAVE or MS a node cannot do `inquiry()` or `connect()`
- `inquiry()` and `connect()` have long delays and no a priori guarantee

Bluetooth only defines single hop Master-Slave data transport
Distributed Bluetooth Piconets

Distributed `inquiry()` and `connect()` is a problem
- nodes are uncoordinated
- limited visibility
- asymmetry: inquired node doesn’t notice

`inquiry()` and `connect()` have long delays
- state change in remote node goes unnoticed
- average delay in seconds [Kasten2001]
- no a priori guarantee for success

`inquiry()` and `connect()` are highly nondeterministic (both in timing and function)
Purpose of this study:

How can we construct ‘arbitrarily’ large trees in a robust and distributed way?
TreeNet simple tree construction

Every node executes algorithm
- until single tree is reached

Formation of large topologies
- robustness
- simplicity
- redundancy
- distribution
- self-healing

Services and applications can break up trees later
- forming other topologies
- optimizing topology

```
loop {
    inquiry();
    forall (nodes_found) do {
        while (not_max_degree)
            connect();
    }
}
```
TreeNet simple tree construction

Every node executes algorithm
- until single tree is reached

Formation of large topologies
- robustness
- simplicity
- redundancy
- distribution
- self-healing

Services and applications can break up trees later
- forming other topologies
- optimizing topology

```plaintext
loop {
    inquiry();
    forall (nodes_found) do {
        while (not_max_degree)
            connect();
    }
}
```
TreeNet simple tree construction

Every node executes algorithm
  – until single tree is reached

Formation of large topologies
  – robustness
  – simplicity
  – redundancy
  – distribution
  – self-healing

Services and applications can break up trees later
  – forming other topologies
  – optimizing topology

```
loop {
      inquiry();
      forall (nodes_found) do {
        while (not_max_degree) {
          connect();
        }
      }
}
```
TreeNet simple tree construction

Every node executes algorithm
  – until single tree is reached

Formation of large topologies
  – robustness
  – simplicity
  – redundancy
  – distribution
  – self-healing

Services and applications can
break up trees later
  – forming other topologies
  – optimizing topology

```
loop {
  inquiry();
  forall (nodes_found) do {
    while (not_max_degree)
      connect();
  }
}
```
TreeNet simple tree construction

Every node executes algorithm
- until single tree is reached

Formation of large topologies
- robustness
- simplicity
- redundancy
- distribution
- self-healing

Services and applications can break up trees later
- forming other topologies
- optimizing topology

```java
loop {
    inquiry();
    forall (nodes_found) do {
        while (not_max_degree) {
            connect();
        }
    }
}
```
TreeNet simple tree construction

Every node executes algorithm
  – until single tree is reached

Formation of large topologies
  – robustness
  – simplicity
  – redundancy
  – distribution
  – self-healing

Services and applications can break up trees later
  – forming other topologies
  – optimizing topology

```c
loop {
    inquiry();
    forall (nodes_found) do {
        while (not_max_degree) {
            connect();
        }
    }
}
```
TreeNet simple tree construction

Every node executes algorithm
  - until single tree is reached

Formation of large topologies
  - robustness
  - simplicity
  - redundancy
  - distribution
  - self-healing

Services and applications can break up trees later
  - forming other topologies
  - optimizing topology

```
loop {
  inquiry();
  forall (nodes_found) do {
    while (not_max_degree) {
      connect();
    }
  }
}
```
TreeNet discussion

Nodes must all be in visible range

Might not fully connect if multiple max_degree roots form
  – rebuilding of partial trees necessary if nodes cannot connect at root

Simple greedy algorithm reduces inquiry() and connect()
  – better performance by caching and time-stamping inquiry() and connect()
  – try to connect() to node-last-seen first
  – exchange of topology data and adaptive connect() retries

In reality a 5 line algorithm ends up to be ~2000 lines of code!
Large scale distributed deployment

So why do we actually need ~4000 lines of code?
– 5 lines -> 2000 lines + system software + debugging and monitoring
– result in an ~87 kB program (un-optimized)

Necessary node functions
– data exchange
– timing and time-stamps
– data storage and handling (persistent and volatile)
– stepwise testing and deployment
– distributed reprogramming
– distributed debugging and monitoring

A backbone infrastructure like TreeNet only enables to deploy and test the ‘interesting’ distributed ad hoc networking algorithms…
Acknowledgements

TreeNet collaborators in IP9
- Oliver Kasten, Friedemann Mattern, Matthias Ringwald, Kay Römer, Frank Siegemund
- Regina Bischoff, Roger Wattenhofer, Aaron Zollinger
- Jan Beutel, Martin Hinz, Lothar Thiele

Related publications:
To probe further…

Come and play the TreeNet puzzle in the poster session tomorrow

Posters
- Prototyping Applications with BTnodes
- The Lighthouse Location System for Smart Dust

BTnode platform
- online documentation and support
- mailing list
- BTnode rev3 development

http://www.btnode.ethz.ch