Robust Topology Formation using BTnodes

Jan Beutel [1]

Computer Engineering and Networks Laboratory
Swiss Federal Institute of Technology (ETH)
Zurich, Switzerland

Abstract

Research of the past years has brought about many new developments and proposals for wireless sensor networks, mostly in the areas of algorithms and theory, backed up by extensive simulation work. In order to fully understand the complexity of these issues from a system perspective we will highlight the idiosyncracies of multi-hop networking implemented on real devices. The BTnode is an autonomous wireless communication and computing platform based on a Bluetooth radio and a microcontroller aimed at fast prototyping and research in mobile computing. The TreeNet example implements a self-healing, distributed Bluetooth Scatternet formation on the BTnode platform. We will discuss our example implementation including devices used, prerequisites, limitations, algorithmic ideas and implementation results.

Key words: Wireless Sensor Network Deployment, Bluetooth Scatternets, Topology Formation and Control, Robustness
PACS:

1 Introduction

Wireless sensor networks (WSNs), ad hoc networks, pervasive and ubiquitous computing, mobile computing, mobile communications and wearable computing are all trends of the past years that have aroused broad interest and that are forecast to persist by many prominent visions alike. Within these themes, multi-hop networking has been in the focus of research and industry for many

Email address: beutel@tik.ee.ethz.ch (Jan Beutel).

[1] The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322.

years but it is almost inexistent in todays commercial applications. While recent research has brought about many new developments and proposals for multi-hop networking, mostly focused in the areas of algorithms and theory, backed up by extensive simulation work few have actually implemented and deployed these on real devices. In this case study we are raising the question why that is and identify some key obstacles and problems that appear in practical multi-hop networking for WSNs.

In the experience gained from the implementation of WSN experiments and deployments such as documented by Szewczyk [1], Cerpa [2], Simon [3], Zhang [4] and others it has been shown that the system aspects are much more complex than initially anticipated. In order to fully understand the complexity of the matter from a system perspective it is necessary to not only model and/or simulate but also to implement and test on real-world systems. Since WSNs are usually assumed to be implemented as distributed systems using small, resource constrained devices that are remotely deployed, meticulous care has to be taken to design simple, robust and failure tolerant algorithms. The features and application support offered on highly integrated devices such as the Berkeley Motes [5, 6], the Intel Imotes [7] or the BTnodes [8] is quite the opposite when compared to the ever increasing performance available in the traditional computing world.

In the work presented here, we want to highlight the idiosyncrasies of multi-hop ad hoc networking when implemented on real devices. More specifically, we will focus on the area of Bluetooth Scatternet formation, where many different algorithms for mesh structures either conforming to the Bluetooth standard [9], with star [10] or tree topologies [11] have been proposed. A recent distributed algorithm with bounds on complexity proposed by Law [12] has been validated by simulations as well as a comparative study by Basagni [13]. Often such work takes into account assumptions on physical prerequisites not readily available in real devices and systems, or simply perfect performance, which is not encountered in wireless systems embedded in the physical world. The extensive work on ad hoc networking simulations is most suitable to answer specific, singular questions and tradeoffs but seemingly inappropriate when approaching system level topics. Generally accepted assumption for simulation work like an 802.11 physical model or an example 1500x1300 grid of evenly distributed nodes and random way-point models certainly lack when relating simulation to real-world experimentation with interference, random failures, imperfections, and human interaction. Unfortunately there is not much work available today on the system integration issues that we will be dealing with here. So, our focus will be very much on the functional side of multi-hop topology formation and not so much on the algorithmic analysis.

The BTnode prototyping platform introduced in Section 2 will be used in the experimental part of this study. In order to be able to deploy such a multi-hop

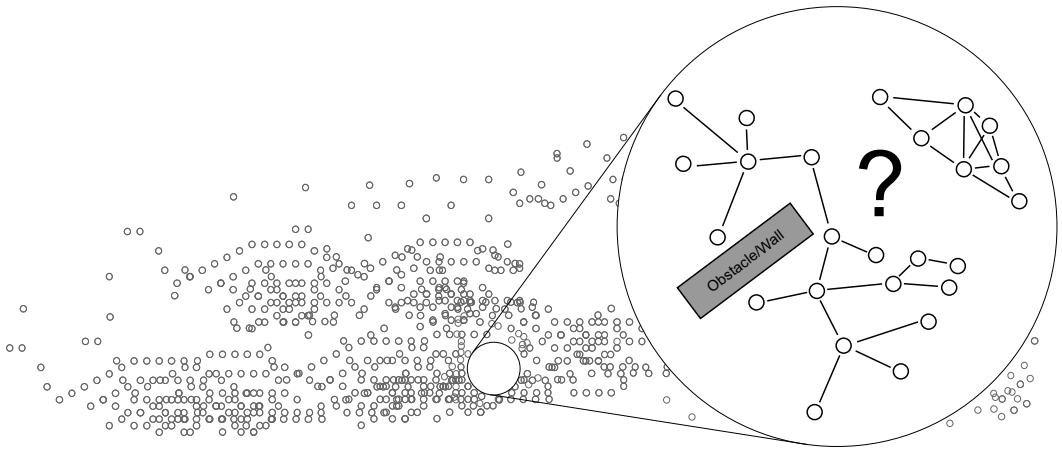


Fig. 1. A view on an example WSN population scenario with highly clustered nodes, scarcely populated areas, obstructed and separated regions. In order for all nodes to be able to communicate with each other they need to form multi-hop topologies in either mesh, ring or tree structures that are discussed in our study.

network we will construct robust tree topologies with Bluetooth Scatternets using a lightweight, self-healing, distributed algorithm introduced in Section 3. We will then discuss the implementation requirements and main obstacles in Section 4.

2 The BTnode Prototyping Platform for ad hoc Networks

The BTnode is an autonomous, lightweight wireless communication and computing platform based on a Bluetooth radio module and a microcontroller [8, 14]. The device has no integrated sensors, since individual sensor configurations are required, depending on the application. Instead, with its many general-purpose interfaces, the BTnode can be used with various peripherals, such as sensors, but also actuators, DSPs, serial devices (like GPS receivers, RFID readers, etc.) and user interface components. An interesting property of this platform is its small form factor of 6x4 cm while still maintaining a standard wireless interface. Clearly the focus here is not on a production-ready, ultra low-power WSN device but on a platform to be used in fast prototyping, early deployment and experimentation in the area of wireless networking systems.

2.1 BTnode rev2 Hardware

The BTnode rev2 hardware (see Figure 2) is built around an Atmel ATmega128L microcontroller with on-chip memory and peripherals. The micro-

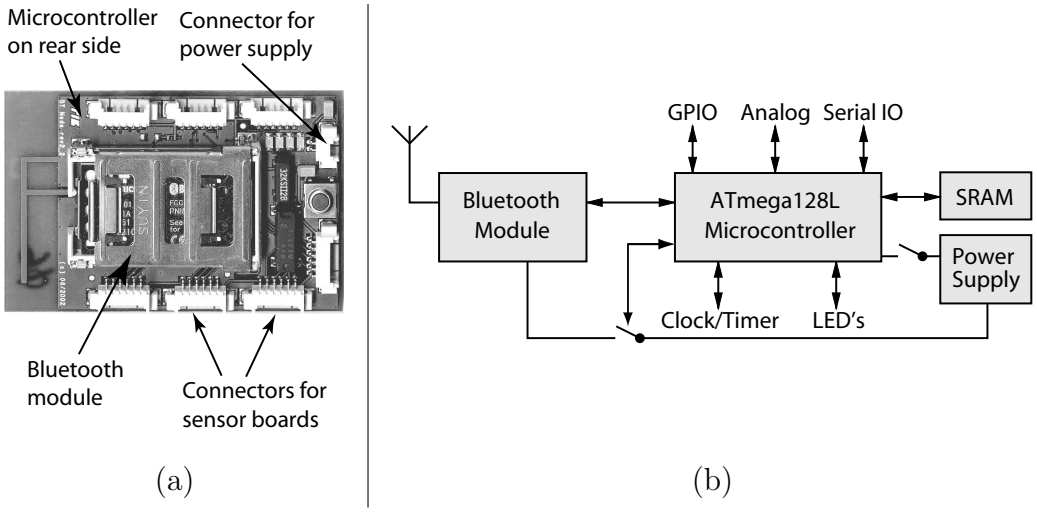


Fig. 2. (a) BTnode hardware (b) BTnode system overview.

controller features an 8-bit RISC core delivering up to 8 MIPS at a maximum of 8 MHz. The on-chip memory consists of 128 kbytes of in-system programmable Flash memory, 4 kbytes of SRAM, and 4 kbytes of EEPROM. There are several integrated peripherals: JTAG for debugging, timers, counters, pulse-width modulation, 10-bit analog-digital converter, I2C bus, and two hardware UARTs. An external low-power SRAM adds an additional 60 kbytes to the microcontroller RAM as well as 180 kbytes of data cache to the BTnode system.

An Ericsson Bluetooth module is connected to one of the serial ports of the microcontroller using a detachable module carrier, and to a planar inverted F antenna (PIFA) that is integrated into the circuit board. This partitioning of communication frontend and computing core at the Bluetooth Host Controller Interface (HCI) allows to build onto an abstract, standardized, high level digital interface that hides most of the underlying radio and baseband complexity. This interface offers a link layer API with asynchronous commands and events to configure the underlying frontend, connections and data transport. Bluetooth also has the advantage of being available today for experimentation in ad hoc and multi-hop networking and compatibility to interface to many consumer appliances for easy integration in user interaction scenarios and demo applications.

The BTnode is typically powered from 3 cell battery packs with a DC input range from 3.6-16 V. The power figures given in Table 1 show that the predominant part of the system power consumption is the Bluetooth radio. Using a separate power-gating circuit (see Figure 2-b), the Bluetooth module can be disconnected from the power supply completely when not in use. Typical demo examples using radio duty cycles of up to 10% have resulted in battery lifetimes of multiple days to a few weeks on a standard 840 mAh Li-ion battery pack.

Table 1
BTnode rev2 power consumption characteristics measured at 3.3 V.

	$P_{typ}[mW]$	$P_{max}[mW]$
CPU power down, Bluetooth off	0.04	–
CPU standby, Bluetooth off	2.35	–
CPU idle, Bluetooth off	3.66	–
CPU active, Bluetooth off	15.0	30.9
CPU idle, Bluetooth idle	23.4	–
CPU idle, Bluetooth Tx/Rx	123	131
CPU idle, Bluetooth Inquiry	136	138
LEDs only	13.6	51.9
SRAM	0.591	14.15

2.2 Lightweight Operating System Support

The BTnode system software is a lightweight OS/application framework written in C and assembly language using a standard libC and the gcc compiler suite. The drivers, which are available for many hardware subsystems, provide convenient APIs for application developers. The system provides coarse-grained cooperative multitasking and supports an event-based programming model. Within this framework, applications are typically partitioned into a Bluetooth link layer that is communicating with the Bluetooth frontend through commands and events, keeping track of point to point connections with individual state machines, a command line terminal for control and debugging and an application core. The whole application is defined at compile time and then programmed into the flash memory of the microcontroller using either an in-system programmer (ISP/JTAG) or a resident bootloader.

The hardware resources available on this miniaturized embedded device are quite reduced when comparing to systems like modern cell phones or PDAs running full blown Windows derivatives on multi-scalar processors. The performance of the BTnode microcontroller core and OS can be somewhat compared to the first PC's in the 80's, but without features such as dynamic memory allocation and dynamic program loading and execution. Thus, extreme care needs to be taken to make use of the scarce resources efficiently.

The BTnode platform has been successfully deployed with over 200 units. Many researchers have been using these for their implementation work and experimentation in domains such as wearable [15] and ubiquitous computing demo applications [16], smart spaces using RFID tags [17], user interactions

with cell phones [18] or sensor networks research [19] and local positioning [20]. Currently the platform is undergoing a major hard- and software revision that is made publicly available through a contract manufacturer [21].

3 Constructing network topologies using Bluetooth

In the remainder of the article we want to give an example of how to construct connected ad hoc network topologies using Bluetooth. The goal of this work is twofold: We want to (i) form large, connected networks consisting of many devices, supporting transparent multi-hop transport and (ii) to understand the limits and benefits of Bluetooth technology in the context of ad hoc networking and the relevant implementation issues.

Throughout this example we will be using the BTnode rev2 prototyping platform introduced in the previous section as our underlying infrastructure. We will first talk about the Bluetooth prerequisites available on the BTnode rev2 that we can actually use as starting points to develop an appropriate topology maintenance algorithm. This algorithm will be subsequently discussed in our example implementation. The Bluetooth prerequisites contain general definitions made in the Bluetooth standard and specific device capabilities found in the BTnode rev2 devices. Nevertheless, the concepts of this work can be applied in a straightforward manner to other generations of devices with differing capabilities.

3.1 BTnode networking prerequisites

Networking in Bluetooth is organized in master-slave configurations called Piconets. Up to seven active slaves can be connected to one master at a time. Multiple Piconets can be interconnected by nodes taking on dual roles of slave-slave or master-slave forming a Scatternet (see Figure 3). While the interconnection of nodes in these different configurations is part of the Bluetooth standard, the formation and control of multi-hop topologies is not governed by the standard. Also, the data transport is only defined on each single hop (from master to slave or vice versa) and not over multiple links, e.g. from slave to slave over an intermediate master in a standard Piconet or even across a Scatternet. This means that a functional layer running on the host controller must take care of general connection management and all multi-hop packet forwarding.

The Bluetooth standard specifies a set of required features, of which devices typically are not required to support all. Likewise the degree of freedom in im-

plementations is reduced considerably from that used in most theoretical studies. The Ericsson devices available in the BTnode rev2 used in our experiment are capable of forming Scatternets interconnected via a single master-slave dual role as can be seen in Figure 3.

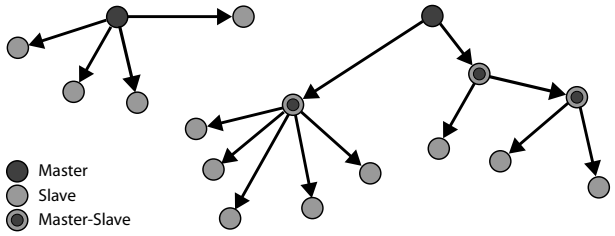


Fig. 3. Bluetooth organized in Pico- and Scatternets.

For the development of a topology control algorithm on the BTnodes we will be able to use the four states disconnected, master, slave and master-slave as well as the operations

$inquiry(duration) \rightarrow addressList$: scan the environment for other nodes
$connect(address) \rightarrow link$: open link to given address
$disconnect(link)$: close a given link
$roleSwitch(link, role)$: change local master/slave relation
$sendData(link, data)$: send data buffer over link.

The operation *inquiry()* is used to scan the environment for other nodes and returns the addresses found in the respective scan. A *connect()* tries to establish a link to a node with a given address, typically found by a previous *inquiry()*. In order to switch the master/slave relation a *roleSwitch()* can only be performed on single point to point links since it changes the location of the master and thus would break the structure of a Piconet with multiple slaves. A *sendData()* operation finally sends a buffer of data across the link specified. These operations are initiated by sending an HCI command to the Bluetooth device and completion is signalled by an HCI event received by the microcontroller. Although multiple HCI commands can be queued at the Bluetooth device they are blocking in function until completed requiring a careful implementation.

In our case, the capability for device discovery and connection control is hardware specific: A node can only (i) perform or (ii) answer to an *inquiry()* or *connect()* while in the states disconnected or master and (iii) is only visible to other nodes while not performing one of the above operations.

Since wireless channels have many variant properties there is generally no guarantee for the success for any operation. In general, the wireless medium can be assumed to be an unreliable medium. Again, when applied to Bluetooth technology this means that operations such as *inquiry()* and *connect()* can exhibit considerable delays and have no a priori guarantee for success. This is especially a problem when operating in a distributed and uncoordinated environment with many peers. An *inquiry()* of a given duration does not always return the whole list of neighbors present and a *connect()* might not succeed even if before the *inquiry()* resulted in exactly this node. As the operations rely on best effort, no assumptions on the state or even the presence of other nodes can be made prior to a successful search or connection setup. A node that was visible before might not be visible anymore, be busy with an operation or subject to other random sources of interference.

In a theoretical worst case distributed scenario where e.g. all nodes would try to perform an *inquiry()* or *connect()* simultaneously for an equal length duration using the properties described earlier, we would not be able to detect the presence of any node, independent of the node density (see Figure 4).

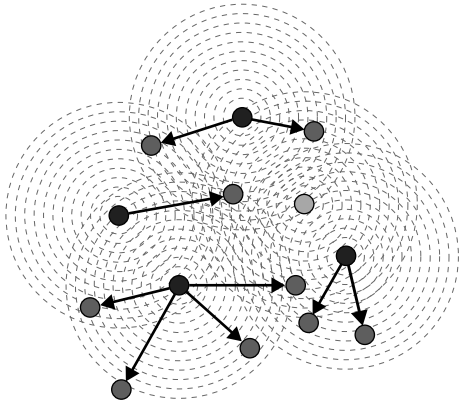


Fig. 4. Theoretical worst case scenario: Simultaneous *inquiry()* of all nodes will not give any results, since all nodes are busy transmitting *inquiry()*.

Traditional problems of limited visibility (hidden station problem) are further influenced by asymmetric effects that are specific to Bluetooth: The inquired node doesn't notice the presence of the inquiring party. Typically, average rates of success are defined and protocols have to take care of retries, retransmitions and back-off on erroneous behavior. Independent studies by Kasten [22], Murphy [23] and Welsh [24] have shown that the average delay for a successful *inquiry()* and *connect()* is in the order of multiple seconds. The recent version 1.2 update of the Bluetooth standard actually contains several improvements in this direction.

What we can learn from such insights is, that *inquiry()* and *connect()* are highly nondeterministic (both in timing and function) and we will have to find means to cope with all these properties to construct robust and reliable services.

3.3 TreeNet Simple Tree Topology Construction

Learning about all these properties we can understand that we cannot construct arbitrary mesh structures in Bluetooth Scatternets. In fact the devices used allow only to construct trees and no circular structures and that we would have to deal with lot's of non-determinisms and failures. So the question in this section is how to construct and maintain arbitrarily large trees in a robust and distributed way using a most simplistic scheme.

A very simple and robust distributed algorithm to be used for connection management based on a search-and-connect scheme that is executed on every node is given in the following:

Algorithm 1 TreeNet Simple Tree Topology Formation

```
loop
  while my_slaves < max_degree do
    found_nodes = inquiry();
    for all nodes in found_nodes do
      connect(node);
    end for
  end while
end loop
```

Starting with all nodes initially disconnected and idle, nodes would perform an *inquiry()* to search for other nodes and subsequently try to *connect()* to all nodes found in this search phase. Upon a successful connection between two nodes, the slave node stops inquiring and connecting (since it cannot inquire and connect anymore), while the master node continues if it has not yet acquired *max_degree* [2] slaves. This means that new connections are opened only between nodes that are not in a slave or master-slave role. Starting with disconnected Piconets, the network topology evolves by connecting more and more Piconets to each other and forming Scatternet trees until, after multiple iterations, only one master node is left at the root of the final tree (see Figure 5). If a link fails, the root of the disconnected subtree is not a slave anymore and will thus start inquiring and connecting again. To speed up the connection process, the information obtained during inquiries (node addresses and clock offsets) is stored locally.

[2] This parameter is used to control the degree and hence depth of the final tree.

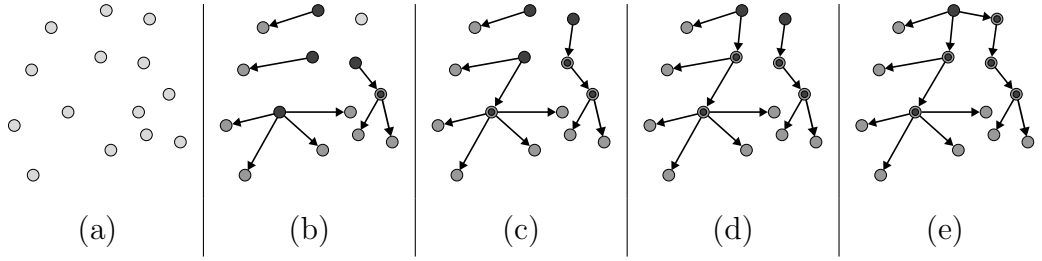


Fig. 5. A schematic view of the TreeNet algorithm operation: Initially disconnected (a) , first Piconets form (b) , then they interconnect to first Scatternets (c), until larger Scatternets are forming and the tree structure gets visible (d) and a single tree is reached (e).

This algorithm allows the formation of large topologies in a robust, distributed and extremely lightweight fashion that is independent of any central controller, exhaustive computation or data exchange between nodes. Based on a simple search-and-connect scheme the TreeNet topology formation offers basic redundancy, self-configuration and even self-healing properties. Since this is clearly a best-effort approach there is no given guarantee for a specific topology or even optimization. If desirable, it is possible to add auxiliary topology control services to break up trees at specific connections and to optimize the topology after the initial construction phase of the TreeNet algorithm.

4 Lessons Learned Through Experimentation

Of course real life in such experimentation doesn't come as cheap as proposed in our 8 line algorithm in the previous section. Major issues that were discovered along the way from concept to final system validation were, that (i) an 8 line high level algorithm leads to about 2000 lines of code in a functional implementation and (ii) it is very difficult to test, deploy and evaluate a large amount of devices. We will now discuss these two issues in more detail.

4.1 Code Size and Complexity

Given the constraints specific to our devices, the implementation and the highly non-deterministic behavior of the environment extensions to the core TreeNet algorithm are close at hand to improve the overall performance.

First of all there are general lockup issues in the formation of the trees, that need to be accounted for. A major restriction that is rooted in the restriction of the Bluetooth hardware to not be able to *connect()* to slave roles is, that all nodes must be in visible range of each other for the algorithm to function and all nodes to be able to connect to a tree. Furthermore a set of nodes might

not fully connect if multiple *max_degree* roots form because they will not be allowed to take on another slave connection due to the *max_degree* limitation.

The problem of distributed *inquiry()* and *connect()* has been discussed earlier. In order to solve this problem and the lockup issues discussed above, a root node that can still see other nodes but has been in a *max_degree* state can try to disconnect one of it's slaves and try to connect to another node. This is of course not a very efficient operation but it allows to gradually include all nodes into a balanced single tree using this simple and robust random back-off technique.

A simple greedy algorithm that time-stamps operations and events can then try to *connect()* to the last node seen on an *inquiry()* first, expecting that node to be available for incoming connections since it has recently answered to an *inquiry()*. Also caching of all *inquiry()* and *connect()* attempts, their respective results and the retry count allows to use further heuristics and to adapt the periodic behavior to nodes with frequent failures. These operations are not very expensive in respect to the overall operation since they are performed completely on the host microcontroller and help to reduce especially the amount and the duration of the time-consuming and blocking *inquiry()* and *connect()* operations.

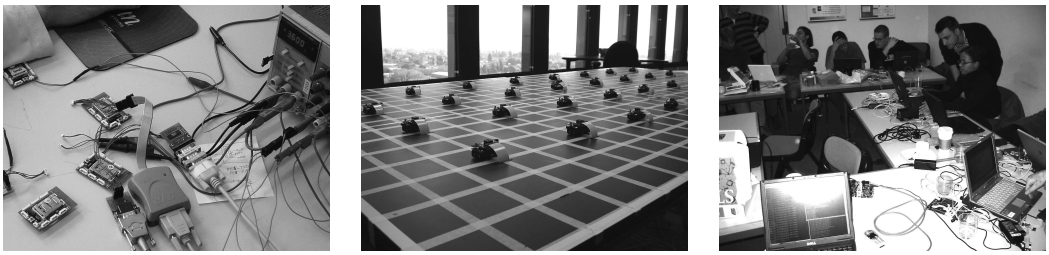
Apart from these enhancements, an application needs appropriate error handling and fall back mechanisms to handle all unsuccessful operations. The basic underlying infrastructure to be able to run such an algorithm thus additionally needs data storage, data exchange, timing functions, time-stamping of events, error handling as well as connection and link management as a minimum subset implemented on every node in the system. In total, this leads to about 2000 lines of additional code in our example implementation for the heuristically enhanced TreeNet application.

4.2 Large Scale Distributed Deployment

When it finally comes to simultaneous deployment on multiple devices, possibly away from the engineers desktop, the situation changes once more: How to debug, quantify, visualize and monitor the operation of a large number of distributed sensor devices without altering the operation of the system and without attaching additional infrastructure? Stepwise testing, deployment and validation are necessary steps in any design process.

Typical problems that appear here are the necessary cables, either for power or debugging and control, batteries, mounting, housing to shield the electronics from harmful influences of the environment, (re-)programming of all devices with software updates, debugging of a distributed concurrent system, devel-

oping for stepwise deployment, visualization and analysis of operations and the online access to nodes. Trial and error, system testing grids or brute force approaches are most common today, some examples are illustrated in Figure 6.



(a) (b) (c)

Fig. 6. WSN development today: A typical engineers desktop (a), a static test-grid with a large serial multiplexer (b) and collective debugging sessions (c).

Extra functionality is necessary to enable to debug and monitor the function of every node until the final functionality can be verified. Today, this is typically done with a serial debugging cable and a control terminal for each node. This is of course not feasible in the field as well as with an increasing number of devices.

In contrast to the design flow proposed for the Berkeley Motes using large simulations [25] as a means for stepwise refinement and collaborative debugging we propose a method based on virtualization and emulation of the embedded BTnode platform on a Linux system [8] prior to deployment on the embedded target device. This approach uses Bluetooth devices attached to a PC to run the BTnode application natively without the need for downloading to the embedded target itself and allowing for extensive debugging an monitoring due to the capabilities of the PC platform. The advantage here is, that the application is embedded into it's real environment and we are using the actual Bluetooth device and not a simplified model as in simulations.

The TreeNet algorithm was implemented on the BTnodes and first demonstrated with over 40 devices in September 2003 at the NCCR-MICS annual project review, to that date and our knowledge one of the largest interconnected Bluetooth Scatternets. In an initial phase the first Scatternet trees appear quickly, on the order of a few seconds. It has shown that to reliably form a single tree out of all nodes is a harder task with durations on the order of multiple minutes. This is attributed to the reduced visibility of nodes as the algorithm progresses until in the final stage, there are exactly two devices left that have to match. The resulting trees are generally very stable and remain connected over hours, even in the presence of heavy interference from other Bluetooth devices and Wireless LAN. Since there are no ambiguous routes in a tree, only a simple packet forwarding mechanism was implemented that would allow a source routing type multi-hop data transfer. A demo application was

implemented as a puzzle with light patterns being broadcast to all nodes from a control terminal for easy visualization and verification of the tree structure.

5 Conclusions

With this work we have demonstrated the implementation of a functional, light weight application that is capable of forming robust and self-healing tree topologies in Bluetooth Scatternets. We have shown that it is feasible to reliably interconnect many Bluetooth devices using simple algorithms. In our experience, Bluetooth has been easy to apply.

The WSN development reality encountered in this example has shown once again that it is very hard to deploy applications anywhere beyond 10-20 nodes. Of course it would be simple to argue that with sufficient manpower all implementation problems can be solved, but we have identified a key problem in this rather young field: Coordinated methods and tools for the complex, cross-layer development, testing, debugging, deployment and validation of systems composed of many devices are missing today.

In our example, a most simplistic 7 line algorithm already bloats to well over 200 lines of code when implemented in a functionally satisfying way. With the additional support for stepwise refinement and debugging capabilities this increases even further, locking up extensive portions of the resources available on a platform such as the BTnode. This should remind algorithm designers to drastically reduce algorithm complexity and the demand for auxiliary functions when designing for resource constrained embedded devices.

Models and methods for the design of such systems usually do not contain abstractions for unreliable operations, e.g. link failures, time-outs or unsuccessful operations. The integration of non-deterministic models into OS and deployment concepts is unclear today and there is no methodology for stepwise refinement and validation or large sensor networks. These are of course major issues for future work that will continue to require further trial-and-error type experimentation.

Specific to the TreeNet implementation on the BTnode rev2 we expect to improve performance considerably with the next generation devices that feature a full fledged Bluetooth 1.2 compliant frontend. This will relieve many of the restrictions encountered on our present devices and allow for much more flexibility and new algorithmic opportunities. Nevertheless to achieve a functional and manageable system we believe that it is of imperative importance to go lean and lightweight in all aspects.

6 Acknowledgements

I would like to thank all the long term contributors that have made the BTnode project such a success. In the context of the TreeNet application special thanks for tireless debugging and testing goes to Matthias Dyer, Martin Hinz, Oliver Kasten, Lennart Meier, Matthias Ringwald and Lothar Thiele.

Biography

Jan Beutel received a Diploma in Electrical Engineering from the Swiss Federal Institute of Technology (ETH), Zurich in 2000. He has been with u-blox AG, Zurich and spent time as a visiting researcher at the Berkeley Wireless Research Center. He is currently working with Lothar Thiele at the Computer Engineering and Networks Lab (TIK) pursuing his Ph.D. His research interests lie in the development and fast prototyping of sensor network applications, integrated application protocols for ad-hoc networks and local positioning algorithms.

References

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

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

[3] G. Simon, G. Balogh, G. Pap, M. Maróti, B. Kusy, J. Sallai, Á. Lédeczi, A. Nádas, K. Frampton, Sensor network-based countersniper system, in: Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004), ACM Press, New York, 2004, pp. 1–12.

[4] P. Zhang, C. Sadler, S. Lyon, M. Mortonosi, Hardware design experiences in ZebraNet, in: Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004), ACM Press, New York, 2004, pp. 227–238.

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

[6] J. Hill, M. Horton, R. Kling, L. Krishnamurthy, Wireless sensor networks: The platforms enabling wireless sensor networks, Communications of the ACM 47 (6) (2004) 41–46.

[7] R. Kling, R. Adler, J. Huang, V. Hummel, L. Nachman, Intel mote: Using Bluetooth in sensor networks, in: Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004), ACM Press, New York, 2004, p. 318.

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

[9] S. Basagni, C. Petrioli, Multihop scatternet formation for Bluetooth networks, in: Proc. IEEE Semiannual Vehicular Technology Conference (VTC Spring 2002), IEEE, Piscataway, NJ, 2002, pp. 424–428.

[10] C. Petrioli, S. Basagni, I. Chlamtac, Configuring BlueStars: Multi-hop scatternet formation in Bluetooth networks, IEEE Transactions on Computers 52 (6) (2003) 779–790.

[11] G. Záruba, S. Basagni, I. Chlamtac, BlueTrees – Scatternet formation to enable Bluetooth-based personal area networks, in: Proc. IEEE Int'l Conf. on Communications (ICC 2001), Vol. 1, IEEE, Piscataway, NJ, 2001, pp. 273–277.

[12] C. Law, A. Mehta, K. Siu, A new Bluetooth scatternet formation protocol, ACM/Kluwer Mobile Networks and Applications 8 (5) (2003) 485–498.

[13] S. Basagni, R. Bruno, G. Mambrini, C. Petrioli, Comparative performance evaluation of scatternet formation protocols for networks of Bluetooth devices, Wireless Networks 10 (2) (2004) 197–213.

[14] J. Beutel, O. Kasten, M. Ringwald, BTnodes - a distributed platform for sensor nodes, in: Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003), ACM Press, New York, 2003, pp. 292–293.

[15] C. Plessl, R. Enzler, H. Walder, J. Beutel, M. Platzner, L. Thiele, G. Tröster, The case for reconfigurable hardware in wearable computing, Personal and Ubiquitous Computing 7 (5) (2003) 299–308.

[16] S. Antifakos, F. Michahelles, B. Schiele, Proactive instructions for furniture assembly, in: Proc. 4th Int'l Conf. Ubiquitous Computing (UbiComp 2002), Vol. 2498 of Lecture Notes in Computer Science, Springer, Berlin, 2002, pp. 351–360.

[17] F. Siegemund, M. Rohs, Rendezvous layer protocols for Bluetooth-enabled smart devices, in: H. Schmeck, T. Ungerer, L. Wolf (Eds.), Proc. of the 1st Int'l Conf. on Architecture of Computing Systems - Trends in Network and Pervasive Computing (ARCS 2002), 2002, pp. 256–273.

[18] F. Siegemund, C. Flörkemeier, Interaction in pervasive computing settings using Bluetooth-enabled active tags and passive RFID technology together

with mobile phones, in: Proc. 1st IEEE Int'l Conf. Pervasive Computing and Communications (PerCom 2003), IEEE CS Press, Los Alamitos, CA, 2003, pp. 378–387.

[19] M. Leopold, M. Dydensborg, P. Bonnet, Bluetooth and sensor networks: A reality check, in: Proc. 1st ACM Conf. Embedded Networked Sensor Systems (SenSys 2003), ACM Press, New York, 2003, pp. 103–113.

[20] K. Römer, The Lighthouse location system for Smart Dust, in: Proc. 1st ACM/USENIX Conf. Mobile Systems, Applications, and Services (MobiSys 2003), ACM Press, New York, 2003, pp. 15–30.

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

[22] O. Kasten, M. Langheinrich, First experiences with Bluetooth in the Smart-It's distributed sensor network, in: Workshop on Ubiquitous Computing and Communication, Int'l Conf. Parallel Architectures and Compilation Techniques (PACT 2001), 2001.

[23] P. Murphy, E. Welsh, J. Frantz, Using Bluetooth for short-term ad hoc connections between moving vehicles: a feasibility study, in: Proc. IEEE Semiannual Vehicular Technology Conference (VTC Spring 2002), Vol. 1, IEEE, Piscataway, NJ, 2002, pp. 414–418.

[24] E. Welsh, P. Murphy, J. Frantz, Improving connection times for Bluetooth devices in mobile environments, in: Proc. Int'l Conf. Fundamentals of Electronics Communications and Computer Sciences (ICFS 2002), 2002.

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