Next-Generation Deployment Support for Sensor Networks

Jan Beutel, Matthias Dyer, Lennart Meier, Matthias Ringwald, Lothar Thiele
TIK-Report No: 207
Computer Engineering and Networks Lab
Swiss Federal Institute of Technology (ETH) Zurich
8092 Zurich, Switzerland

Abstract— **We present a new methodology for the development, test, deployment, and validation of wireless sensor networks. Our approach is a robust, wireless cable replacement offering reliable and transparent connections to arbitrary WSN target devices. Compared to traditional serial-cable approaches, this results in enhanced scalability and flexibility with respect to node location, density, and mobility. This makes the coordinated deployment of WSNs possible. We describe the operation of this novel tool and discuss an exemplary implementation on the BTnode platform.**

I. Introduction

A sensor network is a collection of small, low-resource devices that are distributed in the physical environment. Due to cost and flexibility issues, it is often assumed to be a wireless sensor network (WSN) consisting of a large number of sensor nodes. Each of these nodes collects sensor data, and the network collaboratively provides high-level sensing results.

The recent increase in research interest has led to many new developments and proposals for WSNs. While algorithms, system models, device architectures, and programming abstractions have been investigated for quite some time now, not much has been achieved regarding the support for development, test, deployment, and validation, which we refer to as "deployment" in the remainder of this paper. Coordinated methods and tools for this area are largely missing today. As a result, few implementations using real devices in meaningful scenarios exist.

Two key challenges in the deployment of WSNs are the large number of devices involved and the necessity to embed them in a realistic physical environment. Today, development and debugging are typically done with serial cables connecting the sensor nodes to a control terminal. These serial connections are used for stepwise testing, controlling, monitoring, and (re-)programming of the nodes. This method has become widely accepted for constructing small testbeds. For larger numbers of nodes or deployment in the field, it is unrealistic to connect a cable to each node. A result is, that few people implement and the standard approach for most WSN research is simulation. It is highly scalable and provides information about very specific system properties [1]. But when wireless communication is involved, the chosen simulation models are often too simplistic [2], [3]. For a full understanding of the WSN, it is thus necessary to not only model and simulate, but also to implement and test on real-world systems [4].

The coordinated deployment of a large-scale WSN is only possible if all cabling is replaced by wireless links. Using the WSN radio itself for this purpose is not feasible: In early development stages, it is too unreliable, and in the final validation phase, additional traffic would consume energy and bandwidth, and hence modify the results.

In this paper, we propose a deployment-support network (DSN) as a tool for the deployment of large WSNs. A DSN is a multi-hop wireless network of DSN nodes, which are small enough to be temporarily attached to a target sensor node. This network offers cable replacement by providing reliable virtual connections from a host PC to the target sensor nodes (see Fig. 1). From a networking perspective, this is an overlay network operating independently from the target WSN. This approach is transparent and highly scalable, and does not disturb the target WSN any more than the traditional, cable-based approach. For the engineer, everything actually looks as if the usual cables were in place; he can thus use the same tools. With our approach, we push the limit for large-scale prototyping from virtualization [5], [1] to coordinated real-world deployment.

In Section II, we identify cabling as the key obstacle to a coordinated design flow for WSN deployment, and

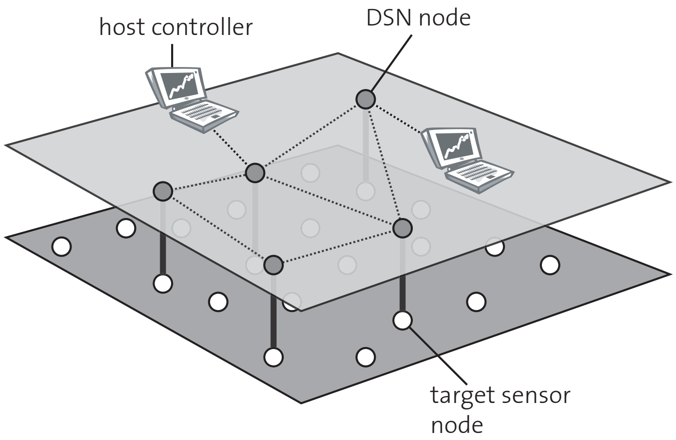


Fig. 1. Deployment-support network. Some or all nodes of the sensor network are attached to a DSN node. The DSN provides wireless virtual connections from one or more host controllers to the target nodes, allowing remote debugging, monitoring, and control.

introduce the deployment-support network as a cable-replacement tool. We have implemented a DSN on the BTnode prototyping platform. The details of our exemplary implementation are presented in Section III, and the experimental results in Section IV. The characteristics of and the experience with our DSN implementation are discussed in Section V. In Section VI, we give an overview of related work.

II. Next-Generation Prototyping of WSNs

The classic approach to developing and deploying WSNs starts with a design specification and initial simulation runs leading to lab setups with only few devices. Here, serial cables are used for program download, control, and monitoring. When moving away from the engineer's desktop and beyond numbers of 10–20 nodes, deployment and testing become increasingly hard. This is because wired connections to every node become infeasible. The loss of these connections reduces the possibilities for control and monitoring considerably, often resulting in trial-and-error procedures. Success and exact results then rely on sufficient manpower [6], individual skill [7], many iterations [8], and also a certain amount of luck [9].

In a coordinated design flow with stepwise refinement and validation (see Fig. 2), it is vital to be able to monitor and control the target systems at all times. It is therefore desirable to be able to connect to any target device as if it were located on an engineer's desktop, ideally with minimal influence on the device's operation.

A step in the right direction are techniques that use the WSN itself to provide a connection to the WSN devices [10], [11]. However, this implies altering the system and thus its behavior. Furthermore, it requires a relatively stable WSN. On failures, which occur frequently in early prototyping phases, a manual recovery is required.

A deployment-support network is useful throughout the entire development cycle: in an early phase, where the focus is on getting the first functions operational and distributing new code to all devices; during system testing in the final stages of development; in a production-like deployment phase where monitoring, validation and measurement with minimal interference are the primary focus. with the DSN approach, operation and testing in the field is made possible, significantly decreasing the amount of necessary cycles in the development process and achieving correct implementations in a timely manner.

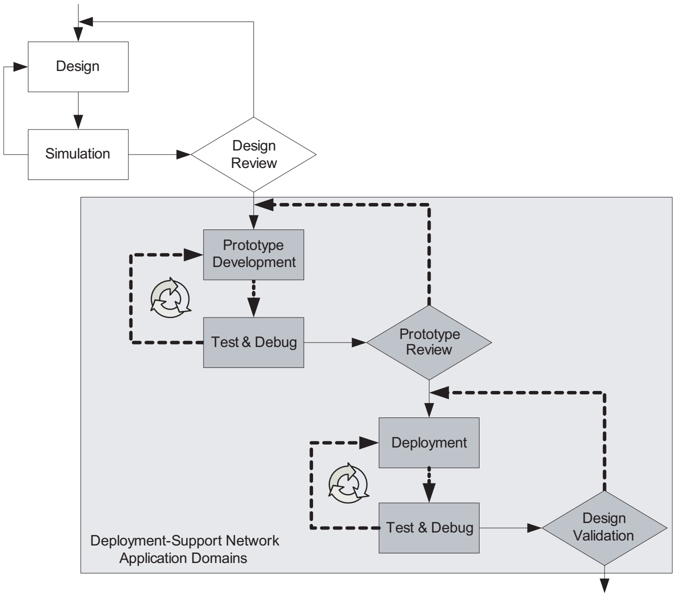


Fig. 2. A deployment-support network enables stepwise refinement. The amount of cycles necessary in the development and deployment process are reduced. Any WSN system can be tested and debugged live and embedded in the physical environment.

A. Basic Principle of a Deployment-Support Network

The ability to reliably connect to a WSN target device without altering it can be achieved by simply attaching a DSN device to every WSN target device, and letting the DSN devices construct and maintain an autonomous multihop network. A host, e.g. a PC, can tap into this network by attaching to one of the DSN devices, and open a virtual connection to an arbitrary DSN node. The host is then able to communicate both with this node and

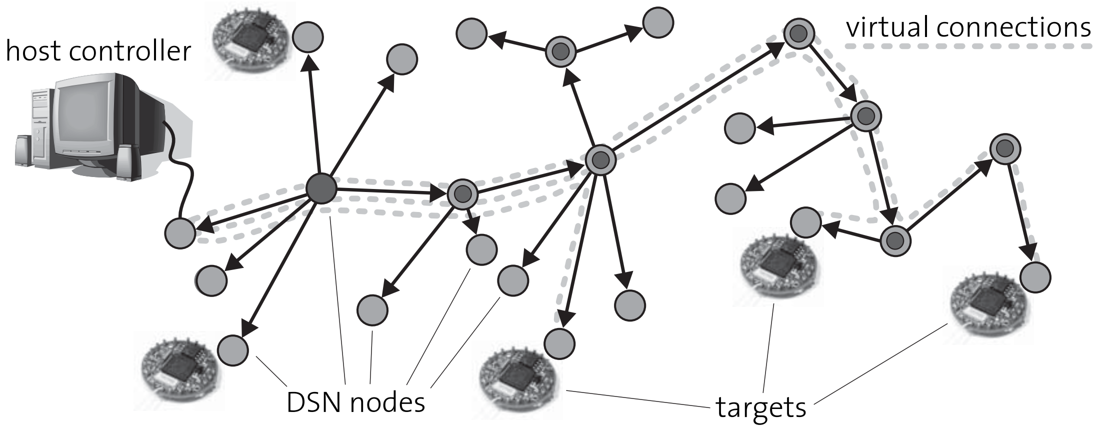


Fig. 3. The deployment-support network with three virtual connections opened from a host to DSN nodes with an attached WSN target: Here, Berkeley Motes are shown as exemplary targets, but arbitrary target devices that can be controlled, monitored and programmed through a serial port are possible.

with its attached target. Multiple virtual connections are possible, either originating at the same host, or at hosts attached to different DSN nodes.

There are different possibilities to interact with the target, such as serial-port connections for target monitoring and control, and remote target programming and resetting. In the case of serial-port *tunneling*, the target's serial port is replicated on the host system. The tunneled data can be used as if the target device was directly attached to the host. Standard tools like an in-system programmer or an in-circuit debugger can be used without modification, although the target may be multiple hops away. For operations requiring access to specific I/O pins of the target device (e.g. remote programming or resetting), additional general-purpose I/O pins of the DSN node have to be used. A general overview of the DSN components and some exemplary host–target virtual connections are shown in Fig. 3.

III. Implementation of a Self-healing DSN

In this section, we discuss our exemplary implementation of a DSN. We first motivate our choice of a hardware platform and then discuss the details of the implementation. The main challenges lie in efficient and robust distributed algorithms, as well as in the formation of a multihop network.

A. Selecting a Hardware Platform

To implement the proposed deployment-support network, we have to use devices that can construct and maintain multihop networks, support efficient packet data transport, and offer a generic target interface.

We chose Bluetooth as the wireless transport layer due to its reliable link-layer, multiplexing, and QoS capabilities, and the rather high bandwidth which allows to tunnel the aggregate traffic of multiple virtual connections. When selecting an appropriate Bluetooth platform, choices range from PC-class systems to tiny embedded devices. Since a large number of DSN nodes are to operate under similar constraints as the WSN target devices, they should be rather cheap, lightweight, and mobile. Power efficiency is not as critical, because the required lifetime of the DSN nodes is much shorter than that of the WSN. Versatility and easy applicability to different kinds of target devices are more important.

The BTnode [12] is an autonomous wireless communication and computing platform based on a Bluetooth radio module and an Atmel ATmega128L microcontroller, similar in performance to a Berkeley Mote. The BTnode is sufficiently small and yet powerful enough to support the intended DSN operation. It has not primarly been designed for ultra-low-power operation and mass production, but rather as a platform suited for fast prototyping and early deployment and experimentation [13].

B. BTnode Multihop Networking

Networking in Bluetooth is organized in master–slave configurations called Piconets. Every Piconet has one master to which up to seven active slaves can be connected at any time. Multiple Piconets can be interconnected by nodes with double roles (slave–slave or master–slave); the resulting network is called a Scatternet (see Fig. 4).

While the interconnection of nodes is part of the Bluetooth standard, the formation and control of multihop topologies is not. Also, only single-hop data transport is defined (from master to slave or vice versa). This means that an additional functional layer must take care of topology control and maintenance, and of all multihop packet forwarding.

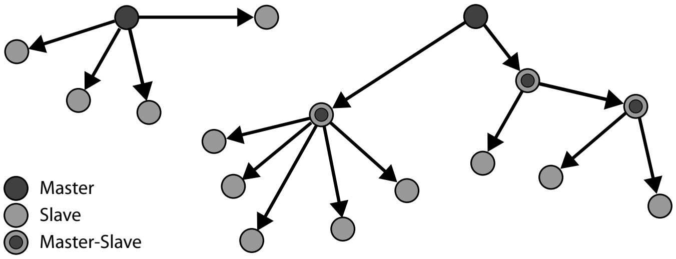


Fig. 4. Several Bluetooth Piconets are connected into a Scatternet using double master–slave roles on 3 nodes.

To ease the adaptation of our system to future devices, we chose a modular structure of the software running on each DSN node. We will now describe the two main modules, first specifying the required functionality, and then illustrating our implementation.

C. Topology Control and Maintenance

The *connection manager* constructs and maintains a multihop network of the DSN nodes. It shall be a simple, robust, and completely local algorithm that automatically takes care of link failures and joining and leaving nodes. The basic principle of a simple, distributed connection manager algorithm is as follows: Every DSN node periodically searches for other nodes, and subsequently tries to connect to all nodes found.

For this exemplary implementation we chose to restrict the connection manager to form tree topologies only. Since in a tree there is only one path between any two nodes, we are relieved from explicit route calculation; this reduces the system complexity considerably for first experiments and evaluation.

The BTnode devices available for experimentation are capable of forming Scatternets via one single master–slave double role, but not using the slave–slave double role or multiple double roles (see Fig. 4). There are some other constraints on the possible connectivity that need to be taken into account: (i) While performing an *inquiry()* or a *connect()*, a node is not visible to other nodes, (ii) while in the slave or master–slave state, a node is not visible, and (iii) while in the slave or master–slave state, a node cannot perform an *inquiry()* or *connect()*.

Treebuilding along the proposed scheme involves the detection of possible cycles before a potential *connect()*. This can be achieved by assigning a unique tree ID to each subtree formed (see Alg. 1). Upon detection of a node, the remote and local ID's are compared, and if they are not equal, a *connect()* is performed. The larger ID is then broadcast to the respective subtree.

Algorithm 1 Tree construction and maintenance

```
loop
  while $my\_slaves < max\_degree$ do
    $found\_nodes = inquiry();$
    for all $node$ in $found\_nodes$ do
      $remote\_id = get\_id(node);$
      if $remote\_id \neq my\_id$ then
        $connect(node);$
        if $remote\_id > my\_id$ then
          $my\_id = remote\_id;$
          $broadcast(remote\_id);$
        else
          $broadcast(my\_id);$
        end if
      end if
    end for
  end while
end loop
```

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[1] slaves. This means that new connections are opened only between nodes that are not in a 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 Fig. 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.

This simple algorithm allows the formation of large topologies in a robust and completely distributed fashion. It does not need central control, or exhaustive computation or communication between nodes. Therefore, it can be expected to scale well to a large number of nodes. Despite its simplicity, our algorithm builds and maintains a self-healing network that tries to reconnect subtrees separated upon disconnects.

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

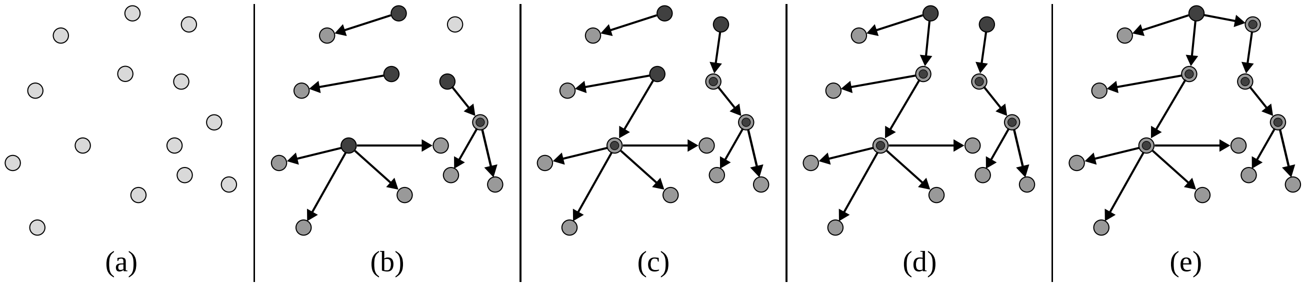


Fig. 5. A schematic view of the connection manager's operation: (a) Initially disconnected, (b) first Piconets form, (c) they interconnect to first Scatternets, until (d) larger Scatternets are forming and the tree structure becomes visible, and (e) finally, a single tree has been constructed.

D. Multihop Packet Forwarding

The *transport manager* takes care of multihop packet forwarding. It receives information about the available connections from the connection manager and uses these connections to route packets. This packet switching at every BTnode is based on ATM virtual-circuit switching and automatically forwards traffic to the appropriate connection based on a virtual-circuit identifier.

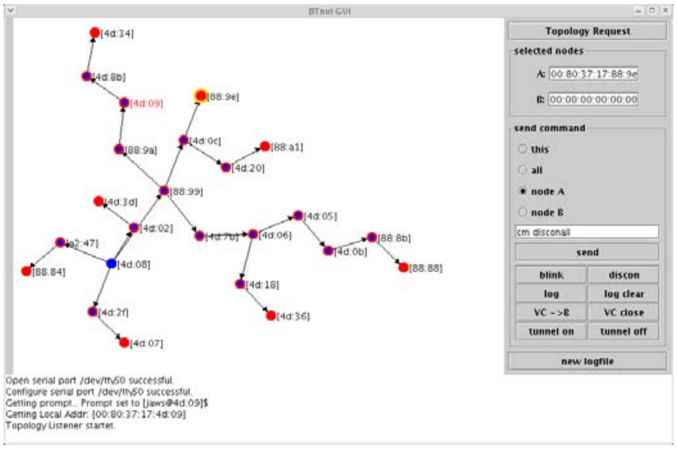
Communication is always initiated by a host, typically by opening a virtual connection to a DSN node. This is done by simply flooding the network with a route-request message. Each DSN node memorizes the connection ID a message arrived on; this is the route to the host to be used on the return path. When the destination node for a virtual connection sends its reply along the return path, the intermediate nodes assign a local virtual-circuit identifier to the connection ID the reply arrived on; this is the route to the destination node. After the setup is completed, packets can be transported over this virtual connection, with only minimal header processing at the intermediate nodes.

In case of link failures, the host and all endpoints of broken virtual connections are notified, and all virtual-circuit identifiers are removed from the local tables. The retransmission of lost packets in case of link failures has to be handled by the host application.

IV. Experimental Results

The results in this section are measured with the following experimental setup: 15–30 BTnodes are scattered randomly on a large desk. All nodes are programmed with the same software. A host PC is connected over a 115kbps serial link to one of the BTnodes. The host PC configures the BTnode to be a host node in order to receive topology information. Each node stores connection-specific events such as new connections and link losses to a local log. The topology information and the logged events are remotely collected by a monitoring and control application running on the host PC (see Fig.6).

We will now discuss two aspects of the implemented DSN: network-topology construction and the per-hop transmission delay.



A: 00:80:37:17:88:9e
B: 00:00:00:00:00:00

○ this
○ all
● node A
○ node B

cm disconall

```
Open serial port /dev/ttyS0 successful.
Configure serial port /dev/ttyS0 successful.
Getting prompt.. Prompt set to [jaws@4d:09]$
Getting Local Addr: [00:80:37:17:4d:09]
Topology Listener startet.
```

Fig. 6. A monitoring tool running on a host allows to monitor the network topology and control the deployment-support network.

a) Network-Topology Construction: – The topology construction depends on the ability to discover other nodes and to successfully connect to them. These are highly non-deterministic operations since no a-priori assumptions about the state of remote nodes can be made. They may be inquiring, connecting, or in a slave role, which means that they are not visible to others at that time. Previous measurements have shown that the time for inquiring is a time-consuming process and in the order of tens of seconds [14], [15] for a reliable discovery of all nodes. Experimentation has shown that for our continuous and iterative connection manager, a short inquiry that is repeated often accelerates the formation of large clusters. Our experiments were conducted with the following values: inquiries last 3 s and pauses between inquiries are chosen randomly between 5 and 20 s.

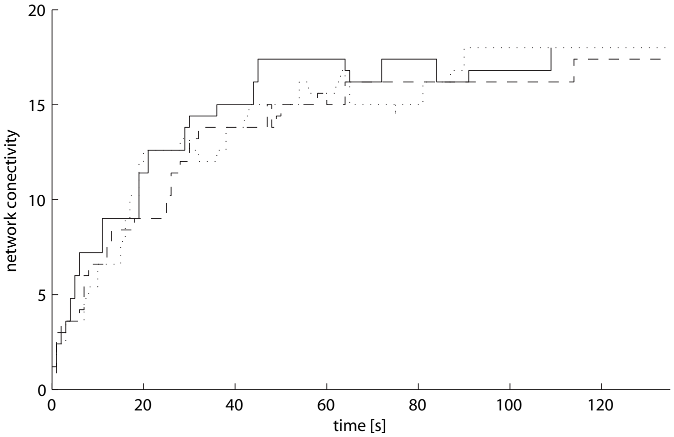


Fig. 7. Initial network-topology construction: Three different experiments are shown here with 19 nodes. The connectivity is the total amount of connections in the network cluster.

Figure 7 illustrates the evaluation of the initial connection events of 19 BTnodes. It shows three test runs of the initial topology construction with the total number of connections in the network[2]. The nodes start to form large clusters within approximately 60 s. The self-healing property can also be seen here: connections that are dropped are subsequently repaired.

Bluetooth connections take time to be set up. Pending connection requests and lost connections that have not been detected by both endpoints are visible as steps of a half in Figure 7. Here, the behavior of connection requests and successful connects can be seen in the quick steps in the left region. Disconnects that rely on the Bluetooth link supervision timeout that is typically set to multiple seconds have much longer half steps as can be seen on the right.

In contrast, in a tree structure without redundant connections, a failure of an arbitrary link may result in the disconnection of a large portion of the network. Figure 8 shows two experiments with a view of the local tree size of two independent nodes connected to a host PC. Since this is the local view from a specific node, local tree sizes can vary over a large range. On subsequent connect and disconnect events, whole subtrees can be affected. Using the self-healing property of the topology control and maintenance algorithm, a recovery to a similar tree size can be accomplished within a rather short time.

b) Per-Hop Transmission Delay: – On a virtual connection spanning multiple hops, the data has to be forwarded hop by hop. The transmission and processing delays add up along the path. We measured the delay by sending time-stamped packets to an endpoint. The packets are looped back to the sender, which measures the round-trip delay. Figure 9 shows the average round-trip delay divided by 2. For different packet sizes we have measured and averaged the delay of 40 packets.

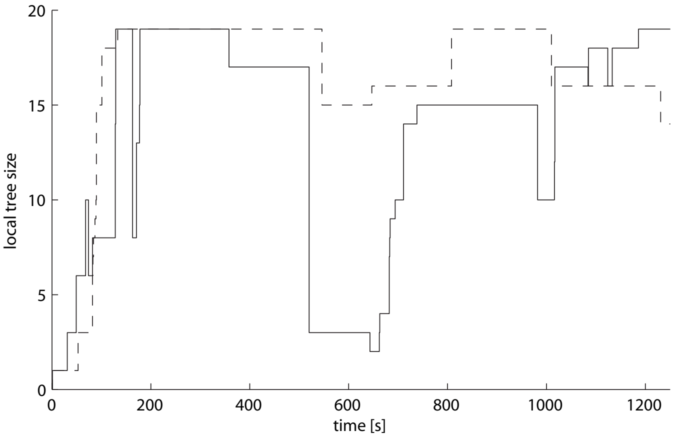


Fig. 8. Network-topology maintenance: Upon link losses the self-healing maintenance reestablishes a connected tree topology. Two different experiments are shown here with the local tree size.

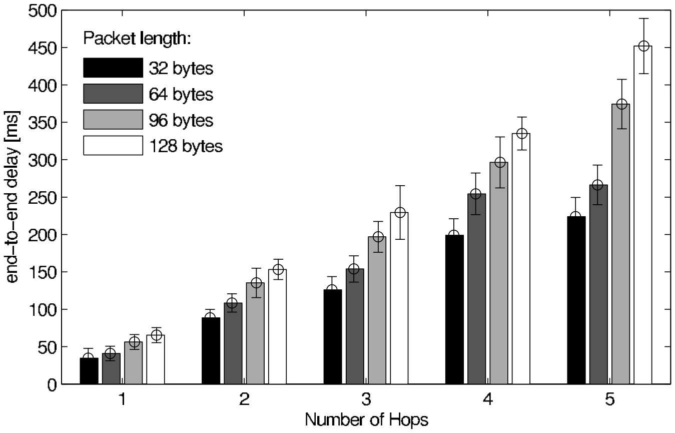


Fig. 9. Per-hop transmission delay: Average multihop transmission delay and standard deviation for different packet lengths.

The per-hop delay was measured to be 35–65 ms for the first hop and 45–90 ms for subsequent hops, depending on the packet size (see Fig. 9). For small packets, this results in approximately 50 ms per hop and per packet.

For serial-port tunneling, the end-to-end delay is of importance. The maximum tolerable delay for a serial connection depends on the application. Typically, a terminal session tolerates minutes before disconnecting, in-

[2] The connections are identical to the edges in the global topology graph of the network.

system programming can cope with a few seconds, and typing on an interactive user interface requires a maximal delay of 100–500 ms. Our measurements show that for the time being, we have to limit ourselves to tasks that do not require low-delay transmissions from the host to a DSN node many hops away. The local connection between DSN node and its attached WSN node can however meet any delay demands of the target WSN node. Delay sensitive applications like reprogramming a WSN node can thus be done by first downloading all the data to the DSN node, and from there with negligible delay to the WSN node.

V. Evaluation

Looking at the overall experimental results, we can see that the concept of a deployment-support network is working out. We have successfully formed connected tree topologies spanning 20–30 BTnodes and operating multiple virtual connections with data rates up to 57.6 kbit/s. Self-healing tree topologies spanning 10–12 hops are autonomously constructed in tens of seconds to a few minutes. The experience from our experiments has shown that the initial tree formation is sufficiently fast and produces large network topologies. In the following discussion, we will focus on the characteristics of the DSN itself and not so much on its interaction with a target WSN.

A major restriction rooted in the device constraints outlined in Section III-B is that all nodes have to be within each other's transmission range. Even when this is the case, it can happen that the root nodes of two or more disconnected trees all have max_degree slaves. In this case, they will not form new connections and thus not form a single final tree. A remedy here would be to introduce auxiliary algorithms to handle such lockup situations.

Due to the properties of our devices and the simplicity of the distributed algorithm, it is impossible to guarantee the formation of a single tree spanning all nodes in a given region. While giving a connectivity guarantee is very hard in a random wireless environment, an optimized topology could be achieved by breaking up trees at specific connections to optimize the topology after an initial construction phase.

The tree topology was a reasonable choice for the initial implementation and the proof of concept. However, the experiments show link losses which result in the disconnection of potentially large subtrees, prohibiting long-term operation of virtual tunnels over larger hop distances.

Some of the problems encountered can be directly attributed to the rather old and sometimes unreliable Bluetooth modules we have used in these initial experiments. Devices making use of a next-generation Bluetooth subsystem, such as the upcoming BTnode rev3, will allow greater flexibility, enhanced stability, and an increased performance. These new devices do not exhibit the limitations documented in Section III-C and can handle arbitrary connections in Scatternets[3].

To achieve greater stability in case of link losses and reliable virtual-connection operation for days, network topologies with redundant links are clearly favorable. This would eliminate many of the limitations discussed earlier, but would require additional functionality to run on the DSN nodes. While redundancy is favorable in respect to robustness and network performance, it comes at a significant price: It will require advanced topology-shaping algorithms and of course more complex routing, which is presently reduced to simple packet forwarding.

The connection and transport manager are already designed to accommodate such functionality and many proposed algorithms for topology control and routing exist. But again success here depends on the implementation details and the concerted behavior of the nodes. Since a wireless network is not static, given that links can break and nodes can fail all components have to be able to operate in a dynamic environment with joining and leaving nodes.

Even in a simple algorithm as presented, several parameters have to be determined through practical experimentation. In our work, a major increase in performance was achieved with the selection of the correct duty-cycle parameters for the inquiry duration and period.

VI. Related Work

Much care has been taken to appropriately design component-based system architectures [16]. TinyOS [17] pursues a network-centric approach to embedded software systems. An easy jump start for fast prototyping was the focus in the design of the BTnode system software [13], where applications can be run in an emulation mode, without download to the target hardware. However, all these approaches are mainly concerned with the single device and its architecture, and not with networks of many devices. Without the appropriate external tools and methodologies, these systems will not be able to cope with more complex applications.

[3] This can be attributed to the recently revised Bluetooth standard and implementation details on Bluetooth chipset and firmware.

Distributed simulation, e.g. TOSSIM [1], is a valuable tool in the development process; it allows to study system-design alternatives in a controlled environment. The problem with simulation is that assumptions have to be made [2], [18], and simplifications have to take place [19]; this inevitably leads to a gap between reality and the simulated, virtual world. In addition, simulation results are known to not always be consistent across different scenarios or tool chains [3].

A significant approach to bridge the gap between simulated and real world is the concept of virtualization pursued by the Emstar architecture [5]. Here, a flow from specification via simulation and emulation to testing on a Linux platform is described. Unique to this approach is the rather tight integration of real and virtual components in one framework, but the capability to capture all properties of the environment and of the devices themselves in the respective virtual counterparts is limited.

Lessons from experimentation, such as outdoor sensor network expeditions [8], [9], university class projects [6], or large-scale testbed deployments [20], have required direct and reliable access to the WSN devices involved. Out of this necessity, many different techniques have been developed to attach temporarily or permanently to WSN devices: Layered architectures with different scales of devices and hierarchical networks [21], [22], serial multiplexing units mounted onto a table, as a semi-mobile laptop unit, or mounted into a ceiling array [23], [5], [20], direct Ethernet attachments such as the MIB600 programming board for the Berkeley Motes [24], used in the moteLab and sMote testbeds. All of these have in common that they require wired connections, although it is common to replace the direct serial cabling by another medium to allow multiplexing. In the case of large mobile or outdoor applications, it is very hard if not impossible to connect a significant amount of devices, whereas setups like a fixed ceiling array can only offer a very synthetic environment for experimentation. This limits the applicability of these approaches to cumbersome testing scenarios where fixed infrastructure is put in place for the duration of testing and removed afterwards.

Tools for remote in-network programming can be operated as a self-regulating code-propagation service inside a sensor network [10], and it has been shown that this technique scales to large populations [11]. However, it requires alterations of the WSN target devices, and offers only remote programming. The lack of dedicated feedback becomes even more important when monitoring data-centric systems without dedicated node identification [25].

Wireless cable replacement is also proposed for monitoring and diagnosing computer systems [26]. The main goal in this work is to reduce cost and system complexity of the testing infrastructure only temporarily used. Here, the wired daisy-chained boundary-scan circuit is replaced by a more robust and flexible wireless network that uses the broadcast medium for direct point-to-multipoint connections to system components.

The Bluetooth standard also offers the RFcomm profile for serial-port communication, and various vendors are offering products based either on RFcomm or on proprietary extensions. The drawback of these solutions is that they offer point-to-point connections between two endpoints only.

The solutions proposed for the automatic formation of Bluetooth Scatternets can be classified according to the preconditions of the algorithms and the properties of the resulting Scatternet. One such precondition is that all nodes must be in each other's transmission range (single-hop). Algorithms of this kind, such as the one presented in [27], are not an adequate choice for a DSN, since a sensor network is usually significantly larger than the transmission range of Bluetooth devices. The BlueMesh [28] algorithm is one solution for the more general multihop case, and guarantees that a connected mesh topology is achieved (if it is achievable). Unfortunately, the authors of BlueMesh assume that the topology of the network does not change. This algorithm can therefore not deal with link failures and leaving and joining nodes.

There are many other proposed solutions referring to the problem of setting up a static Scatternet [29], [30]. However, to the best of our knowledge, implementation reports of a large-scale multihop-Scatternet-formation algorithm are missing so far. Even theoretical analysis has just begun to address the problem of maintaining a connected Scatternet for multihop topologies in a dynamic environment. Recent evaluations of Scatternet-formation algorithms [31], [32] have also referred to the lack of such algorithms.

The properties and performance of the above-mentioned algorithms are determined by simulation. But the parameters chosen for the evaluation are often not realistic. For instance, the performance evaluation in [32] used a variable time for the inquiry operation, randomly selected in the interval from 0.01 s to 0.5 s. This is not realistic on real devices where the duration is specified to be in the range of 1.28 s to 61.44 s by the Bluetooth standard.

VII. Conclusion and Outlook

With the concept of a deployment-support network, we have introduced a valuable tool for the coordinated deployment of distributed sensor networks. In contrast to existing test setups in laboratories, the DSN approach allows large-scale deployment without losing the ability to observe and control all nodes and without the burden of fixed, wired infrastructure or changes to the target system. Scalable communication to large populations of WSN target devices embedded in their designated physical environment will allow to investigate the performance of live sensor systems to enable detailed comparisons and validation. Moreover, a DSN can be used for arbitrary cable replacements in industrial instrumentation and automation control.

We have presented promising experimental results based on the BTnode platform that have given us first insights into the feasibility and performance of the deployment-support network proposed.

The design and implementation has been challenging and stimulated new research issues to be pursued. Applying the extensive experience gained through implementation and experimentation we can identify the following areas for future work: (i) Appropriate control and analysis tools running on the host controller to manage the data delivered by the DSN and (ii) redundant network topologies and robust multihop data transport. This will necessitate new developments and the application of well established techniques from the database and distributed systems community to create appropriate solutions.

In this paper, we propose a new methodology for the design, development, test, deployment, and validation of a large, distributed sensor-network application. The key challenges for achieving functional and manageable systems of many embedded devices will remain to be efficient, robust, and lean implementations.

Acknowledgements

We would like to thank Martin Hinz and the BTnode community for tireless implementation and debugging support.

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.

References

[1] 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)*. ACM Press, New York, Nov. 2003, pp. 126–137.

[2] D. Kotz, C. Newport, R. Gray, J. Liu, Y. Yuan, and C. Elliott, “Experimental evaluation of wireless simulation assumptions,” in *Int’l Workshop Modeling Analysis and Simulation of Wireless and Mobile Systems (MSWiM 04)*. ACM Press, New York, Oct. 2004, p. to appear.

[3] D. Cavin and Y. Sasson, “On the accuracy of MANET simulators,” in *ACM Workshop Principles Of Mobile Computing (POMC 02)*. ACM Press, New York, Oct. 2002, pp. 38–43.

[4] 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)*, ser. Lecture Notes in Computer Science, vol. 2920. Springer, Berlin, Jan. 2004, pp. 307–322.

[5] L. Girod, J. Elson, A. Cerpa, T. Stathapopoulos, N. Ramanathan, and D. Estrin, “EmStar: A software environment for developing and deploying wireless sensor networks,” in *Proc. USENIX 2004 Annual Tech. Conf.*, June 2004, pp. 283–296.

[6] B. Hemingway, W. Brunette, T. Anderl, and G. Borriello, “The Flock: Mote sensors sing in undergraduate curriculum,” *IEEE Computer*, vol. 37, no. 8, pp. 72–78, Aug. 2004.

[7] A. Cerpa, J. Elson, M. Hamilton, J. Zhao, D. Estrin, and L. Girod, “Habitat monitoring: application driver for wireless communications technology,” *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 2, pp. 20–41, Apr. 2001.

[8] 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)*. ACM Press, New York, Sept. 2002, pp. 88–97.

[9] R. Szewczyk, E. Osterweil, J. Polastre, M. Hamilton, A. Mainwaring, and D. Estrin, “Habitat monitoring with sensor networks,” *Communications of the ACM*, vol. 47, no. 6, pp. 34–40, June 2004.

[10] P. Levis, N. Patel, D. Culler, and S. Shenker, “Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks,” in *Proc. First Symp. Networked Systems Design and Implementation (NSDI ’04)*. ACM Press, New York, Mar. 2004, pp. 15–28.

[11] J. 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)*. ACM Press, New York, Nov. 2004, p. to appear.

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

[13] 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)*, ser. Lecture Notes in Computer Science, vol. 2920. Springer, Berlin, Jan. 2004, pp. 323–338.

[14] O. Kasten and 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)*, Sept. 2001.

[15] E. Welsh, P. Murphy, and 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)*, March 2002.

[16] J. 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)*. ACM Press, New York, Nov. 2000, pp. 93–104.

[17] D. Culler, J. Hill, P. Buonadonna, R. Szewczyk, and A. Woo, “A network-centric approach to embedded software for tiny devices,” in *First Int’l Workshop on Embedded Software (EMSOFT 2001)*, ser. Lecture Notes in Computer Science, vol. 2211. Springer, Berlin, Oct. 2001, pp. 114–130.

[18] R. Min and A. Chandrakasan, “Top five myths about the energy consumption of wireless communication,” *Mobile Computing and Communications Review*, vol. 7, no. 1, pp. 65–67, Jan. 2003.

[19] D. Kotz, C. Newport, and C. Elliott, “The mistaken axioms of wireless-network research,” Dartmouth College Computer Science, Tech. Rep. TR2003-467, July 2003.

[20] A. Cerpa and D. Estrin, “ASCENT: Adaptive self-configuring sensor networks topologies,” *IEEE Transactions on Mobile Computing*, vol. 3, no. 3, pp. 272–285, July 2004.

[21] G. Pottie and W. Kaiser, “Wireless integrated network sensors,” *Communications of the ACM*, vol. 43, no. 5, pp. 51–58, May 2000.

[22] J. Hill, M. Horton, R. Kling, and L. Krishnamurthy, “Wireless sensor networks: The platforms enabling wireless sensor networks,” *Communications of the ACM*, vol. 47, no. 6, pp. 41–46, June 2004.

[23] A. Cerpa, N. Busek, and D. Estrin, “SCALE: A tool for simple connectivity assessment in lossy environments,” Center for Embedded Networked Sensing, Univ. of California, Los Angeles, CA, Tech. Rep. 21, Sept. 2003.

[24] J. Hill and D. Culler, “Mica: A wireless platform for deeply embedded networks,” *IEEE Micro*, vol. 22, no. 6, pp. 12–24, Nov. 2002.

[25] S. Madden, M. Franklin, J. Hellerstein, and W. Hong, “TAG: A tiny aggregation service for ad-hoc sensor networks,” in *Proc. 5th Symp. Operating Systems Design and Implementation (OSDI 2002)*. ACM Press, New York, Dec. 2002, pp. 131–146.

[26] H. Eberle, “A radio network for monitoring and diagnosing computer systems,” *IEEE Micro*, vol. 23, no. 1, pp. 60–65, Jan. 2003.

[27] C. Law, A. Mehta, and K. Siu, “A new Bluetooth scatternet formation protocol,” *ACM/Kluwer Mobile Networks and Applications*, vol. 8, no. 5, pp. 485–498, October 2003.

[28] C. Petrioli, S. Basagni, and I. Chlamtac, “BlueMesh: Degree-constrained multihop scatternet formation for Bluetooth networks,” *ACM/Kluwer Mobile Networks and Applications*, vol. 9, no. 1, pp. 33–47, Feb. 2002.

[29] ——, “Configuring BlueStars: Multi-hop scatternet formation in Bluetooth networks,” *IEEE Transactions on Computers*, vol. 52, no. 6, pp. 779–790, Jun. 2003.

[30] G. Záruba, S. Basagni, and 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, June 2001, pp. 273–277.

[31] S. Basagni, R. Bruno, and C. Petrioli, “A performance comparison of scatternet formation protocols for networks of Bluetooth devices,” in *Proc. 1st IEEE Int’l Conf. Pervasive Computing and Communications (PerCom 2003)*. IEEE CS Press, Los Alamitos, CA, Mar. 2003, pp. 341–350.

[32] S. Basagni, R. Bruno, G. Mambrini, and C. Petrioli, “Comparative performance evaluation of scatternet formation protocols for networks of Bluetooth devices,” *Wireless Networks*, vol. 10, no. 2, pp. 197–213, March 2004.