Deployment Support Network
A Toolkit for the Development of WSNs

Matthias Dyer[1], Jan Beutel[1], Thomas Kalt[1], Patrice Oehen[1], Lothar Thiele[1], Kevin Martin[2], and Philipp Blum[2]

[1] Computer Engineering and Networks Laboratory, ETH Zurich, Switzerland
[2] Siemens Building Technologies Group, Switzerland

Abstract. In this paper, we present the Deployment Support Network (DSN), a new methodology for developing and testing wireless sensor networks (WSN) in a realistic environment. With an additional wireless backbone network a deployed WSN can be observed, controlled, and reprogrammed completely over the air. The DSN provides visibility and control in a similar way as existing emulation testbeds, but overcomes the limitations of wired infrastructures. As a result, development and experiments can be conducted with realistic in- and outdoor deployments. The paper describes the new concept and methodology. Additionally, an architecture-independent implementation of the toolkit is presented, which has been used in an industrial case-study.

1 Introduction

The validation and testing of wireless sensor network (WSN) algorithms and software components is an indispensable part of the design-flow. However, this is not a trivial task because of the limited resources of the nodes. Considering further the distributed nature of the algorithms, the large number of nodes, and the interaction of the nodes with the environment, leads to the fact that the nodes are not only hard to debug, but also hard to access.

Access to the state of the nodes, often referred to as visibility, is fundamental for testing and debugging. More visibility means faster development. But not only the amount of state information, but also their quality is important. Simulators for sensor networks for example, provide almost unlimited visibility. But on the other hand they use simplistic models for the abstraction of the environment. They fail to capture the complex physical phenomena that appear in real deployments. Therefore, the visibility of simulations is of low quality.

For this reason, researchers have built emulation testbeds with real devices. Existing testbeds consist of a collection of sensor nodes that are connected to a fixed infrastructure, such as serial cables or ethernet boxes. Testbeds are more realistic than simulators because they use the real devices and communication channels. The problem that remains is that the conditions in the field where the WSN should be deployed in the end can be significantly different from the testbed in a laboratory. In particular, with a cable-based infrastructure it is impossible to test the application with a large number of nodes out in the field.

K. Langendoen and T. Voigt (Eds.): EWSN 2007, LNCS 4373, pp. 195–211, 2007.

In evaluations of real deployment experiments like the ones presented in [1,2,3], a gap between simulation-emulation-results and the measured results of the real deployment has been reported. Measured packet yields of 50%, reduced transmission ranges, dirty sensors and short life-times of nodes did not match the expectations and the results obtained through simulation and emulation. The unforseen nuances of a deployment in a physical environment forced the developers to redesign and test their hardware- and software components in several iterations. It has also been shown that sacrificing visibility, such as switching off the debugging LEDs on the nodes in favor of energy-efficiency is problematic during the first deployment experiments [4].

In this paper, we present the *Deployment Support Network*, a toolkit for developing, testing and monitoring sensor-network applications in a realistic environment. The presented methodology is a new approach, since it is wireless and separates the debugging and testing services from the WSN application. Thus it is not dependent on a single architecture or operating system. In contrast to existing approaches, our method *combines* the visibility of emulation testbeds with the high quality of information that can only be achieved in real deployments. The DSN has been implemented and applied in an industrial case-study.

The remaining of the paper is organized as follows: section 2 presents related work, sections 3 and 4 describe our approach and its realization. In section 5 an industrial case-study is presented and finally, in section 6, we discuss the advantages and limitations of our method and conclude the paper.

2 Related Work

To support the development and test of sensor-network applications various approaches have been proposed. On the one hand, simulation and emulation testbeds allow for observation of behaviour and performance. On the other hand, services for reprogramming and remote control facilitate the work with real-world deployments.

Simulation. Network simulators such as *ns-2* [5] and Glomosim [6] are tools for simulation of TCP, routing, and multicast protocols over wired and wireless networks. They provide a set of protocols and models for different layers including mobility, radio propagation and routing. TOSSIM [7] is a discrete event simulator that simulates a TinyOS mote on bit-level. TOSSIM compiles directly from TinyOS code allowing experimentation with low-level protocols in addition to top-level application systems.

Emulation Testbeds and Hybrid Techniques. Indoor testbeds use the real sensor node hardware which provides much greater sensing-, computation-, and communication realism than simulations. In some testbeds the nodes are arranged in a fix grid, e.g. on a large table or in the ceiling. They are connected via a serial cable to a central control PC. In the MoteLab testbed [8], each node is attached to a small embedded PC-box that acts as a serial-forwarder. The control PC can then access the nodes via the serial-forwarders via ethernet or 802.11.

The *EmStar* framework [9] with its *Ceiling–Array* is also an indoor testbed. It additionally provides the ability to shift the border between simulation and emulation. For instance, the application can run on the simulator whereas for the communication the radio hardware of the testbed is used. This hybrid solution combines the good visibility of simulators and the communication realism of real radio hardware. Another feature of *Emstar* is its hardware abstraction layer. It allows the developers to use the same application-code for simulation and for emulation without modification, which enables a fast transition between different simulation- and emulation modes. The operation mode that provides the best sensing-, computation-, and communication realism within *Emstar* is called *Portable–Array*. It is still a wired testbed but with its long serial cables it can be used also for outdoor experiments.

SeNeTs [10] is in many aspects similar to *Emstar*. Both run the same code on simulation and on the real node hardware and both incorporate an environment model. The main difference is that in *SeNeTs* the simulation part runs on distributed PCs, which improves scalability.

Services for real-world deployments. Deluge [11] is a data-dissemination protocol used for sending new code images over the air to deployed TinyOS sensor nodes. It uses the local memory on the nodes for caching the received images. A disseminated buggy code image could render a network unusable. This problem can be addressed with a *golden image* in combination with a watchdog-timer [3]. The golden image is a know-working program that resides on every node, preferably on a write-protected memory section. Once a unrecoverable state is reached the watchdog-timer fires and the bootloader loads the golden image, which reestablishes the operability of the network.

Marionette [12] is an embedded RPC service for TinyOS programs. With some simple annotations, the compiler adds hooks into the code which allow a developer at run-time to remotely call functions and read or write variables. The main cost of using Marionette is that each interaction with a node requires network communication. Sharing the wireless channel with the application could adversely affect the behavior of the network algorithm that is being developed or debugged.

3 Deployment Support Networks

The Deployment Support Network (DSN) is a tool for the development, debugging and monitoring of distributed wireless embedded systems in a realistic environment. The basic idea is to use a second wireless network consisting of so-called DSN-nodes that are directly attached to the target nodes.

The DSN provides a separate reliable wireless backbone network for the transport of debug and control information from and to the target-nodes. However, it is not only a replacement for the cables in wired testbeds but it also implements interactive debugging services such as remote reprogramming, RPC and data/event-logging.

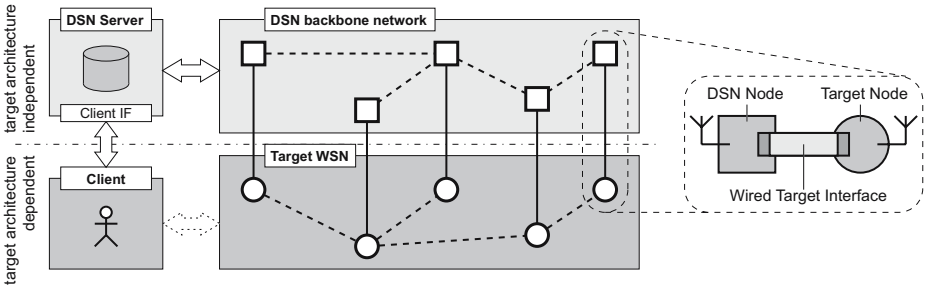

Fig. 1. Conceptual view of a DSN-system with five DSN-node/target-node pairs

3.1 DSN–Architecture

Overview. Figure 1 shows an overview of the different parts in a DSN-system. On the right hand side is the *DSN-node/target-node pair* that is connected via a short cable, referred to as the *wired target interface*. DSN-nodes are battery-operated wireless nodes with a microcontroller and a radio-module, similar to the target-nodes.

In the center of the figure, there is a conceptual view of the DSN with the two separate wireless networks: the one of the DSN-nodes and the one of the target-nodes. The network of the DSN-nodes is a automatically formed and maintained multi-hop backbone network, that is optimized for connectivity, reliability and robustness.

The *DSN-server* is connected with the DSN-backbone-network and provides the client interface, over which the client can communicate and use the implemented DSN-services. The *client* is a target-specific application or script. The information flow goes from the client over the DSN-server to the DSN-nodes and finally to the target nodes and vice versa. The DSN-server decouples the client from the target WSN both in time and space. In particular, data from the target nodes are stored in a database and can be requested anytime, and commands can be scheduled on the DSN-nodes. Separation in space is given through the client interface that allows for an IP-based remote access.

Target-Architecture-Independent Services. A key feature of the DSN-system is the clear separation of the target-system and the DSN-services. As a result, the DSN can be used for the development and testing of different target-architectures. The DSN-services are target-architecture-independent. Only the *wired target interface* and a small part of the software on the DSN-nodes to control it must be adapted. However, this adaptation is typically a matter of I/O configuration which is completed fast.

Client Interface. The DSN-server provides a flexible RPC user interface for the DSN-services. A developer can write his own client scripts and test applications. The client virtually communicates over the DSN with the WSN application on the target-nodes.

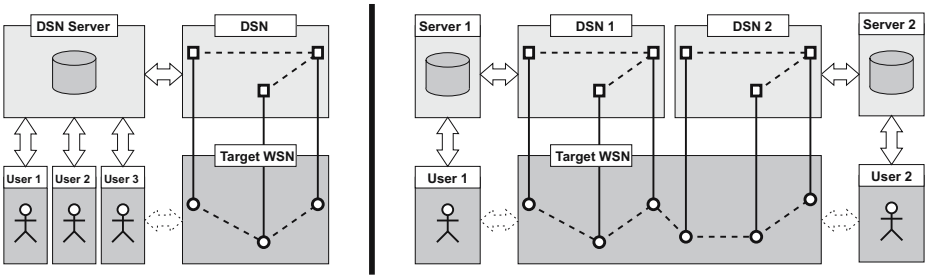

Fig. 2. DSN multi-user and multi-network

The DSN supports multiple users. Figure 2 shows two examples. Multiple users can connect to one DSN-server. Different access privileges can be assigned to users. For example this allows for read-only access or for power-users that are permitted to reprogram the target nodes or to reconfigure the DSN. The second example shows two separate DSN-networks, each with its own server. By this means, two developers can work independently in the same location without interfering each other. Additionally this setup balances the load onto two networks, i.e. yielding a better performance. The separation is achieved by a unique DSN-network-ID that is checked on the DSN connection setup.

3.2 DSN–Services

In this section we describe the debugging services that are provided by the DSN. Each service has a part which is implemented on the DSN-server and a part that is implemented on the DSN-nodes.

Data- and Event-Logging. Probably the most important service of the DSN is the data- and event-logging. It gives the developers insight into the state of the target nodes. The basic concept is as follows: The target-nodes write logging-strings to the wired target-interface (by using e.g. printf-like statements for writing to a debug-UART). The DSN-node receives the log-string, annotates it with a time-stamp and stores it in a local logfile. On the other side, the DSN-server has a logging database, where it collects the log-messages from all the DSN-nodes. For that purpose there are two mechanisms: In *pull-mode*, the DSN-server requests the log-messages, where in *push-mode*, the DSN-nodes sends the log-messages proactively to the server. Finally, the user can query the database to access the log-messages from all target-nodes.

String-based messages are very convenient for debugging and monitoring sensor network applications. They can be transmitted via a serial two-wire cable to the DSN-node with only little overhead on the target-nodes. The majority of the work is done on the DSN-nodes: They run a time-synchronization protocol for accurate time-stamping and they are responsible for transmitting the messages to the server. However, for certain experiments, even the overhead of writing short

log-messages to a serial interface is not acceptable. Therefore, there is a second mechanism which uses I/O and external interrupt-lines to trigger an event. Events are, similar to the log-strings, time-stamped and cached in the logfile.

The caching of messages on the DSN-nodes is an important feature. It allows for a delayed transmission to the server, which is necessary for reducing the interference. The transmission can for example be scheduled by the server (pull-mode) or it is delayed until an experiment has finished. It even allows the DSN-nodes to disconnect entirely from the DSN for a certain time and then after reconnection to send all cached log-messages.

For the sake of platform-independence, the content of the log-messages is generally neither parsed by the DSN-nodes nor the DSN-server. This is the responsibility of the user application written by the developer who knows how to interpret the format. However, the DSN-nodes can optionally classify the messages into classes such as *Errors*, *Warnings* and *Debug*. This can be useful if one wishes to directly receive critical error-messages using the push-mode, while the bulky rest of the debugging messages can be pulled after the experiment.

Commands. The counterparts of the log-messages are the commands. With this service, string-based messages can be sent from the client to the target-nodes. There are two types of commands: *instant commands* that are executed immediately and *timed commands* that are schedulable. A destination-identifier that is given as a parameter, lets the client select either a single node or all nodes. Once the command is delivered to the DSN-server, it is transmitted immediately to the selected DSN-nodes. Then, in the case of an instant command, the message-string is sent over the wired target interface to the target-nodes. For the timed commands, a timer on the DSN-node is started which delays the delivery of the message. Again, the content of the message-string is not specified. It can be binary data or command-strings that are interpreted on the target-nodes.

Together with the data-logging, this service can be applied for the emulation of interactive terminal sessions with the target-nodes. This service sends commands to the nodes while the replies are sent back as log-messages to the user. In other words, this is a remote procedure call (RPC) that goes over the backbone network of the DSN. The wireless multi-hop network introduces a considerably larger delay than direct wired connections. However, even with a few seconds it is still acceptable for human interaction.

Timed commands are necessary if messages should be delivered at the same time to multiple target-nodes. The accuracy of time-synchronization of the DSN-nodes is orders of magnitude higher than the time-of-arrival of broadcasted messages. In addition this service can be used to upload a script with a set commands that will get executed on the specified time.

In addition to the described commands, there is an additional command for the wired target interface which lets the developer control the target-architecture specific functions such as switching on/off the target power, reading the target voltage, and controlling custom I/O lines.

Remote Reprogramming. The DSN has a convenient remote-reprogramming service. The developer uploads a code image for the target-nodes to the server, which forwards it to the first DSN-node. The DSN-nodes have an extra memory section that is large enough to store a complete code image. They run a data-dissemination protocol to distribute the code image to all nodes over the backbone network. The nodes periodically send status-information to the direct neighbors including a version number and the type of the image. By doing so, also newly joined nodes with an old version get updated.

At any time, the developer can monitor the progress of the data dissemination. Once all DSN-nodes have received the code image, he can, with an additional command, select a set of DSN-nodes for the reprogramming of the target-nodes. The DSN-node is connected to the programming-port of the target-node through the wired target interface. This programming connection and its software driver on the DSN-nodes is one of the few parts of the DSN-system that is architecture-dependent. It must be adapted if target-nodes with a new processor type are used.

DSN configuration and DSN status. The DSN is configurable. The user can set different operation-modes and parameters both at setup-time and at run-time. One such mode is the *low-power/low-interference mode*. When this mode is set, the DSN-nodes switch off their radio for a given time-interval. This might be important if the radio of the DSN and the one of the target-system interfere with each other. If this is the case, the DSN radio should be switched off during the experiment. As this mode is also very energy-efficient, it could be set whenever the DSN remains unused for a known time, e.g. during the night. This will significantly increase the life-time because only a timer on the microcontroller of the DSN-nodes needs to be powered.

The DSN further provides the developer with the possibility to gain information about the state of the DSN. In particular, the following information is provided:

- a *list of connected DSN-nodes* with a *last-seen* timestamp,
- a *location-identifier*,
- the *connectivity information* of the DSN-nodes,
- the *versions and types* of the stored code images, and
- the battery voltages of the DSN-nodes

The location-identifier is a string that is stored on every DSN-node containing e.g. the coordinates for positioning. It can be set via the user-interface when the DSN is deployed.

The gathering of the DSN-status requires bandwith on the DSN backbone network. To minimize this overhead, the DSN-server only fetches the information from the DSN-nodes when it is explicitly requested by the client. Otherwise it provides a cached version of the DSN-status.

3.3 Test Automation

It is often not enough to have an interactive terminal session to the nodes. There is a need for automation and scripting support for the following reasons:

(a) Experiments have to be repeated many times, either for statistically validity or to explore different parameter settings. (b) The execution of an experiment has to be delayed, e.g. since interference caused by human activity is minimized during nighttime. (c) Experiments last longer than an operator can assist.

One possibility to automate tests is to send once a set of timed commands to the DSN-nodes (see section 3.2). However, a more sophisticated method for test automation is to use scripts that interact with the DSN-server. This has the advantage that a script can evaluate the state of the targets during test execution and adapt its further actions. For example, only when a particular message from node A is received, a command to node B is sent.

The above described DSN-services are accessible as RPC functions and can therefore be called easily from scripts. Table 1 shows some functions for the different parts.

Table 1. Pseudo-syntax of the RPC functions

```
test-setup:
   loadImage(type, version, code image)
   targetFlash(selected-nodes)
   dsnConfig([selected-nodes], property, value)
   setLogMode(selected-nodes, class, push|pull)
test-execution:
   instantCommand(selected-nodes, command)
   timedCommand(selected-nodes, command, time)
result gathering:
   getDSNStatus()
   getLog(filter)
```

4 Realization

In the previous section, we described the general concept and methodology of the DSN. In this section we present our implementation. Figure 3 shows an overview of the technologies used in our implementation. See also [13] for details.

For the DSN-nodes we use the BTnodes rev3 [14]. This platform has proven to be a good choice since it has a relatively large memory and a robust radio.

4.1 Bluetooth Scatternets

Bluetooth is not often seen on sensor-networks, due to its high energy-consumption (BTnode: 100 mW). However, it has a number of properties that the traditional sensor-network radios do not have and which are very important for the DSN backbone network. Bluetooth was initially designed as a cable-replacement. It provides very robust connections. Using a spread-spectrum frequency-hopping scheme, it is resilient against interference and has a high spatial capacity. Robustness and spatial capacity are mission-critical for the DSN.

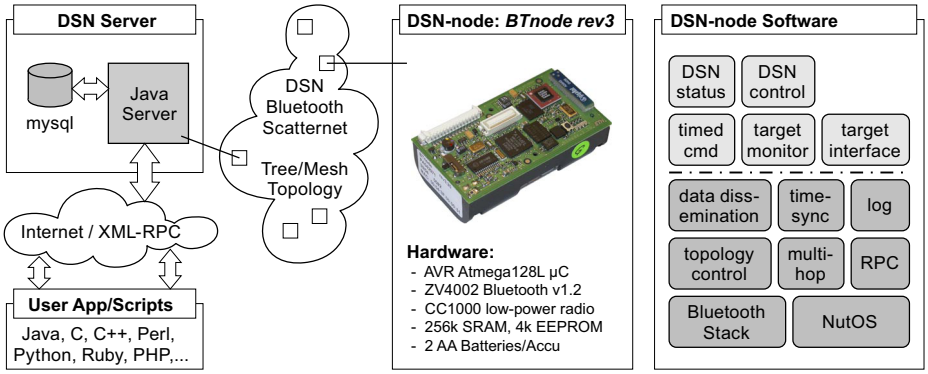

Fig. 3. Technology overview of the DSN implementation on the BTnodes rev3

Using Bluetooth for the DSN has further the benefit that it is potentially accessible from PDAs and mobile phones. Although this not utilized in our current implementation, it opens interesting new usage scenarios for the future.

We use the BTnut system software on the BTnodes which comes with an embedded Bluetooth stack. For the automatic formation and maintenance of a connected Bluetooth scatternet, we have implemented two topology control and routing algorithms: a simple, fast tree-builder and a more sophisticated mesh algorithm. They are both adaptive algorithms, i.e. taking network changes due to link-losses and leaving or joining nodes into account. For more details on these topology control implementations see [15] and [16].

4.2 Wired Target Interface

The only part of the DSN-node software that must be adapted for new target architectures is the *target interface*. Common to most platforms is that data transport is possible through a serial RS232-like connection. The BTnode provides a hardware UART for this purpose. Porting this interface to similar platforms consist of adapting the bitrate and configuring flow control.

More problematic is the programming connection, since different platforms have quite different solutions. Some target-architectures provide direct access to the programming port (ISP). For this case the target-interface must execute the ISP protocol of the appropriate microcontroller type. We have this currently implemented for the AVR and the MSP430 microcontroller family. Some other target-architectures use the same serial port both for data transport and programming. The appropriate control signals for the multiplexing must then be issued by a custom function of the target interface on the DSN-node.

A third programming method is applied on the *Tmote Sky* target: We had to programm the targets with a custom bootloader that is able to receive the code image over the external pins (instead of the USB-connector). Figure 4 shows four different DSN-node - target-node pairs for which our implementation supports the general DSN-services. For the BTnode- and the A80 target we

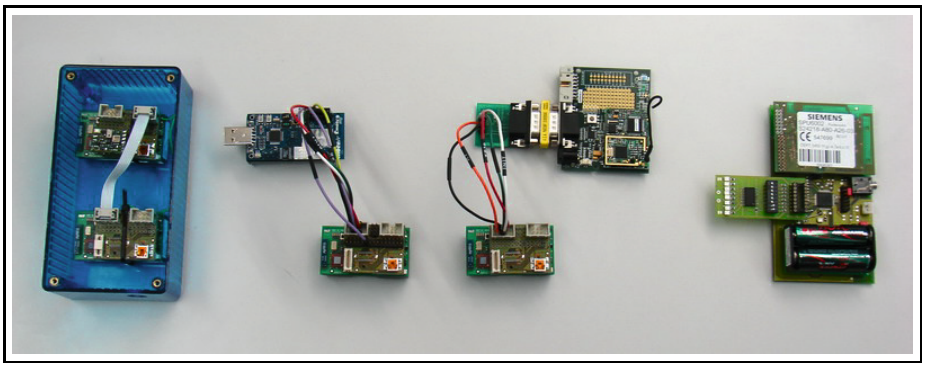

Fig. 4. Realized target interfaces: 4 different DSN-node–target-node pairs are shown (from left to right): BTnode rev3, Moteiv Tmote Sky, Shockfish TinyNode 584, and Siemens SBT A80

added additional target-monitoring functions such as target-power sensing and control to the wired target interface.

4.3 Client Interface

All DSN-services are accessible as RPC functions. We use XML-RPC, since it is platform independent and there exist API-libraries for a large number of programming- and scripting languages. The application or script which uses the DSN-services is target-architecture dependent and must therefore be written by the user. The script in Figure 5 demonstrates how simple automation and experimental setups can be written. In this example, a code image is uploaded to the server, the data dissemination is started, and then the targets are programmed. The DSN does not perform a version check of the targets since this is dependent on the target-application. This must therefore be a part of the client script. In the example it is assumed that the targets write out a boot-message, such as "`Version:X375`". The script uses both a time-string and a text-filter-string to query the server for the corresponding log-messages.

4.4 Performance Evaluation

The performance of our implementation on the BTnodes is mostly limited by the packet processing soft- and hardware. In fact, the microcontroller is too slow for the packet processing at full Bluetooth-speed. Incoming packets are stored in a receive buffer. If the arrival rate of packets is higher than the processing rate for a certain time, packets are dropped due to the limited capacity of the receive buffer. This affects the performance of the DSN in several ways: (a) pushed log-messages might get lost, (b) pulled log-messages might get lost, (c) commands might get lost, and (d) data-dissemination packets might get lost. The probability of these cases increases with the amount of traffic on the DSN backbone network. For many scenarios the user can control what and when data is sent on the DSN. He

user script example (Perl)

```
require RPC::XML::Client;

# creates an xml-rpc connection to the dsn-server
$serverURL='http://tec-pc-btnode.ethz.ch:8888';
$client = RPC::XML::Client->new($serverURL);

# sends the code image to the dsn-server with xml-rpc
$filename = 'experiment2.hex';
$handle = fopen($filename, 'r');
$req = RPC::XML::request->new('dsnService.uploadFile',
                              $filename,
                              RPC::XML::base64->new($handle));
$resp = $client->send_request($req);

# initiates the data-dissemination on the dsn-nodes
$type = 1; # code image is for target nodes
$resp = $client->send_request('dsnService.loadFile', $filename, $type);

# wrapped function that uses 'dsnService.getDSNstatus' to wait until
# all targets have received the code image
waitDataDisseminationComplete($filename);

# programm the targets
$flashtime = $client->send_request('dsnService.getServerTime');
$resp = $client->send_request('dsnService.targetFlash', 'all');
sleep(5);

# collects the target-versions sent as boot-message by targets
$resp = $client->send_request('dsnLog.getLog',
                 'all', 31, 18, 'Version: X', $flashtime, '');
@versions = ();
for $entry (@{$resp}){
  $entry{'LogText'} =~ m/^Version: X(\d+)/;
  push(@versions, {'node' => $entry('DSNID'), 'version' => $1});
}
```

Fig. 5. User script example in Perl

can e.g. wait for the completion of the data-dissemination bevor he starts pulling messages. In general, cases (b)-(d) are not critical, as they can be resolved with retransmission. However, in a scenario, where all nodes periodically generate log-messages that are pushed simultaneously to the server, the log-messages can not be retransmitted. So for case (a), the user wants to know the transport-capacity of the DSN, such that he can adjust the parameters of the setup.

In Figure 6, we show the measured yield of correctly received log-messages at the server. We varied the message-generation rate from 0.5 to 4 packets per node per second and the DSN size from 10 to 25 nodes. Each message carries 86 bytes payload. We left this value constant, because sending the double amount of data would result in sending two packets, which is the same as doubling the message rate. The measurements are performed on random topologies that were generated by the integrated tree topology algorithm. We observed slightly different results for different topologies, but all with the same characteristic: Starting with slow message rates, all packets are received correctly. However, there is a certain rate, from which on the yield decreases very quickly. This cut-off point is between 0.5 and 1 messages per second for 25 and 20 nodes, between

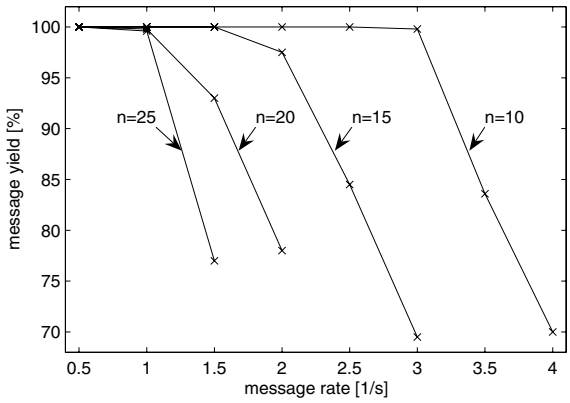

Fig. 6. Yield of correctly received log-messages that are pushed periodically to the server for different message-rates and different sizes of the DSN. For the measurement each node sent 100 log-messages with 86 bytes payload.

1.5 and 2 for 15 nodes, and between 2.5 and 3 for 10 nodes. Thus, if for this streaming scenario a developer needs all pushed log-messages, he must set the message-rate or the DSN size below this cut-off point.

5 Case-Study: Link Characterization in Buildings

Some wireless applications in buildings require highly reliable communication. A thorough understanding of radio propagation characteristics in buildings is necessary for a system design that can provide this reliability. Engineers of Siemens Building Technologies use a BTnode-based DSN system to measure and evaluate link characteristics. To this purpose, the DSN system remotely controls the target nodes and collects measurement data. In the following, the measurement setup is described and the *type* of results that can be obtained is presented. The purpose of the case study is to proof the concept of the DSN system, therefore the obtained data is not discussed in detail.

Experiments. The measurement setup consists of up to 30 target nodes, each connected to a DSN node (see Figure 7). Nodes are placed at exactly the locations required by the actual application (see Figure 8). We measure signal strength (RSSI) and frame error rates for every link between any two target nodes. Additionally, noise levels and bit error rates are evaluated. One target node is sending a fixed number of test frames while all the others are listening and recording errors by comparing the received frame to a reference-frame. Two messages are generated per second. During and after the reception of every frame, the RSSI is recorded in order to provide information about the signal and noise levels.

Fig. 7. Siemens "Blue Box" with BTnode, A80 target node and batteries. The Adapter Board acts as a connecting cable.

Fig. 8. Blue Box placed at location defined by the application

In the *detailed mode*, receiving target nodes create a message after every frame reception. Figure 9 shows the data collected by one target node in detailed mode. In *summary mode*, receiving target nodes only create a message after a complete sequence of frames has been received. Figure 10 shows the data collected by 14 nodes in summary mode. In summary mode, the amount of data is significantly reduced compared to detailed mode. This allowed us to concurrently evaluate all 30 target nodes in the test setup. On the other hand, only detailed mode (with maximally 10 nodes, see also Figure 6) allowed us to analyze the temporal properties of collected data. E.g. Figure 9 shows that frames with low signal strength do not occur at random times, but are concentrated towards the end of the experiment. Thus channel properties seem to vary over time.

The procedure described above provides data for the links between one sending target and all other targets. Test automation is used to repeat this procedure with different senders such that finally the links between any two nodes in the

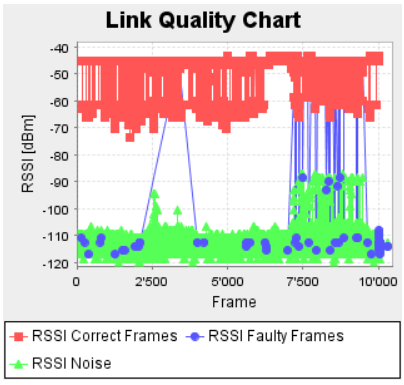

Fig. 9. Detailed mode

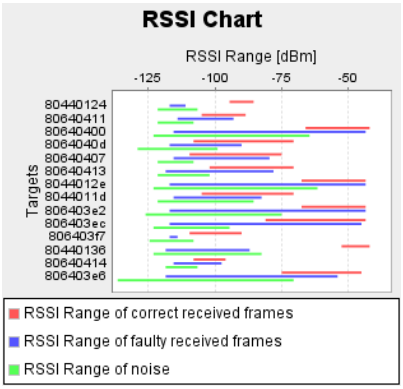

Fig. 10. Summary mode

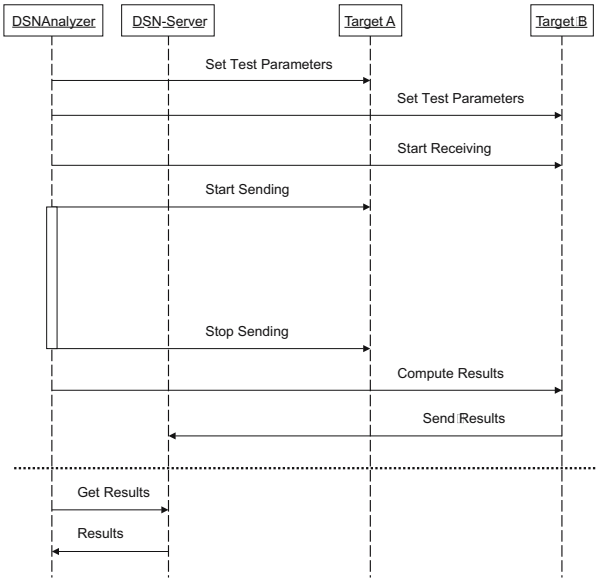

Fig. 11. The automated execution of a simple test from the *DSNAnalyzer* which is a client of the DSN

target system are evaluated. Test automation is also used to repeat tests with different WSN-system parameters like e.g. transmit power. Finally tests are executed during day- and nighttime to observe the influence of human interference. In this case a Java application acts as the user of the DSN-system (see Figure 1). The interaction of this application with the DSN system is illustrated in Figure 11.

Discussion. The BTnode based DSN system has proved to be very useful for SBT's development teams. The following advantages are most relevant to us:

- The simple interface between DSN and target nodes makes it possible to work with existing target platforms. Alternative systems require specific hard or software on the target side.
- The DSN-node/target-node pairs are completely wireless and thus can be deployed quickly and even in inaccessible locations. This is important in our use-case since we are collecting data from a wide range of buildings, some of them in use by our customers, which excludes wired installations.

6 Conclusion

We have presented the Deployment Support Network. It is a new methodology to design and test sensor-network applications in a realistic environment. Existing

solutions fail at providing at the same time both visibility and the high quality information from real deployments.

The DSN is wireless, which is the key difference to existing emulation testbeds. The deployment of DSN-node/target-node pairs is much easier than handling hundreds of meters of cables. This means that the positions of the nodes and thus the density of the network can be chosen and adjusted quickly according to the application requirements and is no longer dictated by the testbed setup.

However, using wireless ad-hoc communication instead of cabled infrastructure introduces also new limitations. One is the limited range of the DSN radio. If the range of the targets radio is larger than the one of the DSN-nodes and if a sparse deployment with maximal distances between the nodes is to be tested, additional DSN-nodes have to be inserted that act as repeaters. Another limitation is obviously the lower throughput for debugging and control information. The researcher must be aware of this and choose the rate of generated pushed messages accordingly or change to pull mode if possible. In our implementation Bluetooth provides the necessary robustness and reliability needed for the DSN. With its high spatial capacity it allows not only for large deployments, but also for very dense ones.

Compared to existing services for real-world deployments such as Deluge and Marionette, the DSN is different in the sense that the services run on a separate hardware and not on the target-nodes itself. This solution causes less interference since debugging services and the sensor-network application are clearly separated and do not share the same computing and radio resources. The resources demand of the DSN-services is different from the resources demand of the target-application which asks for different architectures. If in an application scenario the nodes only have to transmit a few bits once every 10 minutes with best effort, the developer would choose an appropriate low-power/low-bandwidth technology. Running the DSN-services over such a network is not feasible. Another approach is over-engineering. One could use more powerful nodes for the sake of better visibility and flexibility during development. Running a data-dissemination service on the target-nodes would require additional memory that is large enough for a whole code image. Expensive extra memory that is only used for development is no feasible option for industrial products.

During development and test, the DSN-nodes execute the services on dedicated optimized hardware. After that, they can be detached from the target-nodes. Since the services are implemented on the DSN they can be used for different target architectures and independently of their operating system.

Both hard- and software of our BTnode-based implementation of the DSN are publicly available at [13].

Acknowledgement

The work presented in this paper was partially supported by Siemens Building Technologies Group Switzerland and 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. Szewczyk, R., Polastre, J., Mainwaring, A., Culler, D.: Lessons from a sensor network expedition. In: Proceedings of the First European Workshop on Sensor Networks (EWSN). (2004)
2. Tolle, G., Polastre, J., Szewczyk, R., Culler, D., Turner, N., Tu, K., Burgess, S., Dawson, T., Buonadonna, P., Gay, D., Hong, W.: A macroscope in the redwoods. In: Proc. 3rd International Conference on Embedded Networked Sensor Systems (SenSys), New York, NY, USA, ACM Press (2005) 51–63
3. Dutta, P., Hui, J., Jeong, J., Kim, S., Sharp, C., Taneja, J., Tolle, G., Whitehouse, K., Culler, D.: Trio: enabling sustainable and scalable outdoor wireless sensor network deployments. In: Proc. of the fifth international conference on Information processing in sensor networks (IPSN), ACM Press, New York (2006) 407–415
4. Langendoen, K., Baggio, A., Visser, O.: Murphy loves potatoes: experiences from a pilot sensor network deployment in precision agriculture. In: Parallel and Distributed Processing Symposium20th International. (2006) 8 pp.
5. ns-2: (The network simulator - ns-2) Available via http://www.isi.edu/nsnam/ns/ (accessed July 2006).
6. Zeng, X., Bagrodia, R., Gerla, M.: Glomosim: a library for parallel simulation of large-scale wireless networks. In: Proc. Twelfth Workshop on Parallel and Distributed Simulation (PADS). (1998) 154–61
7. Levis, P., Lee, N., Welsh, M., Culler, D.: TOSSIM: Accurate and scalable simulation of entire TinyOS applications. In: Proc. of the 1st int'l conference on Embedded networked sensor systems (SenSys), ACM Press, New York (2003) 126–137
8. Werner-Allen, G., Swieskowski, P., , Welsh, M.: Motelab: A wireless sensor network testbed. In: Proceedings of the Fourth International Conference on Information Processing in Sensor Networks (IPSN'05), Special Track on Platform Tools and Design Methods for Network Embedded Sensors (SPOTS), IEEE, Piscataway, NJ (2005)
9. Elson, J., Girod, L., Estrin, D.: Emstar: development with high system visibility. Wireless Communications, IEEE [see also IEEE Personal Communications] **11** (2004) 70–77
10. Blumenthal, J., Reichenbach, F., Golatowski, F., Timmermann, D.: Controlling wireless sensor networks using senets and envisense. In: Proc. 3rd IEEE International Conference on Industrial Informatics (INDIN). (2005) 262 – 267
11. Hui, J.W., Culler, D.: The dynamic behavior of a data dissemination protocol for network programming at scale. In: SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems, New York, NY, USA, ACM Press (2004) 81–94
12. Whitehouse, K., Tolle, G., Taneja, J., Sharp, C., Kim, S., Jeong, J., Hui, J., Dutta, P., Culler, D.: Marionette: using rpc for interactive development and debugging of wireless embedded networks. In: IPSN '06: Proceedings of the fifth international conference on Information processing in sensor networks, New York, NY, USA, ACM Press (2006) 416–423

13. BTnodes: (A distributed environment for prototyping ad hoc networks) http://www.btnode.ethz.ch.
14. Beutel, J., Dyer, M., Hinz, M., Meier, L., Ringwald, M.: Next-generation prototyping of sensor networks. In: Proc. 2nd ACM Conf. Embedded Networked Sensor Systems (SenSys 2004), ACM Press, New York (2004) 291–292
15. Beutel, J., Dyer, M., Meier, L., Thiele, L.: Scalable topology control for deployment-support networks. In: Proc. 4th Int'l Conf. Information Processing in Sensor Networks (IPSN '05), IEEE, Piscataway, NJ (2005) 359–363
16. Dyer, M.: S-XTC: A signal-strength based topology control algorithm. Technical Report TIK Report Nr. 235, ETH Zurich, Switzerland (2005)