
Andreas Achtzehn, Elena Meshkova, Junaid Ansari, and Petri Mähtönen
Department of Wireless Networks, RWTH Aachen University
Kackertstrasse 9, D-52072 Aachen, Germany
Email: {aac, eme, jan, pma}@mobnets.rwth-aachen.de

Abstract—Testbeds are necessary to evaluate the performance of network applications for a wide range of deployment scenarios. Especially in wireless sensor networks, where the performance highly depends on environmental conditions, traditional computer-based simulations provide validity to only limited extent. Different approaches have been carried out to develop generic testbeds for protocol evaluation, but they suffer from automation deficiencies such as the ability for fast iterations over different topologies or parameter settings. Those features are nevertheless necessary to study the performance of a developed solution in a variety of conditions. Furthermore, the legacy data storage of other testbeds make later processing less tractable.

We present a scalable wireless sensor network testbed that provides a generic interface which can easily be incorporated into existing network research and data mining software. It uses a generic data storage format that allows ease of analysis. Higher utilization of resources by means of parallelisation of individual user tasks and multi-user support makes this testbed a valuable tool. It consists of inexpensive commercial off-the-shelf hardware and as it is open-source, requires no additional investment in legacy software.

I. INTRODUCTION

The evaluation of new protocols for wireless networks and modification of existing ones is a challenging task for the research community. There exist interdependencies among different protocols in a network stack and each protocol provides a number of adjustable parameters. The performance of protocols also depends on the network topology, which might change rapidly. Due to these obstacles, protocols are often evaluated only for a particular use case with few different topologies and pre-set static configurations.

Wireless sensor networks (WSNs) belong to a special class of communication networks. They are known to have performance characteristics that heavily depend on the deployment scenario. Relevant knowledge on their behavior is therefore difficult to derive and often WSNs tend to perform suboptimal. Since WSNs are still a comparably young field of study, no single network protocol stack composition has yet become an accepted standard, leaving sufficient space for further studies and standardization.

There exists a high demand for tools to conduct a preliminary assessment of freshly developed solutions before carrying out any deployment. Performance evaluation for WSNs has traditionally been conducted by means of building a network simulator to derive a configuration that performs desirably across various deployment scenarios. Their abstract models are usually unable to incorporate each relevant aspect of the real hardware and they reflect the real environment only rudimentarily. Therefore, the obtained results often turn out to be unrealistic and valid to only limited extent in actual deployments [1].

One way to make up for this shortcoming is the use of a real-world testbed. The data gathered from such an evaluation closely resemble the behavior of the real network. However, testbeds are difficult to build and the assessment of different setups is usually a tedious task. Many efforts (e.g. in [2], [3]) for generic WSN testbeds have been made, but their focus remains on long-term statistical evaluation instead of rapid iteration over several configurations. The parallel execution of evaluation runs needs manual administration, as a result testbed resources often lie fallow and are not optimally utilized. Rapid testbed evaluations are nevertheless necessary as protocol behavior regularly needs to be studied over a wide range of parameter settings, topologies, and in combination with various network protocol stacks.

We have implemented and tested a new WSN sensor testbed design that allows multi-user interaction and makes better use of existing resources by means of parallelisation. It promises advantages especially in the case of rapid deployment of different software setups and topologies. Special attention was paid to have a generic interface that allows it to be connected easily to existing research tools. Data gathered from the testbed is stored in a unified manner to support efficient data mining. The testbed consists of inexpensive commercial off-the-shelf components. All the software components used in the testbed are open-source and thus allow easy maintenance and modification. We have made them available on our project’s website [4] and list basic requirements in Table I.

II. SYSTEM OVERVIEW

Special requirements exist for the use of testbed systems in an academic environment. In previous protocol evaluations, we experienced huge problems in building realistic topologies due to limited space and difficulties in adjusting the power levels appropriately. On the contrary, our testbed is suitable for small-scale setups that need rapid iteration over different topologies and parameter settings.

The overall system design is depicted in Fig. 1. Central administrative entity of the testbed is a relational database. It stores all the information relevant to an evaluation cycle and
Relational database
Dispatcher
File server
Backend infrastructure

Testbed structure and components.

Fig. 1. Testbed structure and components.

<table>
<thead>
<tr>
<th>Dispatcher</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: Pentium-III class, 2 GHz</td>
</tr>
<tr>
<td>Memory: 2 GB</td>
</tr>
<tr>
<td>Storage: 4 GB HDD</td>
</tr>
<tr>
<td>Software: Linux, TinyOS 2.x, GCC, Python</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mote manager</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU: Pentium-II class, 1 GHz</td>
</tr>
<tr>
<td>Memory: 256 MB</td>
</tr>
<tr>
<td>Storage: 1 GB HDD</td>
</tr>
<tr>
<td>Software: Linux, Python</td>
</tr>
<tr>
<td>Components: USB 2.0-compliant hubs, TelosB motes</td>
</tr>
</tbody>
</table>

Table I
MINIMUM HARDWARE REQUIREMENTS.

keeps track of the available hardware. Data gathered during each of the testbed runs is also kept in this database.

Unlike previous approaches [3], [2], [5], communication between the users and the testbed is conducted solely through the database. Hence, end users do not need any knowledge of the underlying operating system or the software parts of the testbed. There exist many different libraries to connect to the database, therefore it is easy to add testbed functionalities to existing research tools. For example, in our demonstration we connect to the testbed via the ODBC interface of MATLAB. There is no need for strict time synchronization in the testbed as all the information is stored in a centralized and synchronous manner.

Evaluation runs are defined in a command and control scripting language (CCL) that we developed specifically for the testbed. The CCL format is straightforward and can easily be incorporated into research tools that want to make use of the testbed, e.g., through ODBC.

Hardware management is achieved via a dispatcher that is responsible for assigning the physical resources such as the sensor nodes to the test runs (simulations). The dispatcher assigns resources to simulations according to a predefined schedule strategy. Currently, we are using FIFO assignment upon availability, but more sophisticated scheduling strategies can also be implemented. The utilization of the testbed hardware is a critical point in long-running experiments. The immediate strength of the dispatcher is its support for multiple parallel test runs. Since channel assignments are done in a dynamic fashion, simulations do not interfere with each other. We have introduced the notion of sensor node groups, which are lists of physical sensor nodes that can be used to impersonate a certain virtual mote for a particular test run. In the presence of regular deployment failures and fulfilling the requirements for high parallelism in this scheme, it has been beneficial for successful simulation completion.

The number of devices connected to a single machine is limited by cable lengths and the underlying communication protocol. Since we intended to design the network in a highly scalable fashion, we distribute the sensor nodes among several machines, each running a lightweight instance of the mote manager. Upon startup, the mote manager registers with the database and updates its hardware information. It then continuously queries the database for new simulation tasks. If the mote manager is assigned a task, it spawns an independent simulator instance that deploys the new firmware it retrieves from the file server and starts gathering data reported back from the sensor nodes. A flowchart of the control structure is depicted in Fig. 2.

The way data is gathered from the sensor nodes eases post-processing. Information on the message format is stored with the simulation in the database. Fields of incoming messages are mapped to type-adjusted columns of matching tables, alleviating the need for computationally demanding string operations. The data storage method is efficient as it adjusts the table structure to the simulation needs. If the programmers decide to modify the message format at a later stage, they simply need to adjust the CCL simulation definition script. No recompilation or direct adjustment of the testbed software is necessary.

Rapid adjustments of parameters that are needed to quickly test different setups are conducted in two ways. TinyOS 2.0 [6], the operating system used by the sensor nodes, offers the replacement of values within already compiled binaries as well as classical macro-processing of the source as known from the C-programming language. This feature is used, for example, to adjust the node identifier in the binary to the value of the node identifier.
currently deployed. A second way of adjusting the parameters is through the use of application overlays for the sources. The Makefile or any other file that contains parameter settings is overwritten by a new version that includes the updated adjustments.

III. Demo Description

One of the strong arguments for using our testbed is its good interoperability with existing research tools. We will therefore present the connection to MATLAB, a widely used engineering tool with strong signal-processing and visualization capabilities. The MATLAB client uses the Database Toolbox and ODBC to send CCL scripts to the relational database. The CCL and data storage functionalities are implemented on the open-source database mySQL.

We provide the user with a graphical interface where she can select an arbitrary topology for the network and choose several application-specific settings. The network consists of several TelosB sensor nodes[7] connected to two laptops. We have modified the operating system to support virtual topologies by selectively discarding packets at MAC layer level.

After successful completion of the test run, results are returned to MATLAB through the database. A simple visualization tool will show how MATLAB benefits from the convenient access to the data. In another demonstration we have evaluated time-critical scenarios where post-processing was shifted to the database by means of stored procedures [8].

Two off-the-shelf laptops will represent two independent deployment sites, each equipped with a number of sensor nodes. One machine, besides running a mote manager instance, also runs the dispatcher, a NFS and Samba file server for file distribution and the mySQL database the client connects to.

During the long-term protocol evaluation sessions we experienced that sensor nodes became unusable. They had a tendency to stop reprogramming after a certain number of test runs. To overcome this obstacle, we integrated some resilience against node losses by defining backup sensor nodes in the simulation. Besides adding to the parallelisation of simulations, this also solved the shortcomings of the sensor hardware.

IV. Further Research Directions

In later versions of the testbed we will add runtime features to the simulations, where the user can send specific commands to the sensor nodes at a particular event.

A library for connecting MATLAB to the testbed as well as a legacy C# interface is already available. We would like to extend our repository further to support hybrid simulator-testbeds. More sophisticated scheduling will offer a higher number of parallel simulation runs and thereby shorten the waiting times (e.g. through [9]).

Acknowledgements

We would like to thank DFG and RWTH Aachen University for a partial financial support through the UMIC excellence cluster.

References