DOI:10.1145/2063176.2063198

Virtual testbeds model them by seamlessly integrating physical, simulated, and emulated sensor nodes and radios in real time.

BY GEOFF COULSON, BARRY PORTER, IOANNIS CHATZIGIANNAKIS, CHRISTOS KONINIS, STEFAN FISCHER, DENNIS PFISTERER, DANIEL BIMSCHAS, TORSTEN BRAUN, PHILIPP HURNI, MARKUS ANWANDER, GERALD WAGENKNECHT, SÁNDOR P. FEKETE, ALEXANDER KRÖLLER, AND TOBIAS BAUMGARTNER

Flexible Experimentation in Wireless Sensor Networks

WIRELESS SENSOR NETWORKS (WSNs) play a key role in the emerging "real-world Internet," with several large-scale WSNs being deployed; see, for example, Bernat[2] and Dudek et al.[9] However, WSN development is inherently complex, involving hardware design, embedded and distributed programming, heterogeneity, scale, and unpredictable environmental changes. Addressing this complexity, testbed-based experimentation (recommended by Weiser[32]) is increasingly the norm for developing and optimizing WSN systems in a controllable environment prior to deployment.

The WSN research community has historically relied on three main approaches to testbed-based experimentation: physical, simulation, and emulation. However, researchers appreciate that each involves significant drawbacks when used in isolation (see the sidebar "Physical Testbeds vs. Simulation vs. Emulation"). Therefore, they seek to combine all three to enable a more complete evaluation of the system being developed. Unfortunately, each approach requires different coding styles and tools, forcing researchers to expend significant effort reimplementing their systems for different tools/platforms/approaches. As a remedy, techniques have been developed to reduce the transitioning effort among the three approaches, but further work is needed to address the emerging requirement for more flexible experimental facilities.

Our work abstracts the concept of testbeds to yield virtual testbeds (VTBs) programmed similarly regardless of whether their underlying realization is physical, simulated, or emulated. VTBs are private, custom-designed, per-experiment, virtualized testbed instances that enable developers to seamlessly combine and/or interchange physical elements, including sensor nodes and radios, with simulations and emulations of these elements.

We are developing a reference implementation of the VTB abstraction on top of a large-scale federated physical testbed infrastructure (see Figure 1), augmenting the inherent flexibility of the VTB abstraction in terms of scal-

» key insights

- **Physical, emulated, and simulated elements of wireless sensor networks can be seamlessly mixed to gain massive-scale "virtual testbeds" with desired trade-offs involving fidelity, repeatability, and network size.**
- **The relative speed difference between Internet links and low-power wireless radios allows sensor nodes at physically distant testbed sites to interconnect (virtually) over the Internet.**
- **The WISEBED approach of recursively and hierarchically applying a common Web Services API to physical, simulated, and federated or hybrid testbeds facilitates transparent composition of virtual testbeds that can be accessed and controlled by common tools.**

ILLUSTRATION BY JASON COOK

ability (VTBs combine physical nodes from different federated sites) and heterogeneity (VTBs incorporate a variety of equipment types).

Research Challenges

In this article, rather than survey the many purely physical WSN testbeds (such as Chun et al.[4]), we focus on work that seeks to "virtualize" WSN testbeds to some extent or federate physical testbeds; as far as we know, no previous work has sought to do both. Note that the term "virtualization" is commonly used in a variety of contexts, as in "slicing" in systems like PlanetLab and the simultaneous running of multiple operating systems on a single processor, as in VMware. We reserve the term exclusively for systems that mix physical WSN testbeds with elements of simulation and emulation.

In terms of virtualization, several research efforts have sought to bridge the gap between physical execution and simulation. For example, Li et al.[23] and TOSSIM[22] used TinyOS component graphs to generate discrete-event simulations that can be augmented with a small number of attached physical nodes (such as three reported in Li et al.[23]). Node software in Girod et al.[15] and Park et al.[27] was executed inside a simulator, but the simulated nodes communicated through physical radios. And in Österlind et al.[26] and Wen et al.[33] node software was executed iteratively on physical devices and in simulation, with various arbitration and timing schedules applied. All these projects involved a useful mix of physical and simulated elements, but that mix was fixed; for example, TOSSIM cannot support simulated nodes with real radios, and Wen's cannot mix physical nodes with simulated nodes. In contrast, we strive for generality, so physical, simulated, and emulated elements can all be combined as desired.

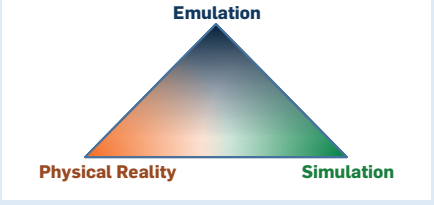
Another feature of these efforts is that programmers must significantly adapt their code to work with the given mix of physical and simulated elements. In contrast, we strive for transparency so the VTB looks as much as possible like a purely physical testbed, with the goal being to minimize the overhead of transferring an experimen-

Physical Testbeds vs. Simulation vs. Emulation

Here we outline the three main approaches to providing WSN testbeds. Our notion of VTBs allows users to arbitrarily combine elements of all three (see the figure here).

Physical testbeds (such as those described in Chun et al.,[4] Dutta et al.,[11] Ertin et al.,[13] Handziski et al.,[18] Tutornet,[12] and Werner-Allen et al.[34]) excel at high-fidelity evaluation of mature WSN designs, as well as detailed planning for real-world deployments. However, physical testbeds for WSN systems tend to be small in scale, expensive to maintain, and time-consuming to set up. They also typically lack flexibility, often offering only a single, fixed, connectivity topology and limited heterogeneity (such as only a single type of sensor node, radio, operating system, or programming language). They also tend to be limited in their programmability at lower levels of the system; for example, many use fixed operating systems and networking stacks. They are also often unsuited to experimentation scenarios requiring repeatability of experiments, as many relevant operating parameters are beyond user control (such as local radio interference due to infrastructure and other experiments).

Three-cornered testbed design space for WSN experimentation.



Simulation (such as described in Fekete et al.[14] and Levis et al.[22]) is useful for quickly trying out new ideas and for investigating the behavior of new protocols and mechanisms in varied topologies at large scale and in a repeatable manner. The most notable drawback is a lack of fidelity, often making it unrealistic to simulate fully at the instruction-execution level and with high-fidelity radio or power-consumption characteristics. While such limitations are not necessarily problematic in traditional network environments, where simulators (such as *NS*-2[21]) are prominent, they represent significant drawbacks in WSN environments where resource scarcity and incidental physical characteristics are of the essence. Simulation alone is therefore of limited use in planning for real-world WSN systems and deployments.

Emulation (such as described in Girod et al.[16] and Wu et al.[36]) is situated between physical reality and simulation. Whereas simulation abstractly models target systems, emulation duplicates the functionality of one system in terms of another system and is therefore capable of much greater fidelity than simulation while potentially offering greater flexibility than a purely physical testbed. Emulation is a much less exploited approach in the WSN testbed context despite much potential; for example, emulation in the form of network overlay technology could be used to support different inter-node connectivity patterns in a physical testbed. Alternatively, a battery-based power supply on a physical node could be emulated by interposing a mains electricity-powered hardware module degrading power over time.

tal system to real-world deployment. Finally, all these projects were small in scale and therefore of limited application in the development of large-scale WSN deployments.

Turning now to work that applies the federation concept to physical WSN testbeds,[a] Ohio State University[17] combined the infrastructure and software of the Kansei testbed with the GENI facility to provide a unified solution; the Senslab project[30] aimed to unify four discrete heterogeneous testbeds into a single testbed of 1,000 nodes; and Cooperating Objects Network Of Excellence (CONET)[6] employed a REST API to provide uniform Web-based access to different testbeds. These efforts highlight the promise of testbed federation in pursuit of scalability, but none support integration of simulated and emulated elements or deliver the transparency of federation offered by our VTB abstraction.

VTB Abstraction

VTBs provide the abstraction of a user-designed private WSN testbed in which some testbed elements are physical, some are simulated, and others are emulated. Users design their VTB in such a way that its mix of physicality, simulation, and emulation is appropriate to their goals. They then instantiate their VTB, deploy their software onto it, and observe the outputs and behavior of their experimental systems as if they were running on a dedicated physical testbed.

VTB elements can include:

Sensor modality. Examples are temperature sensors and pollutant sensors;

Sensor data. What the sensors observe (such as the current temperature and pollutant levels);

Nodes. CPU+memory+radio devices to which sensor modalities are attached;

Node power. Power-supply characteristics (such as battery or solar) and remaining energy of nodes;

Node connectivity. Nodes in broadcast range of each node and the volatility of their relationships; and

Node mobility. The movement patterns nodes are subjected to if, for example, they are mounted on public-transport vehicles.

Users augment VTB physical nodes by attaching emulated sensor modalities not supported by hardware. Alternatively, they can foster experiment repeatability by feeding emulated (scripted) sensor data to their nodes (physical, simulated, or emulated). They can also increase the scale of their VTB by selecting physical nodes from different sites in the underlying federation, combining simulated nodes with a core set of physical nodes, and even multiplexing multiple emulated nodes onto a single physical node. They can also employ emulated node power to facilitate repeatable battery-life investigations.

The emulation of node connectivity is the biggest single source of VTB flexibility, realized through virtual links offering emulated real-time connectivity between pairs of nodes according to the characteristics of different types of radio hardware. Using virtual links, testbed users are able to explore different connectivity patterns atop a set of physical nodes with fixed physical connectivity. For example, they might add emulated connectivity between pairs of nodes with no physical connectivity, including across sites and between any combination of physical, emulated, and simulated nodes. They can also remove unwanted physical connectivity to, say, build an emulated ring topology on top of an underlying physical mesh. Taking this to extremes, they can even

a The concept of a federated testbed is not new; projects (such as PlanetLab and Panlab) have long applied similar ideas, though not in a WSN context.

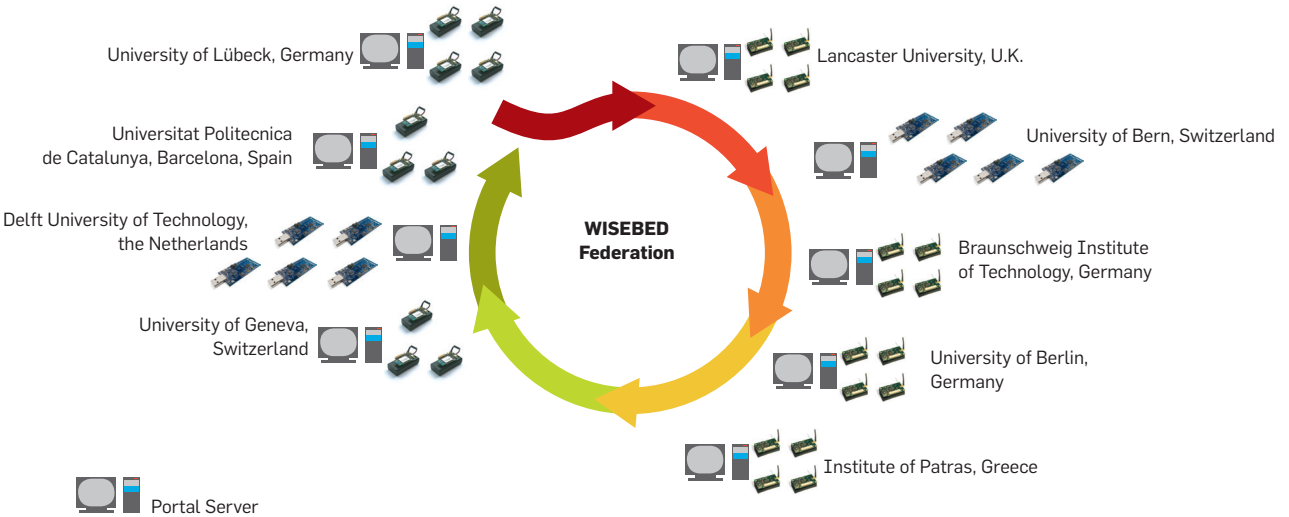


Figure. 1. WISEBED physical testbed environment. In this federation of physical WSN testbed sites (nine today), each differs in its choice of hardware, software, and physical layout.

emulate node mobility (of physical, simulated, and emulated nodes) by dynamically changing the emulated connectivity between nodes according to a time-based script.[3]

From the user's point of view, a VTB would appear as an instance of a Web services interface called iWSN supporting a comprehensive set of testbed-related operations, including loading experimental code and collecting results. Users typically employ a GUI-based front-end to mediate access to their VTB rather than interact with their iWSN instance directly.

The user's view of the entire VTB-based WSN experimentation process is captured in a series of (potentially iterated) steps (see Figure 2):

Experimental software development. Users employ our software development kit (SDK, http://www.wisebed.eu) to develop and synthesize software that can be deployed in a VTB;

VTB specification. They specify a custom VTB that meets the requirements of their experiment in terms of its physical, simulated, and emulated elements;

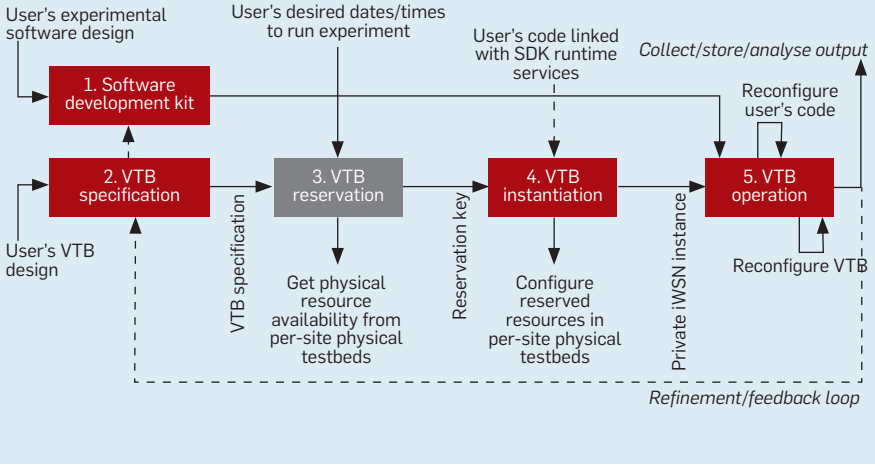
VTB reservation. They contact our reservation system to request their custom-specified VTB be instantiated at a particular time for a particular duration. The reservation system reserves a set of underlying resources able to support the VTB and returns a "reservation key" that uniquely identifies the reservation and serves as a promise that a VTB (as specified) will be made available at the requested time;

VTB instantiation. When the prespecified reservation time arrives, users ask the instantiation system to redeem their reservation keys. In response, the instantiation system builds the user-specified VTB using the previously reserved underlying resources, loads the user's software onto the nodes of the VTB (along with any SDK runtime services), and returns a dedicated iWSN instance; and

VTB operation. Users run their experimental software on their newly instantiated VTB under the control of a GUI that mediates access to the VTB in real time.

To show the applicability of this approach, consider an experimental scenario in which experimenters want to select a broadcast protocol for use in a large-scale WSN deployment.[2] This experimentation requires a testbed with three properties: scale and topology

Figure 2. User view of VTB-based WSN system development.



close to that of the target deployment environment; significant use of physical nodes and radios (important, as the forwarding decisions made by broadcast protocols typically rely on physical features like RSSI thresholds difficult to simulate with high confidence); and repeatability. This combination would be extremely challenging for most existing testbed environments.

Using it, experimenters might begin by designing a simple VTB underpinned by simulated nodes and connectivity. Such a VTB could be executed on a desktop PC backed by a suitable simulator engine (such as Shawn[14]). Running their experimental code would give an initial feel for the general behavior of the candidate protocols and likely critical areas. They might then create a second, more sophisticated VTB underpinned by federated physical testbeds augmented with virtual links to yield a configuration with scale and topology close to that of the target deployment environment. The same experimental code can then be executed on this higher-fidelity VTB, using emulated sensor input to drive the experiments in a repeatable manner. It could be argued that using virtual links here might compromise fidelity somewhat, but the flexibility of the approach makes running a range of what-if experiments straightforward with different virtual-link configurations to build confidence in the stability and fidelity of the results.

Software Infrastructure

Here, we expand on key aspects of the underlying functionality required to support the VTB abstraction in our federated environment. We omit consideration of VTB reservation (less central to the structure and performance of VTBs and implemented like a number of existing purely physical testbeds), as well as discussion of our security provision (to ensure users are properly authenticated and authorized to use specific resources at specific times); for more, see the WISEBED Project.[35]

The SDK includes tools for design, implementation, synthesis, and deployment of the experimental software that will run atop a user's VTB. It is based on a lightweight software component model[7] offering a simple-to-use modular development process. The component model is runtime reconfigurable, so different components can be deployed as the experiment proceeds, facilitating exploration of what-if scenarios.

The SDK is also the primary means by which we address the key research challenge of transparency, whereby users are protected from having to reimplement their code when moving from physical to simulated to emulated and mixed testbed environments. This transparency is achieved through an abstraction layer that selectively exposes low-level APIs (such as drivers) so both application-level software and systems-oriented software can run unchanged atop physical, simulated, or emulated nodes; our Lorien operating system[28] is a good example of systems-oriented software that runs over our low-level APIs.

Besides its abstraction layer, the SDK provides a set of runtime services selectively configured into the user's software build by the instantiation system, comprising these components:

Generic node management. Supports implementation of the generic node-management operations supported by the iWSN interface, including, for example, functions that support the pinging of the node or respond to requests for current battery status;

Sensor emulator. "Pretends" to be sensors attached to the node and supply a stream of emulated sensor data (generated internally or driven by an external source); and

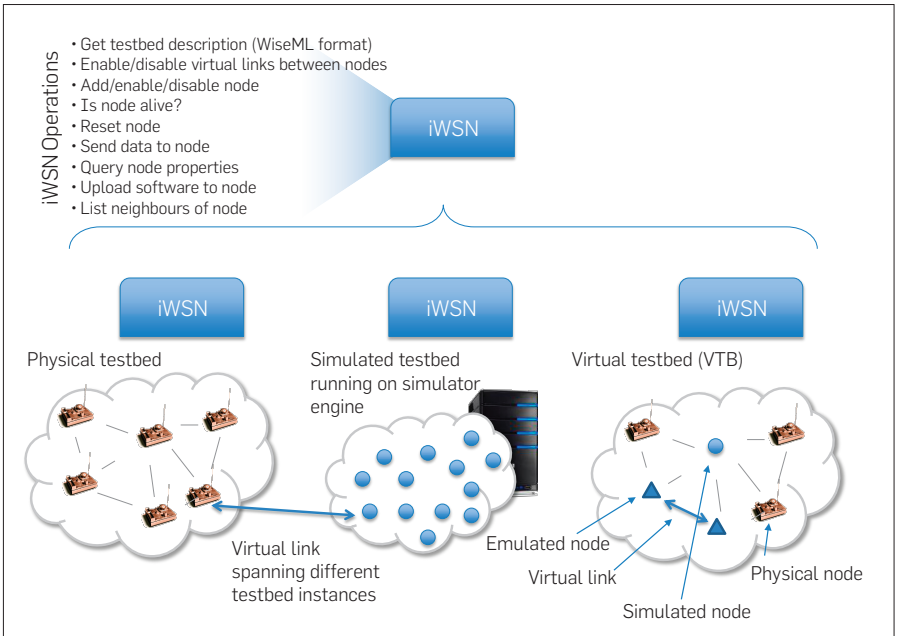
Radio stacking framework. Inserts pseudo-network-device-driver components in front of the bottom-level radio device driver, a key element of our "virtual links" implementation.

The SDK currently operates across a range of platforms, including Contiki,[10] iSense,[5] Lorien,[28] TinyOS,[19] and the Shawn WSN simulator.

VTBs are specified through an XML schema called WiseML,[35] a multi-purpose format also used to encapsulate experimental output data. For VTB specification, WiseML supports specification of the following system elements:

Node-related information. Included is information on node type, whether the node should be physical, simulated, or emulated, the coordinates of the node in a global 3D space, and the sensor modalities supported; and

Connectivity information. Helps define potential connectivity between pairs of nodes in terms of virtual links; specifying a virtual link between two



The interface provides operations for acquiring a full WiseML specification of the associated testbed, managing virtual links, adding new nodes, enabling/disabling/resetting nodes, checking node liveness, sending commands or uploading software to a node, and querying node properties. Virtual-link management operations include virtual links spanning testbed instances, the basis of our approach to testbed federation.

Figure 3. The iWSN interface, a Web services interface providing uniform management access to all testbed types, whether VTBs, the underlying physical testbeds of the federated physical environment, or purely simulation-based testbed instances (simulator engines).

nodes in the VTB enables one-hop unidirectional communication between these nodes.[b]

Users typically generate WiseML-based VTB specifications through a GUI-based tool. Such tools are independent of the rest of the software infrastructure, and it is possible to imagine a range of them with varying degrees of sophistication. A typical tool might display graphical representations of the available physical testbeds laid out in space, with each testbed represented as a graph and one-hop reachability by directed edges. With these graphs, users could select a subset of the available physical nodes to be allocated to their VTB. They could also add simulated nodes from a drop-down menu and selectively add and delete edges between arbitrary nodes to modify the topology of their VTB.

The VTB instantiation system gets involved when users redeem a reservation key issued by the reservation system. The system gathers the physical resources derived from the reservation stage, configuring/connecting them according to their original VTB specification. The instantiation system (and operation step) relies on the iWSN interface providing uniform access to all testbed realizations and supporting the federation and hierarchical composition of testbeds (see Figure 3).

The instantiation process involves creating an empty VTB, then using its iWSN interface to populate it with the required nodes, connectivity, and other parameters. For example, where a user wants a VTB to include physical nodes from multiple sites, the instantiation system creates an overarching VTB into which it inserts selected nodes from the various sites; it then sets up virtual links between nodes to create the required federated configuration.

A similar approach to federating underlying testbeds is applied in experiments involving simulated nodes whereby the instantiation system executes one or more simulator engines. These engines are simulator processes instantiated on a suitably located and resourced server machine; we primarily use the Shawn WSN simulator mentioned earlier. As in physical sites, the simulator engine is abstracted as a testbed in its own right, exporting an iWSN instance that can be federated and "virtually linked" with testbed elements in other VTB instances to build up the user's required configuration.

The software behind the iWSN interface's virtual link operations uses the SDK's radio-stacking framework discussed earlier to deploy "pseudo" radio drivers that appear to software on the node as "real" radio drivers. To disable physical connectivity between two nodes, the pseudo-driver on one node simply drops packets originating from the other. To establish a virtual link where there is no physical connectivity, the pseudo-driver on the sending node transparently diverts (selected) outgoing packets to virtual-link software running on a server, and the pseudo-driver on the receiving node inserts incoming packets arriving from the virtual-link software.

Note this virtual link creation delivers only the basic "plumbing." On top of it, the instantiation system inserts, where required, an additional server process that models a specific radio channel. For this purpose, we are experimenting with the OMNeT++ simulator, finding it fast enough to handle virtual link packets in real time.[31] For a more detailed discussion of the implementation of virtual links see Baumgartner et al.[1]

On completion of the instantiation step, users are given a dedicated iWSN instance through which they interact with their newly instantiated VTB as if it were a private physical testbed; for example, they can then call iWSN services to deploy, monitor execution, and dynamically reconfigure their code. These operations may be carried out programmatically through the iWSN interface directly via Web services calls but are more commonly carried out through a program interacting with the iWSN interface on behalf of the user. We refer to such programs as controllers; like VTB-specification tools, they are distinct architectural elements decoupled from the rest of

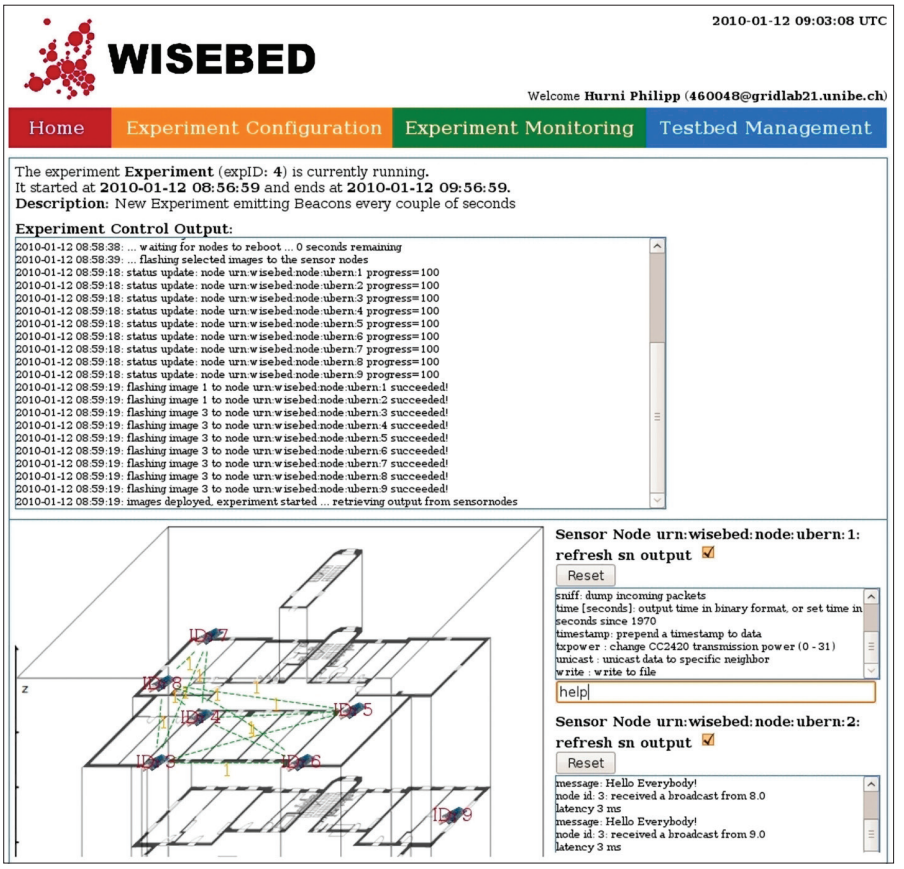


Figure 4. TARWIS controller graphical user interface.

b Two virtual links are needed to represent mutual connectivity between two nodes, allowing users to model situations in which a node A can send to a node B, but B cannot send to A. The WiseML specification of a virtual link includes the unique identifiers of the nodes that will participate in the link, as well as the link's characteristics in terms of its "link quality indicators," packet error rate, and other parameters.

the system and can evolve or be replaced independently.

We have implemented two: The first is a scriptable command-line controller that enables batch-mode control of experiments and is particularly useful in experiments that must be executed repeatedly with varied parameters. The second is an interactive GUI-based system called Testbed Management Architecture for Wireless Sensor (TARWIS) networks[20] that offers a Web-based user interface; it displays the attached VTB and allows users to interactively issue iWSN commands to individual nodes (such as to reset, flash, reprogram, and reboot nodes and to send commands) and store their output. Individual sensor-node output can be displayed in individual windows, with connectivity/topology information displayed graphically (see Figure 4). Experiment results are stored in a WiseML file for post-experiment archiving and analysis. Note, due to the common use of the iWSN interface, TARWIS, or indeed any controller, can also be used to interact in exactly the same way with physical sites and simulator engines, as well as with VTBs.

The implementation just outlined represents a substantial proof-of-concept demonstration of the viability of the VTB abstraction, though there is room for further development, especially in terms of VTB-specification tools; other interesting areas (such as node mobility and emulated sensors) have also yet to be explored in detail. More generally, the implementation highlights and validates the modular and pluggable nature of the software infrastructure; alternative implementations of many of the architectural elements (such as specification, reservation, instantiation, and operating systems) can be provided independently. We see this modular design as crucial in testbed environments that aspire to grow and co-evolve with their user base.

Table 2. Time for receiving packet bursts sent from physical to simulated nodes.

Number of pairs	Rx time (seconds)
1	0.109
5	0.359
10	0.452
15	0.710
20	0.881
30	1.001

Table 1. Message latencies in virtual links under various scenarios; results are averaged over more than 1,000 messages in each case.

Hardware/OS	Physical Radio	Intra-site	Inter-site
ScatterWeb/TinyOS	75ms	5ms	53ms
TelosB/Contiki	40.2ms	5ms	53ms
iSense/iSense	7ms	5ms	53ms

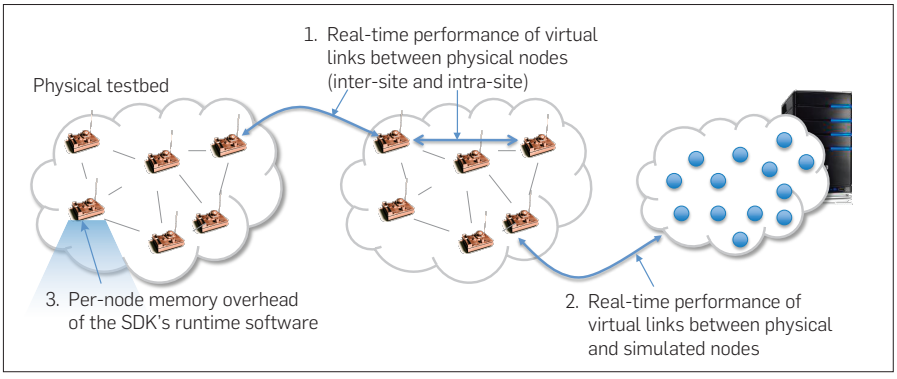


Figure 5. Overall evaluation strategy.

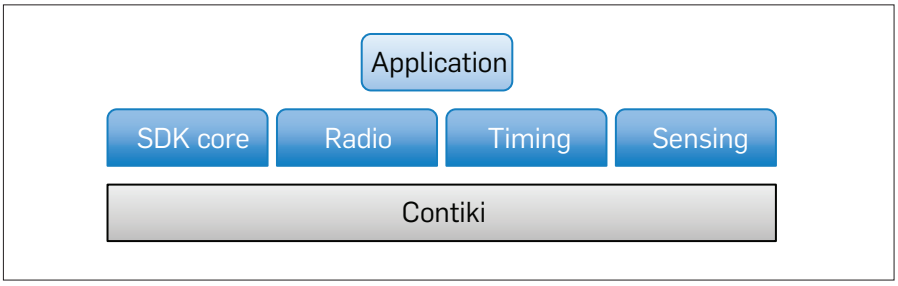


Figure 6. Node software configuration used in an SDK evaluation.

Experimental Validation

The central issue for the viability of the VTB concept is the extent to which physical, simulated, and emulated nodes are able to work and communicate seamlessly with one another in real time. Seamless operation implies an experimental validation of VTBs (see Figure 5) should focus on three areas:

Real-time performance of virtual links between physical nodes. Is the connectivity offered by virtual links a suitable basis for emulating the characteristics of real physical radio-based connectivity? Is it possible to support a virtual link capable of being indistinguishable from a real link in terms of, say, message-transmission rates and latencies?;

Real-time performance of virtual links between physical and simulated nodes. Can nodes running in a simulator server "keep pace" with physical and emulated nodes elsewhere in the VTB in terms of sourcing and sinking messages at a rate and latency, whereby the system as a whole executes seamlessly?; and

Per-node resource overhead of the SDK's runtime software. The VTB abstraction imposes this overhead on sensor nodes as an experiment runs on the VTB. The overhead is mainly subsumed within the SDK's abstraction layer and runtime services, manifesting mainly in terms of memory occupancy.

Performance of virtual links between physical nodes. The evaluation strategy here is to establish that virtual links perform at least as well as real radio links, giving users the potential to

use any spare capacity to model real radio characteristics (such as using a network simulator like OMNeT++ mentioned earlier).

Specifically, we measure the time required to transmit a message between two physical nodes in three cases:

- Baseline, via one-hop physical radio link;
- Intra-site, via a virtual link that traverses from one node to a PC via direct universal asynchronous receiver/transmitter (UART) connection, then to a second PC via Gigabit Ethernet, and finally to the second node via UART again; and
- Inter-site, via a virtual link, as with intra-site, except it uses the Internet instead of Gigabit Ethernet, interconnecting a site in Lübeck, Germany, with another site in Lancaster, U.K.

We examine these three cases on three different hardware platforms: ScatterWeb, TelosB, and iSense (see Table 1). In both the intra-site and inter-site cases the back-end message transport systems represent the dominant cost, so the results across all platforms are similar. The main observation is that intra-site is significantly faster than using physical hardware radios for all three platforms considered, demonstrating the viability of emulating sensor-node communication in VTBs. Due to the raw speed advantage a generous amount of spare capacity is available for the modeling-link characteristics and radio-contention scenarios prior to packet delivery. Inter-site offers less spare capacity; the relatively slow times are attributable to our current VTB implementation's use of Simple Object Access Protocol (SOAP) encapsulation for messages sent across the Internet. Even so, at least in the case of ScatterWeb/TinyOS, the VTB provides significant spare capacity in which to model radio characteristics. In the future, we expect more efficient transport protocols to provide considerably larger spare capacity; raw ping times between the two sites in question suggest approximately 23ms of one-way latency on average.

Virtual links between physical and simulated nodes. In this experiment we use a physical testbed of 30 iSense nodes connected via Ethernet to an Intel dual-core 2.5Ghz PC with 3GB of RAM running the Shawn WSN simulator engine. We execute scenarios in which varying numbers of physical nodes connect one-to-one to "partnered" simulated nodes in the Shawn environment. We then measure how well the simulated nodes "keep up" with the physical nodes sending packets at realistic rates. To model a realistically demanding scenario, we employ packet bursts of 10B x 20B packets with 5ms inter-packet gaps. These bursts represent a far higher data rate than would be encountered in a typical WSN-based experimental scenario.

Given this setup, we obtained the results in Table 2 for different numbers (one to 30) of active sender-receiver pairs. The "Rx time" figure, or time to receive all packets, for the single-pair case shows a 10-packet burst is received in 0.109 seconds. Given that most realistic low-power WSN application scenarios employ sample periods on the order of seconds, this figure is perfectly acceptable. Furthermore, as the number of pairs increases, the overhead scales very well, so despite the inherent serialization imposed by the simulator (any event-based simulator would be the same), the receiving rate remains well within the operating range of typical experimental scenarios.

Per-node memory overhead. The per-node memory overhead of the SDK's runtime software is incurred primarily by its common abstraction layer. To evaluate memory overheads, we instrument a typical sensor network application in which nodes sample data every five seconds, sending it to a parent node and also forwarding data received from child nodes. We implement the application twice: once using only native OS facilities, representing the baseline, and once using the SDK along with the chosen operating system.[c] As outlined in Figure 6, the SDK-based implementation uses abstraction-layer APIs for radio communication, including the radio-stacking framework, sensing data input, and timing. Both implementations run on TelosB motes with 48KB of program memory and 10KB of RAM.

c We used Contiki[10] for the measurements reported here but found similar results through other operating systems.

Within these categories of memory the SDK's overhead is measured as follows:

Program memory overhead. From the machine code implementing the common abstraction layer; in our test application, it consumes 5,331B (10.8% of total program memory) compared to 24,750B by Contiki and 950B by the application itself; and

RAM Overhead. The amount associated with the SDK in the example application is 356B (3.6% of total RAM) compared to 1,200B by Contiki and 368B by the application itself.

These figures show that achieving the convenience of a common abstraction layer over all platforms and VTB operation modes—physical, simulated, and emulated—comes at very reasonable cost; the overheads are sufficiently low as to be a minimal impediment on current hardware. Moreover, memory costs are incurred on a pay-for-what-you-use basis depending on which parts of the common abstraction layer are employed in a given experiment, per our component-based approach.

Conclusion

We have proposed and motivated the VTB abstraction, offering comprehensive fidelity/flexibility trade-offs in WSN testbed-based experimentation, as well as the benefits of a federated physical testbed environment.

The VTB implementation is operational and used by an increasing number of experimenters worldwide. We are also developing additional extensions to "harden" it into a more widely available public facility. We are also exploring more experimental scenarios, particularly those involving virtual mobility.[3]

The underlying federated platform is growing beyond the nine original sites established by the WISEBED project. In particular, a "smart city" site in Santander, Spain, developed by the SmartSantander project (EU FP7 project ICT-257992, www.smartsantander.eu), is being added to the federation and will make available physical sensors already deployed in a real-world environment. In addition, universities in Brazil and Argentina are in the process of becoming WISEBED sites and will enable us to further broaden the

federation approach, push scalability, and increase heterogeneity.

Full documentation of all systems and interfaces discussed here, along with the source code of the implementations, is available at http://www.wisebed.eu.

Acknowledgments

This work was partially supported by the European Union under contract IST-2008-224460 (WISEBED). We would like to acknowledge all those who contributed to the design and implementation of the VTB concept in particular and to WISEBED in general.

References

1. Baumgartner, T., Chatzigiannakis, I., Danckwardt, M., Koninis, C., Kröller, A., Mylonas, G., Pfisterer, D., and Porter, B. Virtualising testbeds to support large-scale reconfigurable experimental facilities. In *Proceedings of the Seventh European Conference on Wireless Sensor Networks, Lecture Notes in Computer Sciences, vol. 5970*, J. Silva, B. Krishnamachari, and F. Boavida, Eds. (Coimbra, Portugal, Feb. 17–19). Springer, Berlin/Heidelberg, 2010, 210–223.
2. Bernat, J. SmartSantander: The path towards the smart city vision. In *Proceedings of the First ETSI M2M European Telecommunications Standards Institute Machine to Machine Workshop* (Sophia Antipolis, France, Oct. 19–20, 2010).
3. Braun, T., Coulson, G., and Staub, T. Towards virtual mobility support in a federated testbed for wireless sensor networks. In *Proceedings of the Sixth Workshop on Wireless and Mobile Ad Hoc Networks* (Kiel, Germany, Mar. 10–11, 2011).
4. Chun, B.N., Buonadonna, P., AuYoung, A., Ng, C., Parkes, D.C., Shneidman, J., Snoeren, A.C., and Vahdat, A. Mirage: A microeconomic resource allocation system for sensornet testbeds. In *Proceedings of the Second IEEE Workshop on Embedded Networked Sensors* (Sydney, May 30–31). IEEE Computer Society Press, Washington, D.C., 2005, 19–28.
5. Coalesenses GmbH. iSense. Lübeck, Germany, 2006; http://www.coalesenses.com
6. Cooperating Objects Network of Excellence. *Common Abstractions for Testbed Federation*. FP7-ICT-2007-2-224053, 2011; http://www.cooperating-objects.eu/fileadmin/research/testbed-federation/CONET_D33.pdf
7. Coulson, G., Blair, G., Grace, P., Taiani, F., Joolia, A., K. Lee, K., Ueyama, J., and Sivaharan, T. A generic component model for building systems software. *ACM Transactions on Computer Systems 26*, 1 (Mar. 2008), 1–42.
8. Crossbow. *MicaZ*. Milpitas, CA, 2007; http://www.xbow.com/Products/productdetails.aspx?sid=164
9. Dudek, D., Haas, C., Kuntz, A., Zitterbart, M., Krüger, D., Rothenpieler, P., Pfisterer, D., and Fischer, S. A wireless sensor network for border surveillance (demo). In *Proceedings of the Seventh ACM Conference on Embedded Networked Sensor Systems* (Berkeley, CA, Nov. 4–6). ACM Press, New York, 2009, 303–304.
10. Dunkels, A., Gronvall, B., and Voigt, T. Contiki: A lightweight and flexible operating system for tiny networked sensors. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks* (Tampa, FL, Nov. 16–18). IEEE Computer Society Press, Washington, D.C., 2004, 455–462.
11. Dutta, P., Hui, J., Jeong, J., Kim, S., Sharp, C., Taneja, J., Tolle, G., Whitehouse, K., and Culler, D. Trio: Enabling sustainable and scalable outdoor wireless sensor network deployments. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks* (Nashville, Apr. 19–21). ACM Press, New York, 2006, 407–415.
12. Embedded Networks Laboratory. *Tutornet Project: A Tiered Wireless Sensor Network Testbed*. University of Southern California, Los Angeles, 2009; http://enl.usc.edu/projects/tutornet/
13. Ertin, E., Arora, A., Ramnath, R., Naik, V., Bapat, S., Kulathumani, V., Sridharan, M., Zhang, H., Cao, H., and Nesterenko, M. Kansei: A testbed for sensing at scale. In *Proceedings of the Fifth International Conference on Information Processing in Sensor Networks* (Nashville, Apr. 19–21). ACM Press, New York, 2006, 399–406.
14. Fekete, S. P., Kröller, A., Fischer, S., and Pfisterer, D. Shawn: The fast, highly customizable sensor network simulator. In *Proceedings of the Fourth International Conference on Networked Sensing Systems* (Braunschweig, Germany, June 6–8). IEEE Computer Society Press, Washington, D.C., 2007.
15. Girod, L., Ramanathan, N., Elson, J., Stathopoulos, T., Lukac, M., and Estrin, D. Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks. *ACM Transactions on Sensor Networks 3*, 3 (Aug. 2007).
16. Girod, L., Stathopoulos, T., Ramanathan, N., Elson, J., Estrin, D., Osterweil, E., and Schoellhammer, T. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of the Second International Conference on Embedded Networked Sensor Systems* (Baltimore, Nov. 3–5). ACM Press, New York, 2004, 201–213.
17. Global Environment for Network Innovations. *GENI-fying and Federating Autonomous Kansei Wireless Sensor Networks Technical Report*. Ohio State University, Columbus, OH; http://groups.geni.net/geni/wiki/KanseiSensorNet
18. Handziski, V., Kopke, A., Willig, A., and Wolisz, A. *Twist: A Scalable and Reconfigurable Wireless Sensor Network Testbed for Indoor Deployments*, Technical Report TKN-05-008. Technische Universität Berlin, Berlin, Germany, Nov. 2005.
19. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., and Pister, K. System architecture directions for networked sensors. *SIGOPS Operating Systems Review 34*, 5 (Nov. 2000), 93–104.
20. Hurni, P., Wagenknecht, G., Anwander, M., and Braun, T. A testbed management architecture for wireless sensor network testbeds (TARWIS). In *Proceedings of the Seventh European Conference on Wireless Sensor Networks, Posters, and Demos* (Coimbra, Portugal, Feb. 17–19). Springer-Verlag, Berlin/Heidelberg, 2010.
21. Information Sciences Institute. *The Network Simulator NS-2*, 2007; University of Southern California, Los Angeles; http://www.isi.edu/nsnam/ns/
22. Levis, P., Lee, N., Welsh, M., and Culler, D. Tossim: Accurate and scalable simulation of entire TinyOS applications. In *Proceedings of the First International Conference on Embedded Networked Sensor Systems* (Los Angeles, Nov. 5–7). ACM Press, New York, 2003, 126–137.
23. Li, W., Zhang, X., Tan, W., and Zhou, X. H-TOSSIM: Extending TOSSIM with physical nodes. *Wireless Sensor Networks 1*, 4 (Nov. 2009), 324–333.
24. Lipphardt, M., Hellbrck, H., Pfisterer, D., Ransom, S., and Fischer, S. Practical experiences on mobile inter-body-area-networking. In *Proceedings of the Second International Conference on Body Area Networks* (Florence, Italy, June 11–13). Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Brussels, 2007.
25. Memsic Corporation. *TelosB*. Andover, MA. 2005; http://www.memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=152%3Atelosb
26. Österlind, F., Dunkels, A., Voigt, T., Tsiftes, N., Eriksson, J., and Finne, N. Sensornet checkpointing: Enabling repeatability in testbeds and realism in simulations. In *Proceedings of the Sixth European Conference on Wireless Sensor Networks* (Cork, Ireland, Feb. 11–13). Springer-Verlag, Berlin/Heidelberg, 2009, 343–357.
27. Park, S., Savvides, A., and Srivastava, M.B. Sensorsim: A simulation framework for sensor networks. In *Proceedings of the Third ACM International Workshop on Modeling, Analysis, and Simulation of Wireless and Mobile Systems* (Boston, Aug. 6–11). ACM Press, New York, 2000, 104–111.
28. Porter, B., Coulson, G., and Roedig, U. Type-safe updating for modular WSN software. In *Proceedings of the Seventh IEEE Conference on Distributed Computing in Sensor Systems* (Barcelona, June 27–29). IEEE Press, Washington, D.C., 2011, 1–8.
29. Project Sun SPOT. Oracle Labs, Redwood Shores, CA, 2006; http://www.sunspotworld.com
30. SensLAB. *Very Large-Scale Open Wireless Sensor Network Testbed*. Lyon, France, 2010; http://www.senslab.info/
31. Staub, T., Gantenbein, R., and Braun, T. Virtualmesh: An emulation framework for wireless mesh and ad hoc networks in omnet++. *Simulation: Transactions of the Society for Modeling and Simulation International* (special issue on software tools, techniques, and architectures for computer simulation) (Jan. 2011), 66–81.
32. Weiser, M. Some computer science issues in ubiquitous computing. *Commun. ACM 36*, 7 (July 1993), 75–84.
33. Wen, Y., Zhang, W., Wolski, R., and Chohan, N. Simulation-based augmented reality for sensor network development. In *Proceedings of the Fifth International Conference on Embedded Networked Sensor Systems* (Sydney, Nov. 6–9). ACM Press, New York, 2007, 275–288.
34. Werner-Allen, G., Swieskowski, P., and Welsh, M. Motelab: A wireless sensor network testbed. In *Proceedings of the Fourth International Conference on Information Processing in Sensor Networks* (Los Angeles, Apr. 25–27). IEEE Press, Piscataway, NJ, 2005.
35. WISEBED Project. *Wireless Sensor Network Testbeds*. Lübeck, Germany; http://www.wisebed.eu/
36. Wu, H., Luo, Q., Zheng, P., He, B., and Ni, L.M. Accurate emulation of wireless sensor networks. In *Proceedings of Network and Parallel Computing* (Wuhan, China, Oct. 18–20). Springer-Verlag, Berlin/Heidelberg, 2004, 576–583.

Geoff Coulson (geoff@comp.lancs.ac.uk) is a professor of distributed computing in the School of Computing and Communications in the Faculty of Applied Sciences at Lancaster University, Lancaster, U.K.

Barry Porter (barry.porter@comp.lancs.ac.uk) is a research associate in middleware and systems research in the School of Computing and Communications in the Faculty of Applied Sciences at Lancaster University, Lancaster, U.K.

Ioannis Chatzigiannakis (ichatz@cti.gr) is director of Research Unit 1 of the Computer Technology Institute & Press and adjunct faculty in the Computer Engineering & Informatics Department of the University of Patras, Rion, Greece.

Christos Koninis (koninis@cti.gr) is a research fellow at Research Unit 1 of the Computer Technology Institute & Press and a Ph.D. candidate in the Computer Engineering & Informatics Department of the University of Patras, Rion, Greece.

Stefan Fischer (fischer@itm.uni-luebeck.de) is a professor of networks and distributed systems in the Institute for Telematics of the University of Lübeck, Lübeck, Germany.

Dennis Pfisterer (pfisterer@itm.uni-luebeck.de) is a senior researcher and tenured lecturer in the Institute of Telematics at the University of Lübeck, Lübeck, Germany.

Daniel Bimschas (bimschas@itm.uni-luebeck.de) is an assistant researcher in the Institute of Telematics at the University of Lübeck, Lübeck, Germany, and lead software engineer of the WISEBED WSN testbed reference implementation and co-author of the WISEBED APIs.

Torsten Braun (braun@iam.unibe.ch) is professor of computer science and head of the research group Computer Networks and Distributed Systems at the University of Bern, Bern, Switzerland.

Philipp Hurni (hurni@iam.unibe.ch), **Markus Anwander** (anwander@iam.unibe.ch), and **Gerald Wagenknecht** (wagen@iam.unibe.ch) were all Ph.D. students in Braun's research group where they developed the TARWIS system.

Sándor P. Fekete (s.fekete@tu-bs.de) is a professor of algorithmics in the Computer Science Department of the Braunschweig Institute of Technology, Braunschweig, Germany.

Alexander Kröller (a.kroeller@tu-bs.de) is an assistant professor of algorithmics in the Computer Science Department of the Braunschweig Institute of Technology, Braunschweig, Germany.

Tobias Baumgartner (t.baumgartner@tu-bs.de) is a research scientist in the Algorithms Group of the Computer Science Department of the Braunschweig Institute of Technology, Braunschweig, Germany.