Shimmer, Cooja and Contiki: A New Toolset for the Simulation of On-node Signal Processing Algorithms

Patrick Kugler, Philipp Nordhus and Bjoern Eskofier
Digital Sports Group, Pattern Recognition Lab, Department of Computer Science
Friedrich-Alexander-University Erlangen-Nuremberg, Erlangen, Germany
Email: patrick.kugler@cs.fau.de

***Abstract*—Wearable sensors are widely used for data collection in many applications. Ssensor nodes have also been applied for real-time applications, e.g. for ECG analysis or activity and fall detection. Processing of the sensor data is either done on an external device or on the node itself. While on-node processing reduces data rate and increases battery life, development and testing can be time-consuming. To allow faster implementation of such algorithms, we propose a simulation framework for the Shimmer platform using the Cooja simulator, MSPSim and the Contiki operating system. We provide the simulator and example applications compatible with the ShimmerConnect protocol, allowing streaming of raw and pre-processed sensor data to MATLAB, LabView and Android. Additionally, a simple activity and fall detection algorithm was implemented on the sensor node and evaluated using both the simulator and real hardware. In the future this will allow rapid development and testing of on-node pre-processing algorithms.**

I. Introduction

Wearable sensors usually integrate physiological, biochemical or inertial sensors with a low-power microcontroller and a wireless transmitter [1]. Body Sensor Networks (BSNs) have been widely used for monitoring and medical applications [2], [3]. Recently, such wearable sensors have also been proposed for real-time applications [4]. Examples include biosignal processing [5], [6], fall detection for the elderly [7] as well as sport applications [8], [9], [10]. All these purposes require that the data is processed online with as little delay as possible.

In order to implement real-time solutions, one possible approach is to stream the raw sensor values from the wireless sensor nodes to a central processing unit, e.g. a computer or a smartphone [6], [3]. In this case no processing is done on the sensor node and all data is streamed to the host for processing (off-node processing). This is often easy to implement, as the host is simple to program and more powerful than the microcontroller of the sensor node. However, this can have a serious impact on the battery life, as the sensor nodes must constantly transmit high volumes of raw sensor data over a wireless link [4].

A different approach is to perform the processing directly on the wireless sensor node (on-node processing) to save battery life [11]. These techniques have been successfully applied to ECG processing [5], [12] and activity recognition [7]. Methods range from filtering and downsampling operations [13] to implementing the entire classification algorithm on the sensor node [7]. As only the processing results are transmitted, this approach can significantly increase battery life. However, on-node processing can be difficult to realize, as it requires implementing and testing the processing algorithms on the embedded microcontroller.

To simplify on-node software development, a series of software tools exists for many widely used platforms such as Shimmer, MikaZ or Telos [1]. Embedded operating systems like TinyOS [14] and Contiki [15] and their corresponding simulation tools TOSSIM [16] and Cooja [17] can help with the rapid development of applications. However, most operating systems and simulation tools focus on network simulation and do not support Bluetooth connections or streaming of sensor data. Hence, existing simulation tools do not address the need of the BSN community for simulation tools for the automated testing of on-node pre-processing algorithms.

The purpose of this paper was to provide a complete simulation framework for faster and easier development and testing of on-node pre-processing algorithms for BSN applications (Fig. 1). The main contributions of this paper are as follows. Firstly, the Contiki operating system was ported to the widely used Shimmer hardware platform. Secondly, the simulators Cooja and MSPSim were extended to support Bluetooth connections and the streaming of previously collected sensor data. Thirdly, we developed three different Contiki example applications for Shimmer and evaluated them both in the simulator and on real hardware.

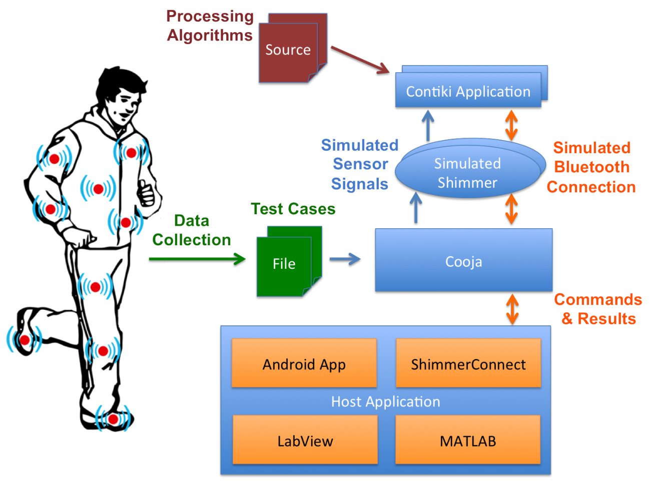


Fig. 1. The workflow for developing on-node pre-processing algorithms using Shimmer, Cooja and Contiki. Data from the wearable Shimmer sensors is collected and stored in a file (left). Using Cooja, this data is then replayed to a simulated Shimmer node running the Contiki-based pre-processing firmware. To evaluate if the firmware is working as intended, any compatible host application can connect to the node using a simulated Bluetooth connection.

II. Background

A. Shimmer Wireless Sensor Platform

Shimmer [18] is a commercially available wireless sensor platform, which supports a wide variety of physiological and kinematic sensors. It was chosen, as it is easily available, commercially supported and has been used in many BSN applications [1]. The current Shimmer 2R nodes contain a low power Texas Instruments MSP430F1611 microcontroller with 8-channel 12-bit ADC, a lithium-ion battery and a Roving Networks RN-42 Bluetooth module. Each node also contains a Maxim Integrated DS2411 ID-chip, a Texas Instruments CC2420 IEEE 802.15.04 wireless module and a Freescale Semiconductors MMA7260Q 3-axis accelerometer with 6g range. Additional extension boards are available for the recording of GPS positions, kinematics and physiological signals.

To help developers, the Shimmer community provides simple firmware examples for logging data on SD card and for streaming data over Bluetooth. For data streaming the custom *ShimmerConnect* protocol is employed, for which host applications are provided as well. This includes support for LabView, MATLAB and Android phones, which allows easy implementation of applications using off-node processing.

For on-node processing, Shimmer supports applications written for TinyOS [14], an embedded operating system providing all necessary hardware drivers. Applications for TinyOS are written in nesC [19], a new component based programming language derived from C. Due to this requirement, extending or writing new applications is often quite challenging.

For testing without hardware, TinyOS provides a simulator called TOSSIM [16]. However, it does not support all features of the Shimmer platform necessary for BSN applications. For example, it is not easily possible to simulate sensor data or to establish Bluetooth connections, hence most on-node processing applications can not be simulated and tested without changing the code.

B. Contiki Operating System

The Contiki operating system (ContikiOS) is an embedded operating system focused on sensor networks [15]. It is written in C and designed for small wireless sensor nodes with limited processing power and memory. Primary target platforms are Texas Instruments MSP430 and Atmel AVR, but the operating system has been extended to other platforms as well. Applications are written in plain C as a series of lightweight threads and event callbacks. By default, the scheduler provides only cooperative multitasking, however a preemptive scheduler is available as an external library. As Contiki is a network centric operating system, it provides a full stack for IPv4 and IPv6 and many example applications for web servers and clients.

Energy consumption is an important parameter for sensor networks, hence Contiki supports energy efficient low-power modes for the supported microcontrollers. Additionally the operating system directly provides a number of energy estimation routines [20]. These allow the user to measure run-time spent in each of several predefined power levels (full-power mode, low-power mode, communication receive, communication transmit, etc.), which can be used to estimate energy consumption during execution of the application program.

C. Cooja Network Simulator

Besides the operating system ContikiOS, the Contiki project also provides the network simulator Cooja [17] to speed up development and testing of Contiki applications. Written in Java, Cooja allows simulation of a single wireless sensor node or a whole network of such nodes on a standard computer without the sensor node hardware. Application execution, network communication and peripherals are simulated in real-time for as many nodes as needed. The user can interact with the nodes using a graphical user interface (GUI), which allows placement of sensor nodes, modifying sensor-inputs and disturbing network communication by increasing the noise level. Cooja supports different levels of simulation (network level, code level and instruction-set level). Simulation on the instruction-set level executes the compiled firmware binaries using an external microcontroller simulator, e.g. MSPSim [21] for MSP430 and AVRORA for the AVR platform. This allows evaluation of the accuracy and run-time of the implemented algorithms on a very detailed level. When using ContikiOS, Cooja can additionally provide estimates on the energy consumption of the running applications.

III. Methods

A. Extension of ContikiOS

ContikiOS supports different hardware configurations by providing the *Platform* abstraction. Each platform is a combination of a microcontroller and specifically wired sensors or wireless modules. To add support for the Shimmer platform, we extended the existing *sky* platform, as many components including the MSP430 microcontroller were similar. Using the TinyOS implementation as a reference, the new *Shimmer* platform was then modified with the correct I/O-ports and clock configuration. Driver support for all Shimmer-specific peripherals was added. This included the CC2540 wireless module, the RN-42 Bluetooth module, the ID-chip, the on-board accelerometer and the serial connection to the dock. Power saving and multiplexing of the serial bus between the different peripherals was implemented as in the TinyOS version. Support for all biophysical extension boards was provided by the ADC implementation, while support for the gyroscope of the kinematics extension board was added. The GPS extension board was made available as a serial interface.

B. Extension of MSPSim

To allow instruction-set level simulation of the Shimmer node, the platform had to be added to both MSPSim and Cooja. In MSPSim, each platform is represented by a class derived from *Node*. To represent the hardware on the Shimmer node, a new class *ShimmerNode* was created. In this class, the microcontroller type and its peripherals were configured. Support was added for all hardware described in section III-A.

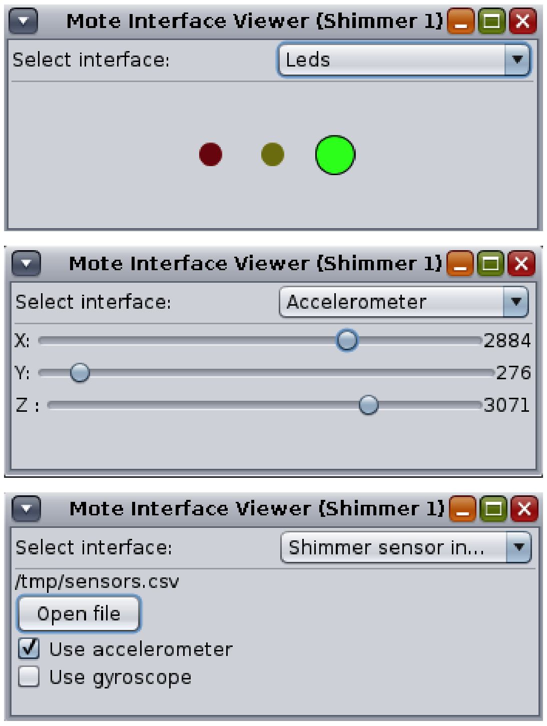
Additionally to the new *ShimmerNode* class, MSPSim was extended to allow simulation of Bluetooth connections. This was realized by mapping the communication over the Serial Port Profile (SPP) of the RN-42 Bluetooth module to a serial interface class. This provided a simple interface to communicate with the simulated sensor node. The same was done with the serial interface to the Shimmer dock, which is commonly used for debugging.

To allow simulation of sensor inputs, both the accelerometer and the gyroscope were represented as new types of *Chip* in MSPSim. These classes allowed setting of virtual sensor values in the simulator. Additionally, a helper class called *SensorInput* was created, which read a text file containing sensor values with timestamps and allowed automatic replay of the sensor signals to the simulated Shimmer node. The timestamps in the file were needed to synchronize the sampling rate of the signal to the simulation time in MSPSim. This was realized using a callback function, which got called after the required amount of simulated time had passed. The callback function changed the sensor input to the current value in the file, computed the time offset to the next timestamp in the file and set the callback-timer accordingly.

C. *Extension of Cooja*

In Cooja, each simulated sensor node has a specific *Mote*-type, which represents the user interface to the node. After the *ShimmerNode* class was available in MSPSim, a corresponding *ShimmerMote* class was added to Cooja. It provided a basic interface to the LEDs and buttons on the Shimmer node and represented the node on the screen (Fig. 2, top). Additional interface elements were added to allow manual setting of the simulated sensor inputs (Fig. 2, middle), and to allow selection of a file containing sensor signals (Fig. 2, bottom).

By default, the output of the serial dock interface and the Bluetooth SPP connection are connected with the Cooja console for debugging. Using the Cooja mote tool *SerialSocketServer* the Bluetooth connection of each simulated mote can be mapped to a dedicated TCP port. This port can then be mapped to a virtual SPP port, emulating a real sensor node


Mote Interface Viewer (Shimmer 1)
Select interface:
Open file
Mote Interface Viewer (Shimmer 1)
Select interface:
Mote Interface Viewer (Shimmer 1)
Select interface:

Leds

Accelerometer
X: 2884
Y: 276
Z : 3071

Shimmer sensor in...
/tmp/sensors.csv
☑ Use accelerometer
☐ Use gyroscope

Fig. 2. Graphical controls for the simulated Shimmer node in the network simulator Cooja, including LEDs (top), manual accelerometer controls (middle) and controls to automatically read sensor data from a file (bottom).

connected via Bluetooth. Any host application can use this port to communicate with the simulated node in the same way as with a real node.

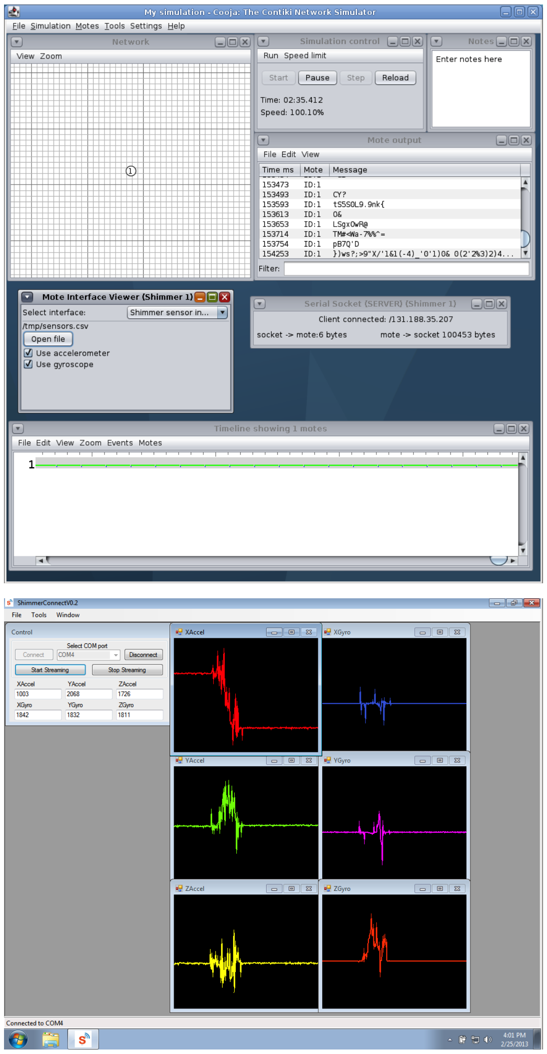
D. *Example Applications*

In order to test the simulation framework and to provide a complete Contiki-based software solution for the Shimmer platform, three different example applications were created. All applications were implemented in plain C using ContikiOS.

1) Basic Streaming: The first example application was intended as a full emulation of the original TinyOS based *BoilerPlate* firmware for Shimmer. This *BasicStreaming* firmware allowed streaming of sensor data over a Bluetooth connection, as it would be done for off-node processing. The employed *ShimmerConnect* protocol allowed different commands to be sent from the host application to the sensor node. Commands were available to toggle the LED on the node, to start/stop streaming and to configure the streaming parameters, i.e. to select the desired sensors, the sampling rate and the required sensitivity range. The Contiki implementation supported all sensors that are used in the original Shimmer firmware, including accelerometer, gyroscope, magnetometer and biophysical extension boards. As the *ShimmerConnect* protocol was used, the Contiki based firmware was directly compatible with all host applications using this protocol, including *ShimmerConnect* itself and the Shimmer instrument drivers for LabView, MATLAB and Android.

2) Filtered Downsampling: The second example application *FilteredDownsampling* was built to provide a simple example on how on-node pre-processing can be used to reduce transmission over the wireless link. This application was based on *BasicStreaming* and employed exactly the same protocol and behavior. However, this application activated a pre-processing algorithm after the toggle-LED command was sent to the node. When this algorithm was active, instead of returning the raw sensor values, the values were first processed using a 4-point moving average filter, followed by a decimation (down-sampling) by a factor of 4. This processing effectively reduced the transmitted data rate by a factor of 4, while possible aliasing artifacts were reduced using the simple low-pass filter.

3) Activity and Fall Detection: The third example was a separate application implementing a complete classification system to distinguish between rest and activity and to detect falls [7]. The *ActivityFallDetection* application sampled the 3D accelerometer at 100 Hz and applied a 3-point median filter to remove acceleration spikes. Then a high-pass filter with a cut-off frequency of 0.25 Hz was used to remove the gravitational component from the accelerometer signal. According to [7], the resulting signal was split into 1-second intervals and used to compute the sum over all absolute accelerometer values in this interval. Using a predefined threshold, the 1-second interval was then classified into either low or high activity. Additionally the euclidian norm of the acceleration vector was computed for each sample. A fall is detected if this value was over a separate threshold for at least two consecutive samples [7]. If a Bluetooth connection was present, the result of the classification decision was transmitted every second as either 0 (low activity), 1 (high activity) or 2 (fall).



Time: 02:35.412
Speed: 100.10%

```
Time ms  Mote  Message
153473   ID:1
153493   ID:1  CY?
153593   ID:1  tS5S0L9.9nk{
153613   ID:1  O&
153653   ID:1  LSgxOwR@
153714   ID:1  TM#<Wa-7%%^=
153754   ID:1  pB7Q'D
154253   ID:1  })ws?;>9"X/'1&1(-4)_'0'1)0& 0(2'2%3)2)4...
```

Select interface: Shimmer sensor in...
/tmp/sensors.csv
[✓] Use accelerometer
[✓] Use gyroscope

Client connected: /131.188.35.207
socket -> mote:6 bytes   mote -> socket 100453 bytes

Select COM port: COM4

XAccel	YAccel	ZAccel
1003	2068	1726
XGyro	**YGyro**	**ZGyro**
1842	1832	1811

Fig. 3. The Cooja simulator is streaming sensor data from a file to a simulated Shimmer sensor node running the *BasicStreaming* firmware. Data is then received by the original ShimmerConnect host application using a simulated Bluetooth connection (bottom).

IV. Evaluation

A. Basic Streaming

The goal of the first evaluation was to show if the Contiki implementation of the *BasicStreaming* application (section III-D1) was able to stream data from the sensors to all host applications using the *ShimmerConnect* protocol. For this a Shimmer node was programmed with the *BasicStreaming* application and was used to stream data to a host using Bluetooth. While streaming, the Shimmer node was worn on the ankle and the subject was walking back and forth. Testing was performed using different sensor configurations (accelerometer, gyroscope, magnetometer, biophysical sensor) and sampling rates (10 Hz, 50 Hz, 100 Hz, 200 Hz, 500 Hz). Data was recorded on a laptop PC using the *ShimmerConnect* host application. After recording, the sensor values were visually inspected and timestamps were inspected to check for missing samples. Additionally, streaming was tested with the Shimmer MATLAB instrumentation driver and an Android application running on a mobile phone.

B. Cooja Simulation

To test the Cooja modifications, a simulated Shimmer node was created inside the Cooja simulator running on a standard laptop PC (Intel Core i5, 2.4 GHz, 4GB RAM). Both the original TinyOS streaming application as well as the new *BasicStreaming* application were tested on this node. Sensor data was simulated using a previously recorded file containing 60 seconds of accelerometer and gyroscope signals during gait recorded at 200 Hz. The original *ShimmerConnect* host application was then used to connect to the virtual Bluetooth serial port and to stream and record the sensor data from the simulated sensor node (Fig. 3). The recorded files were then compared to the original file.

C. Filtered Downsampling

The *FilteredDownsampling* application (section III-D2) was tested in the simulator using the same sensor data file as in the previous test. The processed results were recorded using the *ShimmerConnect* host application. After recording, both the original file and the recorded file were imported into MATLAB. The original sensor data was then processed in MATLAB using a 4-point moving average filter and down-sampled by a factor of 4. The result was compared to the data processed by the Shimmer application. Additionally, the application was tested on a real Shimmer node.

D. Activity and Fall Detection

To test the *ActivityFallDetection* application (section III-D3), a recording was performed with one subject wearing a Shimmer node at the belt. The subject performed multiple trials containing resting, walking and simulated falls. The recorded data was then used in the simulator as input for the activity detection application. Using Cooja, the output of the application was recorded and compared to ground truth. Additionally, the application was tested on a real Shimmer node.

V. Results

A. Basic Streaming

The *BasicStreaming* application was able to correctly stream data in all configurations. The firmware worked with all tested host applications, including the MATLAB instrumentation driver and the Android application. Recorded data quality was equivalent to previous recordings using the TinyOS firmware and did not show any signs of missing samples.

B. Cooja Simulation

Run-time of all simulations was faster than real-time. The comparison of the recorded files from the simulated nodes to the original file showed that both files contained exactly the same sensor data. As the start of the playback and the recording was not synchronized, the file recorded from the simulated node contained additional values before and after the simulated signals.

C. Filtering and Downsampling

The signals processed by the *FilteredDownsampling* application and the signals processed by MATLAB were identical except for small rounding errors. In every case, the correlation between both signals was 0.999998 with an absolute root-mean square raw value error of less than 0.5, which translates to less than 1 mG for the accelerometer. The test on the real Shimmer node showed that processing was done in real-time and that the data rate was reduced by a factor of 4 as soon as processing was activated.

D. Activity and Fall Detection

The *ActivityFallDetection* application assigned 98 % of the labels correctly when evaluating the collected data set in the simulator. On a real node, classification was running in real-time with a wireless transmission rate of one byte per second.

VI. Discussion

The main goal of this paper was to provide faster and easier development and testing of on-node pre-processing algorithms. This was realized by creating a complete Cooja based simulation framework for the Shimmer platform. The software is available on request and it is planned to integrate it into the Contiki open-source project.

The implemented toolset is ready to be used for rapid development and testing of BSN applications. Debugging is simplified, as the simulator allows simple output of debugging messages and full debugger support is provided without dedicated and expensive hardware. Using sensor data read from a file allows reliable and reproducible testing of processing algorithms without altering the firmware. This simulated sensor data can be pre-recorded real data or artificial data, enabling test-driven development and easy evaluation of on-node processing algorithms. Using Cooja, the workflow for development and testing of such applications is greatly improved. If an error in the algorithm is found, it can be corrected in the source file and a simple click in the Cooja GUI triggers a reload. This automatically recompiles the application, loads it onto the simulated sensor node and repeats the test case with the previous sensor input. Many data sets can be evaluated quickly, as the simulator can run the application simultaneously on multiple sensor nodes or even simulate the nodes faster than real-time. Additionally, the simulator can give feedback on the feasibility of algorithms on the low-power microcontroller as well as estimate the impact of the processing on the battery life.

The implementation of Shimmer as a Contiki platform was successful and all peripherals were usable. As Contiki is written in plain C, it is now easier to extend functionality and to create applications for the Shimmer platform without having to learn nesC. Additionally, the port to Contiki enables the use of a full IPv6 stack and tools for energy estimation. However, the energy estimation routines must be calibrated for each hardware platform, which was not yet done for Shimmer. This will be subject of future work. As no schematics were available for the Shimmer hardware, some parts could only be implemented by reverse-engineering the provided TinyOS implementation. Especially the correct multiplexing of the SD-card and the dock proved to be complex and difficult to implement. As not all possible use-cases have been tested, there might still be some unidentified problems and the SD-card had to be disabled to avoid data corruption. As the SD card is important for many BSN projects, this has to be addressed in future work.

The provided implementation of Shimmer as a mote and node in Cooja and MSPSim was almost complete and worked as intended in all tested cases. Complete magnetometer and GPS modules are still missing, but they can easily be added in future work. The evaluation showed that the simulator was able to simulate the tested firmware images correctly and faster than real-time. The instruction-set level simulation provides a high accuracy on the run-time and accuracy of the tested algorithms. However, one has to keep in mind that the simulation is not perfectly accurate in all cases and some driver-related bugs could only be reproduced on real hardware. But this is often only a problem for the developer of the hardware abstraction layer and should not be a problem for the development of applications and pre-processing algorithms. The replay of sensor data was reliable and no samples were missed. However, to allow fully reproducible test-cases a synchronization feature must be added to start the replay of the sensor data at a consistent time point, e.g. 2 seconds after reset.

The implementation of the Bluetooth connection as a serial interface was quite simplistic, however it was sufficient to connect and communicate with the simulated sensor nodes. In future work this could be extended by incorporating a full Bluetooth simulation, but this is quite complex and not necessary for most applications. One drawback with the current implementation is that an additional mapping from the TCP port to a virtual serial device is necessary to work with most host applications. Additionally, it is not yet possible to use the simulated sensor nodes from an Android phone or from the Android simulator without altering the Android application. This problem could be solved by modifying the Android simulator to allow streaming from a TCP port to a virtual Bluetooth serial port.

The provided example applications demonstrated how to use Contiki to build applications for off-node processing, simple on-node pre-processing and full on-node processing. The evaluation has shown that the *BasicStreaming* application was fully compatible with the *ShimmerConnect* protocol and that it was usable as a reliable drop-in replacement for the TinyOS based *BoilerPlate* application. It was compatible with all available host applications for MATLAB, LabView and Android. While the used protocol ensures maximum compatibility with existing applications, it is not the most efficient or extensible protocol. In the future, it should be replaced by a more sophisticated protocol, e.g. supporting activation and deactivation of on-node pre-processing algorithms. During testing, we have also found that the *ShimmerConnect* host application was not very reliable for recording at high sampling rates. Recording at 500 Hz was only possible with the visualization turned off.

The *FilteredDownsampling* application also worked as intended. The resulting signals were identical to the exact values except for small rounding errors. Using this application, the data rate can be reduced by a factor of 4, increasing battery life while maintaining a high sampling rate. This could be beneficial if high frequencies are present in the input signal, which would lead to aliasing artifacts with lower sampling rates. In the tests, the rounding error was almost exactly 0.5, which is due to the use of integer values for the sensor data. While this was no problem for the implemented mean filter, especially IIR filters will require higher precision to be stable. For this floating or fixed point arithmetic is required in the Shimmer firmware, which will increase processing demand.

The *ActivityFallDetection* application was intended as a simple demonstrator how to implement a complete classification system on Shimmer using Contiki. The algorithm was a simplified version of [7] and had a high classification rate during the short test. By implementing the whole algorithm on the node, communication was reduced to one byte per second, compared to the 1.2 kByte/s of raw sensor data. This shows how on-node processing can reduce or eliminate wireless communication and extend battery life. Development was straightforward, as only few lines of C-code were required to implement the algorithm. Testing of the algorithm and adaptation of the threshold values was done using the simulator and the recorded sensor data. As the classification results were immediately visible in the simulator, no real Shimmer nodes were needed during development.

VII. Summary and Outlook

In this paper we have presented an implementation of a complete Contiki simulation framework for the Shimmer platform. The network simulator Cooja was extended to support Bluetooth connections and the simulation of sensor data. The presented Contiki example applications are directly usable as drop-in replacements for the standard Shimmer firmware and work with all host applications supporting the *ShimmerConnect* protocol. As they are written in plain C, they can easily be extended using custom algorithms for signal processing and classification. Together with the Cooja simulator, this allows faster and easier development and testing of on-node preprocessing algorithms for wireless sensor nodes.

In the future this will allow rapid development of BSN applications while extending the battery life of the nodes.

Acknowledgment

We wish to thank Moritz Strübe from the Distributed Systems and Operating Systems Group, Erlangen. This work was supported by the Embedded Systems Institute (ESI) Erlangen, the Bavarian Ministry for Economic Affairs and the European Fund for Regional Development. Additional financial support was provided by the adidas AG, Herzogenaurach, Germany.

References

[1] M. Johnson, M. Healy, P. Van de Ven, M. Hayes, J. Nelson, T. Newe, and E. Lewis, “A comparative review of wireless sensor network mote technologies,” in *IEEE Sensors 2009*, 2009, pp. 1439–1442.

[2] B. Latré, B. Braem, I. Moerman, C. Blondia, and P. Demeester, “A survey on wireless body area networks,” *Wireless Networks*, vol. 17, no. 1, pp. 1–18, 2011.

[3] J. Klucken, J. Barth, P. Kugler, J. Schlachetzki, T. Henze, F. Marxreiter, Z. Kohl, R. Steidl, J. Hornegger, and B. Eskofier, “Unbiased and Mobile Gait Analysis Detects Motor Impairment in Parkinson’s Disease,” *PLoS ONE*, vol. 8, no. 2, p. e56956, 2013.

[4] M. Hanson, H. Powell, A. Barth, K. Ringgenberg, B. Calhoun, J. Aylor, and J. Lach, “Body Area Sensor Networks: Challenges and Opportunities,” *Computer*, vol. 42, no. 1, pp. 58–65, 2009.

[5] H. Mamaghanian, N. Khaled, D. Atienza, and P. Vandergheynst, “Compressed Sensing for Real-Time Energy-Efficient ECG Compression on Wireless Body Sensor Nodes,” *IEEE Transactions on Biomedical Engineering*, vol. 58, no. 9, pp. 2456–2466, 2011.

[6] S. Gradl, P. Kugler, and B. Eskofier, “Real-time ECG monitoring and arrhythmia detection using Android-based mobile devices,” in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society EMBC 2012*, 2012, pp. 2452–2455.

[7] D. Karantonis, M. Narayanan, M. Mathie, N. Lovell, and B. Celler, “Implementation of a real-time human movement classifier using a triaxial accelerometer for ambulatory monitoring,” *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, no. 1, pp. 156–167, 2006.

[8] B. Eskofier, P. Kugler, D. Melzer, and P. Kuehner, “Embedded classification of the perceived fatigue state of runners: Towards a body sensor network for assessing the fatigue state during running,” in *Ninth International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 2012, pp. 113–117.

[9] B. Eskofier, M. Oleson, C. DiBenedetto, and J. Hornegger, “Embedded surface classification in digital sports,” *Pattern Recognition Letters*, vol. 30, no. 16, pp. 1448–1456, 2009.

[10] L. M. Stirling, V. von Tscharner, P. Kugler, and B. Nigg, “Classification of muscle activity based on effort level during constant pace running,” *Journal of Electromyography and Kinesiology*, vol. 21, no. 4, pp. 566–571, 2011.

[11] M. A. Hanson, H. C. Powell, Jr., A. T. Barth, and J. Lach, “Application-Focused Energy-Fidelity Scalability for Wireless Motion-Based Health Assessment,” *ACM Transactions on Embedded Computer Systems*, vol. 11, no. S2, pp. 50:1–50:21, 2012.

[12] F.-T. Sun, C. Kuo, and M. Griss, “PEAR: Power efficiency through activity recognition (for ECG-based sensing),” in *5th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*, 2011, pp. 115 –122.

[13] L. Au, M. Batalin, T. Stathopoulos, A. Bui, and W. Kaiser, “Episodic sampling: Towards energy-efficient patient monitoring with wearable sensors,” in *Annual International Conference of the IEEE Engineering in Medicine and Biology Society EMBC 2009*, 2009, pp. 6901 –6905.

[14] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, “TinyOS: An Operating System for Sensor Networks Ambient Intelligence,” in *Ambient Intelligence*, W. Weber, J. M. Rabaey, and E. Aarts, Eds., 2005, ch. 7, pp. 115–148.

[15] A. Dunkels, B. Gronvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *29th Annual IEEE International Conference on Local Computer Networks 2004*, 2004, pp. 455–462.

[16] P. Levis, N. Lee, M. Welsh, and D. Culler, “TOSSIM: accurate and scalable simulation of entire TinyOS applications,” in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys ’03, 2003, pp. 126–137.

[17] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, “Cross-Level Sensor Network Simulation with COOJA,” in *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, 2006, pp. 641–648.

[18] A. Burns, B. Greene, M. McGrath, T. O’Shea, B. Kuris, S. Ayer, F. Stroiescu, and V. Cionca, “SHIMMER - A Wireless Sensor Platform for Noninvasive Biomedical Research,” *IEEE Sensors*, vol. 10, no. 9, pp. 1527–1534, 2010.

[19] D. Gay, M. Welsh, P. Levis, E. Brewer, R. V. Behren, and D. Culler, “The nesC language: A holistic approach to networked embedded systems,” in *In Proceedings of Programming Language Design and Implementation (PLDI)*, 2003, pp. 1–11.

[20] J. Eriksson, F. Osterlind, T. Voigt, N. Finne, S. Raza, N. Tsiftes, and A. Dunkels, “Demo abstract: Accurate power profiling of sensornets with the COOJA/MSPsim simulator,” in *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems*, 2009, pp. 1060–1061.

[21] J. Eriksson, F. Österlind, N. Finne, N. Tsiftes, A. Dunkels, T. Voigt, R. Sauter, and P. J. Marrón, “Cooja/mspsim: interoperability testing for wireless sensor networks,” in *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*, ser. Simutools ’09, 2009, pp. 27:1–27:7.