International Scholarly Research Network
ISRN Communications and Networking
Volume 2012, Article ID 180369, 10 pages
doi:10.5402/2012/180369

Research Article

Experimental Performance Evaluation of POBICOS Middleware for Wireless Sensor Networks

Jouni Hiltunen, Mikko Ala-Louko, and Markus Taumberger

Converging Networks Laboratory, VTT Technical Research Centre of Finland, Kaitoväylä 1, 90590 Oulu, Finland

Correspondence should be addressed to Jouni Hiltunen, jouni.hiltunen@vtt.fi

Received 13 December 2011; Accepted 16 January 2012

Academic Editors: N. Abu-Ghazaleh, Y. Jiang, R. Montemanni, and S. Weller

The advances in the theory of wireless sensor networks have been remarkable during the past decades, but there is a lack of extensive experimental evaluations. In this paper we present performance-evaluation methods and results for POBICOS (platform for opportunistic behaviour in incompletely specified, heterogeneous object communities), which is an advanced middleware for wireless sensor networks (WSNs). The measurements concern energy consumption, duty cycle, and OS task profiling as well as communication characteristics such as round trip time (RTT) and throughput. In addition, a bandwidth analysis during a long-term experiment of fully functional POBICOS network and application is studied. Based on the evaluation results, power mode and data cache improvements are presented as well as CPU clock frequency optimizations.

1. Introduction

The research done in the field of WSNs has advanced a lot in the past decades. The achieved performance of a WSN implementation is inevitably tied to the characteristics of the used platform, and therefore, the performance evaluation cannot rely solely on the theoretical background. Our study presents an experimental performance evaluation of POBICOS which is an advanced opportunistic WSN middleware implemented on TinyOS operating system and Imote2 hardware platform.

The performance evaluation methods and results are related to energy consumption, duty cycle, and OS task profiling as well as communication characteristics such as round trip time and throughput. In addition, a bandwidth analysis during a long-term experiment of fully functional POBICOS network and application is included. Based on the evaluation results, power mode and data cache improvements are presented as well as CPU clock frequency optimizations.

Energy consumption of battery-powered sensor motes is a very crucial implementation issue which affects the operational costs of the WSN. The energy consumption is mainly affected by the achieved duty cycle and power modes of the motes. The overall operational energy consumption of the motes may be obtained through online energy consumption monitoring or through a hybrid method, in which the results of offline energy consumption measurements and online duty cycle monitoring are combined.

The duty cycle investigation is based on CPU usage monitoring which, in case of the Imote2 platform, can be implemented through performance monitoring unit (PMU) events. The duty cycle optimization can be achieved through monitoring the CPU usage of each running task with a task profiler. This usually requires modifications to the OS source code, but in the case of TinyOS the implementation of the task profiler is straightforward because of TinyOS's simple concurrency model which is based on a single thread and nonpreemptive tasks.

The communication performance of WSN middleware depends on the underlying physical and media access control layers. IEEE 802.15.4 is such a standard widely used in WSNs. The current implementation of the POBICOS supports ZigBee which adds tree topology routing on top of the 802.15.4. Since POBICOS implements services such as reliable transport and packet fragmentation the RTT and throughput measurements were conducted to find out how much additional delay and overhead the POBICOS middleware adds to those of ZigBee.

The middleware internal protocols perform tasks, such as network management, that require control messages to

be sent amongst the nodes. Therefore, bandwidth analysis is an important performance metric when comparing different middleware solutions. We have done a network-wide bandwidth analysis to determine the bandwidth usage of the middleware when running a typical application.

2. Middleware Description

Opportunistic applications are developed without knowledge of the resources that will be present at deployment environment. They use the resources that happen to be available in an environment to achieve the application goals. In POBICOS, such applications are built of collections of microagents that work in an event-driven manner. Microagents can be created and released dynamically, and they can communicate with each other according to the application logic. The microagents are arranged in a tree-based hierarchy, where each microagent has a parent and optionally one or more children. Microagents can only communicate with their parent and children. In case a microagent becomes orphan, for example when the network gets partitioned, it releases itself to ensure a consistent state of the application.

The open-source POBICOS middleware [1] offers mechanisms to host microagents on different hardware platforms by executing them in a virtual machine [2]. It automatically handles the placement of microagents onto the actual hardware according to their resource requirements. A resource discovery is performed each time a microagent is created by the application. The middleware can also migrate microagents to other nodes depending on parameters, for example, to reduce their communication distance [3]. A middleware-level heartbeat protocol detects the disappearance of microagents and propagates this event to its parent and children. Other main features of the middleware are transparent inter-agent communication, security mechanisms [4], and multi-faceted resource access.

The middleware is fully decentralized and each node is running its own instance of it, which makes performance aspects very critical.

3. Performance Evaluation Methods

3.1. Energy Consumption Measurements. The measurement setup for the energy consumption measurements is depicted in Figure 1, and the used hardware is listed in Table 1. A nonintrusive method for measuring the current is achieved by using a probe that measures the current in a conductor through inductive coupling with no electrical contact. The output signal of the probe is amplified by the current probe amplifier, and the amplified signal is displayed in the oscilloscope. Power consumption is then calculated by multiplying the measured current with the 3.3 V operating voltage.

3.2. Task Profiler. The implemented task profiler enables online monitoring of the middleware, see Figure 2. The task profiler collects and stores the CPU PMU events for each running task and timer interrupt. Each record contains only the prevailing PMU counter values and the task/timer IDs which are stored in sequential order. Most of the task monitoring processing is done in the aggregator which is connected to the mote through UART. The task profiler data is requested from the mote on demand to the aggregator which calculates absolute PMU values and converts the task and timer IDs to descriptive names. The names are derived from app.c file which is produced by the nesC compiler from the application code. The presented system provides a lightweight solution for performance monitoring.

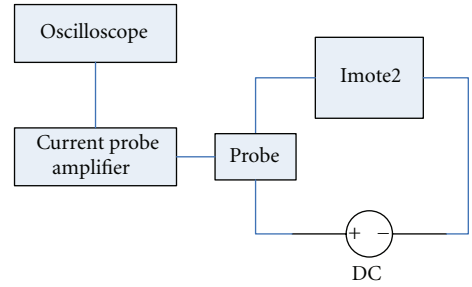

Figure 1: Energy measurement setup.

Table 1: List of the used hardware.

Imote2 battery board	IBB2400
Imote2 processor board	IPR2400
Imote2 radio board	CC2480 board based on TI's Z-Accel Demonstration Kit
Imote2 sensor board	ITS400
Oscilloscope	Tektronix TDS3052 500 MHz/ 5GS/s
Current probe amplifier	Tektronix AM503B
Current probe	Tektronix A6302

3.3. Communication Measurements

3.3.1. RTT. RTTs were measured on two different levels of the POBICOS communications stack:

(i) PoHWCommM—a low-level communication component using the ZigBee subsystem that gives a reference point for comparison.

(ii) PoCommM—the main POBICOS communication component responsible of implementing the reliable transport service with fragmentation of messages.

In both cases, RTT was measured over one hop using two Imote2 motes with empty and full payload lengths. When node A sends a message to node B a timestamp is taken. Upon receiving the message, node B responds by sending the exact same message back to node A. Another timestamp is taken when node A receives the message from node B. RTT is calculated by subtracting the two timestamps. The time accuracy of the measurement was 1 ms so each payload length was tested 10 times and the RTT's averaged.

3.3.2. Throughput. Imote2 network of two nodes over one hop is used in the measurements. Node A sends maximum

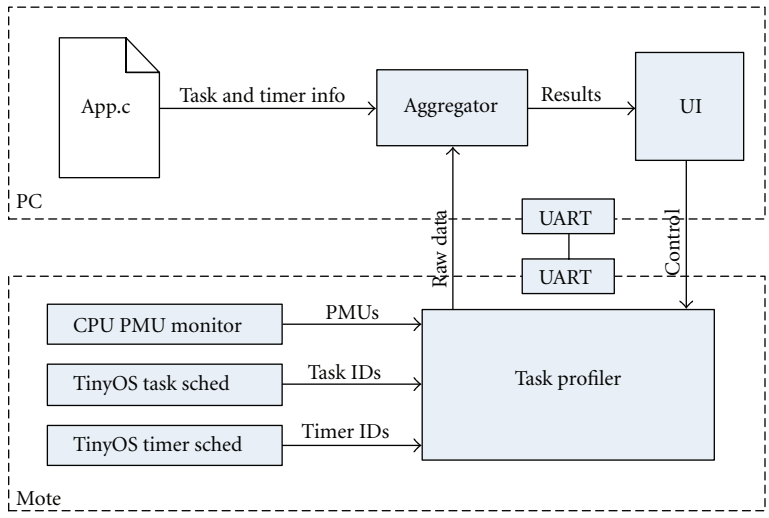

Figure 2: Task profiler block diagram.

length packets of 51 B to node B. The throughput is then calculated every 1 second for ZigBee and for the POBICOS best-effort mode. For POBICOS reliable mode, which introduces acknowledgments and retransmission in case of lost packets, the throughput calculation was done every 1 minute.

3.3.3. Bandwidth Analysis. The bandwidth analysis data was obtained directly from the main communication component (PoCommM) of the middleware through which all the traffic of the higher-level protocols of the middleware traverses. PoCommM provides a parameterized TinyOS interface to the higher-level protocols which means that each protocol is assigned a unique ID that can be used to identify the active protocol for each message sent. PoCommM logs the message lengths, interface IDs, and the transmission mode (unreliable versus reliable) with timestamps. This log can then be used to derive the perprotocol bandwidth usage. The time interval for the bandwidth calculation was chosen to be one minute. The layered architecture of the POBICOS communication components is illustrated in Figure 3. The middleware internal protocols wire to PoCommM which utilizes PoHWCommM to gain access to the ZigBee subsystem. PoCommM logs the communication service usage statistics.

4. Measurement Results

4.1. Energy Consumption Measurements. The energy consumption measurements were conducted with and without the middleware, using different power modes and CPU clock frequencies. To investigate the energy consumption of different power modes extensively, support for the standby power mode was implemented since the used TinyOS platform supported only active and idle power modes. Figure 4 presents the power consumption measurements with Imote2 battery, processor, and sensor boards as well as additional ZigBee radio board included.

In the power consumption measurements presented above, the active mode means running a processor-bound TinyOS task (an empty for-loop). The memory-bound task is an exception where the Imote2 internal memory is continuously accessed. Both the active and nonactive modes were measured within the same test run using periodic duty cycle where 1 s activity was followed by 1 s of inactivity. The measured current consumption multiplied with the operating voltage equals the momentary power consumption of the mote. The total energy consumption is then obtained by integrating the measured power-consumption curve over time.

The comparison of 13 MHz and 104 MHz CPU clock frequency modes shows that energy efficiency is better with 104 MHz mode if the standby mode is in use. Although the 13 MHz mode consumes ~100 mW less in the active mode, the same processor-bound task takes eight-times longer to complete. In the standby mode, the power consumption does not depend on the operating frequency. Without the standby mode, the 13 MHz should be preferred since the middleware is expected to be in idle state most of the time and the idle mode energy efficiency is poorer at the higher CPU clock frequency. However, the implementation of standby mode is essential since it saves 55 mW in 13 MHz mode and 110 mW in 104 MHz mode compared to the idle mode.

For the middleware energy efficiency it is important that the mote remains in nonactive mode most of the time since all middleware tasks consume ~180 mW more than in the standby mode. The radio reception, radio transmission, LEDs and light sensor reading consume additional ~16 mW compared to the processor-bound task. The memory-bound task consumes 13 mW less than the processor bound task.

The ZigBee radio board is a significant energy sink of the POBICOS node since it constitutes over 20% of the total energy consumption. The middleware initializes the ZigBee radio to the active mode, but without the middleware the radio remains in the idle state. The radio consumes 33 mW

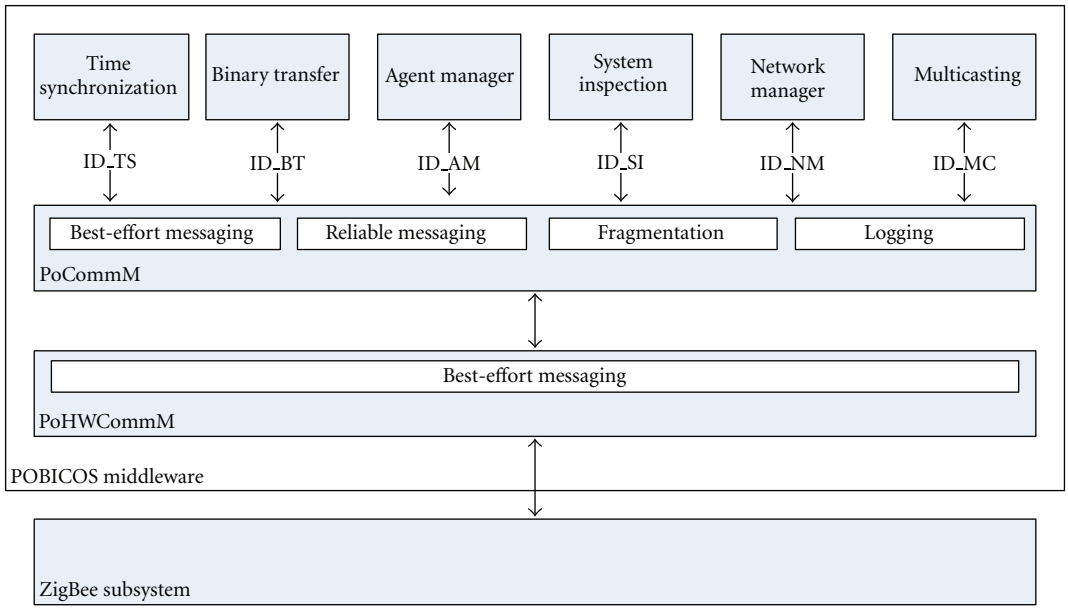

Figure 3: POBICOS communication components.

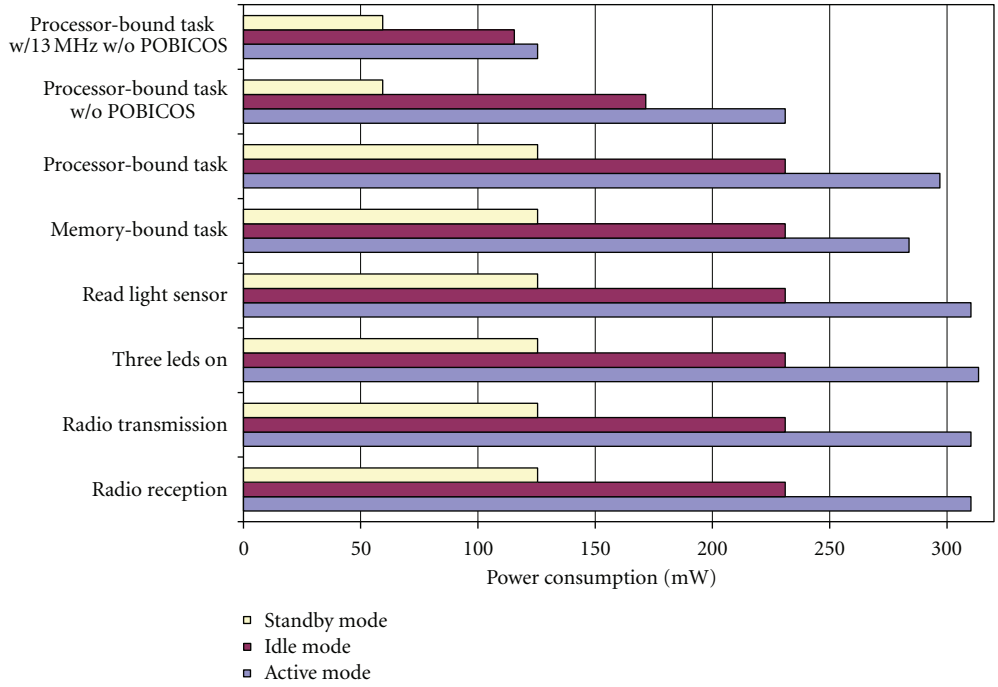

Figure 4: Power consumption with middleware, CPU clock frequency 104 MHz.

when not initialized, 100 mW in listening state, and 110 mW in transmitting and reception states.

The worst case energy profile concerning the initialization sequence of a node in an empty network is presented in Figure 5. The initialization energy consumption is dominated by the radio board especially when the node is a ZigBee coordinator since it consumes 17 s while the total initialization duration is 21 s. However, the radio initialization sequence is faster if there are other nodes in network. The TinyOS initialization lasts 2 seconds and POBICOS initialization duration is < 0.5 second.

Real-world data regarding the duty cycle of a POBICOS node was gathered from an experiment of running a POBICOS system in an office building continuously for four days. In the experiment, temperature and light sensing nodes were used that periodically poll the sensor values and send them via the radio channel to be processed. The CPU loads of the nodes were obtained using the methods presented in

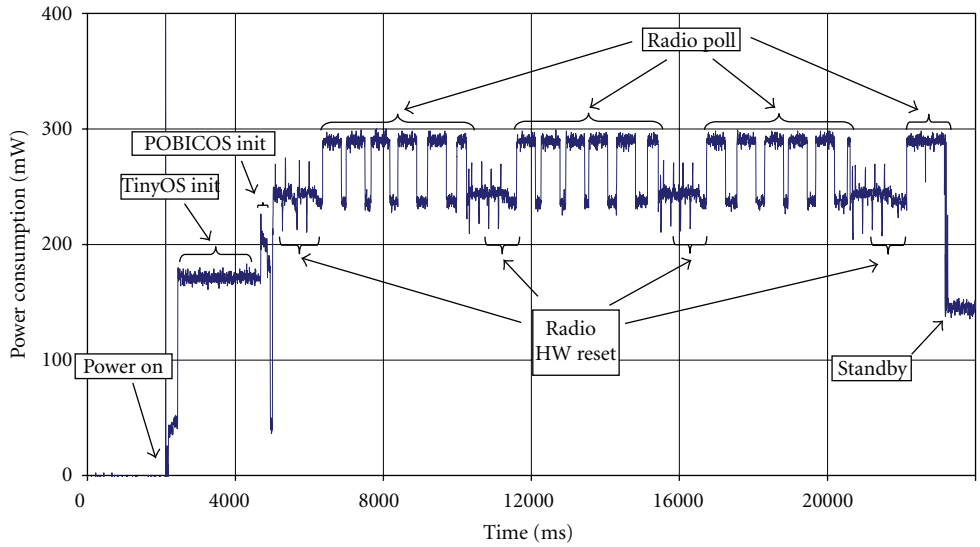

Figure 5: Worst case initialization sequence energy profile of a ZigBee coordinator.

Section 3. The locally measured load values of each node were sent every 10 seconds over the air to a monitoring node that was connected to a POBICOS administration and monitoring tool (PAM). PAM was used to create a log file of the reported load values.

Analysis of the log file of one node revealed that the CPU load during the experiment was close to constant except for the first load report that includes the execution of the initialization sequence of the middleware. The CPU load value of the first report was 44.100% after which, it stabilized to 0.040%. The average CPU load during the whole experiment was 0.057%. The results obtained from other nodes were observed to be similar.

We are now able to estimate the energy consumption of a POBICOS node during the experiment when we combine the online measurements of the duty cycle of a node with the offline measurements of the power consumption presented in Figure 4. If we assume the respective power consumptions during active and standby modes to be constant, the total energy consumption can be calculated with the following equation:

$$E(t) = \int P(t) \cdot dt = (\mathrm{DC} \cdot P_A + (1 - \mathrm{DC}) \cdot P_S) \cdot t, \quad (1)$$

where $E(t)$ is the energy consumption at time t, $P(t)$ is the power consumption at time t, DC is the duty cycle, P_A is the constant power consumption in active mode, and P_S is the constant power consumption in standby mode.

From the measurement data (Figure 4) we can obtain a P_A of 310.2 mW and a P_S of 125.4 mW. Furthermore, we use the abovementioned value of 0.00057 for DC. This yields us a daily energy consumption of 10.844 kJ per one POBICOS node. In one year this adds up to a consumption of 3.958 MJ which corresponds to 1.099 kWh.

4.2. Task Profiler. The results obtained with the task profiler in a two-node network without an application running are presented in Table 2 and the task descriptions in Table 3. In this case the duty cycle is less than 1%. The results indicate that a significant number of the used CPU cycles are wasted by dependency stalls because the data cache is not supported by the TinyOS platform. The support for the data cache was implemented later, and the results of the data cache measurements with TinyOS Blink application are presented in Table 4. The data cache was observed to save a significant number of CPU clock cycles.

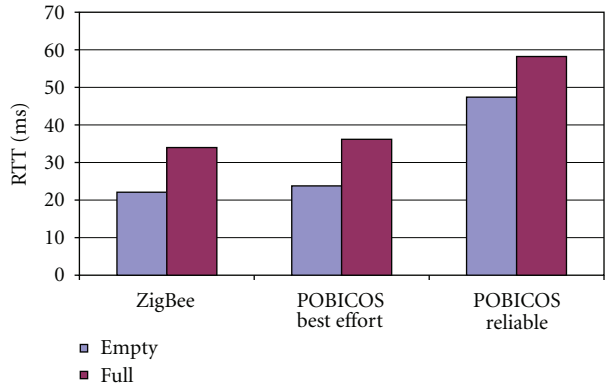

Figure 6: Results of the RTT measurements.

4.3. Communication Measurements

4.3.1. RTT. The results of the RTT measurements are presented in Figure 6. It depicts the RTT values for the ZigBee subsystem and for the POBICOS best-effort and reliable modes. Whereas in the first two tests the RTT constitutes solely of two messages, the reliable mode includes also the acknowledgment messages sent automatically for each received message.

Table 2: 1.5 second Task Profiler sample run in a two-node network without an application running.

Task name	Data cache accesses	Dependency stall (cycles)	Run time (ms)	CPU instructions	CPU clock cycles
McuSleep.sleep	0	0	458.4375	0	0
PoPerfInspM$CpuUsageMilliTimer$fired	N/A	N/A	0.1250	N/A	N/A
McuSleep.sleep	0	0	0.0312	0	0
VirtualizeTimerC0updateFromTimer	68	3492	0.1250	943	11557
McuSleep.sleep	0	0	15.0938	0	0
PoCommTimersM$BaseTimer$fired	176	6239	0.2188	1647	22157
McuSleep.sleep	0	0	0.0312	0	0
VirtualizeTimerC0updateFromTimer	68	3492	0.1250	943	11481
McuSleep.sleep	0	0	69.2500	0	0
PoReliableTransportIstub$Timer$fired	11005	234337	6.0312	78984	610469
VirtualizeTimerC0updateFromTimer	68	3493	0.1250	943	11665
McuSleep.sleep	0	0	8.1875	0	0
CC2480P$sendDoneTask	22	1674	0.0938	594	8577
McuSleep.sleep	0	0	13.0312	0	0
CC2480P$receiveTask	10794	222670	4.3750	65291	443417
McuSleep.sleep	0	0	8.4688	0	0
CC2480P$sendDoneTask	23	1646	0.0938	599	8525
McuSleep.sleep	0	0	13.0312	0	0
CC2480P$receiveTask	3175	65010	1.8750	20568	190225
McuSleep.sleep	0	0	126.9688	0	0
PoNetworkMngrM$MilliTimer$fired	126	5162	0.1875	1281	19181
McuSleep.sleep	0	0	0.0312	0	0
VirtualizeTimerC0updateFromTimer	68	3493	0.1250	943	11549
McuSleep.sleep	0	0	846.4688	0	0

Table 3: Middleware and TinyOS task descriptions from the sample run.

Task name	Source	Description
McuSleep.sleep	TinyOS	Command called in task scheduler when there is no tasks to be run.
PoPerfInspM$CpuUsageMilliTimer$fired	POBICOS	Timer task to measure CPU load periodically. Default measurement period is 3 seconds. Because this task resets the performance counters details of the profiler cannot measure all metrics.
VirtualizeTimerC0updateFromTimer	TinyOS	Task to manage TinyOS timers.
PoCommTimersM$BaseTimer$fired	POBICOS	Task to manage reliable transport timers. Default period is 4 seconds.
PoReliableTransportIstub$Timer$fired	POBICOS	Timer task to manage reliable transport transmissions.
CC2480P$sendDoneTask	POBICOS	Radio transmission task.
CC2480P$receiveTask	POBICOS	Radio reception task.
PoNetworkMngrM$MilliTimer$fired	POBICOS	Task for network management. Default period is 1 second.

We can see that the RTTs of ZigBee and POBICOS best-effort mode are very closely equal with a minor increase observable in the POBICOS best-effort mode. The overhead introduced by the POBICOS reliable mode can be mainly explained by the acknowledgment mechanism of the reliable transport service. The transmissions of the acknowledgments from node B to node A precede the transmission of the response message, therefore increasing the RTT.

4.3.2. Throughput. The results of the throughput tests are presented in Figure 7. Again, we compare ZigBee with the two POBICOS transport modes.

From the figure we immediately observe that the POBICOS best-effort mode seemingly outperforms ZigBee, while both achieve a throughput around 42 kbps. Obviously, this must be considered as measurement inaccuracy, and we can conclude that the overhead introduced by the POBICOS

Table 4: Data cache measurement results with TinyOS Blink application.

	Data cache off	Data cache on	Improvement (%)
Data cache accesses	58062	58062	N/A
Total CPU cycles	536046	137423	74.364
Data dependency stall cycles	395451	36	99.991
Data cache miss cycles	58054	2	99.997

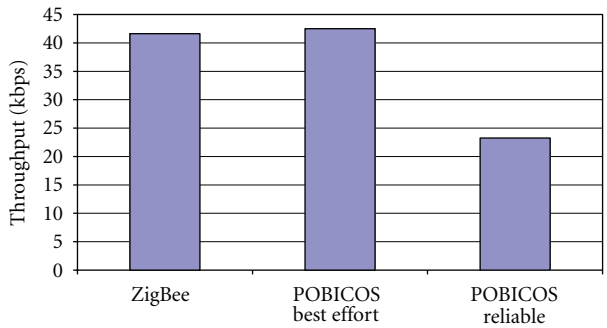

Figure 7: Results of the throughput tests.

best-effort mode is negligible. The POBICOS reliable mode achieves a noticeably lower throughput of 23.2 kbps which can be explained by the occasional packet loss during the measurements and the relatively long resending timeout of 7 seconds.

4.3.3. Bandwidth Analysis. Similarly as the duty-cycle measurements, the results presented in this section are extracted from the real-world experiment. In the experiment, a total of 61 POBICOS nodes were used to run an example application. This bandwidth analysis is performed at the network level, that is, we present the bandwidth usage of the whole network instead of individual nodes. The node bandwidth usages were calculated with one minute intervals and summed together to form the network-level bandwidth usage. It must be noted that the results do not include the automatic retransmissions of the POBICOS reliable transport protocol. Figure 8 presents the bandwidth usage during the whole experiment which lasted four days.

As seen from the figure, the bandwidth usage is close to constant with the exception of the peak at the application start-up phase where most of the microagent creations and resource probings take place. The reduction in the bandwidth usage at approximately 5 : 30 on the second night is merely a statistical anomaly. It is caused by resource probing multicasts distributing over two measurement periods whereas in the start of the experiment all the multicast messages are sent within one measurement period.

Next, we will take a closer look at the bandwidth usage in the system start-up phase, which is depicted in Figure 9. The figure shows the individual bandwidth usages of the middleware's internal protocols with different colours stacked on top of each other while the envelope of the curve corresponds to the total bandwidth usage.

The small system inspection and multicasting load between 15 : 24 and 15 : 27 is caused by the PAM tool upon its start-up phase where it collects information from the nodes of the network. After that, we can see that the middleware idles as there is no application running. The application is started at 16 : 08 which introduces a bandwidth peak that reaches its peak around 850 Bps. The peak is mostly caused by the microagent host probing messages, sent via the multicasting protocol, and microagent binary transfers from the application pill to the host nodes. The application deployment finishes at 16 : 33 after which we see small agent-manager traffic that encompasses the application-level messages.

The bandwidth usage during one hour of normal operation is plotted in Figure 10. Again, the plot is stacked so the envelope of the curve corresponds to the total bandwidth usage.

The bandwidth during normal operation comprises agent manager messages that originate from the application. The multicast peaks are also caused by the application logic, which polls for new temperature and brightness sensor microagent-candidate hosts every 10 minutes.

The results of the experiment startup and the normal operation suggest that there would be room for optimization in the multicast-based host-probing protocol as it dominates the bandwidth usage compared to the application traffic which averages below 10 Bps. Another major bandwidth user is the microagent binary transfer protocol. This is expected as all the microagent binaries are transmitted at runtime over the air from the application pill.

5. Related Work

The research done in overall performance evaluation of WSN middleware implementations is rather limited. The most relevant scientific overall study to our best knowledge is the work by Ribeiro et al. [5] in which the performance of SensorBus is studied. SensorBus is a message-oriented adaptive middleware running on Crossbow's MICAz motes with TinyOS. The measured metrics include throughput, packet delivery fraction, motes' energy consumption, and policy initialization response time in case of an external service request. The throughput and packet delivery fraction results can be used to compare the performances of different multihop routing protocols while response time and energy consumption results provide comparable results with our study.

Santos et al. and Bertocco et al. [6, 7] provide measurement results from simple WSN experiments without a middleware layer. The setup in [6] consists of Crossbow's TelosB motes with Contiki OS. The results obtained in the study provide basic reference to one-hop data-gathering WSN application without advanced self-adaptation functionalities. The experimental evaluation conducted in [7] is based on Moteiv's Tmote Sky motes running custom high-layer, single-hop, master-slave, industrial-monitoring

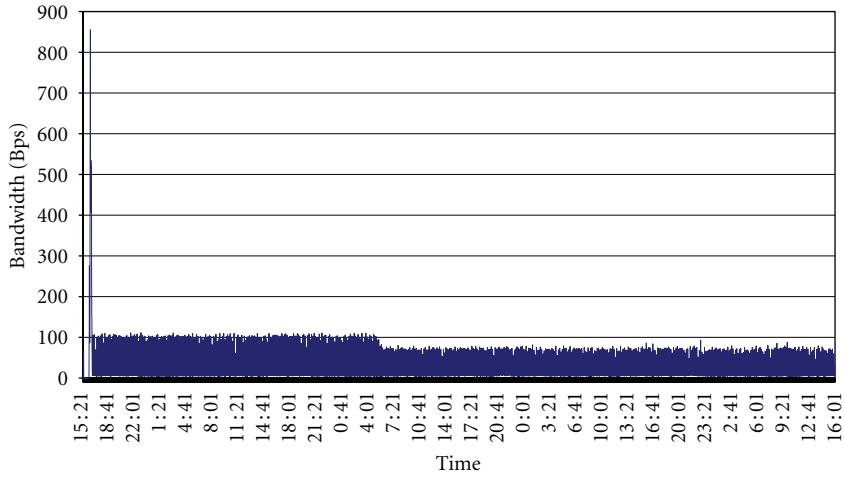

Figure 8: Bandwidth usage during the whole experiment.

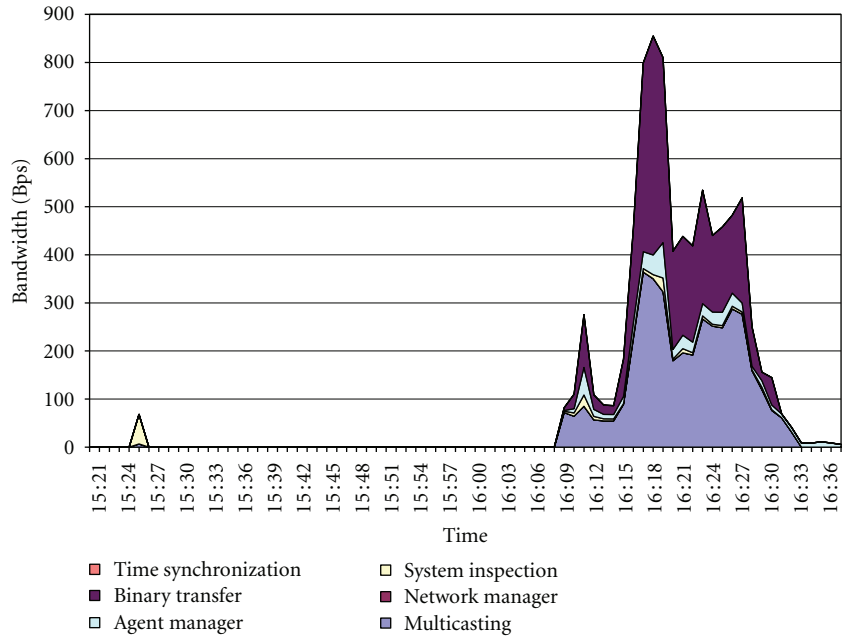

Figure 9: Bandwidth usage during the experiment start-up phase.

protocol which performs two types of tasks: periodical slave polling for receiving sensor data and asynchronous alarm transmissions. The results can be used to estimate the effect of radio interference on both types of tasks.

The performance of the underlying physical and MAC layers under real-world conditions has a dramatic effect on the WSN overall performance. Therefore, the studies performed in [8, 9] provide valuable resources to analyze our measurements. The testbed used in [8] consists of MICAz motes with TinyOS. It was shown that applying the testbed to practical environments is feasible, and the guidelines for the placement of the motes were given. Woon and Wan [9] present realistic experiments on both one-hop and multihop topologies with Freescale MC13193 Evaluation Kit. It presents comparable performance metrics such as throughput, packet delivery ratio, and delay, and it also shows that experimental results are valuable compared to normal simulated environments.

There are some published WSN performance measurement and verification tools such as [10–12]. Rost and Balakrishnan and Ramanathan et al. [10, 11] introduce online network management tools but their main focus is on failure detection. On the other hand, Zheng [12] proposes to apply formal verification techniques to ensure the correctness of the implementation using model checking techniques. These techniques provide valuable knowledge

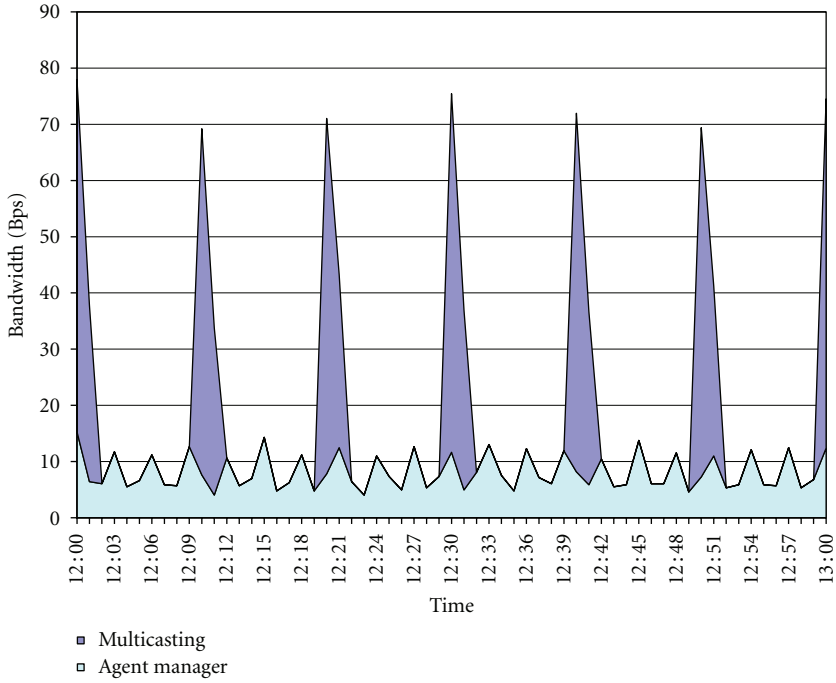

Figure 10: Bandwidth usage during 1 hour of normal operation.

on real-time behavior details of the system, but we are more interested in performance of the distributed WSN application as a whole.

The duty cycle of the motes is an important aspect of the WSN performance. Profilers can be used to obtain the details of the processor usage such as in [13] where the OS is interrupted frequently to collect the currently running task and in [14] where activity tracking across the network is monitored. The perceived duty cycle combined with offline energy measurements can be used to estimate the overall WSN energy consumption. Also real-time energy measurements can be achieved by utilizing hardware built-in switching regulators as in [15] and by using XScale PMU events as in [16]. The reference results for offline measurement with Mica2 motes can be obtained from [17].

6. Conclusions

The full performance evaluation of a WSN middleware implementation requires an extensive set of methods and tools which are able to measure low-level operations such as PMU events and high-level effects such as communication overheads. In addition, the measurements should not interfere with the operation of the running middleware. Distributed methods were found to be efficient when combined with offline measurements such as the presented energy measurements.

The preferred CPU clock frequency in terms of energy efficiency was found to be dependent on the available power modes. The influence of the data cache to the CPU usage performance was found to be dramatic. Our implementation shows also some deficiencies in terms of energy efficiency and initialization sequence duration that can be caused by the usage of separate ZigBee radio board.

The communication measurements suggest that the most crucial target for optimization would be the multicasting protocol which is used by the binary transfer and host object probing services of the middleware. In addition, it was observed that the underlying ZigBee network may induce heavy packet loss which severely affects the throughput of the reliable transport mode of POBICOS due to a long resending timeout.

The future work includes improvements in energy efficiency. For energy-efficient operation of POBICOS the radio board power-saving modes should be taken into use. Currently, all the motes are acting as ZigBee routers and for the routing purposes they are in continuous listening state. Energy savings would be achieved if some of the motes were ZigBee end devices or if the ZigBee routers had their power-saving modes enabled with synchronized sleeping periods.

Acknowledgments

The authors would like to thank the POBICOS consortium partners involved in the middleware design and implementation, Warsaw University of Technology (Poland), and Center for Research and Technology Thessaly (Greece). This work was done in the framework of the EU FP7 Project POBICOS supported by European Commission and VTT Technical Research Centre of Finland.

References

[1] http://www.ict-pobicos.eu/.

[2] A. Pruszkowski, T. Paczesny, and J. Domaszewicz, "From C to VM-targeted executables: techniques for heterogeneous sensor/actuator networks," in *8th IEEE Workshop on Intelligent Solutions in Embedded Systems (WISES '10)*, pp. 61–66, July 2010.

[3] N. Tziritas, T. Loukopoulos, S. Lalis, and P. Lampsas, "Agent placement in wireless embedded systems: memory space and energy optimizations," in *IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum (IPDPSW '10)*, 2010.

[4] P. Tarvainen, M. Ala-Louko, M. Jaakola et al., "Towards a lightweight security solution for user-friendly management of distributed sensor networks," in *9th International Conference on Next Generation Wired/Wireless Networking, and 2nd Conference on Smart Spaces*, S. Balandin, D. Moltchnov, and Y. Koucheryavy, Eds., vol. 5764 of *Lecture Notes in Computer Science*, pp. 97–109, September 2009.

[5] A. R. L. Ribeiro, L. C. Freitas, C. R. L. Francês, and J. C. W. A. Costa, "Middleware performance evaluation in wireless sensor networks," in *International Telecommunications Symposium (ITS '06)*, pp. 207–212, September 2006.

[6] A. Santos, A. Cardoso, and P. Gil, "Poster abstract: a case study on performance enhancement in WSN using Contiki OS," in *European Conference on Wireless Sensor Networks (EWSN '10)*, 2010.

[7] M. Bertocco, G. Gamba, A. Sona, and S. Vitturi, "Experimental characterization of wireless sensor networks for industrial applications," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 8, pp. 1537–1546, 2008.

[8] K. E. Tepe, P. R. Casey, and N. Kar, "Design and implementation of a testbed for IEEE 802.15.4 (Zigbee) performance measurements," *EURASIP Journal on Wireless Communications and Networking*, vol. 2010, Article ID 103406, 2010.

[9] W. T. H. Woon and T.-C. Wan, "Performance evaluation of IEEE 802.15.4 wireless multi-hop networks," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 3, no. 1, pp. 57–66, 2008.

[10] S. Rost and H. Balakrishnan, "Memento: a health monitoring system for wireless sensor networks," in *3rd Annual IEEE Communications Society on Sensor and Ad hoc Communications and Networks (SECON '06)*, pp. 575–584, September 2006.

[11] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the Sensor Network Debugger," in *3rd International Conference on Embedded Networked Sensor Systems (SenSys '05)*, 2005.

[12] M. C. Zheng, "An automatic approach to verify sensor network systems," in *4th IEEE International Conference on Secure Software Integration and Reliability Improvement Companion (SSIRI-C '10)*, pp. 7–12, June 2010.

[13] M. K. Watfa and M. Moubarak, "Building performance measurement tools for wireless sensor network operating systems," in *7th International Conference on Advances in Mobile Computing and Multimedia (MoMM '09)*, pp. 599–604, December 2009.

[14] R. Fonseca, P. Dutta, P. Levis, and I. Stoica, "Quanto: tracking energy in networked embedded systems," in *8th USENIX conference on Operating systems design and implementation (OSDI '08)*, 2008.

[15] P. Dutta, M. Feldmeier, J. Paradiso, and D. Culler, "Energy metering for free: augmenting switching regulators for real-time monitoring," in *International Conference on Information Processing in Sensor Networks (IPSN '08)*, pp. 283–294, April 2008.

[16] G. Contreras and M. Martonosi, "Power prediction for intel XScale® processors using performance monitoring unit events," in *International Symposium on Low Power Electronics and Design (ISLPED '05)*, pp. 221–226, August 2005.

[17] M. Calle and J. Kabara, "Measuring energy consumption in wireless sensor networks using GSP," in *17th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC '06)*, September 2006.