Hindawi Publishing Corporation
International Journal of Distributed Sensor Networks
Volume 2015, Article ID 607143, 9 pages
http://dx.doi.org/10.1155/2015/607143

Research Article

Native Process Migration in Wireless Sensor Networks

Syed Ishtiaq Hussain,[1] Huma Javed,[1] Tehseen Khan,[2] Sara Shazad,[1] and Falak Naz Khalil[1]

[1]*Department of Computer Science, University of Peshawar, Peshawar, Pakistan*
[2]*Department of Computer Science, FAST National University Peshawar, Peshawar, Pakistan*

Correspondence should be addressed to Syed Ishtiaq Hussain; ishtiaquop@yahoo.com

Received 27 April 2015; Revised 15 August 2015; Accepted 16 August 2015

Academic Editor: Mohammad M. Hassan

Copyright © 2015 Syed Ishtiaq Hussain et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

This paper presents a novel architecture for native process migration (PM) in wireless sensor networks (WSNs) without the use of virtual execution environment. Resources in WSN are scarce; therefore creating virtual execution environment puts extra burden on already stringent resources. In addition, the proposed architecture is migrating with complete process instead of code only which also saves resources. The proposed architecture makes process migration decisions by continuously monitoring resources, such as remaining battery life and free memory space on a node. The architecture is suitable for networks with fewer expensive sensor nodes as it allows for better utilization of network resources. Transferring a live executing process from one node to another to meet processing demands dynamically improves fault tolerance, resource utilization, and network management in WSN. The architecture has been successfully tested and implemented on both COOJA simulator and a test bed of TelosB motes.

1. Introduction

Wireless sensor networks (WSNs) are distributed networks of sensors nodes typically comprising of a large number of communicating nodes. A sensor node is designed to be physically small and of low cost; therefore it has limited computational power and small memory space. It is equipped with short range wireless radio [1] for exchanging sensed data to other nodes on the network. WSNs find their use in applications needing measurements and monitoring of environmental attributes such as temperature, humidity, light, pressure, sound, acceleration, orientation, vibration, smoke, radiation, and geographical location. They find their applications in health, medicine, military, environment, and industrial environments for measuring and monitoring physical attributes.

In WSN nodes failure is a very frequent and common phenomenon. If a node fails the process or data are lost. In case of physical destruction the data and programs cannot be recovered but in case of depleted batteries recovery is possible. As soon as the batteries life crosses a certain threshold the sensors can transfer data and code to another node having more energy or resources. It can transfer not only data and programs but also processes which have been processed partially and can save more energy.

Process migration is the task of moving a process between nodes during execution. Advantages of process migration include dynamic processing load distribution [2], improved fault tolerance [3], easier system administration, resource sharing, data access locality, and distributed computing [4].

Process migration is generally accomplished in the following generic three steps [4]: (a) suspending a process to be migrated at the source node, (b) sending process state, memory space, and executable code over the network to destination node, and finally (c) restoring process using information received from source at target node.

Process migration has been done in WSN in agent based systems such as Agilla using virtual environment. Virtual environment consumes resources which is an expensive overhead in WSN because of its very limited resources. An implication and comparison of energy consumption of different strategies including native and virtual machine based code execution are given in Dunkels et al. [5]. The authors have compared Java VM based code and a native code energy consumption and conclude that native code outperforms VM code in terms of energy efficiency for extended duration tasks. It is therefore established that native code is better at energy saving than agent code.

Table 1: Comparison of WSN operating systems and middleware.

	TinyOS	BerthaOS	MagnetOS	Agilla	Mate	Contiki	Proposed system
Code migration	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Process migration	No	No	No	Yes	No	No	Yes
Platform heterogeneity	No	No	Yes	Yes	Yes	No	No
Virtual machine	No	No	Yes	Yes	Yes	No	No
High energy and memory space consumption	No	No	Yes	Yes	Yes	No	No

The work presented in this paper focuses on creation of a resource aware architecture for process migration in homogenous WSNs [6] without using virtual environment. The proposed process migration architecture enables native live (running) processes to migrate between nodes. We have used Contiki as a platform for implementing and experimenting with process migration and further extend runtime dynamic linking [5] in Contiki.

The rest of the paper is organized as follows. Section 2 reviews process migration in various operating systems as well as middleware and motivates the development of process migration in WSNs by highlighting several application scenarios for process migration in practical applications. Section 3 shows our proposed architecture for process migration in WSNs. Evaluation of our process migration architecture is presented Section 4. We also compare our architecture to other known architectures of WSNs in Section 4.1. Finally future work and conclusion are presented in Section 5.

2. Process Migration in WSN

This section presents a brief review of process migration. In the past process migration has been extensively used in heterogeneous and homogenous distributed systems [4, 7]. Recent efforts focusing on process migration include LinuxPMI [8] for Linux clusters, [9] CRIU (Checkpoint/Restore in Userspace) for Linux systems [9], DMTCP (Distributed Multithreaded Checkpointing) [10], and BLCR (Berkeley Lab Checkpoint/Restart) [11]. In smart phones and mobile computing a framework for migration of android application to Cloud has also been proposed [12].

Process migration architectures can be divided into two types. The first type of architectures is based on native process migration, whereas the second type of architectures is based on virtual machines where applications run on virtual machines and are migrated from one machine to another while the application is active. When a process has to be migrated, both the code and the current state of the process have to be transferred across the network. The most important components which are transferred are the following: data segment, code segment, stack segment, and resource handles (unique identifiers to resources on the system) such as open file and network sockets. In addition, process control-block contains instruction pointer, process priority, process metrics, and privileges, as well as thread control block. In the virtual machine based systems the applications are often implemented as agents.

An agent is a program which runs on a virtual machine. The use of virtual machine makes agent programs highly portable and more secure. The agent can move from one machine to another during execution on a virtual machine [7].

Agent and code migration have been implemented in various WSN middleware and operating systems. Table 1 gives a brief comparison of the WSN middleware and operating systems which include BerthaOS [13], MagnetOS [14], Agilla [15], Mate [16], and Contiki [17]. Several Java virtual machines have also been implemented on embedded systems which include AOT (Ahead-of-Time) compilation [18, 19], Darjeeling JVM [20, 21], and TakaTuka JVM [22, 23]. Agilla [15] is a middleware supporting self-adaptive applications for WSN using mobile agents and allows agents to move from one sensor node to another during execution. Besides WSNs, agents also find widespread roles in gathering high quality information on the Internet, such as database searches, online transactions in e-commerce, and system diagnostics [15].

Although agents are very useful, the primary disadvantage of agent based paradigm, such as Agilla, is execution inefficiency such that programs take longer time to execute because the program has to be interpreted by the virtual machine, thereby taking more time, using more energy, and using more memory space. In comparison, native code is much faster and uses less energy and space [5], and it can be suggested that short duration tasks are more suitable to be run as agents (virtual machine based application programs), while long running tasks should be executed as native code.

Being resource constrained, the sensor nodes can be affected by the following inherent vulnerabilities:

(a) Low battery.

(b) Low memory.

(c) Hardware failure.

Common energy sources for powering sensors nodes are battery and solar cells. Because battery has a finite amount of energy to deliver, energy efficiency is very critical to the operation of sensor node. In practical deployments, the battery levels on nodes in a deployed WSN differ from one node to another, as all nodes cannot have the same level of remaining energy at a given moment, because of varying communication patterns and workloads on individual nodes in the network. If on a given sensor node battery is low and cannot sustain node operation any longer, a mechanism can be devised to transfer long running processes by means of process migration to another node which is idle and has more or continuous energy supply from a solar cell [8].

Nodes with continuous energy source are better for running computationally intensive jobs when the battery is being

charged and excess energy is used up for running application on sensor node. Moreover, on solar powered sensor nodes, some nodes may be under full sunlight, whereas some nodes might be in partial or full shade. For nodes which are not receiving sunlight, conserved use of battery is a requirement to remain functional for a longer duration. In such a situation, if there are several processes that are executing on the node, highly active processes should preferably be moved to those nodes that have continuous energy supply and might run the processes at full processor speed. This can help in preserving energy resources of those nodes which do not have access to continuous energy source such as sun. In addition, it is more efficient to compute information locally instead of sending it to base station.

Similarly memory is also of limited size and is very crucial resource in WSN. If a wireless sensor node is running several processing threads simultaneously, very little free memory may be left. In addition, if a process demands more memory which is not available on the host node, the requesting process can be migrated to another node having more memory.

Process migration can offer the following broad advantages in WSNs.

2.1. Ease of Deployment. New sensor applications can be programmed dynamically using process migration. An instance of process can be cloned across a group of nodes to set up new sensor application. This can also help in dynamically changing the configuration of the network. Some processes may move through the network and perform configuration or maintenance tasks on sensor nodes in an agent like manner. Such processes do not need to be fully installed on each node but migrate from node to another which saves code space on the entire WSN.

2.2. Resource Management and Distributed Processing. Nodes in a wireless sensor network may have different resource levels, such as battery level or memory available. A process can move to more resourceful nodes if it needs more resources. Long running specially can benefit from process migration as it allows transparent resumption of computational work after migration to another node. Process migration can also help in distributing tasks among nodes in a distributed processing by cloning initialized tasks among sensor nodes. In data aggregation tasks such as cluster heads, clustering tasks can be migrated from node to another instead of being installed on every node in the network, thus saving significant amount of code space across WSN nodes.

2.3. Ubiquitous Computing. The ubiquitous computing [24] and more specifically Internet of Things (IoT) [25] emphasize computing everywhere and all the time by small powerful communicating nodes. WSNs are part of ubiquitous computing and IoT environments, allowing nodes to sense their environment, communicate with each other, and react to changes in the surrounding environment. Process migration can allow long running processes to migrate from node to another and allow administrative tasks to be performed in ubiquitous networks. Hence process migration can play vital role in these systems by allowing task distribution and management of network wide resources.

Process migration can be difficult when a resource dependency exists; security is also a concern. In the case of resource dependency, when a process migrates, it should transfer from one node to another along with all of its moveable resources. An unmovable resource creates resource dependency and can prevent a process from migrating to another node. Instances of such resources dependencies include open files and node specific sensors (only available on a particular node). Finally the cost can increase when migrating a large process, because its code has to be transmitted; however, data compression techniques can be used for the purpose of reducing the amount of data to be transmitted. Security issues with process migration can be solved by using security protocols such as SPINS [26] and LEAP [2].

3. Process Migration Architecture

Although process migration is a relatively old feature in various Unix-like operating systems, no known process migration mechanism has been developed for any of the WSNs operating systems. The main component of all process migration architectures is process checkpoint and restore [9] mechanism. Checkpoint stores the state of a live running process and restore is responsible for restoring an instance of checkpointed process from stored state of a process. The process state data is typically stored in a file which is later on used for restoring the process. The file typically stores stack, heap, and registers. The restorer can recover the state of a process from this file and resume the execution of checkpointed process.

Checkpointing can be implemented by user library or may be provided as an operating system service. When implemented in operating system kernel as a service, it can facilitate the suspension and resumption of running process for very long durations over multiple reboots of machine. Example of actively developed system for checkpointing is CRIU (Checkpoint/Restore in Userspace) [27], which provides checkpoint/restore for Linux processes. CRIU allows the user to checkpoint and restore a running process or groups of processes. Checkpointing is the core component of process migration implementation. In our architecture checkpoint and restore have been implemented as a middleware service and are used for storing process execution state for a later resumption. Checkpoint and restore concept can be applied in wider context to the whole WSN. Brouwers et al. [21] present an architecture for checkpointing and restoring the state of full WSN.

The proposed architecture for process migration has been split into several individual components. Figure 1 shows the complete architecture of process migration middleware for WSNs. The architecture is based on generic steps common to all process migration system [4].

The central component of our architecture is *migration manager*. It provides support and functionality for various tasks required in migration of a process, such as checkpointing, restoration, serialization, broadcasting, listening to incoming migration requests, and transportation of process code image and state data. The resource monitor has two

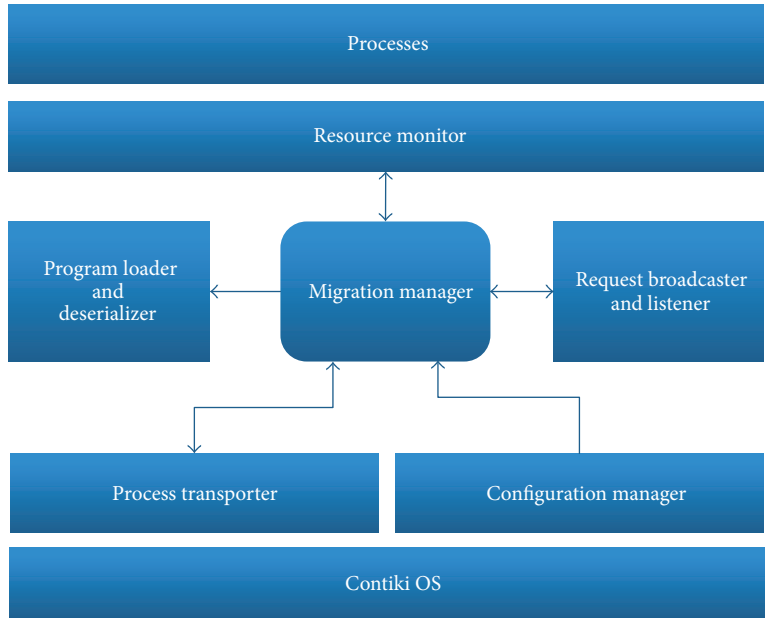


Figure 1: Process migration architecture for WSN.

responsibilities, estimating battery charge and monitoring free memory. Battery charge estimation is done by battery estimator which is responsible for predicting the amount of charge left in the battery and determines how much time the battery can sustain the node operation. It uses battery voltage sensor to periodically sample the battery voltage. The *request broadcaster and listener component* is responsible for broadcasting migration request to other nodes on the network. The listener waits for migration request broadcasts being "request broadcast listener." It continuously listens to requests on broadcast channel. After the request has been received by the *request broadcast listener* it replies back with acceptance message to the sender. The sender then selects a target node and then transmits the process state and executable code using *process code image transporter*. After the process state and code have been received at the target node, the program loader initializes process space from image received, and then state deserializer restores process's local and global variables. After process has been restored on the target node, it is resumed by the migration manager by adding it to the scheduling queue.

Steps for migrating a process are shown by activity diagram in Figure 2.

This is described in detail by steps (a) to (h). The Source_Migration_Manager suspends the process to be migrated and issues a migration request to Target_Migration_Manager which then replies with acceptance message. When the acceptance message is received by the Source_Migration_Manager, it saves a *process image* containing process code and state. The *process image* is sent to Target_Migration_Manager which then creates a new process and initializes it with the code and state of migrated process. Next the migrated process resumes execution. At the same time an acknowledgment of successful migration is sent to Source_Migration_Manager, which then destroys the original process.

(a) The source node wanting to migrate out a process continuously monitors its battery and memory space. The battery charge is estimated by sampling battery voltage at regular fixed intervals. In the event of detecting low battery power or memory space beyond a user specified threshold it initiates process migration procedure. Process migration begins with node "A" broadcasting a request for migrating a process *P*. Any node that has the requested resources available (such as battery) can send back acknowledgement. The reply from accepting node is sent using a unicast reply channel. The replying node first makes sure that it can accommodate the migrated process in memory and storage. After checking for resource availability, migration request is accepted or may be rejected in case resources cannot be allocated. The sender node selects the physically nearest destination node if multiple replies are received. On the receiver side, if multiple requests are received, they are processed in the order of arrival one at a time. Before proceeding to next request the receiver has to complete the current migration request.

(b) The process being migrated is removed from parent source node. Process P is suspended and marked for migration. At this stage the parent node is ready for redirecting/rerouting any open communication channels held/opened by process P.

(c) Communication connections are redirected temporarily. This is done by storing incoming messages to the process P in a message queue and by delivering the message queue to process P after migration of P is complete. This step runs in parallel with steps (d), (e), and (f).

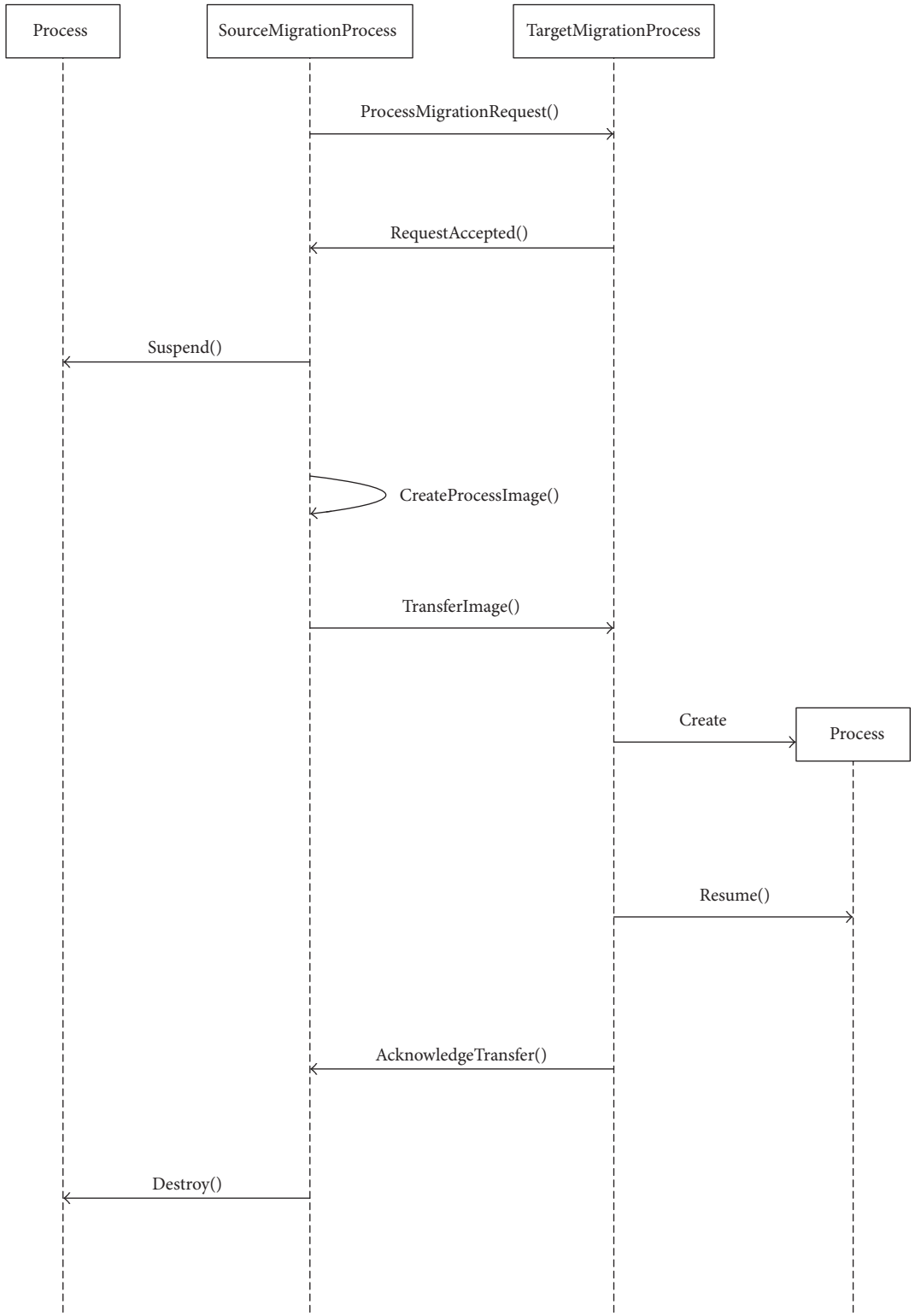


Figure 2: Basic process migration control flow.

```
WHILE (true)
BEGIN
  SET battery_charge = get_estimated_battery_charge()
  SET free_memory = get_free_memory_left()
  IF battery_charge < battery_threshold
OR free_memory < free_memory_threshold THEN
SET p = choose_a_process_to_migrate()
      suspend_process(p)
      initialize_message_queue(q)
SET p_data = get_process_local_data()
save_process_local_data(p_data)
SET R = create_request_for_process_migration()
      open_broadcast_connection()
      send_broadcast(R)
      wait_for_N acceptance_replies_from_other_nodes()
SET best_reply = choose_nearest_reply()
SET target_node = get_target_address_from_best_reply()
SET unicast_con = open_unicast_connection(target_node)
      send_data(p.image, target_node, unicast_con)
      send_data(p.state, target_node, unicast_con)
      close(unicast_con)
      close_all_process_data_connections()
      broadcast_new_location(p.name, target_node)
      send_message_queue(q, target_node)
      remove_process(p)
END
```

Algorithm 1: Algorithm for sender node.

```
WHILE (true)
BEGIN
   wait_for_incoming_broadcast_requests()
   IF migration_request_arrived() THEN
R = get_migration_request()
SET requested_memory = R.requested_free_memory
SET free_memory = get_free_memory()
IF free_memory ≥ requested_free_memory THEN
SET reply_node_address = R.source_address
          send(accept_reply, reply_node_address)
SET unicast_con = open_unicast_connection(listening_port)
      wait_for_incoming_connection(unicast_con)
SET rp = receive process(unicast_con)
SET p = create_process()
      save_process_image(rp.image, p)
      allocate_memory(sizeof(rp.state), p)
      copy_process_state(rp.state, p)
      close(unicast_con)
      link_symbol_table()
SET q = get_message_queue()
      open_process_data_connections()
      deliver_message_queue(p, q)
      add_to_scheduler(p)
END
```

Algorithm 2: Algorithm for receiver node.

(d) The state of process is copied (check-pointed). The process state includes process heap, global variables, process stack, processor state (register contents), and communication state (e.g., opened files and open network connections). This step is also called checkpointing as described in Section 2. The process state is kept on the source node until the successful completion of migration. If migration attempt fails due to some communication problem it may have to be reattempted.

(e) An instance of process is created at the target node into which the transferred state of process P will be copied. The new instance at the destination node is not started until complete state of process P has not been transferred from the source node.

(f) Process execution state is copied into the newly created instance (restored instance) on the remote node. This includes global variables, heap, and stack of process P. At the receiving end the target node unpacks the memory space and code from the received data. The migrating process is loaded into memory and resumes execution from the point of last suspension at the source node. The sender and receiver ends execute identical algorithms whose pseudocode is listed in Algorithms 1 and 2, respectively. The configuration parameters store permissions and user settings for controlling process migration. Configuration parameters allow the system to selectively block and unblock migration of different processes. The settings are stored in a text.

(g) Reference forwarding is done at the source node after the last step (f) to maintain open communication channels created by process P to other connected nodes in WSN. The new location of process P is broadcast in WSN by the source node which enables all other nodes previously connected to P to close connections to process P. Process P now opens new connections to premigration connected nodes. The message queue from step (c) is also delivered at this stage to the process P.

(h) The migrated process P is resumed at the destination node. This marks the successful completion of migration for process P. The resources held by process P after migration can now be released at the source node.

4. Implementation and Evaluation

The implementation of process migration architecture was tested on COOJA [17]. The communication between nodes was achieved using Contiki's Rime Communication Stack. The main thread in the program named pmig_process opened broadcast channels and unicast channels and performed node monitoring and managed the process migration steps. The first program tested was the LED blinker program which migrated for several times between randomly selected nodes.

In Figure 3 the lower panel of COOJA simulator window shows the led blinking on node 1 and node 2. Half of the red-led blinks are on node 1 and remaining half are on the node 2

Table 2: Comparison of proposed process migration architecture with other architectures.

	TakaTuka VM [22, 23]	Darjeeling [20, 21]	Dunkels et al. [17]	Agilla [15]	Proposed architecture
Based on Agents	No	No	No	Yes	No
Virtual machine	Yes (JVM)	Yes (JVM)	No	Yes	No
Dynamic linking and loading	Yes	Yes	Yes	Yes	Yes
Compact ELF	Class file(s)	Class file(s)	Yes	No	Yes
Live transfer	No	No	No	Yes	Yes
State transfer	No	No	No	Yes	Yes
Resource monitoring	No	No	No	No	Yes
Energy aware	No	No	No	No	Yes
Live process migration	No	No	No	Yes	Yes

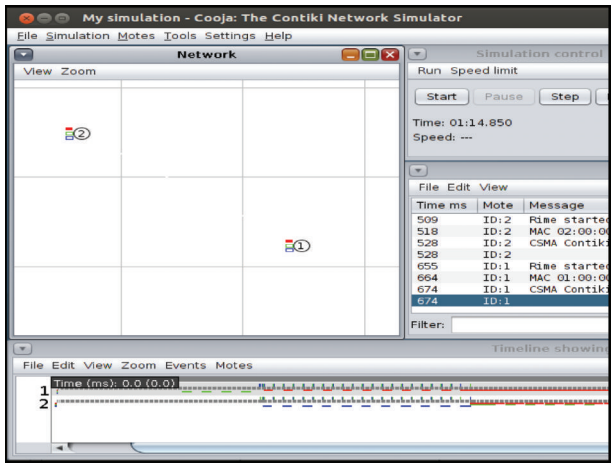


Figure 3: Process migration between two nodes in COOJA.

after migration completed which was initiated while the blinker process was running at node 1.

We also tested the proposed process migration system on a test bed of 25 TelosB nodes. In our implementation on TelosB nodes, we measured battery voltage using battery sensor in TelosB mote. Multiple readings were averaged to get approximate voltage reading. The measured voltage was used for estimating the battery remaining charge percentage.

4.1. Comparison with Related Architectures. In practical scenarios, nodes may be physically unreachable and reprogramming of the sensor nodes can be impossible or very difficult. To enable reconfiguration and reprogramming of software on WSN nodes, WSN middleware provides code management. Table 2 shows comparison of our work with other architectures including Dunkels et al. [17], Agilla [15] based on Mate [16], Darjeeling JVM [20, 21], and TakaTuka JVM [22, 23] for wireless sensor motes.

4.1.1. Code Migration. Code management in WSNs is typically achieved by code migration [16, 28] which allows the migration of application code from one node to another. This facilitates the remote installation of the software on nodes with ease and without the need of having physical access to nodes. Our implementation differs from code migration, because in addition to code migration it also supports the migration of the execution state which means that the whole process state is migrated. This allows the execution of migrated process to be resumed seamlessly on a different host node after migration has completed. This saves precious resources from being wasted.

4.1.2. Virtual Machine Agents. Process migration uses code migration. It incorporates the migration or transfer of state of executing process along with code from one sensor node to another in the WSN. By definition a live process has both native executable code image and state. The main focus in this work has been on the migration of natively executable processes along with state so that after migration the migrated process can resume execution at the target node. The native code is faster and more energy efficient [5]. This is similar to agent based systems in which code as well as the state of executing agent is migrated; however agent code is run on virtual machine and the code has to be interpreted at runtime. Therefore it requires extra resources to run the virtual environment.

An example of this strategy is Agilla [15] middleware which allows the mobile agents running on a byte code interpreter virtual machine to migrate state and executable code from one node to another node at runtime. It has been suggested that migrating small executable code modules consume less energy than a full application [29] and for shorter tasks this strategy is energy efficient, but for long running tasks the nature of code interpretation makes it less energy efficient. In real world situations virtual machine based applications typically consume more resources including time, space, and energy than the native applications. A comparison of the running times of different algorithms has been provided by Ellul [18, 19] and summarized in Table 3, which benchmarks Ahead-of-Time (AOT) compilation, with native and TakaTuka JVM. The increased time of execution indicates more energy consumed compared to native code.

4.1.3. Long Running Tasks. Levis and Culler [16] suggest that it is more energy efficient to implement long running and complex application programs with frequent executions as native code rather than as virtual machine based code, because of the overhead caused by virtual machine's interpreted execution. Furthermore the energy saving in the long term outweighs the energy consumed during the migration

Table 3: Comparison of VM with native code [18, 19] using BubbleSort16 benchmark for sorting 256 values.

	Execution time	Binary size	Energy consumed
TakaTuka	120 seconds	1050 bytes	587.76 mJ
AOT	13 seconds	110 bytes	63.67 mJ
Native Code	2 seconds	230 bytes	9.769 mJ

of large and complex processes needing frequent executions. The authors recommend that for smaller applications with small sized executable code and frequent migration requirements mobile agents are suitable. And for larger and complex applications, process migration is preferred.

5. Conclusion and Future Work

This paper proposed architecture for native process migration in WSNs. The proposed architecture incorporates resource monitoring and dynamic process migration. This architecture improves fault tolerance and energy consumption of a WSN network without using virtual environment which is an extra burden. We have tested and implemented our architecture on both simulation and real test bed using Contiki and TelosB successfully.

In the future we will incorporate a security mechanism in our architecture to make native process migration secure. Energy consumption can also be improved even further in the proposed architecture by using a compression algorithm for compressing process images before transfer takes place; see Ansel et al. [10]. We also intend to work on solving resource dependency problem and on improving resource monitoring to better estimate the resource capacity of wireless sensor nodes.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 2002.

[2] S. Zhu, S. Setia, and S. Jajodia, "LEAP+: efficient security mechanisms for large-scale distributed sensor networks," *ACM Transactions on Sensor Networks*, vol. 2, no. 4, pp. 500–528, 2006.

[3] Y. Artsy and R. Finkel, "Designing a process migration facility: the Charlotte experience," *Computer*, vol. 22, no. 9, pp. 47–56, 1989.

[4] D. S. Milojičić, F. Douglis, Y. Paindaveine, R. Wheeler, and S. Zhou, "Process migration," *ACM Computing Surveys*, vol. 32, no. 3, pp. 241–299, 2000.

[5] A. Dunkels, N. Finne, J. Eriksson, and T. Voigt, "Run-time dynamic linking for reprogramming wireless sensor networks," in *Proceedings of 4th International Conference on Embedded Networked Sensor Systems (SenSys '06)*, pp. 15–28, November 2006.

[6] J. Yick, B. Mukherjee, and D. Ghosal, "Wireless sensor network survey," *Computer Networks*, vol. 52, no. 12, pp. 2292–2330, 2008.

[7] H. Jiang and V. Chaudhary, "Process/thread migration and checkpointing in heterogeneous distributed systems," in *Proceedings of the 37th Annual Hawaii International Conference on System Sciences*, p. 10, January 2004.

[8] LinuxPMI, 2008, http://linuxpmi.org/trac/.

[9] C Team, "CRIU," http://criu.org/.

[10] J. Ansel, K. Arya, and G. Cooperman, "DMTCP: transparent checkpointing for cluster computations and the desktop," in *Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS '09)*, pp. 1–12, IEEE, Rome, Itlay, May 2009.

[11] P. H. Hargrove and J. C. Duell, "Berkeley lab checkpoint/restart (BLCR) for Linux clusters," *Journal of Physics: Conference Series*, vol. 46, no. 1, pp. 494–499, 2006.

[12] S.-H. Hung, J.-P. Shieh, and L. E. E. Chen-Pang, "Virtualizing smartphone applications to the cloud," *Computing and Informatics*, vol. 30, no. 6, pp. 1083–1097, 2012.

[13] J. Lifton, D. Seetharam, M. Seltzer, and J. Paradiso, "Bertha: the os for pushpin computers," Tech. Rep., 2002.

[14] R. Barr, J. C. Bicket, D. S. Dantas et al., "On the need for system-level support for ad hoc and sensor networks," *ACM SIGOPS Operating Systems Review*, vol. 36, pp. 1–5, 2002.

[15] C.-L. Fok, G.-C. Roman, and C. Lu, "Agilla: A mobile agent middleware for self-adaptive wireless sensor networks," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 4, no. 3, article 16, 2009.

[16] P. Levis and D. Culler, "Maté: a tiny virtual machine for sensor networks," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 85–95, 2002.

[17] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki—a lightweight and flexible operating system for tiny networked sensors," in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pp. 455–462, IEEE, November 2004.

[18] J. Ellul and K. Martinez, "Run-time compilation of bytecode in sensor networks," in *Proceedings of the 4th International Conference on Sensor Technologies and Applications (SENSORCOMM '10)*, pp. 133–138, July 2010.

[19] J. Ellul, *Run-time compilation techniques for wireless sensor networks [Ph.D. thesis]*, University of Southampton, Southampton, UK, 2012.

[20] N. Brouwers, P. Corke, and K. Langendoen, "Darjeeling, a Java compatible virtual machine for microcontrollers," in *Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion (Companion '08)*, pp. 18–23, ACM, Leuven, Belgium, December 2008.

[21] N. Brouwers, K. Langendoen, and P. Corke, "Darjeeling, a feature-rich VM for the resource poor," in *Proceedings of 7th ACM Conference on Embedded Networked Sensor Systems (SenSys '09)*, pp. 169–182, November 2009.

[22] F. Aslam, *Challenges and solutions in the design of a Java Virtual Machine for resource constrained microcontrollers [Ph.D. dissertation]*, Department of Computer Science, Faculty of Applied Sciences, University of Freiburg, Breisgau, Germany, 2011.

[23] F. Aslam, C. Schindelhauer, G. Ernst, D. Spyra, J. Meyer, and M. Zalloom, "Introducing TakaTuka: a Java virtualmachine for motes," in *Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems (SenSys '08)*, pp. 399–400, ACM, Raleigh, NC, USA, November 2008.

[24] U. Hansmann, *Pervasive Computing: The Mobile World*, Springer, 2003.

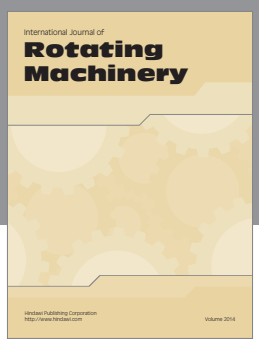
[25] J. Holler, V. Tsiatsis, C. Mulligan, S. Avesand, S. Karnouskos, and D. Boyle, *From Machine-to-Machine to the Internet of Things: Introduction to a New Age of Intelligence*, Academic Press, 2014.

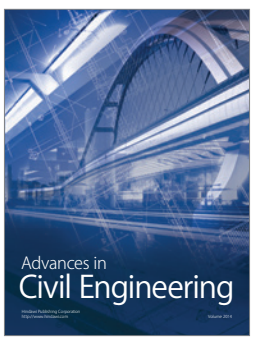
[26] L. Jing, F. Liu, and Y. Li, "Energy saving routing algorithm based on SPIN protocol in WSN," in *Proceedings of the 3rd International Conference on Image Analysis and Signal Processing (IASP '11)*, pp. 416–419, IEEE, Hubei, China, October 2011.

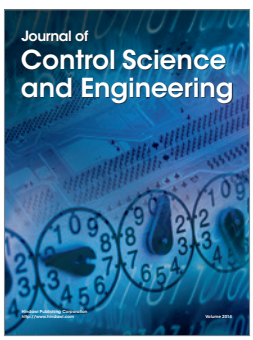
[27] W. Zhang, L. Chen, Q. Lu, P. Zhang, and S. Yang, "Flexible component migration in an OSGi based pervasive cloud infrastructure," in *Proceedings of the Service-Oriented Computing Workshops (ICSOC '13)*, pp. 505–514, 2014.

[28] M.-M. Wang, J.-N. Cao, J. Li, and S. K. Dasi, "Middleware for wireless sensor networks: a survey," *Journal of Computer Science and Technology*, vol. 23, no. 3, pp. 305–326, 2008.

[29] S. Hadim and N. Mohamed, "Middleware: middleware challenges and approaches for wireless sensor networks," *IEEE Distributed Systems Online*, vol. 7, no. 3, p. 1, 2006.







Hindawi
Submit your manuscripts at
http://www.hindawi.com