3-19-2012

A Unified Wireless Sensor Network Framework

Joshua Kiepert
Boise State University

Sin Ming Loo
Boise State University

 DOI: 10.1109/SysCon.2012.6189450

A Unified Wireless Sensor Network Framework

Joshua Kiepert
Department of Electrical and Computer Engineering
Boise State University
Boise, USA

Sin Ming Loo
Department of Electrical and Computer Engineering
Boise State University
Boise, USA

***Abstract*—Wireless sensor networks (WSNs) have been a significant area of research over the past decade. WSN systems are used in a wide range of applications such as surveillance, environmental monitoring, target tracking, wildlife tracking, personal health monitoring, machinery monitoring, and many others. With such wide ranging applications, there is active research in nearly every facet of the field including network topologies, communication protocols, node localization, time synchronization, and sensor data processing. This movement has largely been the result of the advances in microelectronics and low-power radio systems. These advancements have enabled the design and implementation of small, powerful, low-power, wireless sensor network systems. Like any emerging technology, a standard needs to be established to allow the advances in the field to be directly leveraged rather than requiring reinvention. This paper outlines the traditional approaches to WSN system design, and in contrast, proposes the necessary components of a unified WSN framework that would support the majority of present applications as well as providing the foundation for further advancements in the field.**

***Index Terms*—Wireless networks; wireless sensor networks; software architecture; operating systems; sensor systems**

I. Introduction

Wireless sensor network (WSN) systems have been a significant area of research over the past decade. This movement has largely been the result of the advances in microelectronics and low power radio systems. These advancements have enabled the design and implementation of small, powerful, and low-power WSN designs. Wireless sensor networks are composed of many individual nodes, each of which has (at a minimum) a small microprocessor, a wireless transceiver system, and a collection of sensors. The capabilities of the hardware and software present in WSN systems vary greatly depending on the application. WSN systems are used in a wide range of applications such as surveillance [1], environmental monitoring [2], [3], target tracking [4], [5], [6], wildlife tracking [7], personal health monitoring [8], [9], machinery monitoring [10], [11], and many others.

Active research is being conducted in nearly every facet of the field including network topologies, communication protocols, node localization, time synchronization, and sensor data processing. So, we must ask ourselves: How has researched shaped the development of WSN technology? In general, we see technology developed for many independent applications and research areas that tend to be designed with only a particular application or research need in mind. This thinking leads to vertically integrated designs. Of course, there are certainly advantages to vertical integration. Perhaps most noteworthy is the fact that such systems generally meet the needs of their intended application very effectively, and in many cases this may be more important than any other consideration. The problem with vertical integration is that it inherently prevents the research community from directly leveraging the engineering work of previous systems, thus slowing the further development of the technology as a whole. It does this by preventing interoperability between similar systems, restricting functionality to such specific domains that they can not be easily adapted to other areas.

Given these considerations, the need for a cross-platform, general purpose framework to support WSN systems becomes apparent. A similar problem presented itself during the development of modern computers and the Internet [12]. During that time, many different "standards" were developed by many vendors, all of which were designed to meet very similar goals. One could easily argue that the success of modern computers and the Internet are a direct result of the industry as a whole embracing a set of standards that meet the needs of the vast majority of applications. It is with the same motivation that we seek to define a practical, general purpose framework for wireless sensor networks. In the following we will take a look at the issues that are intrinsic to WSN systems in an effort to identify the key elements that would be required to establish a general purpose framework capable of both supporting WSN applications as we understand them today and provide the foundation for future advances in field. With the primary issues identified, this paper will conclude with a discussion about what research is still needed to successfully implement a unified, general purpose framework for WSN systems.

II. WSN Data Management

To establish a unified framework, it becomes necessary to identify the ways wireless sensor networks are being utilized and by what means the independent solutions solve those problems. Perhaps one of the most significant issues facing WSN systems is how to manage the vast amount of data that is available from sensor networks. Sensor networks range in size from just a few to thousands of nodes, but there are basically three schools of thought (although hybrid designs exist, e.g. [13]) as to how to manage the sensor data:

- Stream data to high-performance machines outside the sensor network to subsequently be processed [14]

Table I
WSN DATA MANAGEMENT SYSTEMS

	WSN Data Management System		
Characteristic	Stream Processing	In-Network Query	In-Network Processing
Average Network Traffic	High	Moderate	Application Dependent
Data Access Latency	Low	Moderate	Application Dependent
Computation on Sensor Data	No	Limited	Hardware Dependent
Scalability	Limited	High	High

- Treat the WSN like a distributed database and require the WSN to send out only the data required to answer specific structure query language (SQL) database type queries [15]
- Have the WSN itself process the data in-network, sending out only relevant information or alerts as needed [16]

Table I summarizes some of the primary differences between these data management systems.

As seen in Table I, each approach to WSN data management is compared across several characteristics: amount of network traffic, sensor data access latency, the ability to do computations on sensor data, and network scalability. The network traffic characteristic is based on what level of network traffic must occur for sensor data, or the results of sensor data processing, to be made available outside the WSN. The data access latency characteristic refers to time it takes for sensor data, or sensor data processing results, to be made available outside the WSN. The computation characteristic compares the relative ability to apply complex algorithms to sensor data by the WSN itself. Finally, the scalability characteristic provides a comparison of how effectively the network may be expanded to an arbitrary number of sensor nodes.

From Table I, we can see that in-network processing inherently provides much more control of characteristic behavior as compared with the other methods of sensor data management. Both the average network traffic and data access latency are controlled by the WSN application. The computation capabilities are limited only by the available computational resources on the WSN hardware platform, and as with in-network query, in-network processing can be highly scalable. However, other issues need to be considered. In the following sections we will briefly analyze each data management system and identify issues involved with each methodology.

A. Stream Processing

Stream processing provides several benefits but potentially many more liabilities. Primary benefits to stream processing include its relatively simple implementation and its effectiveness in small networks with a limited number of sensors. "Smart" control of the wireless channel can reduce the inherent network congestion. However, there is still an upper limit to the scalability of a network implementing direct streaming. This limit is determined by the bandwidth of the receiver and the bandwidth requirements of each sensor node. Other issues include packet-loss due to network congestion, inefficiencies both in terms of transmit power and data duplication, and finally, the throughput may be too slow to be processed and acted upon from outside the network. Processing latency issues arise due to the fact that, typically, no data analysis is done within a streaming system by the WSN itself. Instead, an outside system must aggregate data, perform analysis, and finally, act upon the results. These delays may be significant depending on the WSN system, and as such, a limited number of systems can effectively operate in this mode.

B. In-Network Query

In-network query processing provides a high-level abstraction for interaction with a sensor network. As discussed above, this system treats the WSN as a database which responds to SQL-type queries. When a query is injected into the network (typically one node is responsible for receiving such queries), the query is propagated through the network so that each node may supply their part of the result. To make this more efficient, the network internally keeps tables/indexes updated (even when no active queries are being processed). This makes it unnecessary for all sensor nodes to be involved in response to each individual query. The primary benefit of this type of communication is that it can allow the natural application of data aggregation algorithms such as *min*, *max*, *avg*, etc. In-network query also has the means to potentially reduce overall network traffic as compared to stream processing since most of the communication only takes place to answer an injected query. As we have previously alluded to, one area where in-network query falters is that, depending on implementation, it may still require significant continued communication outside of queries to main up-to-date indexes and tables. Additionally, there is significant software overhead required for implementation as well as somewhat limited usefulness for general WSN applications which require advanced analysis of sensor data beyond the simple, database-like, aggregation algorithms.

C. In-Network Processing

In-network processing effectively applies both the concepts of parallel processing and distributed systems to the embedded systems domain. It requires that each sensor node be capable of both responding with data directly when requested as well as accepting computation jobs which must be completed and returned. The benefits of this type of framework are many: duplicate data can easily be silenced, data can be aggregated before sending it out of the network, network communication can be directly controlled (and thus reduced), advanced

algorithms can be applied to the sensor data, and high-level abstractions such as in-network query can be supported. Moreover, this type of management system can support nearly all common uses of WSN systems as it provides a low-level control of each sensor node that can easily be abstracted to provide simple high-level management of the network as a whole. As with the other methods, in-network processing has some disadvantages. With sensor nodes participating in computation there are several side effects. Namely, the sensor nodes must have greater computational performance, and the power requirements may be increased since computation must be done in addition to normal sensor management. Since the programming interface is flexible, it can be inherently more complicated to develop programs in such an environment. Despite the drawbacks, we believe in-network processing is the best candidate for implementation of a unified framework that can be applied to the majority of WSN applications.

III. In-Network Processing Requirements

Having established a processing mechanism that can support many WSN applications, it then becomes necessary to not only look at the requirements for in-network processing, but also what other capabilities are needed in a system framework to support WSN systems as a whole. For in-network processing itself, there are several key features that are needed including error tolerant communication, a versatile parallel processing application programming interface (API), and a time synchronization mechanism.

A. Communications

The communications stack must support point-to-point, multi-cast, and multi-hop modes. Additionally, as this communication takes place over a potentially unreliable ad-hoc wireless link, we can only expect that it operates with a best-effort guarantee of delivery. As WSNs tend to be more dynamically formed than traditional networks, automatic network self-organization is important. The communications stack must support a wide range of network hierarchies to ensure that there is great flexibility for different applications and environments.

B. Parallel Processing API

The unreliable nature of communications in WSNs is a serious issue that must be directly addressed in any parallel processing API that is implemented for WSNs. Virtually all parallel processing APIs designed for traditional computer systems assume guaranteed delivery of messages. Thus, directly applying the same methods to the WSN environment is not feasible. Another complication for parallel processing within a WSN is the potential for a multi-hop network architecture. Any parallel processing API for WSNs must take into account the possibility of messages and data having to traverse several nodes before being received by the intended recipient.

C. Time Synchronization

Time synchronization in a WSN system is needed to provide a means of coordinating communication, correlating sensor data, and establishing coordinated power down sequences for optimizing power consumption of sensor nodes. The key features of an effective time synchronization system for WSNs include low overhead, accuracy, and an in-band implementation. Traditional time synchronization techniques such as network time protocol (NTP) do not perform well in the WSN environment, and the reasons for this are well outlined in [17]. In general, it is the unreliable nature and multi-hop architecture of wireless sensor networks that make the usual algorithms impractical. The required time-keeping accuracy is generally application specific, and thus, this metric should be optimized after meeting other requirements. The absolute accuracy of the time-keeping is not as important as having the sensor network self consistent to a fine degree of resolution since we are primarily concerned with scheduling and differentiating events within the sensor network itself. Finally, an in-band (i.e., operating within the same channel as all other communications) implementation is important to allow the framework to operate on a wide range of hardware platforms.

IV. Embedded Parallel Processing

In-network processing can only be successful if the embedded systems hardware is capable of providing the necessary computational power. One of the many benefits to the continuous advancement in microelectronics is that microprocessors continue to get faster, less expensive, and more power efficient. As such, developing WSN hardware that can meet the computational and power requirements of many applications is highly feasible. Moreover, the purpose of implementing parallel processing within WSNs is not to provide a high-performance parallel processing system but rather to provide a means of dealing with the distributed nature of the data collected within a sensor network. Much work has been done to show that parallel processing is possible within wire-networked embedded systems [18], [19] and FPGAs [20]. Significantly less research has been done on general purpose parallel processing with wireless sensor networks. Research on parallel processing within WSNs has primarily been done on purpose-built systems, e.g. [21]. While some work proposes general purpose parallel processing APIs such as [22], [23], [24], few provide frameworks that can be practically implemented. A common issue identified by such work is that implementation is not possible due to poor support for parallel processing within embedded operating systems. With the limited research done on general purpose parallel processing in WSNs, it seems clear that it will be necessary to expand work in this area.

V. A Unified Framework

Up to this point we have discussed the general direction of research in WSNs, and some general characteristics of WSNs have been outlined. A unified WSN framework will consist of an embedded operating system (OS), a fault-tolerant

communications stack, a parallel computing API, and a time synchronization system. In the following sections we will summarize the characteristics we believe are necessary to provide an effective WSN framework that could directly suit the needs of many applications and provide the foundation for new advancements in WSNs.

A. Embedded Operating System

As discussed previously, an embedded operating system capable of supporting the unique requirements of WSNs, including parallel processing, is needed. To be effective as a general solution to WSN systems, a number of features must be present in the embedded OS . An embedded OS must be portable to a wide range of hardware, provide a hardware independent driver interface, support both real-time and event-based multi-tasking, support dynamic application loading, and support a parallel processing API. It is highly probable that an existing embedded OS can be modified to fit these needs. Many embedded operating systems implement a subset of the requirements previously mentioned, thus, simply identifying those systems and assimilating the essential features is all that is required.

1) Hardware Portability: The operating system should be portable to a wide range of hardware, enabling its use on many different WSN systems. This requires that the operating system utilize as little hardware specific assembly language as possible. Additionally, a hardware independent driver interface is needed to ensure that WSN applications can be loosely coupled to the hardware that it was originally written to use. By providing a well defined driver interface, application programming is simple and identical across hardware platforms. Of course, hardware portability is often a primary goal of operating system designers. Thus, this requirement is easily met with most general purpose embedded operating systems.

2) Multi-tasking System: Support for both real-time and event-based tasks are important as there are many applications where one or both are required. In WSN systems, certain tasks may allow loose timing constraints. In such cases, tasks will operate correctly on a purely event-based system. On the other hand, tasks such as controlling certain types of sensors require servicing at precise intervals to avoid inaccurate results or damage to a sensor. When timing is critical, real-time task control is essential. Many WSN specific embedded operating systems choose to exclusively implement one or the other task management system. However, many general purpose embedded operating systems provide facilities for both type of scheduling.

3) Dynamic Applications: Dynamic application loading is needed particularly for large WSN systems. It is logistically difficult to update application software on each WSN node by taking each node offline. Often, sensor nodes are not easily accessible which compounds the problem. Remote application loading provides a solution to these problems. This feature can be difficult to implement as most embedded operating systems require that the application software be compiled with the operating system and then deployed to the hardware. Dynamic loading requires that the operating system, or at least a small part of the operating system is resident in a boot-loading section of memory so that, on power-up, the resident program may load the new application.

4) Data Storage: Typical embedded microprocessors have RAM capacities on the order of 0.5-96KB and program memory capacities on the order of 1-512KB. As such, it is necessary that the operating system work with limited memory resources. With such constraints, support for additional data storage is needed to temporarily buffer and process sensor data. Many embedded microprocessors support off-chip RAM expansion and physical interfaces for high-capacity storage mediums. In general, most any microcontroller will be capable of utilizing storage mediums such as Secure Digital (SD) or MultiMediaCard (MMC) to store data. The operating system should have support for easily storing and recalling data from a wide range of storage mediums.

B. Fault-Tolerant Communications

Fault-tolerant communication is an important feature for WSNs. WSNs are often dynamically formed, and furthermore the nodes themselves may not be statically located. The transient nature of WSN makes reliable communication difficult. If a node moves out of range of the current network formation then any outstanding results from sensor data requests or computing jobs can not be relied upon by the network. Much work has been done to build reliable communications in this type of environment utilizing low-power wireless systems such as IEEE 802.15.4 (Zigbee) [25]. Zigbee provides fault-tolerance in that it is able to dynamically re-route traffic around unresponsive nodes, and it is resistant to RF and multipath interference [26]. A WSN centric communications management system is still needed to ensure successful high-level interaction with any parallel processing framework that operates over the wireless channel.

C. Parallel Computing Framework

With an embedded operating system that supports parallel processing we, of course, need to outline a general purpose parallel computing framework to enable in-network processing applications. For parallel processing to be useful within WSNs, the programming environment must be similar to larger parallel computing frameworks, allowing researchers to leverage existing software knowledge when developing applications for the WSN environment.

1) Message Passing Interface: Perhaps the most ubiquitous platform for general purpose parallel processing is the message passing interface (MPI). For many years parallel processing was only performed within proprietary programming environments. Now, the MPI framework may be considered the de facto standard of cluster programming APIs. MPI is a language independent framework that supports a wide range of communications methods for inter-node communications. While MPI provides all of the required facilities to implement in-network parallel programming, the specification makes assumptions about its operating environment that are not necessarily true within WSNs.

2) MPI in WSNs: Several characteristics if WSNs make direct implementation of MPI difficult. As described earlier, MPI assumes there is guaranteed delivery of messages since it was designed to operate over TCP/IP based wired networks. As discussed in section V-B, such guarantees can not be made in wireless systems as there are many reasons that node-to-node communication may fail in a WSN. Power consumption and sensor tasks must take precedence over MPI tasks to ensure each sensor node remains effective. Sensor networks do not have the data storage resources of typical cluster systems, and moving data across the network is expensive in terms of both power and time. Clearly, each of these issues will need to be addressed for an MPI implementation within WSNs.

Limited work has focused on MPI implementations for WSNs. Most MPI implementations applied to embedded systems do not utilize a wireless communications channel. Some work utilizes message passing algorithms in WSNs, but the systems are highly focused on particular applications rather than on providing a general purpose framework that can be used for a wide range of applications.

D. Time Synchronization

As discussed in section III-C, we believe a successful time synchronization system will operate in-band to dispose of special purpose hardware. Of course, the WSN system designer has the prerogative to include special hardware for time synchronization, but a general purpose framework should not depend on such hardware. Several in-band protocols are promising for use in WSNs including RBS [27], TSPN [28], FTSP [29], and others. Various algorithms will need to be tested to establish a solution that provides the best compromise in terms of accuracy, resolution, overhead, and power consumption.

VI. Assembling the Framework

As outlined in section V, many potential components of the framework are already well researched, however, several practical issues need to be solved before a unified framework can be effectively implemented. The primary issues include: 1) developing a new framework architecture that provides both complete abstraction of underlying hardware and online application loading, 2) adapting a fault-tolerant MPI API to the WSN domain, and 3) identifying a method to allow computation and sensor operations to coexist on a WSN node while operating on a strict power budget.

A. Framework Architecture

The architecture of a framework is pivotal to providing hardware independent and thus, portable applications. Essentially the goal is to provide hardware independence for WSN applications similar to the way traditional computer systems provide hardware independence for general applications. Traditional computer system programming environments allow applications to be deployed to many different hardware platforms because the underlying operating system hides the differences from the application layer. With the correct framework architecture, we believe that online application loading can also be achieved while meeting the other requirements previously discussed. A standardized hardware driver interface packaged in combination with the operating system and application loader would allow new applications to be deployed to sensor nodes without removing them from the network. Figure 1 shows a simplified graphical representation of the envisioned system to better illustrate the relationships between the framework components.

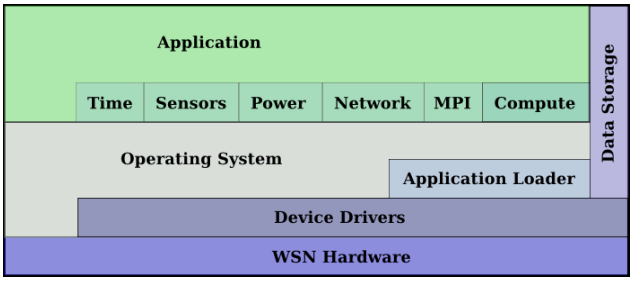

Figure 1. Framework Architecture

As shown in Figure 1, the device drivers provide a standardized interface to the underlying WSN hardware for the application loader and operating system. Only the device drivers and the operating system need to interact with the WSN hardware directly. Only a small part of the operating system needs to be hardware specific. The hardware dependent part of embedded operating systems is typically within the context switching and interrupt handling components. From Figure 1, we also see that the application and application level APIs are completely hardware independent, including time synchronization, sensor device access, power management, network management, compute management, and MPI. Finally, the data storage component in the diagram is shown to highlight the fact that this component will be accessed by most of the other subsystems. However, the data storage component remains hardware independent via the hardware driver interface.

B. MPI for Wireless Sensor Networks

As discussed in section V-C2, adapting an MPI framework to function reliably within a wireless sensor network is not a trivial problem. It is likely that only a subset of MPI will be practical to implement on WSN systems. Several groups have already simplified the standard MPI API for embedded systems, e.g. [18], [19], [20], [30]. Thus, we can build on this work to establish a MPI framework that is optimized for the WSN environment. Optimization of MPI for WSN systems will largely be associated with controlling power consumption and system resources to prevent computation from interfering with other sensor node operations.

C. Measurement with Computation

The issue of assigning wireless sensor nodes the responsibilities of both collecting sensor data and processing data

will require utilizing the multi-tasking features of the underlying embedded operating system. Many embedded operating systems provide robust multi-tasking mechanisms which will make managing both measurement and computation responsibilities easier. Beyond leveraging operating system multi-tasking, a computation management system is needed that can gracefully pause and resume (i.e., checkpoint) its operations (for resource sharing/power management) and interface with the WSN optimized MPI system.

VII. Conclusion

Based on the current state of development in wireless sensor networks, it seems clear that there is a need for a framework that provides a general solution for WSN systems. A number of areas still need to be investigated to establish such a framework. Many of the base components of such a framework are already well researched such as WSN time synchronization and WSN optimized communication protocols. This makes much of the development a matter of identifying components and determining the best way to interface them into a cohesive system. Primary research will need to be focused on expanding an embedded operating system to support both parallel processing and dynamic application deployment. Additionally, a practical MPI-like parallel processing API must be developed that is optimized for in-network processing in embedded systems. The culmination of this research will provide a means to unify WSN operating software across applications, and thus, provide a foundation on which to support current applications and future advancements in the field.

References

[1] F. Viani, G. Oliveri, M. Donelli, L. Lizzi, P. Rocca, and A. Massa, "WSN-based solutions for security and surveillance," in Microwave Conference (EuMC), 2010 European, 2010, no. September, pp. 1762–1765.

[2] A. Singh, L. S. Chyan, and P. Sebastian, "Sensor integration in a Wireless Sensor Network system for environmental monitoring system," in Intelligent and Advanced Systems (ICIAS), 2010 International Conference on, pp. 1–5.

[3] T. Zheng, Y. Qin, D. Gao, J. Duan, and H. Zhang, "A practical deployment of Intelligent Building Wireless Sensor Network for environmental monitoring and air-conditioning control," in Network Infrastructure and Digital Content, 2010 2nd IEEE International Conference on, 2010, pp. 624–628.

[4] J. Rushing, S. J. Graves, S. Tanner, and E. Criswell, "Real time target tracking with binary sensor networks and parallel computing," 2006 IEEE International Conference on Granular Computing, pp. 112-117, 2006.

[5] J. Zheng, W. Jia, G. Wang, and J. Wu, "Target Trajectory Querying in Wireless Sensor Networks," in Communications (ICC), 2010 IEEE International Conference on, 2010, pp. 1–6.

[6] X. Wang, J. Ma, S. Wang, and D. Bi, "Distributed Energy Optimization for Target Tracking in Wireless Sensor Networks," IEEE Transactions on Mobile Computing, vol. 9, no. 1, pp. 73-86, Jan. 2010.

[7] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein, "Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet," in ACM Sigplan Notices, 2002, vol. 37, no. 10, pp. 96–107.

[8] O. Postolache, P. Girao, M. Dias Pereira, G. Ferraria, N. Barroso, and G. Postolache, "Indoor monitoring of respiratory distress triggering factors using a wireless sensing network and a smart phone," in Instrumentation and Measurement Technology Conference, 2009. I2MTC'09. IEEE, 2009, no. May, pp. 451–456.

[9] O. Kwon, S. Shin, and W. Kim, "Design of U-Health System with the Use of Smart Phone and Sensor Network," in Ubiquitous Information Technologies and Applications (CUTE), 2010 Proceedings of the 5th International Conference on, 2010, pp. 1–6.

[10] R. K. Yedavalli and R. K. Belapurkar, "Application of wireless sensor networks to aircraft control and health management systems," Journal of Control Theory and Applications, vol. 9, no. 1, pp. 28-33, Mar. 2011.

[11] T. Becker et al., "Autonomous Sensor Nodes for Aircraft Structural Health Monitoring," IEEE Sensors Journal, vol. 9, no. 11, pp. 1589-1595, Dec. 2009.

[12] D. Culler et al., "Towards a sensor network architecture: Lowering the waistline," in Proceedings of the 10th conference on Hot Topics in Operating Systems-Volume 10, 2005, pp. 24–24.

[13] V. Vaidehi and D. S. Devi, "Distributed database management and join of multiple data streams in wireless sensor network using querying techniques," in Recent Trends in Information Technology (ICRTIT), 2011 International Conference on, 2011, pp. 583–588.

[14] Y. K. Lee, L. Wang, and K. H. Ryu, "A System Architecture for Monitoring Sensor Data Stream," in Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on, 2007, pp. 1026–1031.

[15] M. B. Ahmad, M. Asif, M. H. Islam, and D. S. Aziz, "A short survey on distributed in-network query processing in wireless sensor networks," 2009 First International Conference on Networked Digital Technologies, pp. 541-543, Jul. 2009.

[16] M. Bal, W. Shen, and H. Ghenniwa, "Collaborative signal and information processing in wireless sensor networks: A review," 2009 IEEE International Conference on Systems, Man and Cybernetics, no. October, pp. 3151-3156, Oct. 2009.

[17] J. Elson and D. Estrin, "Time synchronization for wireless sensor networks," Proceedings 15th International Parallel and Distributed Processing Symposium. IPDPS 2001, pp. 1965-1970, 2001.

[18] J. Liu, MPI for Embedded Systems: A Case Study. 2003, p. 12.

[19] R. Subramaniyan, V. Aggarwal, A. Jacobs, and A. George, "FEMPI: A Lightweight Fault-tolerant MPI for Embedded Cluster Systems," in Proc. International Conference on Embedded Systems and Applications (ESA), pp. 26–29.

[20] M. Saldana and P. Chow, "TMD-MPI: An MPI implementation for multiple processors across multiple FPGAs," in Field Programmable Logic and Applications, 2006. FPL'06. International Conference on, 2006, pp. 1–6.

[21] J. Schiff, D. Antonelli, A. G. Dimakis, D. Chu, and M. J. Wainwright, "Robust message-passing for statistical inference in sensor networks," Proceedings of the 6th international conference on Information processing in sensor networks - IPSN '07, p. 109, 2007.

[22] P. Ramanathan, K. Saluja, K. Wang, and T. Clouqueur, "UW-API: A Network Routing Application Programmer's Interface," Draft version 1.2, 2001.

[23] C. Lombriser, M. Marin-Perianu, R. Marin-Perianu, D. Roggen, P. Havinga, and G. Troster, "Organizing Context Information Processing in Dynamic Wireless Sensor Networks," 2007 3rd International Conference on Intelligent Sensors, Sensor Networks and Information, pp. 67-72, 2007.

[24] J. Horey and A. B. Maccabe, Designing a Dynamic Middleware System for Sensor Networks. 2005.

[25] S. B. Attia, A. Cunha, A. Koubaa, and M. Alves, "Fault-Tolerance Mechanisms for Zigbee Wireless Sensor Networks," in Work-in-Progress (WiP) session of the 19th Euromicro Conference on Real-Time Systems (ECRTS 2007), Pisa, Italy, 2007, no. 1, pp. 37–40.

[26] R. Alena, R. Gilstrap, J. Baldwin, T. Stone, and P. Wilson, "Fault tolerance in ZigBee wireless sensor networks," in Aerospace Conference, 2011 IEEE, 2011, pp. 1–15.

[27] J. Elson, L. Girod, and D. Estrin, "Fine-grained network time synchronization using reference broadcasts," ACM SIGOPS Operating Systems Review, vol. 36, no. SI, p. 147, Dec. 2002.

[28] S. Ganeriwal, R. Kumar, and M. B. Srivastava, "Timing-sync protocol for sensor networks," in Proceedings of the first international conference on Embedded networked sensor systems - SenSys '03, 2003, p. 138.

[29] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, "The Flooding Time Synchronization Protocol," Proceedings of the 2nd international conference on Embedded networked sensor systems - SenSys '04, p. 39, 2004.

[30] A. Agbaria, D.-I. Kang, and K. Singh, "LMPI: MPI for heterogeneous embedded distributed systems," in Parallel and Distributed Systems 2006 ICPADS 2006 12th International Conference on, 2006, vol. 1, p. 8 pp.