An Evaluation of Middleware Models with Focus on Constraints and Challenges in Wireless Sensor Networks

Majid Ali Khan

University of Tampere
Department of Computer Science
Computer Science
Master's Thesis
Supervisor: Eleni Berki

University of Tampere
Department of Computer Science
Computer Science
Majid Khan: An Evaluation of Middleware Models with focus on Constraints and Challenges in Wireless Sensor Networks.
Master's Thesis, 92 pages
March 2010

Wireless sensor networks (WSN) are long running distributed systems comprised of tiny devices called nodes. Nodes are capable of sensing, computing and communicating. These networks have been developed for wide range of applications including habitat monitoring, military surveillance, object tracking, building monitoring, smart offices and homes. WSN has many limitations that need to be taken into account, such as hardware constraints, limited energy, dynamic networks and unattended operations. These constraints raise the necessity of adopting a middleware.

Various middleware architectures have been proposed so far to achieve suitable abstraction from underlying hardware and networks. They focus on ease of application development and maintenance of wireless sensor networks. These middlewares provide the capability to support and coordinate concurrent applications and act as a bridge between hardware, operating system, network stacks and applications.

This thesis surveys some of the models and middlewares suggested so far. It evaluates the middlewares with focus on design challenges arise due to the distributed and resource constraint nature of the wireless sensor networks. It presents the comparison among various middlewares and addresses the differences as well as commonalities to find out the strengths and shortcomings in each model. This thesis shows that each middleware addresses only a limited subset of issues and design challenges. Each middleware or a group of middlewares, targets a specific set of issues based on the middleware's own design based on the type of programming abstraction they provide.

Keywords: Evaluation, Survey, Middleware, Middleware Models, WSN

Table of Contents

Table of Figures

Table of Tables

Acknowledgments

I wish to express my sincere thanks and appreciation to my supervisor Eleni Berki, for helping me throughout the thesis work and pushing me till the end and being so patient while I was writing the thesis and always guiding me in right direction. I would also like to thank Timo Poranen for his constructive comments, suggestions and guidance.

I would also like to extend my deepest thanks to Jaakko Kangasharju (Helsinki Universtiry of Technology) who introduced me to the field of wireless sensor networks and for guiding me in the right direction through initial phases of this thesis.

My greatest gratitude to my family who has always being a source of motivation throughout my life, especially my parents who are the guiding force of my existence. I am especially indebted to my wife, Wagma for being so patient for the last two years and her unconditional support all the times. I am grateful to the Almighty for his blessings. In the end my friends also deserve some credit for their ever cheerful presence; Aamir, Naveed, Soahil, Shahzad, and Tayyab.

1 Introduction

With the passage of time, rapid advancements in computing technology has increased computation power and storage capacity exponentially. At the same time, it has also decreased the size of computers and other computing devices many fold. With these advances in computing, the use of wireless media such as radio and optical channels has increased the capability of communication devices and revolutionised the concepts of communication technology. Small devices such as PDAs, mobile phones and sensor devices have high computation and communication capabilities that were not possible a decade ago.

With the advent of these technologies, new paradigms like distributed systems, real time computing and ubiquitous systems have emerged. In distributed systems, data and applications are distributed over devices on the network where each device can potentially communicate with any other device on the network. These devices can be in geographically diverse locations connected though Personal Area Network (PAN), Local Area Network (LAN), Wide Area Network (WAN) or internet using wired or wireless media. When responses of these systems become time bound, where quality of service not only depends on the results but are also critically dependant on the response time, then they become real-time distributed systems.

There is a fast emerging field where real-time distributed systems are created using small sensing devices known as sensors and the underlying network for communication is wireless and operations are unattended. The use of sensors is not a new concept but they are mainly wired devices carefully placed for sensing particular phenomena. In this field, small sensor nodes that are used are capable of computation as well as communication using wireless media. When these devices are placed together in a field, they supposedly self configure a network system which opens new horizon to countless applications that were not possible before. Such systems are known as wireless sensor networks (WSN).

Wireless sensor networks consist of small and inexpensive distributed devices which are capable of local processing, sensing and wireless communication. These tiny, low power devices that contain sensor, processor, memory, power supply and network components are called sensor nodes. Each sensor node is capable of a limited amount of processing along with its basic function of sensing and communicating. WSN is a novel and rapidly developing field. According to the Moore's law, with the passage of time, node's capability will increase and size will decrease. The physical size and cost of each individual sensor node has a significant and

direct impact on the ease and cost of deployment [Hill, 2003]. When nodes are spread over a field, they start to operate autonomously and self configure a network without human interaction. All the distributed nodes in a network then coordinate with each other to gather and communicate the information to the end user or high powered base station. They have the ability to perform bigger tasks in great detail. Thus, a sensor network can be described as a collection of sensor nodes which co-ordinate to perform some specific action. Unlike traditional networks, sensor networks depend on dense deployment and co-ordination to carry out their tasks [Culler, 2009].

The ability to work anywhere without any support provides a wide range of potential applications ranging from monitoring to surveillance, from smart offices and home to traffic and structure management to the military applications but wireless sensor networks face a number challenges due to the resource limitations in sensor nodes. Due to limited power, new routing protocols such as listed in [Karaki and Kamal, 2004], data sampling and data aggregation techniques [Madden et al., 2002] are needed to limit the communication in WSN and increase the in-node computation because communication in WSN is more precious than in other computing environments. Sending a single bit of data can consume huge amount of energy as compared to energy consumed by execution of a single line of a code. The Power cost of producing and computing sensor data is many times lower than the cost of transmitting it [Levis and Culler, 2002].

The advancements and limitations in hardware should be well supported by software to ease the development and take the full advantage of hardware capabilities. To ease the programming burden due to resource constraints and complex nature of the distributed systems, a new layer of software is needed between the application and network which hides the limitations of the hardware, the complexity of the network and presents whole distributed system as one. Such software is called middleware. It not only provides abstraction to the developers but is also used in bridging the gap between applications and the underlying hardware.

Appropriate middleware is needed to provide standardization, system abstraction and the capability to support and coordinate concurrent applications in wireless sensor networks. Programming wireless sensor networks and applications for sensor networks is an extremely challenging task due to their tight integration to the physical world and unique constraints like resource limitation and power efficient sampling and routing algorithms. These constraints make wireless sensor applications more complex than traditional distributed applications and thus a middleware is needed to address these issues. Various middlewares for sensor networks such as database models [Madden et al., 2005; Yao and Gehrke, 2002], VM-based [Levis and Culler,

2002], tuplespace-based [Curino et al., 2005b; Costa et al., 2006a], message-based [Souto et al., 2005], agent-based [Fok et al., 2006], component-based [Costa et al., 2006b] have emerged in recent years. They differ from each other based on the programming abstraction and their assumptions about the topology and other characteristics of the network. These solutions are inspired by middlewares for more traditional distributed systems such as distributed databases, messaging-based and event-based models. The models have been proposed by the research community with emphasis on solving known issues in wireless sensor networks like data fusion, aggregation, information dissemination, routing, network and data management, resource constraint, optimal power consumption, scalability and reliability.

This thesis evaluates all the available models and compares these models with each other to discover the best available solution. It targets the specific areas such as standardization and re-usability which existing middlewares lack currently. This thesis emphasizes on design challenges and constraints arise due to the distributed and resource limited nature of the wireless sensor networks. It is aimed to clearly identify the strengths and weaknesses of the middleware models. It identifies the services that can be provided by each middleware to the new applications and level of abstraction to the developers and designers.

This thesis work presents details of the most common middleware models working in wireless sensor networks and provides comparison of these middleware models working in sensor networks. The middleware solutions are surveyed and compared keeping in view the challenges caused by the constraints and complex nature of wireless sensor networks. The constraints increase the challenges in designing and developing the applications for wireless sensor networks. These constraints and design challenges in WSN are grouped together based on their characteristics and list of middlewares addressing these challenges. The evaluation in each group provides objective comparisons of the middlewares and help in identifying the outstanding solutions. The study also helps in identifying the similarities and differences in the middleware approach towards the design challenges outlined in the tables. The data is grouped in the tables based on energy consumption and energy conservation, usability and abstraction, quality of service to the networks and applications, data management operations and network operations.

This thesis work is divided into eight chapters. Chapter two presents the proposed and existing applications running on wireless sensor networks. Chapter three explains the basic working model of wireless sensor networks and sensor nodes. Chapter four outlines the challenges to address during design and development of WSN. Chapter five explains the concepts of

middleware and the existing models in traditional computing and chapter six presents and explains the existing middleware in wireless sensor network and how the middleware solutions addressed the design challenges. Chapter seven compares all the middleware paradigms against each of the design challenge they need to address. Chapter eight summarizes the conclusions and findings for future work.

2 Applications

The ability of WSN to work anywhere without any support and human monitoring makes the range of applications limitless from environmental to industrial monitoring, from battlefield surveillance to infrastructure health check and logistic monitoring. The emerging field of tiny network sensors offers an unprecedented opportunity of wide spectrum of applications and numbers of applications build for wireless sensor networks are expected to increase with the passage of time. The aim is to push data out from the traditional desktops to the physical world. The applications running on WSN can operate in any environment autonomously without human interference. Some of the applications include infrastructure protection, scientific exploration and smart environments, monitoring air soil and water, condition based maintenance, habitat monitoring (determining the plant and animal species population and behaviour), seismic detection, military surveillance, inventory tracking, smart spaces etc. Current wireless systems only scratch the surface of the possibilities emerging from the integration of low-power communication, sensing, energy storage, and computation [Hill, 2003]. Some of the applications in different domains are indicated below.

2.1 Military applications

Research in this domain is mostly related to surveillance, detection and monitoring based systems. These systems have the potential to reduce casualties incurred in the surveillance of hostile environments. Wireless sensors are deployed outdoors such as the battlefield to scan physical phenomena like detecting and classifying the objects i.e. troops, vehicles and their movement in the field. Due to the hostile nature of the battlefield, WSN should be easily deployable and be able to self organize the multi-hop network and reorganize itself in the event of node failure. Examples of military applications are given in Table 1.

Battlefield Surveillance [Bokareva et al., 2006]	It is a battlefield monitoring and surveillance system based on wireless sensor networks to identify enemy objects and their movement in the given field. After detecting the target, the system should be able to send the data back to the command centre in real time for further processing.
VigilNet [He et al., 2006]	It is an energy efficient surveillance system used to acquire and verify information about enemy capabilities and positions of hostile targets. VigilNet allows a group of cooperating sensor

	devices to detect and track the positions of moving vehicles in the energy-efficient and stealthy manner.
The Counter Sniper System [Simon et al., 2004]	It is an ad-hoc wireless sensor network-based system that detects and accurately locates shooters even in urban environments. The system consists of a large number of cheap sensors communicating through an ad-hoc wireless network. Detecting and accurately locating shooters in challenging urban terrain becomes possible only due to wireless sensor networks. Researchers have tried to use different information sources for sniper detection but so far acoustic signals such as muzzle blasts and shock waves provide the easiest and most accurate way to detect shots. These kinds of systems need to be very robust, otherwise incomplete information due to node failure can result in false calculations.

Table 1: Applications for military applications.

2.2 Habitat monitoring

Habitat and environmental monitoring is a domain where wireless sensor network applications are intensively researched with potential benefits for our environment by studying different habitats and environments.

Wireless communications has enabled the deployment of densely distributed sensor/actuator networks for a wide range of biological, earth and environmental monitoring applications in marine, soil, and atmospheric contexts. With wireless sensor networks, the scientist will be able to study difficult, hard to visit terrains or the habitats where researchers say that the human's visit can disturb the environment for animals as well as plants that live there without any human interference. WNS also tends to be economically more feasible in such studies for longer periods of time. The intimate connection with its immediate physical environment allows each sensor to provide localized measurements and detailed information that is hard to obtain through traditional means [Mainwaring et al., 2002]. Some of the ongoing projects in this field are listed in Table 2:

Great Duck Island [Mainwaring et al., 2002]	It is used to observe the breeding behaviour of a small bird called Leach's Storm Petrel on Great Duck Island, Maine, USA. These birds can easily be disturbed by the presence of humans; hence WSN is used for the better understanding of their behaviour. Their behaviour is monitored by placing sensors outside and inside

	nesting burrows which can monitor humidity, pressure, temperature and ambient light level.
ZebraNet [Juang et al., 2002]	It is a WSN application used to observe the behaviour of wild animals within a spacious habitat. Animals are equipped with different kind of sensors and a GPS receiver to monitor their movements and other behaviours.
Grape Monitoring [Beckwith et al., 2004]	It is used to monitor the conditions that influence plant growth (such as temperature, soil moisture, light, and humidity) across a large vineyard in Oregon.
Virtual Fences [Buttler et al., 2004]	This application is used to assist in cattle herding.

Table 2: Applications for habitat monitoring.

2.3 Traffic management

WSN is useful for managing and monitoring urban traffic. Systems can be used to gather information, for monitoring and scheduling the traffic flow, as well as guiding and controlling the vehicles. Sensors can be spread all across the roads to monitor the traffic activity anywhere and anytime. Table 3 shows some of the example for traffic management applications.

Vehicle Navigation System [Chang et al., 2008]	It is used to determine the optimal routing path for achieving the least travel time and to avoid routing to heavy traffic load roads. These systems can be categorized as specialized surveillance systems.
CarTel [Hull et al., 2006]	This application is well-suited for studying issues related to traffic, congestion, and navigation and is a good example which addresses issues such as commute time, traffic jams and assistance in finding routes using vocal directions and images. CarTel node is a mobile embedded computer coupled to a set of sensors with embedded camera and GPS device to give full assistance for the commute to the users.

Table 3: Applications for traffic management.

2.4 Smart environments (office and home applications)

Low power, self configuring devices such as wireless sensor nodes are easily deployable anywhere in our daily environment ubiquitously and in ad-hoc manner. Such systems enhance our ability to monitor and optimize the environment in which we live and work. Conference room application to find free conference rooms without looking around saves our time. These are also helpful in resource management in offices. Some of the applications are listed in Table 4.

The Follow-Me [Conner et al., 2004]	This is an active visitor guidance system designed to actively assist people in finding places in new locations and environments.
MAX [Yap et al., 2005]	This is a very intuitive idea based on wireless sensors that facilitate human-centric search of the physical world. When a person searches for a particular object, this system provides location information of the desired object in a natural way by giving the address or a location near identified landmarks instead of giving the coordinates of the object. In this system, all the physical objects from documents to clothing can be tagged and people can locate objects using an intuitive search interface.
Smart Kindergarten [Srivastava et al., 2001]	This application is aimed at developmental problem-solving environments for early childhood education and health monitoring.

Table 4: Applications for smart environments.

2.5 Structure health monitoring

Structure Health Monitoring (SHM) is a very actively researched area in the field of wireless sensor networks. Events such as earthquakes and other natural and man-created incidents can damage the civil structures. Structural health monitoring systems are used to detect and localize damage in buildings, bridges, ships, and aircraft. SHM allows the estimation of the structural state and detection of structural change that affects the performance of a structure. Structure monitoring through sensors is not a new concept and currently hardwired sensors are used to gather the data but Wireless Sensor Networks provide comparable functionality at a much lower price, which permits a higher spatial density of sensors [Suzuki et al., 2007]. Table 5 shows few applications for structure health monitoring.

[Ou J P et. al, 2005]	Application for wireless sensor networks based health monitoring system for tall buildings.
Wisden [Xu N et. al, 2004]	A wireless sensor network system for structural-response data for monitoring the health of big structures.

Table 5: Structural health monitoring applications.

3 WSN model

Currently there is large and diverse research going on in this field. Unfortunately much of research is isolated in various organizations and individual components are created, mostly targeted towards the specific application rather than combined, common approach for more generic architecture. Thus resulting in the lack of standard sensor network architecture, general design principles and detailed interfaces that would provide guidance about which components are necessary and how they should fit together. If there were a standard model, components would be largely reusable across applications, and between research and development efforts. Moreover, the resulting designs could more comfortably accommodate new generations of devices [Culler, 2009].

Most of the middleware architectures and models emphasize on the optimal usage of power and the protocols proposed are inherited from the work in real time computing, mobile ad hoc networks and self organizing networks. We present a generalized and simplified model for wireless sensor networks. Sensor networks are different from other wireless and ad hoc networks. It contains small sensor nodes which can self organize to establish the network. These nodes have limited energy, memory, processing power and short of other hardware resources.

3.1 Network model

Wireless sensor networks mainly comprise of sources and sink nodes as shown in Figure 1, where source is a sensor node that gathers information from the environment thus also called information provider. It can either be a typical sensor node, actuator or they even act as only routers to facilitate the transmission of data gathered by sensor to the base station or end user. Each node might have different task but they co-ordinate and act together to achieve a higher common goal.

The gathered information from node is then sent to the sink node where information is collected from different sources. The sink is a device that connects a sensor field with outside network. Sink can be a special device that has higher computation capability and sufficient power such as a base station, or in some cases just one of the sensor nodes that is closest to the user or base station. A sensor network may contain only a single sink as well as multiple sinks. The gathered information passed to the sink is either further processed or it is merely used as high powered

communication node to send data to the base station or end user or to some other network such as internet.

Certain level of computation can be done in sensors as well as in the sink nodes but in order to save the energy resources and increase the life of a node, higher computation is done at base stations or by end users.

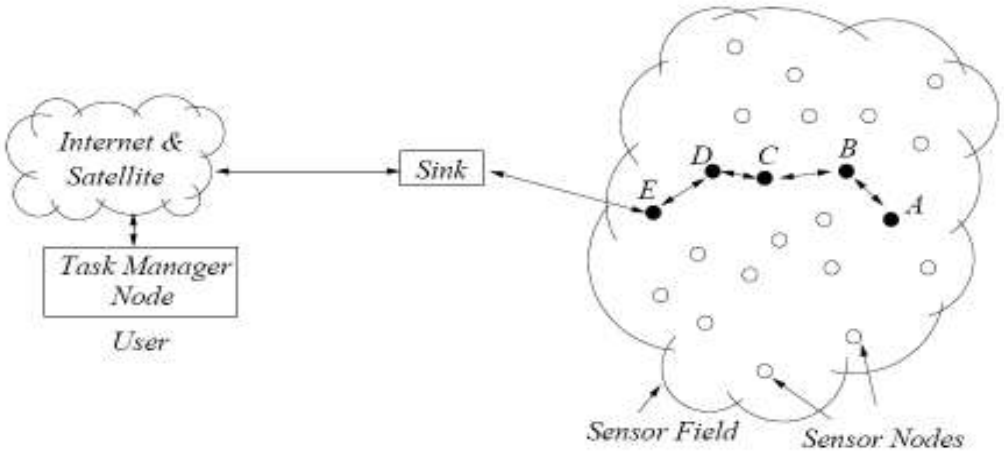

Figure 1: Basic wireless sensor network architecture [Akyildiz, et al., 2002].

These nodes are low cost embedded devices for performing variety of tasks without needing any existing infrastructure. While the capabilities of any single device are minimal, the composition of hundreds of devices offers radical new technological possibilities [Hill, 2003].

The aim of WSN is to be easily deployable low cost networks as no physical connectivity is needed as in wired networks. The network can be extended any time by adding more devices for additional or redundant tasks. Nodes in traditional wireless networks need high power base station to directly communicate, WSN has no such requirement. Each sensor in WSN becomes a part of mesh-like connected network to transfer the data in multi-hop fashion from neighbouring node to the sink node. To compensate node failure or addition of new nodes, network can dynamically adjust and can reconfigure itself but the challenges detecting the relevant quantities, monitoring and collecting the data, assessing and evaluating the information, formulating meaningful user displays, and performing decision-making and alarm functions are enormous [Cook and Das, 2005].

The wireless sensor network organization process starts from nodes deployment. Deployment can be pre-planned or by just throwing the sensors in the field where process of self organization starts. Localization is important to find the geographical location of a node in a network for better working of a system. After localization of all the nodes is complete, each node starts to synchronize with other nodes in a network. Time synchronization is very critical for any distributed system and wireless sensor networks make extensive use of synchronized time in many contexts (e.g. for data fusion, Time Division Multiple Access (TDMA) schedules, synchronized sleep periods, etc.). Once the network is synchronized, it is ready to operate to gather information from environments through sensors. Data is gathered by a sensor or multiple sensors; it is aggregated and transferred to the base station through other sensor nodes, relay nodes and sinks. Wireless sensor networks generally form mesh-like networks where data is transferred to destination in multi-hop fashion from one node to another node. Network can be a single multi-hop mesh network or multiple localized sub-networks that communicate with each other through network head or sink nodes to transfer the data to the destination. This indicates that a network can have more than one source node as well as many sink nodes as shown in Figure 2.

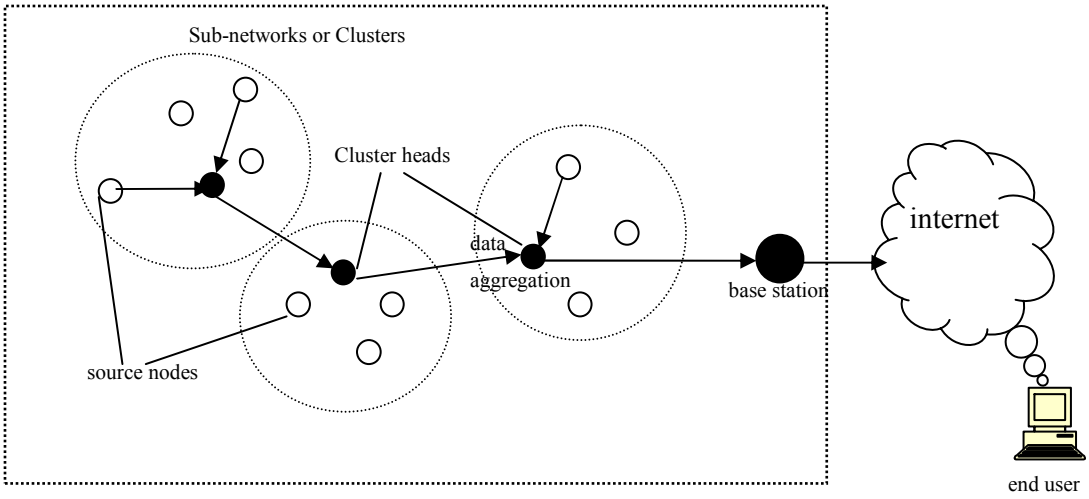

Figure 2: Multi source and sink nodes and sub-nets communicating through group leaders.

The power of wireless sensor networks lies in the ability to deploy large numbers of tiny nodes that assemble and configure themselves. Usage scenarios for these devices range from real-time tracking, to monitoring of environmental conditions, to ubiquitous computing environments, to in situ monitoring of the health of structures or equipment [Hill, 2003].

Another form of sensor network is of mobile sensor networks where source nodes or sink nodes or both are not static and can move around in a network to gather more information from sources if needed. These nodes can be attached to moving phenomena, such as animal collars to gather information all the times. WSN provides opportunity to build smart environments where the sensors are distributed and integrated in our surroundings ubiquitously. Sensors are used to capture the event without being noticed and calculate and transfer the data to the desired location: it can be the end users, backend database or application where data can be stored and further processed or some powerful base station, from where it can be transmitted to the destination.

3.2 Node architecture

Advances in hardware technology and engineering design have led to dramatic reductions in size, power consumption and cost for digital circuitry, wireless communications and Micro Electro Mechanical Systems (MEMS). This has enabled compact, autonomous and mobile nodes, each containing one or more sensors, computation and communication capabilities and a power supply [Kahn et al., 1999]. For example, the popular Berkeley Mica series has an 8-bit micro-processor running at 7.3827MHz, no hardware floating-point support, and only 4KB data memory [Klues et al., 2007].

A node has sensing, computation, communication and locomotion capabilities along with other extra sub components like actuators, GPS devices etc. A basic sensor node usually consists of computing, communication, sensing and power supply subsystems [Bharathidasas and Anand, 2002].

3.2.1 Computing subsystem

The computing subsystem consists of a microprocessor and a memory module which is responsible for the control of the sensors and execution of communication protocols. It is responsible for managing node operations, executing sensor applications and responsible for data processing within a node. To save power, micro-processing unit operates in different modes, mainly active, idle and sleep modes. The energy consumption level for the switching modes should also be kept in mind for optimizing power management.

3.2.2 Communication subsystem

The communication subsystem consists of a short range radio which is used to communicate with neighbouring nodes or with the external communication devices. The radio module is more power hungry especially during the communication, while not actively communicating they go to sleep mode for sake of battery saving. Radios operate in four different modes: Transmit, Receive, Idle and Sleep modes for the sake of power management.

3.2.3 Sensing subsystem

Sensing is fundamental function of wireless sensor networks. Sensing subsystem consists of sensor for sensing the data from outside world and sends it to the computation or communication subsystem for further processing. Sometimes it may also contain actuators to take certain actions in response to the events generated due to sensed data. Sensors gather the data from environment it is interacting with in the form of analogue signals. These signals are passed to the analogue to digital converter (ADC) located within a sensing module to convert the sensed data into digital form to pass it to the processor for further processing.

3.2.4 Power supply subsystem

The power subsystem consists of a battery which supplies power to the node. The power management system regulates the current from battery to the node. The lifetime of a battery can be increased by reducing the current drastically or even turning it off often.

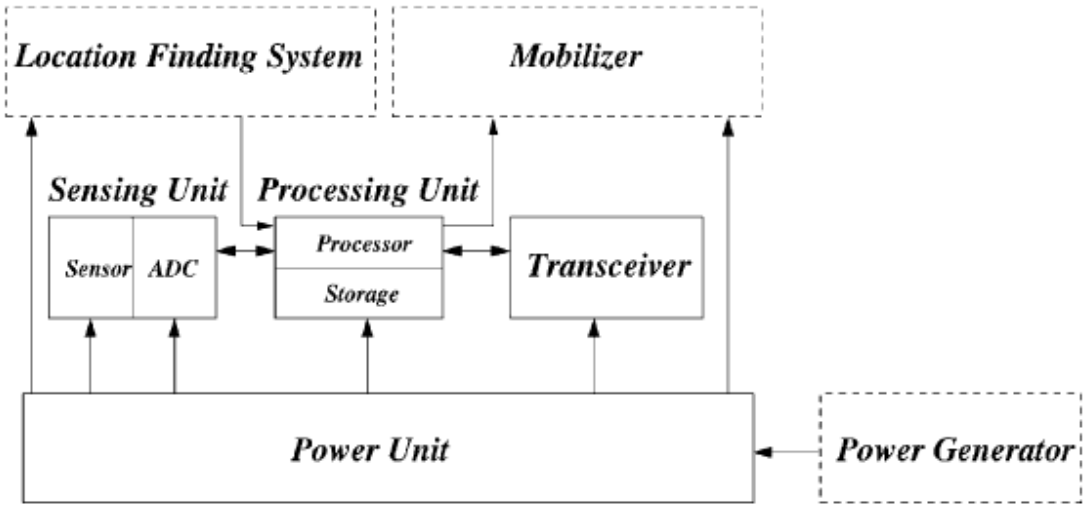

Figure 3: System architecture of a typical wireless sensor node [Akyildiz et al., 2002].

3.3 Operating system

The role of any operating system (OS) is to promote development of reliable application software by providing a convenient and safe abstraction of hardware resources [Culler, 2006]. For sensor networks, operating system must be efficient in use of memory, processor and power to meet the requirements. It should have a power and flexibility to accommodate multiple applications to simultaneously use limited resources.

Currently there are few operating systems available for wireless sensor networks. Contiki [Contiki-online, 2009] is an open source, highly portable, multi-tasking operating systems with an event-driven kernel on top of which application programs are dynamically loaded and unloaded at runtime.

MANTIS [Mantis-online, 2009], a (MultimodAl system for Networks of In-situ) wireless Sensors provides a new multithreaded cross-platform embedded operating system. FreeRTOS [FreeRTOS-online, 2009] is another operating system becoming popular for small embedded devices. It has a scalable, real-time kernel designed specifically for small embedded systems. It is predominantly written in C language that supports both task and sub-routines. Due to very small and simple core of the kernel and use of sub-routines, it has minimal effect on ROM, RAM and processing overhead that makes it a good choice for wireless sensor networks.

TinyOS is explicitly designed for wireless sensor networks and widely used by many researchers, designers and developers in various WSN based systems and application. Some of the platforms where TinyOS resides in are iMote, TMote, Mica, Mica2, BTnode, Mote2, Teslos [TinyOS–Hardware Design, 2008]. TinyOS is commonly used and widely acceptable operating system and rapidly becoming the standard operating system in wireless sensor networks. The advantage of TinyOS compared to other OS is that it has a communication centric modular design which very much suits the requirements of wireless sensor networks. This small sized, resource friendly operating system is feasible for the applications and services distributed on resource constraint nodes of WSN. TinyOS has a structured event driven execution model and component based software design which has high degree of concurrency and low power consumption.

TinyOS is built on software component model with support for component composition and network data types where components provide abstraction to different hardware. These components have sets of well defined bidirectional interfaces that are either written as a whole or built from combining smaller subcomponents. In bidirectional interfaces, a high level software component requests a task. Status of the request is sent back immediately to the requestor even the requested operation is not complete. After completion, the operating component signals an event to the low level component to take further action, thus providing interleaved execution. During this interleaving, processor goes to sleep for power saving, if no other tasks are pending. Unlike traditional operating systems where multiple threads are used to handle different devices and each thread has its own stack which is kept in memory. TinyOS uses a single stack that supports many concurrent activities. The event-driven nature helps the system organized around a single dispatcher, that calls specific even handler depending on the event type. Upon getting the call from dispatcher, the event handler takes a required action and updates the system state. TinyOS uses very efficient execution model, component model and communication mechanisms.

TinyOS execution model

Wireless sensor network make use of event based model which is very useful to achieve high level of performance in concurrency intensive application using a small amount of space. To avoid long running applications to block other subsystems, TinyOS uses tasks. Tasks run in the background without interfering with other events and can be interrupted by low level system events when required.

TinyOS component model

Hardware abstraction maps physical hardware into component model. Synthetic Hardware simulates the behaviour of advanced hardware. The high level software components perform control, routing and all data transfers [Hill, 2003].

TinyOS communication model

TinyOs uses active messaging (AM) model for communication. It is a simple model widely used in distributed systems. The active message reaching at the target node contains application level handler. This handler is to be invoked on target node where it extracts message from the network and then either integrates the data into the computation or sends a response message. When the data arrives at the node, it must be stored in memory buffer and then these buffers are delivered to the application by AM dispatch layer. If the system does not support the dynamic memory

allocation, application returns the message buffer back to the radio subsystem when the message is delivered while radio subsystem has to keep one extra buffer to receive the next message. After receiving the message, receiver will send back a special sequence immediately to the sender if packet passes the cyclic redundancy check. This acknowledgement is very important for reliable delivery system.

4 Design challenges

There are number of technical challenges different from other conventional wireless networks which need to be addressed during design and implementation. The sheer numbers of sensors and the expected dynamics in the environments present unique challenges in the design of unattended autonomous sensor networks. Despite lots of effort and research on WSNs, these networks have several restrictions, e.g., limited energy supply, limited computing power, and limited bandwidth of the wireless links connecting sensor nodes. One of the main design goals of WSNs is to prolong the lifetime of the network and prevent connectivity degradation by employing aggressive energy management techniques. Some of the factors that influence the design and architecture of WSN [Akyildiz et al., 2002; Hill, 2003] are listed below.

4.1 Hardware limitations

Researchers are exploring the limits on size and power consumption in autonomous sensor nodes, a major challenge is to incorporate all these functions while maintaining very low power consumption, thereby maximizing operating life given the limited volume available for energy storage [Kahn et al., 1999]. Currently due to the small size of a sensor node, very limited amount of modules can be embedded in a node. This results in limited power source, memory and processing unit in addition to other application specific modules like location detectors, special kind of sensor etc. this limitation has a big impact on the overall design and network algorithms for wireless sensor networks. With limited processing power and memory, local computation within a node is very restricted and highly distributed algorithms are needed to distribute the computation among different nodes within a network to cater with this limitation. Communicating nodes are linked by a wireless medium. The traditional problems associated with a wireless channel (e.g., fading, high error rate) may also affect the operation of the sensor network. This strict resource limitation makes it very difficult to execute fast and complex network and other signal processing algorithms with moderate or high time / space complexity [Klues et al., 2007], which makes the design of WSN networks and application very complex.

4.2 Power limitations

Due to the autonomous nature and unattended operations of the wireless sensor networks, small sized power supply integrated in the node demands of optimal power usage and optimized

protocols. Wireless links in real sensor networks can be extremely unreliable, deviating to large extents from the perfect reception. Packets transfer on the lossy links can really degrade the delivery rate and increase energy wastage if retransmission is required [Saeda et al., 2004]. Very active research is being done to improve the power management and efficient use of power usage in sensor networks. In this effort, many routing and network protocols are presented for different layers, especially physical, Data link and network protocols are the center of research. Unified architectures are suggested such as MAC Layer Architecture (MLA) [Klues et al., 2007b], network layer architecture NLA [Ee et al., 2006] and unified power management architecture (UPMA) [Klues et al., 2007a]. More research and design effort is needed keeping these restrictions in mind.

4.3 Data aggregation & reporting

Data sensing, aggregation and reporting techniques depend on the time criticality of data and nature of application. Acquisition and reporting of the data can be time-based, event-driven or query-driven depending on the demand of application. Query-driven and event-driven are mostly reactive models which sends the data in response to certain event or request whereas time-based are requirement of proactive systems in which data is sent after particular interval of time. The selection in routing protocols is based on the type of data reporting. In high density networks, similar data will be acquired by more than one sensor which results in sending redundant data to the sink from each node individually. To save power consumption this should be avoided and aggregated data should be sent to conserve energy. Data aggregation techniques also depend upon the nature of application, routing protocols and density and nature of sensor network.

4.4 Deployment

Deployment strategy of nodes in wireless sensor networks is highly application dependant and has affect in selection of network organization and routing protocols. Node deployment can either be deterministic by carefully placing nodes in friendly environments like homes and offices. They can be easily deployed for monitoring the health of civil infrastructure or they can be scattered randomly in a field for surveillance. Here, the tricky situations like incremental self deployment [Howard et al., 2002] of nodes using the information gathered by previously deployed node to determine its target needs to be handled during application design. In mobile sensor networks deployment of sensor nodes is also an energy consuming process and the protocols should be carefully designed.

4.5 Heterogeneity

Depending on the application, sensor nodes may have different roles to play. It may be sensing different type of events and can have different capacity in terms of computation, communication, and power. These sensors can be heat sensors, temperature detectors, sensors to measure pressure and humidity of the surrounding environment, detecting motion via acoustic signatures, audio or video devices. Different types of heterogeneous devices sensing gather various types of data bring lot of flexibility to the system. It increases the complexity in design, programming and maintenance as these devices can no longer be treated as same and raises many technical issues related to data routing and reporting.

4.6 Fault tolerance

Sensor nodes may fail or be blocked due to lack of power, physical damage, or environmental interference. Failure of sensor nodes should not affect the overall functioning of the sensor network and keep on working in any circumstance. This can be achieved by getting explicit knowledge of the overall state of the network from time to time so users can get early warning of failure aid in incremental deployment of sensors [Zhao et al., 2002]. The monitoring techniques should be power friendly and have minimum impact on network life time. In case of self organized, unattended networks, MAC and routing protocols must accommodate formation of new links and routes to the data collection base stations.

Fault tolerance can also be increased by using redundant nodes that takes place and start acting for the failed node upon discover, as ASCENT (Adaptive Self Configuring Sensor Networks Topologies) [Cepra and Estin, 2002] suggests that; as density increases, only subset of nodes are necessary to establish route forwarding backbone. A reliable method is needed for such networks that are using flooding and broadcast protocols like RBP (Reliable broadcast protocol) because lossy links broadcast is not reliable method in wireless sensor networks.

4.7 Network organization and management

Wireless sensor networks are usually deployed on ad hoc basis and suppose to perform unattended operations. These networks should be adaptable to changes in the network and adjust their operations by adjusting nodes, routes and network protocol on the fly. There are few such network challenges which need to be addressed start from time of deployment:

4.7.1 Localization

Localization is the process of finding out the exact geographical location and physical location of a node within the network. Localization of a node starts right after deployment. It is the essence of wireless sensor networks and all the applications should be location aware. There are simple solutions present based on using GPS devices which is a very well known location service in use today but the power, size and cost constraints makes the use of GPS in sensor nodes unsuitable for low cost networks so other beacon-based, beacon-free, range-based and range free solutions are used as identified in [Battelli and Basagni, 2007].

4.7.2 Time synchronization

Time synchronization is critical for any type of distributed system and so is for wireless sensor networks. Extra efforts are needed due to the scope, network life time, energy constraints and precision requirements in wireless sensor networks. Time synchronization is necessary for sensor network, if it needs to work as a whole single entity. When data is captured by the sensors; time of event detection and data transmission help in eliminating the duplicate data and synchronizing the activity throughout the network.

4.7.3 Scalability and re-organization

There can be hundreds or thousands of nodes deployed in the sensing area and any routing scheme should be able to work with large number of nodes and should be scalable enough to respond to changes in the network. In the event of node failure or addition of supplementary nodes due to scalability requirements, routing and network protocols should be able to adjust to these changes and network should be able to self organize or re-organize.

4.7.4 Network dynamics

Sometimes due to the application requirements, network nodes or base stations or both need to be mobile. Thus routing messages from and to moving nodes is a challenging task since route stability becomes an important issue, in addition to energy and bandwidth.

4.8 Security

Security is broadly used term encompassing the characteristics of authentication, integrity privacy and anti-playback [Pathan et al., 2006]. Wireless networks are usually more vulnerable to various

security threats as the unguided transmission medium is more susceptible to security attacks than those of guided transmission medium. The broadcast nature of the wireless communication is a simple candidate for eavesdropping. The major challenge for employing any efficient security scheme in WSN is due to limited processing power, memory and type of tasks expected from the sensors. Following are the types of attacks that can happen on wireless sensor networks and need to be addressed for the uncompromised security [Pathan et al., 2006].

Denial of Service (DoS): The attackers try to exhaust the network by flooding it. They keep on sending extra packets to the nodes on network thus limiting the network users from using resources.

Attack on information in transit: Sensors send all the information to the sink node to send it out of the network. This information in transit can be altered or spoofed by eavesdropper. Any attacker can intercept and alter the data on the sink.

Sybil attack: In wireless sensor networks, the attacker can act as a legitimate node and tries to degrade resource utilization and data integrity. Sybil attacks are very hard to prevent in networks where there is no central high resource base station.

Black holes: This is a fake malicious node in a network, which attracts all the data in network by responding to the requests for routes. The attacker tells the requesting node that it contains the shortest path to the base station thus diverting traffic to its self.

Hello flood attack: The attacker with higher processing and communication power as compared to the nodes in network. It sends hello packets to the nodes on the network thus other nodes identify it as contending node and tries to send their packets to the base station through attacker.

Wormhole attack: The attacker receives the broadcast for routing request from sink or other nodes and replays the same request in its own neighborhood. Each neighboring node receives the replayed packet and considers the attacker as its parent node in the neighborhood. Thus all the packets are sent through the fake node; even it is not in the single hop range thus creating a tunnel to other location in a network.

5 Middleware

Increase in dependence on information technology and rapid improvement in the hardware resulted in the growth of number applications of different types. Customization within these applications has raised the need of integrating expensive legacy systems with their new advance counterparts. The technological advancement also brought the growth in network centric paradigm. With the emergence of distributed systems, there arise needs of new software and tools that can facilitate the integration of different software and distributed components of a system. This special software eases the programming challenges that arise with distributed systems. It provides the abstraction to hide the complexity and heterogeneity of the underlying distributed environment that contains network technologies, machine architectures and operating systems. This new software that lies between many software or services distributed across the network is known as middleware.

A Middleware is software that brokers between different software or components of software. It is composed of set of services designed to manage the complexity and heterogeneity inherent in distributed systems. Middleware is a layer of services which lays between the operating system and the application program or software APIs as shown in Figure 4. The middleware layer in Figure 4 shows the common types of middlewares that can be used, such as Transaction Processing (TP), Remote Procedural Calls (RPC) and so on. Middleware is there to provide a common programming abstraction across a distributed system. It helps the application programmers in reducing the programming overhead by relieving them from concerning about low level physical layer and hardware issues by providing abstraction to the application programmer and dealing with low level network and hardware issues itself.

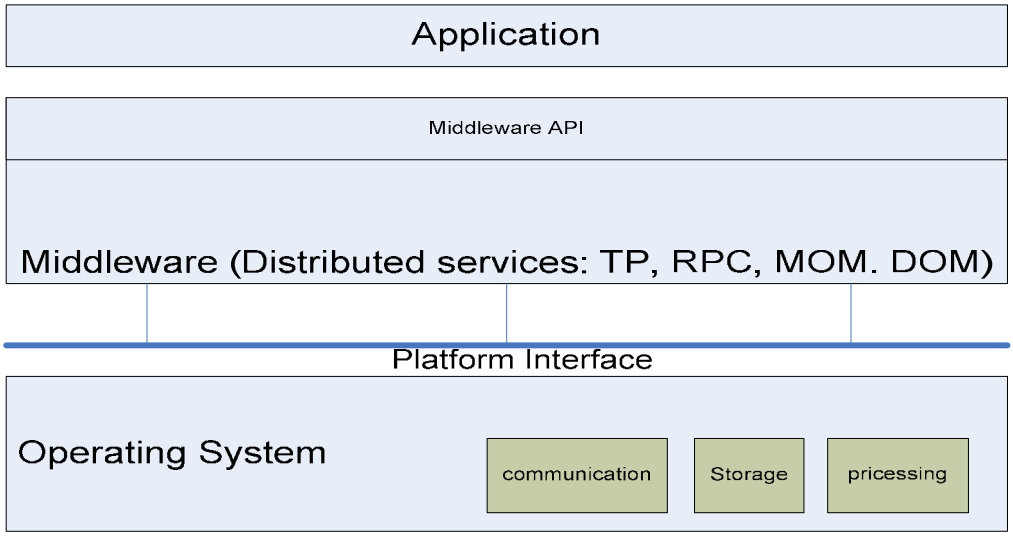

Figure 4: Middleware lies between the operating system and the application program.

Middleware can be considered to be the software that makes a distributed system programmable. Programming a distributed system is in general much more difficult without middleware, especially when heterogeneous operation is required [Bakken, 2001]. With the growth of network based distributed application, middleware software are becoming very important day by day. This results in creation of various types of middlewares encompassing different applications and domains such as distributed objects and components, message-oriented communication, and mobile application support.

5.1 Need of middleware

In distributed systems middleware is a brokering software that lies between the operating system and the applications on each side of the system which provide abstraction from the underneath network and communication protocols for the ease of application development. The middleware approach lies in between. The idea is to extend an existing programming language by introducing a new layer between the application and the network that hides the complexity of communication and data transfer. To the developer, the invocation of a remote procedure should appear no different than the invocation of a local one [FPX–Middleware, 2009].

[Object Web -Middleware, 2009] states few reasons for the need of the middleware.
In a distributed system, interconnected software and services are running in physically distributed locations, where they need to collaborate with each other. Middleware is used to hide this

distribution and show the complete system as single entity to the application developers for the ease of development.

Physical separation of components of a distributed system means they run on different hardware components, run on different operating systems using separate communication protocols. Middleware is used to hide this kind of heterogeneous platforms and resource thus reducing the complexity at higher level.

Middleware helps in providing standard common set of services so that applications can be easily composed, managed and reused. It also helps in avoiding duplicate efforts and facilitate in collaboration between applications.

Due to the abstraction provided by middleware, they provide resource management in distributed systems that are otherwise very hard to manage. Apart from network and communication protocol abstractions, they can be helpful in managing resources like disks, memory and CPU in a centralized clustered environment.

5.2 Types of middleware

Middlewares are used for different type and genre of systems and applications. They can be categorized as database middleware, application server middleware, message-oriented middleware, transaction-processing monitors and Web middleware depending upon the type of application they facilitate. The following categorization is based on types of heterogeneity they provide to the programmers and users.

5.2.1 Distributed tuples & database systems

Distributed relational databases act as a middleware to offer abstraction to the tuples spread across in different locations and different formats. Users then interact with these relational databases (middleware) through SQL language to manipulate the set of tuples. Users do not have to worry about organizing the data and fetching the distributed tuples from certain locations.
These distributed database system are also capable of managing resources for client queries, server side process management and multiple database transactions. The middleware sits between the client and database server like distributed tuples that manages the transfer of data between

multiple terminals and the application programs to ensure data consistency during multiple transactions.

5.2.2 Transaction processing

This middleware sits between the system that provides transaction based services and the clients accessing the system. It is more or less like database systems. It is usually used to handle the clients' connection and disconnection with the server side [Cook and Das, 2005].

5.2.3 Remote procedural calls

Remote procedural calls (RPC) are used to invoke procedures in different locations across the network. This facilitates in creating the application distributed across the network. RPC helps programmers to execute remote routines and processes by simply calling the local routines, without worrying about networks and access issues.

5.2.4 Distributed object middleware

In distributed systems, applications are structured into objects. These objects are located in various locations spread across the network that interacts through location transparent method invocation. Distributed Object Middleware (DOM) provides the abstraction for remote objects that provides synchronous communication between distributed programs and their methods can be called as they are located locally in same address space. The Common Object Request Broker Architecture (CORBA) [Corba-online, 2009] is a standard for distributed object computing, that provide heterogeneity across programming languages and vendor implementations. Microsoft's Distributed component object model (DCOM) [DCOM-online, 2009] and Java RMI/JINI [RMI-online, 2009] are the commonly known example of DOM.

5.2.5 Message oriented middleware

Message oriented middleware (MOM) is used in applications in which messages need to be persistently stored and queued. Unlike RPC and DOM it provides asynchronous form of communication that provides abstraction of the message queues across the network where we can simply put the messages or retrieve messages from the queue. It eases the message exchange between different programs distributed across the network. Typical MOM is a publish/subscribe system where source publish to the entire network and interested node subscribe to the message. Email programs can be seen as specialized form of these queues where messages are asynchronously sent and received from the mailbox. IBM's WebSphere MQ, Sun's Java

Messaging Service (JMS), BEA MessageQ and Microsoft Message Queue Server (MSMQ) are the prime examples of MOM [Middleware-online, 2009].

6 WSN middleware models and architectures

Appropriate middleware is needed to provide standardization, system abstraction and the capability to support and coordinate concurrent applications. Design of wireless sensor networks is highly influenced by resource scarcity and the middleware should be facilitating the development and maintenance of sensing based applications. Programming wireless sensor networks and applications for sensor networks is an extremely challenging task due to their tight integration to the physical world and unique constraints like resource limitation and power efficient sampling and routing algorithms. These constraints make wireless sensor applications more complex from traditional distributed applications and thus a middleware is needed to address these issues. This can help the programmers, developers and users to focus on application development instead of worrying about these problems and underlying network related issues. WSN middleware acts as a bridge between hardware, operating system, network stacks and applications. Middleware should support the implementation and basic operation of sensor network to avoid the above mentioned programming overhead. Traditional middleware is normally heavy weight in terms of memory usage and computation and therefore it's not suitable for wireless sensor networks with scarce energy and computation resources.

Main purpose of middleware is to support development, maintenance, deployment and execution of sensing based applications. The distributed nature of WSN increases the need of middleware to support the applications for wireless sensor networks. Various middlewares for sensor networks have emerged in recent years. Most of the existing middlewares for sensor networks have very limited scope in terms of the application. They are targeting the specific area thus existing middleware models lack the standards and reusability. They differ in terms of their models for querying a data aggregation and their assumptions about the topology and other characteristics of the network. These solutions are inspired by middlewares for more traditional distributed systems. E.g. distributed databases, publish/subscribe, tuple space etc. generally, middleware in sensor networks has to tackle the issues like abstraction, data fusion, aggregation, information dissemination, routing, network and data management, resource constraint, optimal power consumption, scalability, reliability etc.

6.1 Database inspired middleware

The database inspired middleware views the whole network as a virtual distributed database system where data is spread across the sensor devices called nodes. Here nodes act as different sites of a distributed database. The users and programmers are provided with the simple SQL based interface to gather information from the network by issuing SQL-like commands without worrying about the underneath operational logic.

Supporters of this approach believe that declarative queries are preferred way to interact with sensor networks rather than deploying application specific procedural code. They argue that the use of query based approach is better because sensor network applications are naturally data driven. Database inspired middleware abstracts the low level communication away from high level application programmers and they don't need to explicitly define how the information should be collected inside the network rather they can simply submit the queries for data collection and simple data aggregation functions.

Due to data centric nature of the applications, database inspired middleware concentrate on data collection and communicating it to requesting application efficiently with in a required time frame with improved network life.

In this type of middleware main emphasis are on techniques of data aggregation, which is integral part of query based database systems. Other important areas include metadata management for optimized and reliable network operations and data routing techniques with focus on energy conservation. This type of middleware support relatively simple data collection applications with basic data aggregation functions. Some of the middlewares in this category are discussed below.

6.1.1 TinyDB

TinyDB [Madden et al., 2005] is a query processor for the sensor networks that uses Acquisitional Query Processing (ACQP) that runs on each node in a distributed network. TinyDB not only makes use of traditional energy conservation techniques such as reducing communication through snoozing and through reducing the amount of data to be communicated through data aggregation techniques but it also focus on techniques that takes in to the account location of sensors nodes and cost of data acquisition at the nodes. TinyDB focuses on data sampling and how the sampling interleaves with other operations which can significantly reduce power consumption.

TinyDB has a relational table called ‘Sensors’ that stores the data acquired on the sensor nodes. Each sensor node has a single tuple in a sensor table with columns representing attributes of the sensor nodes. Records in this table are stored for short period of time when needed for certain query. Physically the sensor table is spread across all the sensor nodes of a network where each node stores its own reading.

The basic use of middleware is to ease the development and programming. The DB-like structure and use of SQL-like language in TinyDB decreases the effort needed by programmers as it takes away the complexity of structural languages like C that need many lines of complex code to perform simple operation in WSN. E.g. Great Duck Island software consisted of 1000 lines of C code for just to broadcast the collected data over a single hop radio [Madden et al., 2005]. Like SQL, TinyDB's language consists of select-from-where-group by clauses. The FROM clause refers to the sensor table spread across the nodes in sensor network or it may also refers to a stored table, that is created through special logging called materialization point. Normally the sensor table is unbound as it is composed of stream of data, sampled at defined time interval. The bounded subset of streams or a window is specified for operation like sorts and joins. This subset of data on a sensor node is called materialization point.

Aggregation

TinyDB includes support for group aggregation queries for the affective resource management in order to conserve energy. Aggregation reduce the quantity of data to be transmitted though the network. SQL-like aggregate functions like AVG, SUM, and COUNT etc. are used. TinyDB differs from normal SQL as it contains stream of values rather than single aggregates, so each aggregate value in the stream consists of group id, aggregate value pair and time stamp with an epoch number. Tuples in sensor nodes are produced at sample interval that is parameter of the queries. The period of time between the start of each sample period is known as epoch.

Data sampling techniques

TinyDB gathers data using various techniques and queries for optimal power usage to increase life time of the network. Following techniques are used to achieve the desired network life time.

Event based queries:

TinyDB starts gathering data when certain event triggers. Events are generated either by other queries or by low level part of operating system. The query uses ON EVENT clause to generate the event. For example:

```
ON EVENT bird-detect (loc)
Select AVG (light) from sensors AS s
WHERE dist(s.loc, event.loc) < 10m.
```

Explanation: when a sensor detects a bird, take an average of light measure for all the sensor locations (s.loc) in 10 meters range of the event location (event.loc).

Every time an event occurs, the node that detects the event issue the query and data is collected from the neighboring nodes. Such event allows system to be dormant until event is generated due to external stimulus instead of continuously and actively waiting for the some data to arrive, thus providing significant reduction in power consumption.

Besides low level signal event as described in previous paragraph, TinyDB queries may also signal events. For instance:

```
SELECT nodeid, temp
WHERE temp > thresh
OUTPUT ACTION SIGNAL hot (nodeid, temp)
SAMPLE PERIOD 10s
```

Explanation: Node signals an event when temperature rises above some threshold. The output is sent after every 10 seconds.

Such events don't provide the advantage of low level interrupt but we get the programmatic advantage of linking queries to the signaling of events.

Life time based queries:

Life time queries in TinyDB are used to specify life of a query via "LIFETIME" clause. This intuitive technique is used to control power consumption by adjusting sample rate and transmission rate. When "LIFETIME" clause is used in a query, TinyDB performs life time estimation in which it estimates sampling and transmission rate given a number of joules of energy remaining such as the following query.

```
Select nodeid, accel
From sensor
LIFETIME X days.
```

Explanation: Query indicates to sample light and acceleration sensors at a rate that is as quick as possible and still satisfies. the goal of X number of days.

These rates can be computed on a single node by taking into account the cost of accessing sensors, selectiveness of operator, expected communication rate and current battery capacity. Once a lifetime on a node is calculated, it tries to achieve this lifetime goal. The transmission rate needs to be coordinated across all nodes in a routing tree where senders and receivers coordinate the wake and sleep calls according to the new adjusted rate.

Sometimes the sampling rate calculated for the required lifetime is not feasible for application's requirement then a higher sampling rate is needed. TinyDB allows an option "MIN SAMPLE RATE" to be supplied to achieve the requested sampling rate and required lifetime. The system will not actually transmit all the readings, it may be forced to aggregate the value or discard some samples.

Query dissemination and route selection

When data is requested from the node, query is disseminated in the network from root by broadcasting it to all the nodes. When node hears the query, it must decide whether the query applies locally or it applies to the children node. If the query does not apply to the node and its children, the entire sub tree is excluded from the query, thus saving considerable time of disseminating, executing and forwarding the result of the query. Such dissemination needs to maintain information about child attribute values, thus TinyDB uses a data structure called semantic routing tree (SRT).

Semantic routing tree:

This routing tree allows each node to determine whether the nodes below them need to be participating in the query over some constant attribute A. SRT is constructed when a node selects its parent node with the most reliable connection to the root with considering the semantic properties as well. Conceptually, SRT is an index over a certain attribute that can be used to locate nodes having relevant data.

SRT is created in two steps: first build request is transmitted from the root that propagates down to all the nodes in the network. When request over certain attribute reaches a node N, it forwards the request to children. If it has no child, it selects a parent node from available parents and sends backup the requested value of attribute A to the parent in parent selection message. If N has children, it records the value of attribute A along with the id of each child and send the parent

selection message with range of values over attribute A that covers all the children and its own values.

Route re-organization and SRT maintenance

In TinyDB, network organization changes dynamically when the nodes fail due to power drainage or environmental hazards etc. New nodes can be added to existing operational network or change in links quality may cause the node to change the parent node.

When a node starts to reselect its parent, it sends a parent selection message to its new wannabe parent node a. If this message changes the range of values over an attribute of parent node A's interval, then Node A notifies its parent node and so on, in this way updates propagate to the root of the tree.

If the child node is removed, the parent node starts to re-organize the network below; it associates an active query id and last epoch with every child in the SRT. When a parent node P forwards query Q to its child node C, it sets the C's active query id to the id of Q and sets its last epoch entry to zero. Every time P forwards results or aggregate result of Q from C, it updates C's last epoch and if P does not hear from child C for some number of epochs, it removes the child from its SRT. Then P asks its remaining children to send their ranges.

Network maintenance cost:

The benefits of using SRT can be substantial due to reduced messaging but it also has the maintenance and construction cost associated with it. Construction cost of SRT is slightly higher than conventional sensor network because additional parent selection messages are sent back whereas conventional network doesn't need explicit confirmation to the parent. This means that. there is substantial cost of switching parents in SRT.

Data aggregation and time scheduling

During aggregate queries, there is a need of coordinating and synchronizing the time between parent and child nodes transmitting and receiving timings. It is to be made sure that parents are aware and have access to children's reading before aggregating. TinyDB makes use of slatted approach, in which each epoch is divided into number of fixed length time interval. These intervals are numbered in reverse order from higher to lowest such that last interval in the epoch is interval 1. These nodes are assigned to the interval equal to their level such as nodes at level 1

in the first interval and nodes at level 2 are in interval 2 and so on. The level of a node is equal to its distance (in hops) from the root node in a routing tree. If a node is aggregating during the node's interval, it computes the partial state record consisting of the combination of any child values it heard with its own local reading. After computing, it transmits either its partial state record or raw sensor reading up the network. In this way, information travels up the tree to the root.

Metadata management

For the maintenance of network and routing tree, each node maintains a catalog that describes its local attributes, events and user defined functions. Metadata maintained at nodes is transferred to the root from time to time for use by optimizer. Metadata consists of event metadata and attribute metadata.

Event metadata consist of name, a signature and a frequency estimate that is used in query optimizer. In attribute metadata, information about cost and time to fetch data and range of attribute is used in query optimization while the information about semantic properties is used by query dissemination and result processing. The catalog also contains information about aggregate system, such as their names and pointer to their codes that help in data aggregation.

Prioritizing data delivery

Once the data is ready to be sent, they queued onto the radio queue for delivering to the parent node. The node transmits the data to its parent node which is gathered locally as well as the data which is received from its own children.

If there is low data rate and little network contentions, then the data can be sent without any hiccups and transmission queue will drain easily but in case of high data rate and network collision due to higher number of queries or aggregates, the queue will overflow and there are chances of data loss. In this kind of situation, the system should decide whether to discard the overflow tuples or old tuples or to combine the multiple tuple through aggregation. The Acquisition Query Processing (ACQP) in TinyDB makes these very important decisions because the cost of acquiring and delivering data is very high and this decision will restrict the rate of data item, arriving at the node.

Adapting rates for power consumption

TinyDB adjust the sampling and transmission rate to improve the data delivery by avoiding data collision and network related losses. TinyDB can adjust the rates and prioritize the data delivery in case of transmission queue overflows to achieve required quality parameters. Initially the optimizer chooses sampling and transmission rate based on current network load, required sample rate and life time but after running the same query continuously for days, these rates are no more optimized which can degrade the performance by many folds. TinyDB recalculate and adapt to new rates to avoid network contention and for improved power consumption.

In order to avoid network contention, TinyDB may prioritize some of the data over other to reduce the amount of data sent. It may also decrease the transmission rate of nodes but this technique can cause the transmission queue to overflow, forcing TinyDB to discard the tuples. Adaptive tuple delivery rates can also be used to meet specific life time requirements. The system can compute the predicted voltage battery and then compare it with current voltage to re-estimate the transmission rate for optimized power consumption.

6.1.2 Cougar

Cougar [Bonnet et al., 2001; Yao and Gehrke, 2002], like other database-inspired middleware acts as distributed database across the wireless sensor network. The information is retrieved from the sensor network using database style queries known as sensor queries. Cougar uses distributed query approach where different portions of a query execute on separate nodes so the workload is divided across the devices as each query process the subset of data found on particular node.

In Cougar, each type of sensor is modeled as a new abstract data type (ADT) and their signal processing function are modeled as ADT functions that return sensor data fetch from the physical world by sensors.

In Cougar data is characterized as sensor data and stored data. Stored data is represented as relation. Stored data comprise of all the sensors that are part of the sensor database together with characteristics of these sensors. Data collected by sensors from the physical world using signal processing function is sensor data. Time plays very important role in sensor data. Each output of the signal processing function on the sensor nodes is marked with a timestamp and sensor data is represented as a time series.

Device representation

In Cougar, devices are accessed directly by queries and partial processing of data is done at nodes, so physical devices in a network are needed to be presented in database system. There are two types of representations in cougar.

User representation

Devices in Cougar are represented as abstract data types (ADT). ADT is natural reflection of devices in database systems because both ADT and devices provide controlled access to the data through well define interfaces. An ADT object in a database corresponds to the physical device (sensor node) in real world. The ADT function corresponds to the signal processing function of the node. One sensor ADT is defined for each type of sensor and the public interface of sensor ADT corresponds to the specific signal processing function supported by a type of sensor. The language used is SQL-like language in which expressions over sensor ADT can be included in SELECT or WHERE clause of a query.

Internal representation

Internal representation shows, how devices are represented internally. In cougar each node has light weight query execution engine that is responsible for executing signal processing functions and sending back data to the front end. Sensor ADT functions are very important elements for sensor queries and represented internally as virtual relation. Virtual Relation (VR) is a tabular representation of a function where records contain input and output arguments of functions known as virtual records. If a device function takes x arguments, then the corresponding virtual relation has x+3 attributes. The first attribute is unique identifier of a device, second to x+1 attributes are input arguments of a function, x+2 attribute corresponds to the output and x+3 is a timestamp corresponding to the time at which output is obtained. Virtual relation is append-only, i.e. records cannot be updated and deleted. Virtual relation is partitioned across the set of devices and base relations are either stored in central database or partitioned across devices.

Query processing

Users and applications use query without knowing how data is generated and processed to compute results. End users don't know about underlying catalog management, query optimization and processing of sensor data. Cougar proposes a query layer consisting of query proxy lying between network and application layer on each sensor node. This makes use of the important

aspect that, to increase the network life, part of computation and query processing should be done on nodes based on the findings that communication of wireless network uses lot more energy than computation. In-network query processing improves energy consumption and network life time.

The query optimizer is present at the gateway node that generates the query plan according to catalog information and query specification. Query plan specified data flow between sensor nodes and exact computation plan at each node. The plan is the disseminated to all relevant sensor nodes.

Since many applications are intended to monitor environment for longer period of time, cougar supports long running queries as well as periodic queries to produce time bound data. In addition to normal SQL clauses (SELECT, FROM, WHERE), it also supports time based clauses like DURATION and EVERY.

Aggregation

Cougar proposes different aggregation techniques to limit the amount of data sent over the network. These techniques are:

Direct delivery is the simplest technique in which each sensor node sends packets to the leader node and computation only happens at the leader after receiving all the data from sources.

Packet merging technique reduces the energy consumption by merging several records in to a larger packet. This limits the expensive communication cost and only pay the packet overhead once per group of records.

Partial aggregation is used to push the partial computation from leader node to intermediate nodes along the path to reduce the data by partially aggregating on the fly [Yao and Gehrke, 2002].

Network organization and maintenance

The limited communication capability of sensor nodes, low quality of communication channels and frequent topology changes make the wireless sensor network very unstable and require all nodes to participate in routing. Cougar proposes the use of typical reactive routing protocol called ad hoc on demand distance network (AODV) [Perkins, 1999] for packet routing to support in-network aggregation. AODV builds routes only desired by application layer. Reason that cougar proposes to use the reactive protocol over proactive protocol is because it scales to large size

networks with thousands of nodes. AODV does not produce duplicate packets which is very helpful for duplicate sensitive in-network aggregation.

Network interface extension

In order to implement the in-network aggregation, nodes need to have ability to intercept data packets that are not destined for the node itself. The query layer in Cougar network is responsible for query processing and task like aggregation needs to communicate to the network layer in order to catch the packets that are destined for the leader node. A node will transfer the packet to the next node if it is not destined for itself and partial aggregation couldn't be possible to perform without a query layer. The query layer needs to communicate to the network layer to intercept the required data. When a network layer gets the packets, it will first pass the data packets through set of functions which can alter, delete or add the additional data i.e. query layer will intercept the data packets received from children nodes when it is scheduled to aggregate the data. During this process, the node will generate new packets and send it to the leader. All this is transparent to the network layer.

Routing protocols

Routing protocols are designed for point to point to communication and are usually evaluated by selecting two random nodes and establishing and maintaining a communication path between them. A sensor network with query layer has significantly different communication pattern. Route initialization and maintenance in Cougar system is as follows.

Route initialization

The route is being initialized before sensor nodes start sending data packets to the leader or root node. Instead of starting initialization at each node separately, the leader node in aggregation tree broadcasts route initialization message. The initialization message counts hops through which each node determines their depth in a spanning tree. Using this initial broadcast, nodes can save the reverse path to the leader thus saving energy by reducing the amount of communication for initialization.

Route maintenance

Network reliability is very important for in-network aggregation. Dropping a data packet due to a bad link can seriously decrease the result as these packets might contain results from multiple

nodes. Node failure and packet loss should be identified early and corrective actions must be taken quickly. In case of failure, cougar proposes two repair methods: local repair and bunch repair [Yao and Gehrke, 2003].

Query optimization

It is the optimizer's responsibility to determine the number of flow blocks in a query plan and manage interaction and communication between them. Each flow block has tasks to collect data from sensor nodes and coordinate and communicate to the leader in a flow block. Each sensor block has its own set of source nodes, leader selection policy, and the routing structure from source to the leader node and amount of computation that a block should perform. The optimizer has a critical role of improving the network life by limiting the communication cost and achieving the application's quality requirements, such as accuracy of a query result, communicating the result within specified time period through managing the flow blocks and communication and computation within a block.

Multiple query plans can be created for a single query. There can be a flow block for each group or a single flow block can be stored by all the groups. Based on above parameters, optimizer has to decide whether to use single block or multiple blocks e.g. depending on the physical location and scatter of the nodes, if sensors in a single group are physically close then we can use separate flow block to aggregate the data, since the communication cost to aggregate close-by sensors is usually low. However if sensors from different groups are spatially interspersed, it is more efficient to construct a single flow block shared by all groups.

6.1.3 Sensor information network architecture

Sensor Information Network Architecture (SINA) [Srisathapornphat et. al, 2000] differs from Cougar and TinyDB as it uses SQL-like language for expressing queries but also provides functions which are outside the scope of traditional database systems [Hadim and Mohamed, 2006]. It provides support for scripting using language called SQTL and these scripts are disseminated through Sensor Execution Environment (SEE).

SINA provides abstraction from underlying communication and data gathering process. It abstracts the network of sensor nodes as a collection of distributed objects. SINA uses hierarchical clustering for better data gathering, data dissemination and aggregation. Sensor nodes are clustered on the basis of their proximity in the network remaining power. Cluster heads are elected from within the cluster where data fusion, filtering and aggregation are performed. In case of failure of cluster head, new head is elected again from within the network to perform the assigned tasks.

SINA proposes the use of attribute based naming convention as it facilitates the data centric nature of database-type sensor networks in order to find targeted sensor nodes for the query.

Information management

SINA [Srisathapornphat et al., 2000] provides abstraction from how the data is actually stored in a network. Data is conceptually viewed as in the form of data sheet. Attributes of each sensor node are stored in the cells of data sheet and each cell is named based on the attribute instead of x-y coordinate based location of a cell. Initially at the time of deployment, each sensor node contains few pre-defined attributes and later these sheets can be requested by other node after sensor network creation. This information from other nodes can be used for aggregate or can be metadata information for network maintenance. New information is in the newly created cells and each cell should be uniquely named. The information in new cells can be simple data like remaining battery power or compound data like temperature history.

Data extraction and dissemination paradigm

SINA uses Sensor Execution Environment (SEE) running on each sensor node that interpret and examine all incoming SQTL messages and perform appropriate actions specified in the messages. SQTL is a procedural scripting language that plays a role of interface between application and SINA middleware. It has the capability of interpreting simple declarative queries thus can be used as a query interpreter. An SQTL message can be interpreted and executed by any node in the network. The message is encapsulated in the SQTL wrapper, which acts as message header that contains information of sender, receiver and other application parameters. When a node receives a message, SEE interpret the message and checks the receiver's argument to decide whether to forward the message or not to other node. Once the SQTL script is injected into the network from front-end node, the script is then pushed to other nodes to complete the tasks. After a result is

produced on target nodes, a 'tell' message is generated and the result is delivered back to the requesting node.

In SINA, users may also select sensor information using a query language based on SQL and use built-in query interpreter instead on writing SQTL scripts. SQTL also play the role of event handler by providing event handling construct to process asynchronous events. It handles events such as event triggered by the time or by receiving another message.

Data collection and dissemination (information gathering)

SINA uses three main methods for gathering information or data collection. These are:

1. Sampling operation
2. Self-orchestrated operation
3. Diffused computation operation

These operations are used to gather information from nodes and propagating the collected data back to the requested node. SINA chooses the best method for data distribution and collection based on the type of query and current network status.

When front-end node receives user's query, it interpret and evaluate the query by requesting information from other nodes to improve the quality of response by avoiding the data collision and response implosion problem [Srisathapornphat et al., 2000]. The other task is to minimize the usage of network resources to achieve better life time. Three following methods are used by SINA for data collection. It may use any one of the following technique alone or may use them in combination for better results.

Sampling operation

This technique uses the sampling method where few sensors response to the query while others may not need to response if their neighbors respond. The decision whether to respond or not, is based on the response probability calculated on a node. To avoid response implosion from dense areas, the response probability is then computed on a cluster head. The computation is based on the number of replies required from each cluster. This process is known as Adaptive Probability Response (APR).

Self orchestrated operation

In a small network where responses from all the nodes are necessary for accurate results, each node defers the response to avoid response implosion and collision. Delay can be calculated as:

Delay = KH (h^2 – (2h – 1) r) where

h= number of hops
K= compensation constant
H= constant reflecting estimated delay per hop.
r= random number 0 < r <= 1

Diffused computation operation

This technique makes use of aggregation of data on the way back to the front-end node. The information aggregation logic is distributed to all the nodes with SQTL, so they know how to aggregate data while sending back results. Important assumption is that each node can only communicate to the nodes in its immediate vicinity. Since the data is aggregated on the way back, it significantly reduced the traffic so thus collisions and congestions are avoided.

Location Awareness

Information about physical location of a node is very important for initialization, configuration and operation of a sensor network as sensors operate in physical world and interact with specific physical locations. Location information for SINA can be obtained through GPS receiver for precise location information. For economic reasons, only few nodes may be equipped with GPS receivers and periodically transmit their location as beacon signal to other nodes without GPS, so they can roughly determine their position in the terrain.

6.2 Publish / subscribe middleware

Wireless sensor network applications are mostly based on events and data transmission is triggered when the events occurred. For such type of communications, asynchronous publish/subscribe paradigm is more suitable as compared to synchronous communication [Hadim and Mohamed, 2006]. The publish/subscribe (pub/sub) is a central service in publish/subscribe middleware that acts as a coordinator between local application and other services facilitating the application. It also maintains the list of subscribed topics and publishing messages containing data to the related topics. In WSN, sinks are the event subscribers and sink nodes are the

publishers. The pub/sub middleware layer will coordinate the requests subscribed by the sinks and data publish by the sensors.

This type of middleware provides strong decoupling between the senders and receivers. Pub/sub is very flexible to the changes so the new services can be added to the existing system by implementing the required interfaces. Pub/sub is not suitable for the applications where data is continuously gathered and sent to the end user or the base station because it increases the communication overhead due to extra messaging that drains the energy out of resource scarce devices.

6.2.1 Mires

Mires [Souto et al., 2004; Souto et al., 2005] is a message oriented, event based middleware that allows applications to communicate through publish and subscribe messages. Mires facilitates development and maintenance of sensing based applications and is very flexible to changes. New services can be easily added and removed from the existing system thus communication and computation mechanisms provided by the Mires are scalable and energy efficient. Mires is not feasible for the applications where continuous data collection is required as it would need to poll the information providers continuously that may lead to data congestion and network overload. It is better in supporting applications where data fetching is intermittent.

Mires is composed of many loosely couple services; Basic set of services are publish/subscribe service and routing component. It can have other services like aggregation, security, network maintenance etc. these additional services can be easily integrated if appropriate interfaces are implemented. Publish/subscribe is the core of Mires middleware, which is responsible for coordination between various services. It takes care of advertising the topics, maintaining the list of subscribed topics and publishing the messages i.e. data collection. Pub/sub uses multi hop routing algorithm to transfer data to the sink node.

If additional services are interested to be notified for the certain events by the pub/sub service, they need to implement the service handle for the required events. There are three types of notification in Mires: Topic Arrival event notifies that node application has submitted the data collected from the sensors. State Arrival event notifies that data arriving from the network is submitted by the node application.

Topic Arrival event is a subscribe message broadcasted by the user application. It contains both, the subscribe topics and configuration information for the services. When the data is returned from the node to the Mires, pub/sub service then interacts with the routing component to send the messages containing data to the next node and similarly all the way to the sink node.

Pub/sub interacts with different components of the Mires through various interfaces with the help of broadcast component. It interacts though send, receive and intercept interfaces whereas it uses send and intercept interfaces with MultiHopRouter. MultiHopRouter is responsible for routing the messages towards the sink node. Node's application and pub/sub service interact with each other through publish and advertise interfaces. All other services in Mires interact with pub/sub through Notifier and PublishState interfaces.

Working

In Mires, the complete cycle of interaction takes place in three phases; from request by the application till getting the desired results. These are advertise, subscribe and publish.

Advertise: node application advertise its capability to different tasks to pub/sub service. Pub/sub then sends this message to the Network MultiHopRouter component. These messages are intercepted by intermediate nodes where pub/sub service on these nodes extract the information from the message, update its control structure and send it back through MultiHopRouter towards the sink node.

Subscribe: user application broadcasts the subscribed topic through the sink node. Each node that receives the subscribe message, pub/sub service there extracts the setup information and then notify other service components attached to it.

Publish: when node application obtains a reading from sensor, it invokes the publish command of pub/sub service. Pub/sub informs other services using TopicArrival event. Other services then pass the processing result to pub/sub and invoke its PublishState command. Pub/sub then sends the publish message to the network containing the results from other services using MultiHopRouter. When publish message reaches other node, MultiHopRouter signals intercept event and pub/sub extract the information from message and notify other services through StateArrival event instead of TopicArrival event.

Scalability and heterogeneity

Mires middleware is very flexible as it is very easy to add new services to increase the system's functionality. New services just need to provide the required interface with pub/sub service to work in the existing system. This also provides the heterogeneity support to the Mires as new services can be easily added to process the data from heterogeneous nodes. Currently there is no known service developed apart from aggregation.

Aggregation

Aggregation is used to reduce data transmission over network by combining the data from different sources. Mires support in-network aggregation where aggregation is done on each node on the fly. The aggregation service in Mires can perform its duty during subscribe process. There is an aggregation function which tells how to perform the aggregation and aggregation policy that tells when to stop the aggregation. When aggregation policy is met, pub/sub service publishes the result to the next node. This is how local data is aggregated with data from the network. In Mires, different topics have association with different aggregation functions. It is necessary to monitor and control these associations because a node can receive data from different topics.

6.3 Virtual machine middleware

A virtual machine is a software that acts as a complete machine that can provide hardware abstraction to the application and act as an operating system saving lots of effort needed by the programmers and designers to handle the underlying network and hardware system. Virtual machine adds the flexibility through hardware virtualization and also increases support for heterogeneity in wireless sensor networks. It allows executing the code across the range of heterogeneous devices. However there is significant processing overhead in virtual machine execution that effects battery life.

Virtual machines provide very promising programming model for wireless sensor networks devices. However the virtual machine execution has high impact on power consumption and battery life [Oi and Bleakley, 2006]. Applications can be built through small, separate modules of code. These small modules can be injected into the network as a new block of code or program or a new version or it can also be used to adjust simple parameters for the fine tuning of running applications. Virtual machines have a reduced memory footprint.

6.3.1 Mate

Mate [Levis and Culler, 2002] is a small virtual machine to form wireless sensor networks. Its aim is to reduce the communication overhead by decreasing the amount of packets sent during the software updates. Larger programs are broken in smaller capsules that can itself disseminate through the network. It is byte code interpreter running on motes that allows easy and quick installation of wide range of applications with little network traffic. Mate has its own routing algorithm to forward patches/capsules and it also provides flexibility to write new routing algorithms.

Mate uses synchronous event model that makes application programming very simple. Whenever Mate sends/requests the data from sensors, it suspends its execution context and waits for event completion message. Once the event is complete, it resumes its context. Mate has three execution contexts; send, receive and clock timer, each of them corresponds to an event. Each execution context has an operand stack, used by the instructions for handling the data and smaller stack called return address stack. This stack is used by few instructions to control the program flow, such as managing sub-routine calls. The operand stack for send and receive contexts is not consistent across the execution whereas operand stack for clock remains persistent across execution to implement internal clock timer. It means that if one invocation leaves the reading on top of the stack, it remains available to the next invocation and so on. The three types of operands used in mote are values, sensor readings and messages. Mate also contains a one word heap as shared variable among three contexts, which allow these contexts to communicate in a shared state.

Mate conserves energy and resources like RAM by dividing a larger program into smaller capsules of up to 24 instructions large. A capsule can be sent and received in a single TinyOS packet, thus avoiding a need to buffer partial capsules. Mate has four types of capsules: message send capsule, message receive capsule, time capsule and sub-routine capsule. Sub routine capsules are used in bigger and complex applications where as other types of capsules are related to each of the three execution contexts.

Main features provided by Mate are code management, route management and network management.

Code management

In wireless sensor networks, an existing program or code needs to be updated in all the nodes. Mate is very proficient in code updates on the fly with minimal energy usage. When Mate receives a newer version of code in form of capsules, it installs the code and broadcast it to other neighboring nodes. The receiving node installs the code if version is higher than currently installed and ignores the packets that has lower version of capsules. After installing the newer version, packets are transferred to neighbors and the new code will spread throughout the network quickly with minimal communication and energy usage. New code is normally injected into the network through inject the new capsules on the sink or by introducing new node in the network that contains self-forwarding capsules. These capsules are forwarded to other neighbors as discussed above. The process repeats itself until it spreads in the whole network.

Route and network management

Mate uses a variation of adaptive ad-hoc routing protocol BLESS (Beacon less ad hoc routing protocol). This protocol is included in TinyOS. Mate uses a simpler version of BLESS that only keeps track of one parent node instead of all possible nodes, as being done in TinyOS version of BLESS. Nodes use snooping to gather information from packets directed to other nodes and use this information to find suitable routing tree. Every packet has the routing information stored in it. Each packet has source address, destination address and hop count to the source to prevent wasteful sending of packets in a disconnected graph. Maximum count of 16 hops is the limit and then the parent node is changed if not found within specified interval.

Usage recommendation:

Additional consumption of energy due to instruction overhead restricts the usage of Mate to smaller applications. It is preferable to use Mate in applications where number of executions are small. It is not recommended for large and complex applications which are not going to be updated very often. Mate always adds flexibility and energy conservation in version updates of the code instead of installing new application from the scratch.

6.3.2 Bombilla

Bombilla [Levis, 2002] is a byte code interpreter for sensor networks built on Mate's architecture with few add-ons. Bombilla provides a simple programming interface and error detection mechanism to facilitate the programmers.

In addition to three contexts present in Mate, Bombilla has additional context called 'once' context. This context just runs once when it is installed. This context can be used by users to initialize state, adjust constant and perform any operation that needs only one execution. In addition to buffers in Mate, there are additional typed buffers that can hold up to 10 values. Size of heap is increased to 16 as compared to only 1 in Mate to share among the contexts.

6.4 Mobile agents based middleware

An agent is an autonomous program that can migrate across sensors taking along the code and data. Mobile agents add the flexibility to the applications by enabling modularity and adaptability due to the autonomous nature of the agents. This type of middleware can support multiple applications running simultaneously on the network. The applications can be easily updated due to modular nature of the middleware. Also applications can be added and removed to the network on the fly.

Although monolithic programmed middlewares are more energy efficient and compact, yet the agent based middleware adds the flexibility and they are even superior when smaller fraction of nodes are touched during upgrades. Like virtual machines, it allows application upgrade in small chunks as compared to monolithic approaches where whole code needs to be replaced throughout the network.

Agent based middleware is more expensive for typical data gathering because of the messaging overhead. It consumes more energy due to high processing usage and more data is transmitted as it passed from node to node using agents but during updates, amount of communication is lesser when new instructions are delivered to the network. But Szumel et al. [2005] argues that "using agents can reduce aggregate energy cost in the case where the network must be re-tasked several times and re-tasking does not include entire network".

Agent based middleware also provides support to run multiple applications on a network as agents from different applications. This type of middleware also supports in-network re-programming because new agents can easily be inserted to the nodes to replace old agents.

6.4.1 Agilla

Agilla [Fok et al., 2006] is composed of small programs that can run autonomously and freely migrate from node to node, known as agents. Agilla is built on top of TinyOS and Mate. Its instructions are simply implemented as TinyOS tasks. The use of applications, simplify the programming and adds the flexibility to the network and new applications can easily be injected into the network at runtime. Each agent in Agilla is a virtual machine with its own instruction set and memory.

Agilla middleware maintains the network by keeping a list of neighboring nodes, agents and maintains a tuple space. Tuple space acts as a shared memory where agents insert the data in form of tuples or read the data from the tuple space inserted by other agents. Each sensor node in the network has its own local tuple space, which can be read and updated by all the agents running on that particular node. Agilla also provides some flexibility to the agents to access the remote tuple space. Tuples are stored as a set of fields where each field has a type and value combination. Stored tuples are accessed through pattern matching using the templates. Templates are also in the same format as tuples. When an agent on a node wants to access tuple in a tuple space, it provides a template that should have same number of fields as tuple and each field should match the corresponding field in the tuple. If more than one tuple in a tuple space match the template, only one of the matched tuples is returned indiscriminately.

Agilla also uses reactions. Reactions are like triggers and they are fired when a matching tuple is inserted into the tuple space. Reaction consists of a template and a callback function, which is a block or code that executes when reaction fires. When reaction fires, its counter is set to the first instruction of the callback function and the execution starts. Reactions are fired only once for the pre-existing tuples even if more than one matching tuple exist.

Architecture

The Agilla middleware consists of various managing processes (managers) that are coordinated by Agilla engine. These managers are:

Agent manager: an agent manager maintains agent's context by allocating memory on arrival of the agent and de-allocating on the exit. The allocated memory consists of agent's execution state and code. Agent manager determines and notifies the Agilla engine when agent is ready to run.
Context manager: agents identify the location of the host node and its neighbors through context manager.
Code manager: it dynamically allocates the memory to the migrating agents to share the code. Code manager also retrieves next instruction from the code during local execution of an agent.
Tuple space manager: tuple space manager is responsible for managing the contexts and operation of the local tuple space.
Reaction manager: it registers the reaction for agents and when reaction fires, it notifies the agent manager. When a new tuple is added in a tuple space, it checks for all the registered reactions for the new tuples and if match is found, it fires the reaction and notifies the agent manager.
Agilla engine: this is a core component to manage agents' activities on a node. Agilla engine handle the arrival, departure and execution of multiple agent concurrently. In addition to agent control, it also performs remote tuple space operations. It sends request to the remote node with required instructions and template, where operations are performed locally by tuple space manager and result is sent back to the Agilla engine.

Location awareness

Agilla does not use network addressing for nodes; rather it addresses the nodes based on the geographical location. Node's address is its geographical location represented in form of (x,y) coordinates. This type of technique is called localization. Agilla may localize itself using GPS or any other localization technique.

Network reliability and delays

Agilla uses techniques like timers and acknowledgements to achieve the reliable transmission of messages. As the messages are sent in multiple packets, all of them are acknowledged on each hop. If a node does not get an acknowledgement, it waits for short time and resends the packets. All this is done on each hop on a way towards the destination, which also incur extra load on a

network. Agent migration has significantly higher overhead as compare to unreliable transmission.

6.4.2 Impala

Impala [Liu and Margaret, 2003; Liu et al., 2004] is an agent based middleware, primarily designed to achieve modularity, adaptability and energy efficient updates of the applications on the fly in an energy constrained sensor network environment. The modular nature of Impala brings the flexibility and reliability to sensor networks that are deployed in harsh and unchecked environments.

Impala is suitable for environments where data is periodically captured and processed, and updates are less frequent because the overhead in event delivery is high as compare to monolithic programs. Impala can be used efficiently for small updates as well as complete software upgrades or re-installations.

Architecture

Impala middleware consists of three main components.

Application adapter agent: it is used to adapt the application to various conditions and situation to get the optional performance and energy usage. Adapter makes use of different parameter values and based on these values, it selects one of the available routing protocols to increase efficiency or to reduce power usage. Application parameters may include but are not restricted to; the amount of sensor data, battery available, transmitter range, geographical position of the system and the type of hardware available etc.

Application updater agent: updater is used to distribute and install the software updates on nodes. It is responsible for managing all the update activities. It keeps track of different versions of complete and incomplete updates on a node. Nodes can have incomplete, uninstalled updates of different versions if the updates are frequent and high losses are present due to unreliable transmission medium. In addition to versioning and transmitting updates to the nodes, updater is also responsible for installing the updates on all the nodes.

Event Filter agent: This agent is responsible for catching and dispatching events to the adapter agent, updater agent and to all the hosted applications. All the events are processed sequentially for the sake of simplicity and all the events have limited time for processing in order to avoid blocking. There are few events handled by the event handler: Timer event, Packet event, Send done event, Date event and Device event.

Management

The updater agent provides very efficient network re-programming. The adapting nature of Impala can improve routing performance that makes it more energy efficient and robust. The use of agents or the modular nature of the middleware improves the network management.

Energy conservation

The Adapter module gets the optimal energy conservation by selecting the correct routes and nodes for application requests. Middleware forces the application to use certain paths and avoid the nodes where resources are low.

6.5 Tuple space middleware

Tuple space [Costa et al., 2009] acts as a shared memory for the agents, applications and users where data is shared in the form of elementary data called tuples. The collection of these tuples together in a form of virtual shared memory is called tuple space. Tuples can be read and updated by the agents or small set of commands (programs) running on the node. Tuples are stored as set of fields where each field has a type and value combination.

Stored tuples are accessed through pattern matching using the templates. Templates are also in the same format as tuples. Templates either contain values called actuals or wild card variables known as formats. When a template value matches a tuple in a tuple space, it returns the tuple to the requester. If multiple tuples match the template then only of the tuples is returned indiscriminately.

This type of middleware provides abstraction to the users and applications that, data collected by the sensors is readily available in a central shared memory. The programmers don't have to worry about data collection techniques and sampling rate of the sensors. Different individual components of the sensor networks communicate with each other through the shared tuple space. It is basis of the highly decouple coordination among the processes. The tuple generator and

requestor need not to be simultaneously available at one point in time or even at a same location. They even don't need to have mutual knowledge of each other.

Another important feature of tuple space based middleware is the use of reactions. Reactions are like triggers. The reaction contains a template and a set of code or a procedure. When a tuple is inserted in a tuple space and it matches the reaction's template, reaction fires and start executing the procedure to perform particular task. Reaction helps in limiting data collection and processing by providing limits to data collection.

The tuple space can be shared among devices in wireless sensor network or access can be restricted to just local tuple space. In this way tuple spaces can be used to define scope of the data for different components in the sensor network.

6.5.1 TinyLime

TinyLime [Curino et al., 2005b] is a tuple space based middleware from wireless sensor networks. It provides the data to the application through a tuple space interface that acts as shared memory between sensors and applications. Sensors collect data from environment and place them in one virtualized central location in form of organized tuples. This central location is called tuple space.

TinyLime is an extension of popular middleware for ad-hoc mobile networks know as LIME [Murphy et al., 2001], which adapts and extends the tuple space model made popular by Curino et al. [2005a]. TinyLime extends LIME by providing features specialized for sensor networks while maintaining Lime's coordination for ad-hoc networks [Curino et al., 2005b].

TinyLime proposes a new operational model which rejects the need of centralized location like one powerful base station or a powerful sink for collection and transmission of sensor data. TinyLime proposes that it is not necessary for the nodes to communicate the data to a single central location like base station rather there can be more than one base stations and data is collected only from those sensors, which are within one-hop distance from the base station. This removes the burden of multi-hop communication from node towards the base, to save energy.

TinyLime proposes the settings where all the sensors are not necessarily interconnected and sends the data to central sink node. Rather sensor field has many sinks and each communicates to the

small set of sensor nodes. Some of the devices in the network are only clients and they can only communicate with the sink for requesting and transferring data.

Architecture

Basic feature of TinyLime is to provide abstraction to the transiently shared tuple space that contains all the sensor data. In addition to the host, clients and agents, TinyLime introduces a new component called motes. Mote is only visible in TinyLime when it is connected to the base station and is presented as other agents residing on the base station [Curino et al., 2005a]. This kind of abstraction provided by TinyLime is just for the ease of programming.

Tinylime middleware contains client components, base station and mote components. Both client and base station contain MoteLimeTupleSpace (MLTS) component that is primarily used for interaction between client and base station. It gives the abstraction of a single tuple space but actually contains two tuple spaces; mote tuplespace (MTS) and config tuplespace (CTS). MTS provides access to the sensor data. When a client requests the data, MLTS first looks in the MTS for fresh data and if data is not found it asks from the base station through CTS to query the data.

When the base station gets the request for data, it uses two additional components to get the data from the motes: MoteAgent (MA) and TOSMoteAccess (TMA). MoteAgent receives client request and TOSMoteAccess translates this request into packets, understandable by motes. All the communication between base station and motes is done using simple messages. To conserve energy, motes sleep most of the time and wake for nominal wake time. Nominal wake time is the amount of time motes are configured to be awake during each epoch. Motes don't collect data during sleep time and base station needs to repeatedly send packets to request data unless it gets reply from the mote when it awakes. Mote only sends single packet to the base station for the sake on energy conservation.

Mote components in TinyLime Middleware manage the epoch and wakeup time of each mote and are also used for responding to the incoming messages. Logging and aggregation components on mote are used in reading the sensor values and logging them for aggregation purposes. The SensorSubsystem is used to take sensor readings using appropriate TinyOS components.

Energy preservation techniques

TinyLime uses few of the conventional energy saving techniques employed by many wireless sensor network middlewares to make it suitable for sensor networks.

Aggregation: TinyLime uses both local and global aggregation to save energy by decreasing the amount of data sent over the network. Local aggregation uses values collected by single sensor whereas global aggregation is applied over data collected by multiple sensors.

Reactions: reactions also help in restricting the amount of computation and communication in order to save energy. Normally, sensor continuously collects data because it does not know when the data can be requested by clients. If reaction is registered, then sensor only gathers the data when reaction fires thus saving energy.

Sampling: TinyLime provides the flexibility to the sensors to either actively sample the data through continuous collection of data or passively collects the data. In passive sampling, sensor sleeps for the given interval of time to save energy and then wakes up to collect the data and then automatically go to passive mode and sleep.

6.5.2 TeenyLime

TeenyLime [Costa et al., 2006a] gives the concept of wireless sensor networks where computation is not limited to the powerful sinks. It gives the concept of complete decentralized network where all the data collection and processing is done on the sensor nodes. All the devices in sensor networks will participate in computation and processing of data without relying on external devices, such as sink powerful sink nodes.

TeenyLime uses the tuple space model where tuple space is spread over all the devices and shared with single hop neighbors. Each device has its own local tuple space. These local tuple spaces together make the transient tuple space. The transient tuple space is only restricted to single hop neighbors. Each device has different set of neighbors. The information about single hop neighbors is stored by teenylime in a special tuple space.

6.5.2.1 Architecture

TeenyLime has a tuplespace interface through which applications access TeenyLime. Requested operations by the applications are either sent to the 'LocalTeenyLime' component or 'DistributedTeenyLime' component, depending on the scope of the request.

LocalTeenyLime component stores tuples and reactions in the local tuple space of the device. It also handles the tuple returned due to reaction fire and communicates to rest of the TeenyLime system through LocalTupleSpace interface.

DistributedTeenyLime component is used for remote tuple space operations. Reaction on remote devices is handled by this component. When DistributedTeenyLime receives the request, it delegates the request to the LocalTeenyLime which handles the request and returns the matching tuple to the DistributedTeenylime through localtuplespace interface. TeenyLime adds the flexibility to the WSN middleware by providing new features.

Range matching

In range matching, pattern does not only match against a single value in a tuple, rather it can be also match against the range of values. This adds the power to select a tuple for a big range of values in a single go, thus adding the flexibility and limiting the amount of communication.

Capability tuples

Capability tuple is used to indicate that the current device has the capability to return tuples for the requested pattern. Due to presence of capability tuple in a tuplespace, mote does not have to sample data regularly. When the capability tuple matches the pattern, it asks the sensor to gather the data and then build the tuple on the fly and place it in a tuplespace. TeenyLime takes this newly created tuple to the requester. In this way, capability tuple just acts as a place holder for actual tuple and send request for the actual tuple when pattern matches.

Data freshness

Like other tuplespace base models, TeenyLime also has the ability to gather the time sensitive data using epochs of constant lengths. Whenever a tuple is placed in a tuplespace, it is stamped with current epoch value. TeenyLime uses this timestamp to gather fresh data or data in any available time range.

Network management

TeenyLime also creates a special tuplespace where each tuple represents the device in the neighborhood. Each device listens passively to the messages from neighboring devices. These messages contain the tuple called 'NeighborTuple'. This special tuple is used to populate the device tuple space. This way, each device has the knowledge of neighboring nodes and thus is helping in maintaining the network.

When a new node is added or old one is replaced, it performs the read group (rdg) operation to discover the nodes in its neighborhood. In this way newly added device also informs others of its presence. When a new tuple is added to the device tuple space, the reaction is fired to inform the application about the new addition of a new device.

6.6 Other middleware models

Many wireless sensor network middlewares that do not come under any of the models that are presented before in this thesis such as database inspired, agent based, VM based, tuplespace based and publish / subscribe models. The frameworks provided by the following middlewares use concepts different from the above mentioned models. The other models can be a rule based, role based, security based model or it could be emphasizing on reusability component based model or the adaptability of middleware to provide desired Quality of Service (QoS) to the applications. Following are few middlewares that differ in programming abstraction from the middlewares explained above.

6.6.1 MiLAN

MiLAN [Murphy and Heinzelman, 2002] is different from other WSN middleware model as it tries to directly address the issues to provide QoS to the resident applications rather than just providing abstractions for ease of programming. It is a 'Middleware Linking Application and Network'. It is placed between application and network stacks to provide interfaces to low level components. It gives more control over low level network components and provides detailed network monitoring.

MiLAN provides high level abstraction to the programmers by easing design and implementation of the applications. It takes requirements from the applications, monitor the network and adjust

the underlying network resources to fulfill the application requirements that can be used to achieve desired performance, reliability, accuracy etc. MiLAN continuously adjusts the network configuration to meet the application's needs and tries to minimize the cost.

Working

MiLAN will trade off between the available application requirements, network cost and energy consumption. These three metrics make the basis to achieve best performance at the low acceptable cost.

Application performance data: MiLAN needs to know the requirements of the applications and benchmarks for reliability, performance and data gathering parameters. It defines the minimum acceptance level for each QoS parameter depending on the current status of the application. Data provided by the application helps MiLAN in choosing the appropriate set of sensors while also considering power costs.

Network costs: MiLAN also makes its decision on the basis of network limitation and current usage by continuously monitoring the network to determine the set of nodes that can be used in current operations.

Energy costs: like all other WSN middlewares, MiLAN also tries to minimize the energy consumption. It tries to use only those sensors that have high available energy and low usage for on going operations.

6.6.2 RUNES

Reconfigurable Ubiquitous Network Embedded Systems (RUNES) [Costa et al., 2006b] is a component based model aimed to provide mobility, adaptability and heterogeneity. It tries to address each constraint and programming challenge individually by providing completely modular and decoupled sets of component addressing functionality or a set of functionalities. RUNES aim is to provide middleware consists of reusable components to achieve adaptability and modularity.

RUNES middleware consists of components, interfaces and reciprocates encapsulated together in a capsule. Component encapsulates the functionality of a single unit that can be any service

provided to the application by RUNES middleware. Components interact with each other through interfaces. Reciprocates are special interfaces that are required for explicit dependencies on other components.

When similar types of components which address similar functionality area are working together, it is called component framework (CF). New components can be easily added in CF as add-ons to extend or modify the CF behavior. CF can provide any service to the application such as memory management, network services like flooding, multi hop routing, localization etc. only the required services need to be added to the middleware to provide abstraction to the application.

RUNES also contain a reflective meta model which is used for system re-configuration or system adaptation.

6.6.3 FACTS

FACTS [Terfloth et al., 2006] is a rule based event driven middleware that provides triggers to catch the events and actions in response to perform operations. FACTS main focus is to provide easy and flexible abstraction to the programmers and provide some level of QoS to the programs instead of focusing of limiting energy consumption. The middleware uses facts, rules and functions to provide abstractions to the developers and they combine together to provide set of services to the application. Each service can be provided in the form of component that can be added to the middleware on demand thus also focusing on the need of the targeted applications.

All the system's data is represented in the form of facts and stored in centrally in facts repository. A repository can consist of multiple facts with the same name. They can be identified using unique key-value tuples known as properties. There are few mandatory values in each fact; they are owner, time, id and modified.

Rules in the middleware are set of conditions and reactions to the external events in a system. A rule fires when all the conditions are true and at least one of the facts mentioned in the conditions is tagged as modified.

Function acts as an interface between rule engine and underlying operating system or firmware. It is a piece of code that allows implementation of critical performance algorithms.

Each node in FACTS middleware contains its own set of facts, rules, functions and rules engine. All the data on a node is stored in facts repository and it acts as a distributed stored memory for the software. Facts can be dynamically added to the facts repository. Rule acts as event to the application thus providing highly flexible, event-centric programming paradigm and providing abstraction the underlying distributed network and asynchronous event handling.

7 Middleware evaluation and comparisons

This chapter presents the evaluation of different middleware models and architectures that are presented previously. The comparison is done with focus on design challenges arise due to the distributed and resource constraint nature of the wireless sensor networks. This chapter also focuses on the programming abstraction provided by each middleware at the node level as well as at the network level. The middleware solutions are compared with each other keeping in view the challenges pose by the constraints and the complex nature of wireless sensor networks. These constraints increase the challenges in designing and developing the applications for wireless sensor networks.

A WSN middleware needs to address many issues to ease the development of the applications. It should be able to provide optimal power consumption model, support for runtime application upgrades and should be able to provide efficient network and data management techniques. Good middleware solutions are not application specific and provide solutions for bigger set of design challenges. The good designs address the power constraints by providing the solution using adaptive sampling rates and minimizing the communication using optimal routing protocols. A middleware should also focus on secondary goals such as security, adaptability, operation reliability and user friendliness. The primary aim of all the middleware is to achieve high level abstraction for the developers by easing the application development and to conserve the energy whenever possible.

Each table in this chapter represents a set of challenges grouped together and a list of middlewares addressing these challenges. These tables provide clear comparisons among the middlewares and also help in identifying the similarities and differences in the middlewares approach towards the design challenges outlined in the tables.

7.1 Power conservation and usability

This section compares middlewares that address the challenges and design issues inherent due to the nature of WSN based applications. Power conservation is the most important of all the issues that all the WSN middlewares are trying to address. They try to conserve power though changing sampling rates, limiting communication and through distributing computing. Second important issue is the level and type of abstraction each middleware provides to the developers and users.

	Power conservation	Programming Abstraction & Usability	Usage
TinyDB	-Event based queries to restrict transmission and distributed queries on nodes for load sharing. -Many aggregation techniques. -SRT, adaptive sampling and transmission rates are used for conserving power during routing.	-Database abstraction -Easy query based programming interface. -Supports event based, time based, simple and aggregate queries. -No node level control.	-Support small range of applications. -Simple sensing and data collection applications. -Suitable for environmental monitoring.
Agilla	-Conserve power during application upgrades through restricting packets. -Power overhead due to modular code execution.	-Need programming knowledge of agents so little hard to code. -More control due to node level access.	-Supports for all types of applications. -Good for small and frequent updates.
Mires	-Conserves power due to direct node to node messaging. -Communication overhead in continuous data collection.	-Message based asynchronous programming. -Needs programming skills to implement services and interfaces. -GUI available for subscription to existing services but Interfaces for services need to be implemented.	-Supports multiple services and applications. -Not good for continuous monitoring and surveillance applications. -Wide area coverage.
Mate`	-Save power during small upgrades. -Energy overhead in executing VM byte code. -Uncontrolled propagation of packets.	-Need knowledge of assembly language. -Easy programming due to small set of instructions.	-Good for small updates like version updates. -Not suitable for long running continuous applications.
TinyLime	-Conserves energy by limiting communication due to clustering and high data filtering. -Overhead due to agents and	-Provide abstraction of a centralized shared memory. -Database like column/tuple and triggers. -Ease programming as it	-Sparsely distributed networks. -Different types of clients move around to collect data.

	TS management	hides distributed nature of the system. -Small set of supported instructions.	
MiLAN	-Select node with high power availability and less power usage during data collection.	-Abstraction from underlying network protocol stacks and hardware. -No defined user interface available.	-Focus on benchmark requirements from the applications. -Good for all types of applications with continuous network management.
Cougar	-Distributing the computing on nodes and query optimization on base station. -Different aggregation techniques.	-Abstract as a centralized database. -Easy to use using SQL like queries.	-Not suitable for large scale networks. -Suitable for frequent and infrequent sensing and data collection.
Impala	-Small modular application updates save communication.	-Need to programming knowledge of agents for adding services. - Users have more control over the network and nodes.	-Suitable for small and large infrequent updates and long running applications. -Tracking mobile object applications.
SINA	-Use clusters to limit messaging in number of nodes, thus limits the amount of data transfer. -Use of different types of aggregations.	-Easy to use SQL like query interface. -SQTL scripting provides additional flexibility.	-Suitable for massively distributed networks. -Good for data inquiries.
TeenyLime	-Capability tuples and reactions to save energy. -Single hop communication saves node energy.	-Abstraction of database type shared memory and transient sharing of tuple spaces. -Allow node level interactions.	-Large heterogeneous networks. -Computing is spread all over the network.

Table 6: Comparisons among middlewares for power conservation and usability.

From the values in Table 6, we can deduce that DB-oriented middlewares conserve less power because of its autonomous nature as compared to other middleware models. The other

middleware models consume more energy because they are either modular or contain intermediate services or separate byte code interpreter that increases the instruction overhead, especially when code execution is frequent. In the case of energy conservation and preservation, TinyDB stands out from other middlewares based on the variety of techniques used to save energy followed by Cougar and SINA respectively. All the middlewares belong to the same DB-oriented group. Mires has high energy consumption because of extra communication needed in exchanging messages in publish subscribe models. Whereas, continuous usage of byte code interpreter coupled with broadcasting of messages in Mate` has considerable energy consumption overhead which makes it less suitable for energy scarce wireless sensor networks. Other than the database-type middleware, energy consumption in Agilla and TinyLime is more conservative as compare to rest of the middlewares because of their hybrid nature and controlled communication that limits the energy usage despite being modular like rest of the middleware models.

The abstraction provided by the middleware is analyzed in terms of level of abstraction and level of usability. The level of abstraction shows whether the abstraction is provided at the node level or at the system level. Node level access increases the programming complexity but it gives more control over the system and its interactions to the developers. Level of usability is determined by the ease of programming, user friendliness and wide usage of the middleware model. TinyDB and Cougar are very user friendly that have SQL-based interface that make the programming very easy but the level of control over nodes is very limited and the programmers don't have much liberty to manipulate the system. The agent-based and message-oriented programming is complex but they provide more control over the system that can also increase the usability of the middleware expanding to different domains. Agilla, Impala, Mires and TeenyLime have deeper control over the system respectively. Milan also intends to provide low level control to the applications to achieve quality goals that each application intends to achieve.

The DB-type middlewares do not provide node level control to the programmer. They present whole network as a single centralized database. Programmer can just perform the data centric activities over the network without being able to go into the details of data collection from each node. This is automatically handled by the middleware thus providing system level DB-type abstractions. Mate´ is the middleware that is able to concentrate on the node level and programmers can do some activities using assembly language but the level of control is also limited because of the small set of instructions. The tuple space based middleware also provide abstraction similar to the database-type middleware. In tuplespace middleware, data is stored on the nodes and presented in the form of tuples, similar to the database systems and instead of using

SQL-like language, it uses templates to extract the tuples. In tuplespace-based middleware, more control is achieved with the use of agents that facilitate in routing the packets and extracting the data. From the comparison data in the Table 6 we deduce that level of abstraction also drive the usage and coverage support of the middlewares. The simple and data centric approaches, such as TinyDB and Cougar support small networks which limits the range of applications with emphasis on simple sensing and data collection. TeenyLime and TinyLime are also good candidates for data collection and sensing applications. These middlewares are scalable and cover wider area due to absence of base station that decreases the multi-hop communication and node mobility. Agilla, Impala and Mires are able to support wider range of applications where goals are mix of sensing, data collection, and updating, computing and in-network analysis. Due to their modular and decoupled nature, they can be used in widely distributed networks. Mate´ has a limited scope of working. It mainly focuses on updates and upgrades of the code running on the nodes. It is not suitable for data collection thus we can say that it is more of a middleware component rather than a complete middleware. Milan is the only application oriented middleware so it has the capability to support wider range of applications.

Based on the data in Table 6, top picks for power friendliness and usability in my opinion are as follows.

TinyDB – For user friendliness and more power awareness.
Agilla – Due to its hybrid nature, it encompass more types of applications.
SINA - Data centric with the ability of scripting using SQTL and covers wide areas.
TeenyLime – Hybrid due to the use of tuplespaces and agents. Independent of support from sinks and base station.

7.2 Network and system QoS

In this section we show the flexibility, reliability, robustness and adaptability of a network and the applications running on it. These features are very important for all the systems and middlewares should be able to provide them and these are the primary goals that need to be achieved by all the middlewares. These characteristics provide quality to the network and applications and hence increase the quality of the middleware itself.

These are characteristics all the middlewares should strive to achieve in order to add quality to the network. These characteristics are the desired in any WSN middleware and they are considered very important after power awareness and usability.

Mobility and heterogeneity adds flexibility to the network. It increases the usability of the network and also increases the number of targeted applications. Mobility is the ability of sink and sensor nodes to move around in a network. The middleware should be able to support mobile nodes to provide this feature to the networks and the applications running on them. The node movement results in change in network topology so middleware should be able to support the changes and adjust according to the new topology. In addition to the mobile nodes, there are mobile applications and services that move from node to node and change the target users. Heterogeneity also adds flexibility to the applications and scope of the applications where middleware is able to fetch and manage the data from different type of nodes. Security from the external threats in wireless sensor networks is still considered trivial and none of the middlewares presented in this work provide any kind of security to the WSN systems.

	Mobility	Heterogeneity	Security	Adaptability	Scalability
TinyDB	-Static nodes.	-Homogenous.	-No	-Rate adaptation queries for power saving. -Uses reactive adaptation.	-Not scalable. -Cannot add new nodes.
Agilla	-Static nodes. -Agents mobility. Agents move from node to node.	-No support for Heterogeneity. -Support for heterogeneity can be added because of VM interpreter in agents.	-No.	-Supports both proactive and reactive adaptation. -Highly adaptable.	-Highly scalable.
Mires	-Limited node mobility.	-Supports due to asynchronous nature.	-No.	-Adaptable.	-Highly scalable due to decoupled services.
Mate`	-Strong support for mobility.	-Supports heterogeneity.	-Little security due to	-No adaptability support.	-Scalable.

			own VM.		
TinyLime	-Mobile client nodes.	-Heterogeneous.	-No.	-N/A-	-Scalable.
MiLAN	-Static nodes.	-No support for heterogeneity.	-No.	-Adaptable -Proactive route adaptation.	-Not scalable.
Cougar	-No mobility.	-Homogenous.	-No.	-No adaptability.	-No scalability.
Impala	-Strong support for mobility.	-No support for heterogeneity.	-No.	-Highly adaptable -Uses adapter module for auto performance adaptability.	-Highly scalable.
SINA	-No support for node mobility.	-Homogenous.	-No.	-Reactive adaptability for sampling and routing.	-Not scalable -Scalability can be added due to clustering.
TeenyLime	-Little support for mobile nodes.	-Support heterogeneous nodes.	-No.	-N/A-	-Scalable.

Table 7: Comparisons among middlewares for System's QoS attributes.

Data in the Table 7 shows that all the database-oriented middlewares are rigid and static. The DB-type middlewares do not provide any support for node mobility, heterogeneity and scalability. This group of middlewares has no support for runtime enhancements thus these middlewares can only be used in environments where long running unattended operations are required that do not need any updates and future enhancements. Only Impala and Mate´ support node mobility because of their highly modular and decoupled architecture. Agilla and TinyLime do not support node mobility but they support application mobility due to the mobile agents moving from node to node. TeenyLime has a special case of partial node mobility. Mobile nodes in the case of TeenyLime are the clients which join the network only to access information from the nodes but they do not become the working part of the network. Mate´ and TeenyLime also add support for handling heterogeneous nodes in a network. No other middleware listed in the Table 7 support heterogeneous networks. As Agilla runs on top of Mate´, it may be able to handle the heterogeneous nodes in the future but currently there is no such known support available in Agilla.

Scalability is the ability of the network to add new nodes to the existing network at runtime to add to increase the size and coverage of the network. Scalability is important for the future growth of the network and applications, without it the system will be static with no scope for growth and increasing the coverage. From the data in the table, we can infer that scalability in WSN is directly dependant on the modularity of the middleware. Monolithic middlewares like DB-oriented are not scalable and cannot be expanded once deployed. All other middleware such as Mate´, Agilla, Mires, TeenyLime and TinyLime are modular in nature and all these middlewares provide scalability to the network.

Adaptability is the ability to adjust itself based on the existing conditions, requirements and working environments such as based on power consumption, route congestion and so on. There are various levels of adaptability present in different types of middlewares. Some middlewares are more adaptable according to new required situation than others. For example, SINA is more adaptable than other middlewares because it can adapt itself to new sampling rates as well as adapting to the new routes. Adaptability in WSN middleware is classified as reactive and proactive. Reactive adaptability is when a middleware adapts itself to the situation when events occur or in response to some new situation. Proactive is when the middleware adjusts itself before the event or perceived situation. Middlewares in WSN mainly adapt themselves by either changing the sampling rates or by adjusting the routes and paths to avoid unwanted situations.

	Reactive	Proactive
Sampling adjustment	TinyDB, TinyLime, SINA, TeenyLime	Impala
Routes adjustment	Agilla, SINA	Milan, Impala
No adaptability	Mires, Mate´, Cougar	

Table 8: Adaptability matrix for WSN middleware.

Table 8 shows that how middlewares are grouped together based on their adapting technique and level of adaptability. SINA and Impala belong to both groups, showing that Impala has capability to do the proactive route adjustment where as SINA adjusts both sampling rates and routes reactively. Mires, Cougar and Mate´ are grouped together as non adaptive middlewares.

7.3 Application QoS

Middlewares need to provide some services to the residing applications to maintain and improve the quality of the applications and network for the end users. WSN middlewares provide services to the applications that ensure the timely and accurate data collection from the nodes to the application users through reliable packet delivery and data integrity checks. It also improves the quality of the applications using code management at runtime where users and developers don't have to worry about the methodology of how to update the codes without any downtime and redeployment.

	Accuracy (data and messaging)	Runtime support
TinyDB	-Lack of acknowledgements and data checks makes it less accurate. -Queue overflows and data discard decrease the data reliability.	-Not flexible for updates. -Complete re-programming needed for updates.
Agilla	-Use timers and acknowledgements for message accuracy. -Reliable operations.	-Highly modular architecture helps in code upgrades. -Code upgrades are spread using agent migration.
Mires	-No support for data authentication. -Unreliable operations.	-Modular and decoupled that provides runtime support. -New services can be easily added. -No known support for code upgrades.
Mate`	-Unreliable data routing thus lacks accuracy. -Less prone to bugs due to synchronous communication that makes it reliable.	-Small modules of code. -Proficient code updates specially in version updating. -Support runtime re-programming.
TinyLime	-Accurate data delivery. -Uses epochs and acknowledgements during packet exchange.	-Modular due to use of agents but no known runtime support for code updates.
MiLAN	-Lacks accuracy.	-N/A-
Cougar	-Temporal accuracy due to use of timestamps. -Lack of integrity checks makes data less accurate.	-No modularity and runtime updates.

	-Local and bunch repair increase reliability.	
Impala	-Accurate data for local node applications. -Reliable and robust due to adapter module.	-Updater module used for updates. -Supports on-demand updates.
SINA	-Checks for accuracy at sink nodes increase data accuracy.	-No modularity -No code updates. -New queries and scripts can be inserted using SQTL. SEE is used to propagate it to the nodes.
TeenyLime	-Supports both reliable and unreliable operations. -Reliable operations guarantee accuracy.	-No known support for runtime updates.

Table 9: Comparison of middlewares based on WSN applications QoS attributes.

First column in Table 9 shows that whether middlewares provide any support to increase the operation reliability and data accuracy using techniques like acknowledgements, timer and collision detection. Lack of accuracy means less reliable middleware that cannot be used in data critical applications.

TinyDB, Mires, Milan and Mate´ provide unreliable operations due to lack of data integrity checks and acknowledgments on data arrival. Although lack of these checks decrease the communication overhead but it also compromises the data accuracy thus these middlewares can only be used in systems where data accuracy is trivial. Agilla, TinyLime and Cougar provide basic checks such as time checks and acknowledgement on data arrival to provide some accuracy to the data. Impala provides reliable and accurate data to the applications by implementing additional features like route adaptability and congestion control, TeenyLime is more flexible than other middlewares as it provides both reliable and unreliable operations. Reliable operations cause communication overhead but ensure the packet integrity and data accuracy.

WSN middleware can add the flexibility to change the applications on the fly. Users should be able to add and remove the components to the network without bringing down the system. Middleware should be able to provide support for updating the applications or parts of the applications.

We understand from the data provided in Table 9 that decoupled, modular middlewares are better for code updates and runtime support. TinyDB, SINA and Cougar are monolithic in nature. If applications running on these middlewares need to be updated then the whole application needs to

redeployed with new version of code. It is not possible to channel these updates at runtime for these middlewares. Mires is a modular middleware where new services can easily be added to the existing system which make it very good choice for the application changing at runtime. Mate´ is primarily used for updating the applications. It is very good for small version updates. Impala and Agilla are the most flexible among the middlewares listed because of the use of agents. Agents are independent, modular and completely decoupled having there own VM that helps in code updates, application upgrades as well as addition of new modules by simply inducting new agents to the system.

7.4 Data management

Due to the inherent data centric nature of the sensors, WSN applications are naturally biased towards sensing and thus emphasis is on data collection, data filtering, data analysis and data propagation techniques from sensor towards the sink nodes and base stations. This increases the importance of data management and data related operations in wireless sensor networks.

	Acquisition & sampling	Filtering & congestion control	Storage and computation	Aggregation
TinyDB	-Time based sampling. -Query base inquiry. -Event driven sampling. -Continuous sampling. -Full control on adjusting sampling rate.	-Data prioritization and rate adaptation for congestion control. -Uses snooping to decrease communication during aggregation.	-Data is stored in virtual 'SENSORS' table spread across the nodes. -Legacy data is stored on the base station. -Decentralized computation on nodes. -Data management, query optimization on sinks.	-Supports wide range of aggregation functions. -Node aggregation and in-network aggregation using pipelining.
Agilla	-Sample using agents' instructions.	-Use reactions, timers and event filters.	-Data stored on nodes in local tuple spaces. -Updates are inserted at sinks and forwarded to nodes.	-No known aggregation functions available.

Mires	-Event driven sampling. -On subscription by service.	-Data only sent to subscribed services. -Auto filtering due to pub/sub.	-Data is passed to sinks/base station where computation is done.	Basic aggregation functions like avg, max and sum. -In-network aggregation is done when aggregation policy meets.
Mate`	-May support any method.	-No filtering and congestion control.	-Very small storage. -New code is introduced in sinks and forwarded to nodes. -Computation is done on nodes.	-No aggregation.
TinyLime	-Event based collection in passive mode using reactions. -Continuous collection in active mode.	-Reactions are used to limit data. -Passive data collection.	-Data stored in local tuplespace on nodes.	-Global aggregation on the fly. -Basic support for aggregation functions.
MiLAN	-Sampling depends on the application.	-Use cluster to minimize nodes for sampling. -Use network data to minimize data flow.	-Data storage and computation on sink nodes. -Also proposed to move computation on nodes for energy conservation.	-No known support for data aggregation.
Cougar	-Sampling on query. -Continuous time based.	-No good support for data filtering and congestion control.	-Sensor data on sensor nodes. -Stored data on base station. -Computation distributed on sensor nodes. -Query optimization on base station.	-Partial aggregation is done on nodes. -Aggregation is mainly performed on the sink node of a cluster.

Impala	-Event driven sampling.	-Route adaptation to avoid congestion.	-Local storage. -Computation on local nodes. -Final data collection on base stations.	-No support.
SINA	-Query based selective sampling. -Avoid using all nodes for sampling.	-Sampling limited to few nodes. -Response deferred using self orchestration. -Diffused computation.	-Data store on sensor nodes in associative spread sheets. -Mostly processing is done cluster heads and sinks nodes.	-In-network aggregation on the fly. -Aggregation related info is spread in the nodes using SQTL scripts.
TeenyLime	-Event based using reactions. -Active collection in normal mode.	-Limiting sampling using reactions and capability tuples. -Decrease communication through range matching.	-Data store in local tuplespaces on nodes. -Processing on nodes. -Data shared through transiently shared tuplespace.	-N/A-

Table 10: Comparison of middlewares based on data management.

Starting from data acquisition methods and varying sampling rates, Table 10 shows that TinyDB and other database-type middlewares are more flexible in data related operations. TinyDB supports variety of data acquisition techniques and varying sampling rates. It can adjust the sampling rates and increase the life time of a sensor network using life time based queries. In this type of queries, sampling rates are adjusted to achieve the desired life for a node. TinyDB supports the continuous data acquisition on a node, it supports a query based single inquiry and it also supports the event based sampling using event based queries. TinyDB is the only middleware that supports different data acquisition techniques and sampling rates and it clearly stands ahead of other middlewares in the field of data sampling.

Event based sampling is the natural selection for WSN based applications because this method controls the data acquisition and data propagation, thus increasing the efficiency in power usage. Mires, TinyLime, TeenyLime and Impala, all support event based sampling that qualify these

middlewares as suitable choice for wireless sensor networks applications. SINA samples on inquiry and limits the number of nodes used in sampling to minimize the communication.

Data filtering and congestion control is used to avoid excessive packet propagation and packet loss within the network respectively. Aggregation is also used for similar purpose which is commonly implied by majority of the middlewares to restrict amount of data propagation in the network. Mate´ and Cougar transmit the data in raw form without any filtering that can overload the network and thus these middlewares are not good choice for data collection operations. Agilla, TinyLime and TeenyLime filters the data collection using reactions to control traffic over network whereas Impala, SINA and TinyDB uses more efficient methods such as route adaptation along with adjustable sampling rates to control data loss and network congestion.

Aggregation is a data manipulation tool that is used for multiple purposes in wireless sensor network and consider as an important technique to reduce data over the network. It refines the raw data using simple aggregation functions like sum, averages, counts etc. and sends the result over the network to other node and sinks. The aggregation can be done locally on a node just using the locally collected raw data and it can also be done globally on a data collected on multiple nodes. Global aggregation is done on a fly while transferring data from one node to the other towards the sink or base station. Such aggregation is also known as in-network aggregation which saves energy consumption by distributing computation among the nodes and by limiting amount of data sent over the network. In-network aggregation is naturally a preferred method and implied by SINA, TinyDB and TinyLime. Mate´, Agilla and Impala lack support for aggregating data. TinyDB supports both in-network and local aggregation and supports most of the simple aggregation functions.

7.5 Network management

Being a distributed system, network management is vital to achieve desired performance levels for the wireless sensor networks. Middlewares for wireless sensor networks are not primarily aimed to provide the ad hoc protocol and network configuration solutions rather they can be used to provide support for the network management to add flexibility and adaptability to the networks.

	Network organization & re-configuration	Communication & routing methods	Network information management
TinyDB	-Ad hoc deployment. -Network reorganization through maintaining SRT. -Probing messages to check child presence. If missing, reorganization the network below the node. -No support for resource discovery.	-It uses controlled flooding for sending queries. -Uses snooping during aggregation to reduce communication. -SRT is created for routing data to the base station. -SRT is used to recreate routes.	-Logical table 'sensors' contains information about sensors and other parameters. -SRT is used to manage routing.
Agilla	-Uses geographical location based localization after deployment.	-Agilla uses geographic routing and controlled migration of agents for packet transfer. -Also uses unicast on remote tuple spaces.	-Keeps the list of nodes and agents in a tuple space.
Mires	-Uses clustering with cluster head for network organization.	-Uses broadcast for advertising. -Decentralized request response messaging between nodes and sinks.	-Pub/Sub service manages the communication. -Routing is managed by central routing service.
Mate`	-Support ad hoc routing and network reconfiguration.	-Save and forward flooding techniques for updates. -Snooping to gather neighbor's information. -Uses BLESS routing protocol.	-No central information management. -Each node has version manager.
TinyLime	-Supports for ad hoc networking. -Resource discovery using command rdg(p) for group read operation.	-Only communicates with single hop neighbor. -Packet transfer between sink and a	-N/A-

		node.	
MiLAN	-Network level service discovery. -Uses node clustering.	-Multi hop routing. -Switches between protocols to achieve QoS. -Adapts to different routes to achieve QoS.	-Accesses network protocol stack to gather information for management.
Cougar	-Creates clusters with group leaders during localization. -No resource discovery.	-Uses ad hoc on demand reactive routing protocol (AODV). -Multi hop routing -Packet intercept for in-network aggregation. -Routing based on query plans.	-Catalog keeping sensor and environment parameter in sensors database.
Impala	-Ad hoc and mobile network organization. -Supports node discovery and self re-organization.	-Dynamic re-routing. -Uses unicast during updates. -Parameter based adaptive routing.	-N/A-
SINA	-No ad hoc networks. -Localization starts with pre-existing parameters and address using coordinates. -Supports beacon and beacon-less localization.	-Multi hop broadcast for SQTL messages to the nodes. -Uses diffused computation for data routing to base station.	-Spreadsheet for record keeping on nodes. -Each cell in spreadsheet represents node's attributes.
TeenyLime	-Ad hoc deployment. -Self localization. -Node discovery.	-Multi-hop communication but single hop operations. -Remote function invocation using capability tuples.	-Keeping track of neighbors using neighborTuplespace.

Table 11: Comparison of middlewares based on network management.

One of the requirements for WSN is that it should have the ability to be deployed anywhere and at anytime and it should be able to maintain itself without any external support. All middlewares for WSN are more or less able to provide support for ad hoc deployment and ad hoc routing

protocols. TinyDB uses SRT for creating and maintaining routes from nodes to the base station. SRT is also used to re-organize network topology when the parent node changes. Agilla supports both self deployment and network reorganization. It uses GPS-based as well as beacon-less geographical localization techniques for self deployment of a network. Agilla also controls the network traffic through controlled routing with the help of agents. Whereas Mate´, TinyDB, Mires and SINA use broadcast and flooding for downward traffic from sink towards the nodes that can overload the network. Impala uses self deployment and ad hoc routing protocols for network organization. It supports nodes discovery that is helpful in finding new nodes and reorganizing networks without any breaks. All of these features are suitable for WSN network management and makes Impala superior to other middlewares. TinyLime and TeenyLime also support self deployment and resource discovery whereas Cougar just supports the self deployment to add some support to the network organization in wireless sensor networks.

Clustering is another strategy to limit the amount of data propagation over the network to save congestion and minimize communication. In clustering, a group of nodes closer to each other in network form a sub-network and perform all the computation and communication within a cluster to minimize the power usage of the overall network. A cluster communicates with rest of the network using cluster head thus saving the energy of rest of the nodes. Cluster head can be a normal node or a sink node with extra resources. TinyLime, Cougar, TeenyLime and Mires create clusters during network organization.

Last column in Table 11 shows how network information is managed by each middleware. Network wide information is needed by the middleware to keep status of the network updated and to perform reliable operations. Details about the nodes and their parameters are stored in the network to assist network related operations performed by the middleware. The management data is either kept on sink nodes or it can be distributed all over the sensor nodes. It can also be sent to the base station for permanent record keeping because nodes have very limited storage due their small size. This data is used for network organization, route management, resource discovery and all the network related operations.

TinyDB keeps the network information in a logical table called ‘sensors’. The sensor table contains the list of all the nodes in a network and the attributes of each sensor. Each row in the sensor table represents one sensor and the attributes are stored in respective column. It also contains the data collected on each sensor. Records in this table are stored for short period of time and it gets updated with newly collected data. Physically, the sensor table is spread across all the

sensor nodes where each node stores its own parameters and data. All the legacy data is stored on the base station where it is used by the optimizer for optimizing request paths.

Agilla also maintains the list of neighboring nodes and the agents residing on these nodes. This information is stored centrally in a tuplespace. Similar to TinyDB each sensor node stores locally its own data but logically the management data is presented as one federated tuplespace. Mires does not contain any central management repository but it is centrally managed by the publish/subscribe service to coordinate all the messaging from root to node and back again to the root.

There is lack of information whether data keeping and central management in Mate´, TinyLime and Impala is available or not. Mate´ keeps track of versions on local nodes. It records the number of capsules of old version and the new version and the currently installed version.

Cougar keeps the management data in the form of sensor data and stored data. Stored data contain all the information about sensor and their related parameters. All the actual data collected by sensors is stored in sensor data. SINA uses spreadsheets to keep track of all the data. Spreadsheets are stored locally on all the nodes and each cell in a spreadsheet represents the node's attribute. These spreadsheets are shared by the nodes and these are logically presented as one big central datasheet. TeenyLime also keeps track of the single hop neighbors by keeping this information in a special tuplespace called Neighbor tuplespace.

The study in this section shows that DB-oriented middlewares have better central information management due to their data oriented nature as compared to other middleware models. It also shows that the use of tuplespaces also facilitate in organizing central management information which helps in effective management of the networks.

7.6 The evaluation summary

This section shows the simplified reference context based on the above findings. To specify these values in Yes (Y) and No (N) is not a straight forward task. There are many cases where middlewares or middleware models can provide multiple solutions or the middlewares can adjust according to the situations. A middleware can be scalable or Power aware in one situation and else in other scenarios. So the most common scenarios and results are assumed for the table below. In some cases there is no conclusive information available to judge whether a middleware

provides certain functionality or not. In such cases hyphens (-) are used to indicate unavailability or lack of conclusive information.

	TinyDB	Agilla	Mires	Mate	Tiny Lime	MiLan	Cougar	Impala	SINA	Teeny Lime
Power	Y	Y	Y	N	-	N	Y	Y	Y	Y
User friendliness	Y	N	N	N	Y	N	Y	N	Y	Y
Node level access	N	Y	N	Y	-	Y	N	Y	N	Y
Mobility	N	N	N	Y	Y	N	N	Y	N	Y
Accuracy	N	Y	N	N	Y	N	Y	Y	Y	Y
Heterogeneity	N	N	Y	Y	Y	N	N	N	N	Y
Security	N	N	N	N	N	N	N	N	N	N
Runtime support	N	Y	Y	Y	N	N	N	Y	Y	N
Scalable	N	Y	Y	Y	Y	N	N	Y	N	Y
Adaptable	Y	Y	Y	N	N	Y	N	Y	Y	-
Filtering	Y	Y	Y	N	Y	Y	N	N	Y	Y
Congestion control	Y	N	N	N	-	Y	N	Y	Y	Y
Distributed. storage	Y	Y	N	Y	Y	Y	Y	Y	Y	Y
Centralized storage	Y	N	Y	N	N	N	Y	Y	N	N
Event-based sampling	Y	N	Y	N	Y	N	N	Y	N	Y
In-network aggregation	Y	N	Y	N	Y	N	Y	N	Y	N
Network management	Y	Y	N	Y	N	N	Y	-	Y	Y
Self organization	Y	Y	Y	Y	Y	N	Y	Y	Y	Y
Resource discovery	N	Y	N	N	Y	Y	N	Y	N	Y

Table 12: Reference matrix for WSN middlewares.

7.7 The graphical framework

The graphical presentation of the reference framework shown in Figure 5 is based on the component model [Costa et al., 2006b]. The core of the model contains middleware components and network components. Network components are mainly handled by data link layer and network layer. The lowest layer comprise of sensor hardware that contains sensing, computation and power module. The middleware layer resides on top of the operating system. The middleware core contains the level and type of abstraction in use. It also contains network management data and network parameters. Additional functionality can be added to the middlewares by adding new components to the framework shown in Figure 5. Every component framework represents functionality such as scalability, adaptability, usability and power awareness. Each component framework consists of components that provide similar functionality but they can differ in usability or implementation details. Middlewares mentioned in the each framework show that the functionality is currently provided by the specified middlewares. This complete model is presented, based on the evaluation done in this thesis.

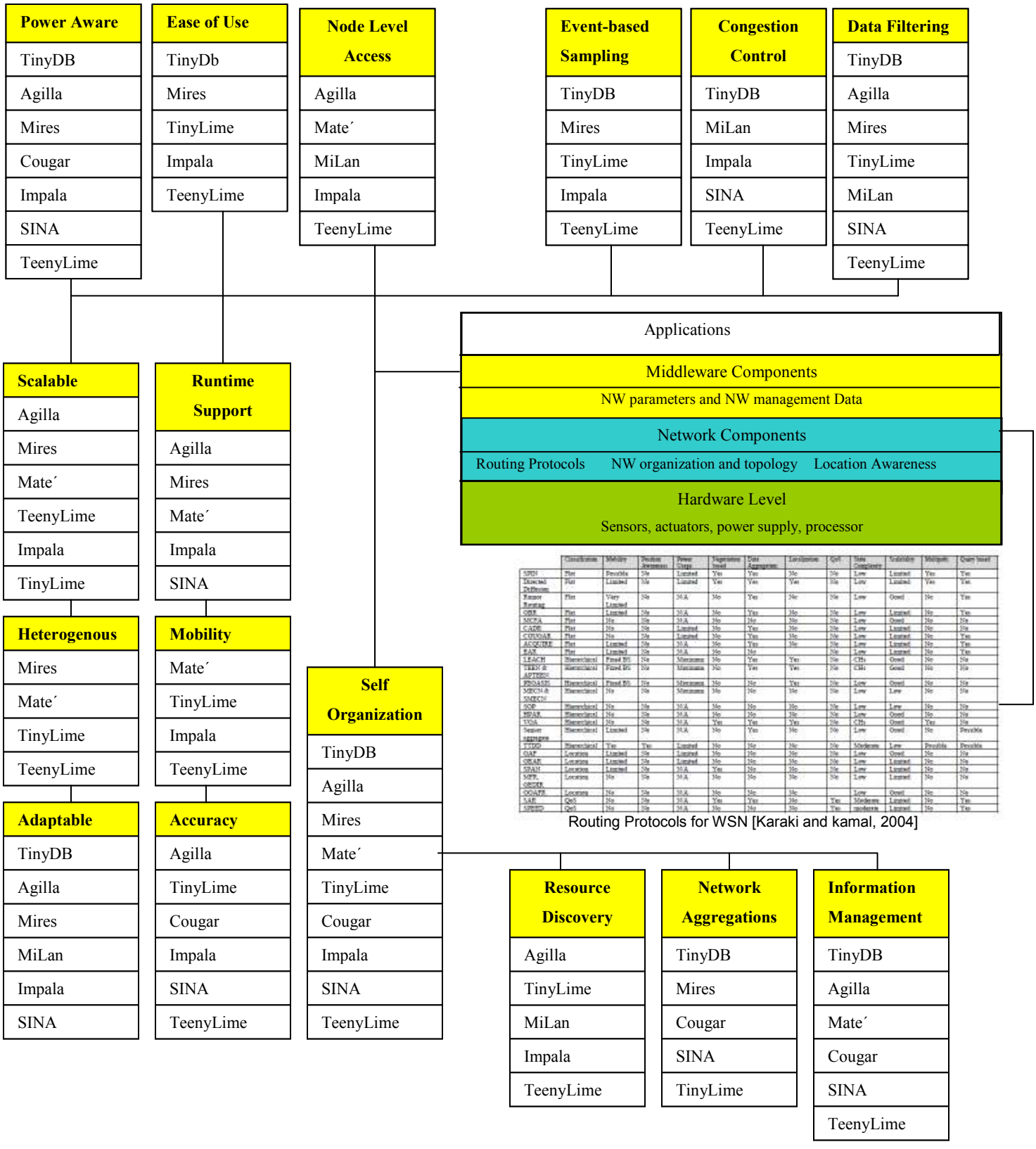

Figure 5: Graphical representation of reference context.

8 Conclusion

This thesis was targeted to identify the strengths and weaknesses of all the available middleware models and the existing middlewares for wireless sensor networks. It also tried to identify the limitations in each of the model and points to the areas which need to be addressed in the future and to find the alternate methods of development. The aim is to benefit developers, designers and project manager working on WSN based applications. The designers and developers can easily evaluate that which model is easy to program and which one provides more flexibility than others. It can help managers in selecting the middleware for their projects and products easily. The reference matrix can help in finding the pros and cons for the available middlewares for quick evaluations and decision making.

This thesis discussed, evaluated and compared various middleware architectures and models such as TinyDB, Cougar and SINA from DB-typed middlewares, Agilla and Impala from agent based middlewares, Mate as a virtual machine for WSN, Mires from messaging based models, TeenyLime and TinyLime from tuplespace models. These middlewares are necessary for running and supporting distributed systems like Wireless sensor Networks. This thesis also presented the design challenges that are faced by the middleware models. These middlewares were then evaluated and compared based on these challenges.

Various middleware solutions have been surveyed and evaluated in this thesis, keeping in view the issues like power awareness, usability, data fusion and aggregation, information dissemination, network and data management and repositories, resource constraints, scalability, reliability, mobility and other QoS related issues. Similar surveys for WSN middlewares were carried out by Wang et al. [2008] and Hadim and Mohamed [2006]. The former lacked the in depth evaluation details and the later only compared the QoS parameters addressed by different middleware models, whereas we evaluated all the available models and compared these models with each other to discover the best available solutions for the problems and targeted the specific areas that existing middlewares lack in standardization and re-usability. Yoneki and Bacon [2005] also thoroughly surveyed various available middlewares in detail but mainly focused on network organization, topology and protocols whereas this thesis focused less on network issues and more on other issues like usability, abstraction and power awareness along with other design challenges but gave very limited overview of network side issues. In particular, this thesis emphasized the design challenges and constraints arising from the distributed and resource limited nature of the

wireless sensor networks. It also presented the details of the most common middleware models working in wireless sensor networks and provides comparison of these middleware models working in sensor networks. The middleware solutions are surveyed and compared keeping in view the challenges caused by the constraints and complex nature of wireless sensor networks.

The evaluation done in this thesis demonstrated that none of the middlewares under discussion addresses complete set of issues and design challenges effectively. Each middleware addresses only a limited subset of issues. Each middleware or a group of middlewares target specific set of issues based on the middleware's own design and the type of programming abstraction they provide. The database oriented middlewares are more data centric that help them in addressing the data management issues effectively, where as modular middlewares focus on Quality of Service and flexibility of the network and applications. Monolithic middlewares such as DB-type middleware are more power friendly and agent based middlewares including Agilla and TeenyLime are generic in nature that can address more issues due their flexible and hybrid nature but less effective than others in addressing particular issues individually.

The DB-based solutions provide simple database type abstractions and user friendly interfaces but mainly they provide primitive data fusion and aggregation function. More advance techniques are needed for sophisticated applications. Publish/Subscribe middlewares provide strong decoupling between the senders and receivers that are more suitable for distributed applications but this approach is not suitable for the applications where data is continuously gathered and sent to the end user or the base station because it increases the communication overhead due to extra messaging that drains the energy out of resource scarce devices. Although virtual machines have a reduced memory footprint yet their execution has high impact on power consumption and battery life. In my opinion VM based middlewares have very limited functionality and they can be best used as part of other middleware models instead of using it separately as a whole middleware. Although monolithic programmed middlewares are more energy efficient and compact, yet the agent based middlewares add the flexibility and they are even superior when smaller fraction of nodes are touched during upgrades. Agent based middlewares are more expensive for typical data gathering because of the messaging overhead. They consume more energy due to high processing usage and higher data is transmitted as it passed from node to node using agents.

Design of wireless sensor networks is highly influenced by resource scarcity and the middleware should be facilitating the development and maintenance of sensing based applications. Most of

the existing middlewares for sensor networks have very limited scope of applications. They are targeting the specific area thus existing middleware models lack the standards and reusability. They differ in terms of their models for querying a data aggregation and their assumptions about the topology and other characteristics of the network. These solutions are inspired by middlewares for more traditional distributed systems.

This thesis also showed that there is lack of standardization in current approaches and need of a collective effort in designing and developing standardized middleware that can address wide range of applications. Appropriate middleware design approach is needed to provide standardization, system abstraction and the capability to support and coordinate concurrent applications. All the existing middleware solutions are aimed to provide complete one-off solutions for the targeted domain instead of standard approach towards re-usability. Future studies should be aimed to address and solve the design challenges and to find generalized solutions to add services to the existing models. A component based approach can be more effective due to component's reusability and focus of efforts on the issues rather than on the targeted applications and domains. Currently the proposal for component based model is very generic with no concrete design and architecture and not yet been implemented. In my opinion more efforts on this model can lead towards the standardization and re-usability of components.

From the study in this thesis, we conclude that almost all the existing models for WSN middlewares are derived from the concepts of existing models for conventional distributed system's middleware. There is a need of more innovative approaches to address the issues purely related to resource scarce nature of WSN. This thesis evaluated only few middlewares and more middlewares can be included in the future for comprehensive study and evaluation. We mainly emphasized on usability, power awareness and data related limitations. Further such evaluations can be done for the use of network protocols and network topologies and their use in the middlewares. More effort is needed towards unified, standard goals and issues like security, heterogeneity and mobility of nodes need more attention. The current middlewares severely lack these features.

References

[Akyildiz et al., 2002] I. F. Akyildiz , W. Su , Y. Sankarasubramaniam and E. Cayirci. “Wireless sensor networks: a survey”, Computer Networks: The International Journal of Computer and Telecommunications Networking, volume 38, pages.393-422, March 2002.

[Bakken, 2001] D. E. Bakken. Middleware, Kluwer Academic Press, 2001.
online: http://www.eecs.wsu.edu/~bakken/middleware.htm (last accessed on 10.08.09).

[Battelli and Basagni, 2007] M. Battelli and S. Basagni. “Localization for wireless sensor networks: Protocols and perspectives”, Canadian Conference on Electrical and Computer Engineering, CCECE 2007, pages 1074 - 1077, April 2007.

[Beckwith et al., 2004] R. Beckwith, D. Teibel and P. Bowen. “Pervasive Computing and Proactive Agriculture”, in Adjunct Proc. PERVASIVE 2004, Vienna, Austria, April 2004.

[Bharathidasas and Anand, 2002] A. Bharathidasas and V. Anand. “Sensor networks: An overview”, Technical report, Dept. of Computer Science, University of California at Davis, 2002.

[Bokareva et al., 2006] T. Bokareva, W. Hu, S. Kanhere, B. Ristic, N. Gordon, T. Bessell, M. Rutten and S. Jha. “Wireless Sensor Networks for Battlefield Surveillance”, In Proc. of the Land Warfare Conference (LWC), 2006.

[Bonnet et al., 2001] P. Bonnet, J. E. Gehrke and P. Seshadri. “Towards Sensor Database Systems,” in Proc. Second International Conference on Mobile Data Management, Hong Kong, pages 3 - 14, January 2001.

[Buttler et al., 2004] Z. Butler, P. Corke, R. Peterson, and D. Rus. “Networked Cows: Virtual Fences for Controlling Cows”, in proc. Workshop on Application of Mobile Embedded Systems (WAMES) 2004, Boston, USA, June 2004.

[Cepra and Estin, 2002] A. Cerpa and D. Estrin. “ASCENT: Adaptive Self-Configuring Sensor Networks Topologies”, In Proc. IEEE Infocom, volume 3, pages 1278-1287. IEEE, June 2002.

[Chang et al., 2008] B. Chang, B. J. Huang and Y. H. Liang. "Wireless Sensor Network-Based Adaptive Vehicle Navigation in Multihop-Relay WiMAX Networks", In Proc. 22nd International Conference on Advanced Information Networking and Applications, pages 56-63, 2008.

[Cook and Das, 2005] D. J. Cook and S. K. Das. "Smart Environments: Technologies, Protocols, and Applications", John Wiley and sons, Inc, 2005, pages 101-114 and pages 13-24.

[Conner et al., 2004] W. S. Conner, J. Heidemann, L. Krishnamurthy, X. Wang and M. Yarvis. "Workplace applications of sensor networks", Technical report, 2004.

[Contiki-online, 2009] http://www.sics.se/contiki/about-contiki.html (last accessed on 02-05-2009).

[Corba-online, 2009] http://www.corba.org/ (last accessed on 28-12-2009).

[Costa et al., 2006a] P. Costa, L. Mottola, A. Murphy and G. Picco. "TeenyLime: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks", In Proc. Workshop on Middleware for sensor networks, pages 43-48, November 2006.

[Costa et al., 2006b] P. Costa, G. Coulson, C. Mascolo, G. P. Picco, S. Zachariadis. "The RUNES Middleware: A Reconfigurable Component-based Approach to Networked Embedded Systems", In Proc. PIMRC, pages 574-577, October 2006.

[Costa et al., 2009] P. Costa, L. Mottola, A. L. Murphy and G. P. Picco. Chapter 11 of Middleware for Network Eccentric and Mobile applications, Springer Berlin Heidelberg, pages 245-264; February 2009.

[Culler, 2006] D. Culler. "TinyOS: Operating System Design for Wireless Sensor Networks", Sensors magazine online, May 2006, online: http://www.sensorsmag.com/sensors/article/articleDetail.jsp?id=324975&pageID=1&sk=&date= (last accessed 10.09.2009).

[Culler, 2009] D. Culler. "Creating an Architecture for Wireless Sensor Networks", online: http://www.citris-uc.org/research/projects/creating_an_architecture_for_wireless_sensor_networks (last accessed on 10.09.2009).

[Curino et al., 2005a] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy and G. P. Picco. "TinyLIME: Bridging Mobile and Sensor Networks through Middleware". In Proc. PerCom, pages 61-72, March 2005.

[Curino et al., 2005b]C. Curino, M. Giani, M. Giorgetta, A. Giusti, A. L. Murphy and G. P. Picco. "Mobile data collection in sensor networks: The TINYLIME Middleware". Elsevier Pervasive and Mobile Computing Journal, volume 4, number 1, pages 446–469, December 2005.

[DCOM-online, 2009] online: http://msdn.microsoft.com/en-us/library/ms809340.aspx (last accessed on 28-12-2009).

[Ee et al., 2006] C.T. Ee, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker and I. Stoica. "A modular network layer for sensornets", In Proc. OSDI, pages. 249-262, November 2006.

[Fok et al., 2006] C. L. Fok, G. Roman and C. Lu. "Agilla: A Mobile Agent Middleware for Sensor Networks", Technical Report, WUSCE-2006-16, Washington University in St. Louis, March 2006.

[FPX–Middleware, 2009] online: http://www.fpx.de/fp/Uni/Diplom/node7.html, (last accessed on 05.09.09).

[FreeRTOS-online, 2009] online: http://www.freertos.org/ (last accessed on 10.08.08).

[Hadim and Mohamed, 2006] S. Hadim and N. Mohamed. "Middleware Challenges and Appraoches for Wireless Sensor Networks", IEEE Distributed System Online, volume 7, number 3, page 1, March 2006.

[He et al., 2006] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui and B. Krogh. "VigilNet: An integrated sensor network system for energy-efficient surveillance", ACM Transactions on Sensor Networks (TOSN), volume 2, number 1, pages 1-38, February 2006.

[Hill, 2003] J. L. Hill. "System Architecture for Wireless Sensor Networks", PhD thesis, University of California, Berkley, Spring 2003.

[Howard et al., 2002] A. Howard, M. J. Matarić and G. S. Sukhatme. "An Incremental Self-Deployment Algorithm for Mobile Sensor Networks", Autonomous Robots, volume 13, number 2, pages 113-126, September 2002.

[Hull et al., 2006] B. Hull, V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan and S. Madden. "CarTel: A Distributed Mobile Sensor Computing System". In Proc. ACM SenSys, pages 125–138, October 2006.

[RMI-online, 2009] online: http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp (last accessed on 28-12-2009).

[Juang et al., 2002] 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 Proc. ASPLOS X, San Jose, USA, pages 96-107, October 2002.

[Kahn et al., 1999] J. M. Kahn , R. H. Katz and K. S. J. Pister. "Next century challenges: mobile networking for "Smart Dust", In Proc. 5th annual ACM/IEEE international conference on Mobile computing and networking, pages 271-278, August 15-19, 1999.

[Karaki and Kamal, 2004] J. N. Karaki and A. E. Kamal. "Routing techniques in wireless sensor networks: a survey", IEEE Wireless Communications, Volume 11, number 6, pages 6-28, December 2004.

[Klues et al., 2007a] K. Klues, G. Xing and C. Lu. "Towards a unified radio power management architecture for wireless sensor networks", International Workshop on Wireless Sensor Network Architecture (WWSNA), April 2007.

[Klues et al., 2007b] K. Klues, G. Hackmann, O. Chipara and C. Lu. “A component-based architecture for power-efficient media access control in wireless sensor networks”, In Proc. 5th international conference on Embedded networked sensor systems, pages 59-72, November 2007.

[Levis and Culler, 2002] P. Levis and D. Culler. “Mate: A tiny virtual machine for sensor networks”, In Proc. 10th Int. Conf. Architectural Support for Programming Languages and Operating Systems (ASPLOS-X), San Jose, USA, ACM Press, pages 85-95, 2002.

[Levis, 2002] P. Levis. “Bombilla: A Tiny Virtual Machine for TinyOS”, Bombilla Documentation, 2002.

[Liu and Margaret, 2003] T. Liu and M. Margaret. “Impala: a middleware system for managing autonomic, parallel sensor systems”, In Proc. ACM SIGPLAN, pages 107-118, June 2003.

[Liu et al., 2004] T. Liu, C. M. Sadler, P. Zhang and M. Martonosi. “Implementing software on resource-constrained mobile sensors: experiences with Impala and ZebraNet”, In Proc. MobiSys, pages 256-269, June 2004.

[Madden et al., 2002] S. Madden, R. Szewczyk, M. J. Franklin and D. Culler. “Supporting aggregate queries over ad-hoc wireless sensor networks”, In Proc. IEEE Workshop on Mobile Computing Systems and Applications, page 49, June 2002.

[Madden et al., 2005] S. Madden, M. J. Franklin and J. M. Hellerstein. “TinyDB: An acquisitional query processing system for sensor networks“, ACM Trans. Database Systems, volume 30, number 1, pages 122-173, March 2005.

[Mainwaring et al., 2002] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk and J. Anderson. “Wireless sensor networks for habitat monitoring”, In Proc. 1st ACM international workshop on Wireless sensor networks and applications, pages 88-97, September 2002.

[Mantis-online, 2009] online: http://mantis.cs.colorado.edu/index.php/tiki-index.php (last accessed on 02-05-2009).

[Middleware-online, 2009] online: http://www.middleware.org/mom/basicmom.html (last accessed on 26-12-2009).

[Murphy et. al., 2001] A. L. Murphy, G. P. Picco and G. C. Roman. "Lime: A middleware for physical and logical mobility", In Proc. the 21st Int. Conf. on Distributed Computing Systems, Orland, USA, pages 524-533, May 2001.

[Murphy and Heinzelman, 2002] A. L. Murphy and W. Heinzelman. "MiLAN: Middleware Linking Applications and Networks", TR-795 University of Rochester, Computer Science, 2002.

[Object Web -Middleware, 2009] online:http://middleware.objectweb.org/ (last accessed on 10.08.09).

[Oi and Bleakley, 2006] H. Oi and C. Bleakley. "Towards a Low Power Virtual Machine for Wireless Sensor Network Motes", In Proc. FCST'06, pages 18-22, October 2006.

[Ou et al., 2005]J. P. Ou, H. W. Li, Y. Q. Xiao and Q. S. Li. "Health dynamic measurement of tall building using wireless sensor network", In Proc. SPIE, volume 5765, pages 205-216, Smart Structures and Materials 2005: Sensors and Smart Structures Technologies for Civil, Mechanical, and Aerospace Systems, May 2005.

[Pathan et al., 2006] A. S. K. Pathan, H. W. Lee and C. S. Hong. "Security in wireless sensor networks: issues and challenges", Advanced Communication Technology, 2006. ICACT 2006. In Proc. 8th International Conference, volume 2, pages 1043-1048, February 2006.

[Perkins, 1999] C. E. Perkins. "Ad hoc on demand distance vector (AODV) routing", Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-manet-aodv-04.txt, October 1999.

[Saeda et al., 2004] K. Seada , M. Zuniga , A. Helmy and B. Krishnamachari. "Energy-efficient forwarding strategies for geographic routing in lossy wireless sensor networks", In Proc. 2nd international conference on Embedded networked sensor systems, pages 108-121, November 2004.

[Simon et al., 2004] G. Simon, M. Maróti, Á. Lédeczi, G. Balogh, B. Kusy, A. Nádas, G. Pap, J. Sallai and K. Frampton. “Sensor network-based countersniper system”, In Proc. 2nd international conference on Embedded networked sensor systems, pages 1-12, November 2004.

[Souto et al., 2004] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa and C. Ferraz. “A message-oriented middleware for sensor networks”, In Proc. 2nd workshop on Middleware for pervasive and ad-hoc computing, pages 127-134, October 2004.

[Souto, E. et al, 2005] E. Souto, G. Guimarães, G. Vasconcelos, M. Vieira, N. Rosa, C. Ferraz and J. Kelner. “Mires: a publish/subscribe middleware for sensor networks”, ACM Proceeding for Personal and Ubiquitos computing, volume 10, number 1, pages 37–44, December 2005.

[Srisathapornphat et al., 2000] C. Srisathapornphat, C. Jaikeo and C. Shen. “Sensor Information Networking Architecture”, Workshop on Pervasive Computing , page 23, August 2000.

[Srivastava et al., 2001] M. Srivastava , R. Muntz and M. Potkonjak. “Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments”, In Proc. 7th annual international conference on Mobile computing and networking, pages 132-138, July 2001.

[Suzuki et al., 2007] M. Suzuki, S. Saruwatari, N. Kurata and H. Morikawa. “A high-density earthquake monitoring system using wireless sensor networks”, In Proc. 5th international conference on Embedded networked sensor systems, pages 373-374, November 2007.

[Szumel et al., 2005] L. Szumel, J. LeBrun and J. D. Owens. “Towards A Mobile Agent Framework For Sensor Networks”, In Proc. 2nd IEEE workshop on Embedded Networks Sensors, pages 79-87, May 2005.

[TinyOS – Hardware Design, 2008] Hardware Design, online: http://www.tinyos.net/scoop/special/hardware (last accessed on 10.08.09).

[Terfloth et al., 2006] K. Terfloth, G. Wittenberg and J. Schiller. “FACTS: A rule based middleware for WSN”, In COMSWARE 2006: First IEEE international conference on communication systems and middleware, New Delhi, India, January 2006.

[Wang et al., 2008] M. M. Wang, J. N. Cao, J. Li and S. K. Das. “Middleware for wireless sensor networks: A survey”, Journal of Computer Science and Technology, volume 23, number 3, pages 305-326, May 2008.

[Xu et al., 2004] N. Xu, S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan and D. Estrin. “A wireless sensor network for structural monitoring”, In Proc. 2nd international conference on Embedded networked sensor systems, pages 13-24, November 2004.

[Yao and Gehrke, 2002]Y. Yao and J. Gehrke. “The Cougar Approach to In-Network Query Processing in Sensor Networks”, SIGMOD Record, pages 9-18, September 2002.

[Yao and Gehrke, 2003] Y. Yao and J. Gehrke. “Query processing in sensor networks”, In Proc. 1st Biennial Conference on Innovative Data SystemsResearch, Asilomar, CA, pages 47-52, Jan 2003.

[Yap et al., 2005] K. K. Yap, V. Srinivasan and M. Motani. “MAX: human-centric search of the physical world”, In Proc. 3rd international conference on Embedded networked sensor systems, pages 166-179, November 2005.

[Yoneki and Bacon, 2005] E. Yoneki and J. Bacon. "A Survey of Wireless Sensor Network Technologies: Research Trends and Middleware's Role", Technical Report UCAM-CL-TR646, University of Cambridge, 2005.

[Zhao et al., 2002] J. Zhao, R. Govindau and D. Estrin. “Sensor Network Tomography: Monitoring Wireless Sensor Networks”, ACM SIGCOMM Computer Communication Review**,** volume 32, number 1, page 64, January 2002.