A NEW STACK ARCHITECTURE FOR SENSOR NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

MUAMMER EROĞLU

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

SEPTEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

_____
Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____
Prof. Dr. İsmet ERKMEN
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____
Prof. Dr. Semih BİLGEN
Supervisor

Examining Committee Members

Asst. Prof. Dr. Cüneyt BAZLAMAÇCI	(METU, EE)	_____
Prof. Dr. Semih BİLGEN	(METU, EE)	_____
Prof. Dr. Hasan GÜRAN	(METU, EE)	_____
Dr. Altan KOÇYİĞİT	(METU, II)	_____
Dr. Şenan Ece SCHMIDT	(METU, EE)	_____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name :

Signature :

ABSTRACT

A NEW STACK ARCHITECTURE FOR SENSOR NETWORKS

EROĞLU, Muammer
M.Sc., Department of Electrical and Electronics Engineering
Supervisor: Prof. Dr. Semih BİLGEN

September 2006, 80 pages

In this thesis, a new stack architecture for sensor networks is proposed. The stack consists of the following layers: application, query, aggregation, network, MAC and physical. Various algorithms are implemented using this stack and it is shown that this stack is modular.

Following an overview of sensor networks, the previous protocol stack suggestions for sensor networks are examined. Sensor network algorithms that can be classified as sensor data management systems are surveyed and compared with each other. Four of the surveyed algorithms, namely, TAG, Synopsis Diffusion, Tributary-Delta and Directed Diffusion are implemented using the introduced stack. The implementation is performed using a sensor network model developed with OMNeT++ simulator. The simulation results are compared to the

original results of these algorithms. Obtaining similar results, the stack and algorithm implementations are validated, moreover, it is shown that the stack does not induce any performance degradation.

Using the implementation details of the algorithms, the modularity of the suggested stack is demonstrated. Finally, additional benefits of the stack are discussed.

Keywords: Sensor networks, protocol stack

ÖZ

ALGILAYICI AĞLARI İÇİN YENİ BİR YIĞIT MİMARİSİ

EROĞLU, Muammer
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Danışmanı: Prof. Dr. Semih BİLGEN

Eylül 2006, 80 sayfa

Bu tezde algılayıcı ağları için yeni bir yığıt mimarisi önerilmiştir. Bu yığıt; uygulama, sorgu, birleştirme, ağ, bağlantı ve fiziksel katmanlar olmak üzere altı katmandan oluşmaktadır. Bu katman yapısı farklı senaryolarda denenmiş ve modüler olduğu gösterilmiştir.

Algılayıcı ağları hakkında verilen özet bilgiden sonra önceki yığıt önerileri incelenmiştir. Algılayıcı veri yönetimi sistemleri adı altında sınıflandırılabilecek algoritmalar ayrı ayrı incelenerek karşılaştırılmıştır. Sunulan yığıt mimarisini içeren bir algılayıcı ağı modeli geliştirilmiş ve bu yapıyla incelenen algoritmalardan dört tanesi uygulanmıştır. Benzetimlerden elde edilen sonuçlar ile uygulanan algoritmaların yayınlarındaki sonuçlar karşılaştırılmıştır. Sonuçların benzerliği sayesinde önerilen yığıt mimarisi ve algoritma

uygulamaları doğrulanmış, ayrıca bu yığıtın performans kaybına yol açmadığı gösterilmiştir.

Algoritmaların uygulama detayları kullanılarak bu tezde önerilen yığıt mimarisinin modülaritesi doğrulanmıştır. Son olarak önerilen yapının diğer getirileri tartışılmıştır.

Anahtar kelimeler: Algılayıcı ağları, protokol yığıtı

ACKNOWLEDGMENTS

I would like to thank Prof. Semih Bilgen for his innovative ideas and valuable supervision.

I wish to thank to my colleagues at ASELSAN Inc. for their continuous support.

I would also like to thank my family because of the encouragement they gave me to complete this thesis.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

IP	Internet Protocol
MAC	Medium Access Control
NIC	Network Interface Card
OMNeT++	Objective Modular Network Testbed in C++
OSI	Open Systems Interconnection
SD	Synopsis Diffusion
SRT	Semantic Routing Tree
TAG	Tiny Aggregation Service
TCP	Transport Control Protocol
TD	Tributary Delta
UDP	User Datagram Protocol

CHAPTER I

INTRODUCTION

Developments in micro sensor technology and low power analog and digital electronics have made possible to build distributed and wireless networks of sensor devices, called sensor networks. These sensor devices, also called sensor nodes, are capable of sensing a particular subject of interest, such as sound, wind, temperature, or a visual object. They are powered with their own batteries and they communicate through wireless medium. Sensor nodes have their own processor, memory and radio modules, and they are expected to be low cost, small sized and robust devices.

Sensor nodes are to be deployed densely and in large amounts, hundreds to thousands of nodes in one network. They will organize themselves in an ad-hoc fashion and will run as a distributed network. They will be able to operate unattended and in harsh environmental conditions. The data of the sensed subject is transferred from the nodes to a sink node, which is responsible to relay this data to a control center.

Some of the key application areas of sensor networks are the following subjects: environmental monitoring, military surveillance and intelligence [1], and patient health care. In these applications, the nodes may have to operate in harsh environmental conditions, such as rain, cold, or radiation. The nodes may not be recollected after the distribution and could be disposed after the mission is completed.

The biggest obstacle of the sensor networks is the limited battery energy of the nodes. Since the radio is the most power draining module of a node, using the radio less often lengthens the life of the nodes. With this idea in mind, instead of using long distance end-to-end routing, it is more energy efficient to use short-distance hop by hop routing while transferring data. Moreover, sensor nodes should keep their radios turned off when not used.

Another efficient method to conserve power is on-board processing. The power usage of a processor is very little compared to the radio [1]; therefore, processing the raw data inside the nodes and transmitting the processed result having fewer bytes decreases power consumption.

Since the incoming data of a node travels through other nodes on its path to sink, merging data in certain nodes proves useful to conserve energy by eliminating redundancies and minimizing the number of transmissions [20]. This is called data aggregation, where the sink requests the aggregate value of the sensed data, instead of the particular readings of the sensors. The aggregate requested by the sink could be the minimum or average value of the readings, or it might be the most frequent reading.

Sensor networks are used in various applications, and unification is very difficult. However, recently, it is proposed that they can be thought as distributed databases producing a continuous streaming data [2,3,4]. A typical operation of this system would be as follows: the sink distributes a query for the required information and the response is collected back from the network, either directly or by data aggregation inside the nodes. The algorithms working in this manner are called data-centric sensor network algorithms, and they are studied in detail in Section 2.4.

Four of these data centric algorithms are implemented in this thesis. They are Tiny Aggregation Service (TAG) [2], Directed Diffusion [4], Synopsis Diffusion [9] and Tributary-Delta [11].

TAG and Directed Diffusion are widely known algorithms which are also implemented in real life. TAG is an algorithm that queries a data where all of the nodes in the network can contribute. It uses a routing tree rooted at sink node

and constructed hierarchically as the query propagates through the network. The readings of the nodes are aggregated at every node and sent to sink using this routing tree backwards. Every node keeps the address of its parent. After receiving the results from the child nodes, the nodes aggregate these results with their own readings and send the new aggregate to their parents.

Directed Diffusion is an algorithm which requests data generated by only some of the nodes inside the network. It basically uses flooding to distribute a query to the network. Whenever the node having the required data is found, it is transmitted back to the sink using the reverse path of the query. Then, upon arrival of data from different sources, the sink node selects one of them and enforces it by sending a new query using the direct path to the node.

Synopsis Diffusion and Tributary-Delta are recent and advanced techniques that try to improve fault tolerance of tree based sensor networks algorithms, such as TAG. Synopsis Diffusion uses multi-path routing and duplicate-insensitive summaries, which are called synopses. Synopsis Diffusion uses rings topology, where the sink is at the center and nodes reside in one of the rings according to their distance to the sink in number of hops. When the synopsis generated in a node is broadcasted, all the nodes at the inner ring that can hear this message process it, realizing multi-path routing. The disadvantages of SD are; synopses have larger size compared to the TAG aggregates and they have a fixed approximation error.

Tributary-Delta combines Synopsis Diffusion and TAG by using them in different parts of the network. The central part around the sink is called delta region and Synopsis Diffusion is performed. The outer parts are called tributary region and TAG is performed. Since the central regions are more critical to errors, multi-path routing is used. In order to minimize transmission and lower approximation errors, TAG is used at the outer parts of the network, where a link error does not cut off a large tree branch. Moreover, Tributary-Delta algorithm adapts the tributary and delta regions according to the network contribution rate.

It is claimed that traditional TCP/IP protocol stack does not fit well to sensor networks because of their novel requirements [1, 8]. There have been

complete sensor network protocol stack suggestions in the literature by Akyıldız et.al. [1] and Kumar et.al. [8]. Akyıldız et.al. extend the TCP/IP stack by appending three vertical planes that can operate in every layer. They are responsible for power, mobility and task management functions. Kumar et.al. introduce a data fusion layer under the application layer, and also information exchange and helper service layers. The details of these stacks are given in Section 2.3.

The objective of this thesis study is to propose a modular protocol stack for sensor networks and implement various sensor network algorithms using this architecture. Modularity is defined as the property of system modules being cohesive inside and minimally coupled with each other. In this sense, the introduced stack will be a modular stack architecture for sensor networks. The stack will be used in the implementation of four different algorithms, namely, TAG, Synopsis Diffusion, Tributary Delta and Directed Diffusion by making simulations using the sensor network model developed with OMNeT++ simulator [16]. The results of the simulations will be compared with the original results of these algorithms. By the help of these comparisons, it will be shown that the proposed stack is applicable and does not induce any performance degradation to the sensor network algorithms. Using the implementation details, the modularity, reusability, implementation and maintenance advantages will be demonstrated. Moreover, it will be shown that this protocol stack has the potential to be a framework in order to classify and compare different algorithms. The proposed stack diagram is shown in Figure-1.

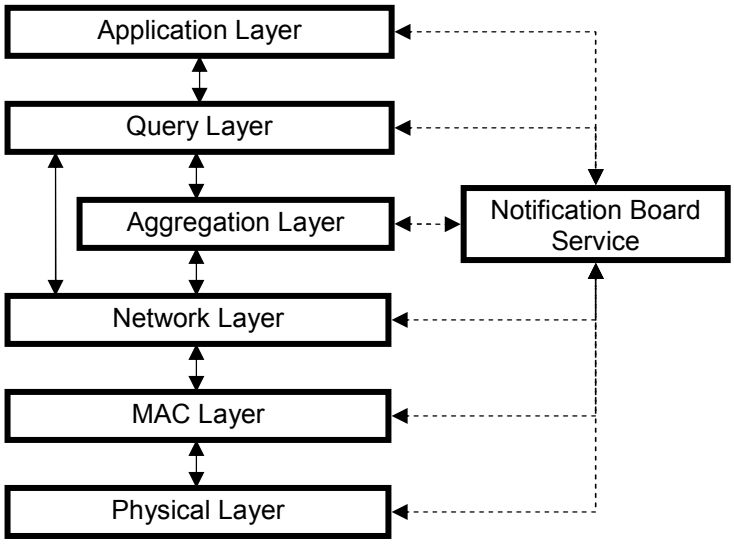

Figure-1 The sensor network stack proposed in this thesis

The main differences from the traditional TCP/IP stack are the introduction of the query layer, the aggregation layer, and the notification board service, replacing the transport layer. Query and aggregation layers can directly communicate with the network layer. All of the layers can communicate with the notification board through publish and subscribe methods, thereby realizing cross-layer communication. Similar to [8], transport layer is omitted from the stack, because the flow control of transport layer is not a generic requirement for all sensor network applications.

Yao & Gehrke [6] also suggested a query layer to be used under the application layer of TCP/IP. The proposed query layer in this thesis is similar to their suggestion, since both of them are responsible for all query optimization and distribution tasks. However, their work is based on TCP/IP and does not have special layers for data aggregation and cross layer communication.

Kumar et.al.'s SensorStack architecture [8] offers a data fusion layer placed under the application layer. This layer is responsible for in-network data fusion operations. While this idea was innovative for the aggregation layer in this thesis, SensorStack's data fusion layer only handles data aggregation tasks at

specific nodes, and when there is no data fusion requirement, this layer is not used at all (e.g., Directed Diffusion). This thesis suggests that the aggregation layer should be responsible for data aggregation, but it should also perform data collection and transmission tasks. Thus the entire query response process will be performed in this layer.

The network layer in this thesis will be able to filter packets based on the addresses of sensor nodes. A query distribution will be coordinated by the query layer communicating directly with the network layer, and data collection will be coordinated by the aggregation layer again communicating with the network layer. Releasing the network layer from the query distribution and data collection route control gives us the opportunity to add actual routing decisions using a routing table similar to TCP/IP. This table might be based on geographical information, or the kind of sensed data. Consequently, it is possible to implement query-informed routing, which is also proposed in SensorStack.

The cross layer communication will be performed by the notification board service, which is quite similar to information exchange service in SensorStack. Notification board service will serve all of the layers and other possible services using publish and subscribe functions for a particular data. Using this service, layers will be able to communicate directly, and also they will be able to receive information from other services, for example, the battery and radio conditions, or the location of a node.

The remainder of this thesis is organized as follows. The background of the thesis is established along with the literature review in Chapter 2. The suggested protocol stack is explained in detail in Chapter 3 along with the implementation details of the simulated algorithms and simulation results. Discussions, conclusions and future work are given in Chapter 4.

CHAPTER II

LITERATURE SURVEY

2.1 SENSOR NETWORKS

Sensor networks are composed of low-cost electronic devices called sensor nodes. These nodes are mostly battery powered and small sized devices and they have sensing, processing and communication capabilities. A sensor network has the purpose of collecting information about some observable phenomenon and relaying this information to a data center. Usually, the required information is difficult or costly to obtain by normal means. Using sensor nodes, it is possible to lower the cost and simplify the task of collecting information, since sensor nodes are cheap, robust, easily deployed and scalable. [1], [13], [14], [20], [21], [22]

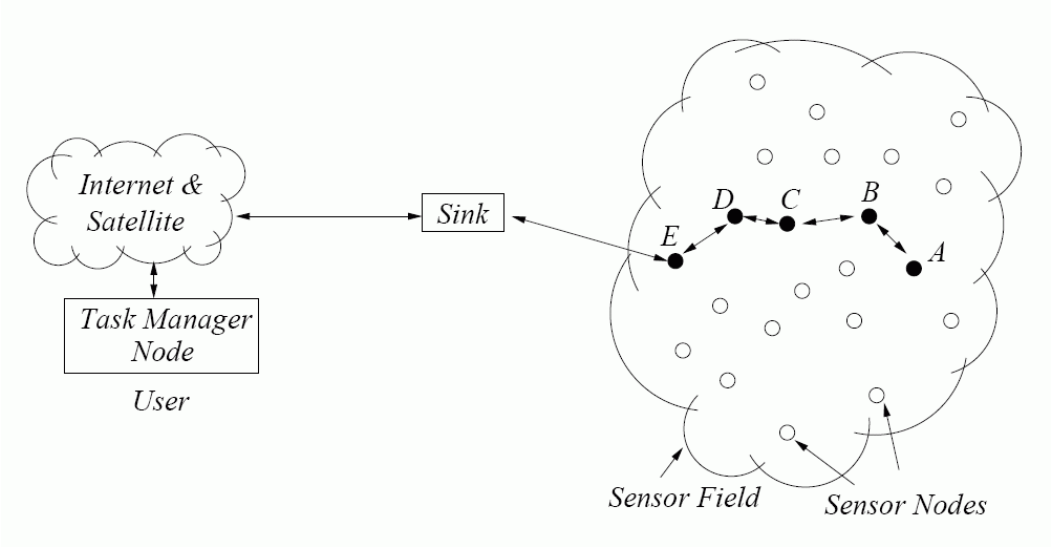

Figure-2 Sensor network diagram [1]

Sensor networks can be used in many different application areas, since the network and applications can be tuned to the purpose. Environmental monitoring, military applications and biomedical research are the outstanding subjects of today. In the future, it is envisioned that sensor networks will find far more application areas in our lives.

Sensor nodes have limited power, computation and communication capabilities since they should be low-cost and small-sized. Therefore, these limited capabilities should be optimized using efficient strategies.

A basic sensor node should have the following components; a sensing unit, a processing unit, a transceiver unit and a power unit. For a complex sensor node, additional components such as a location finding system, solar cells or a mobilizer can also be included. The block diagram of a sensor node is shown in Figure-3.

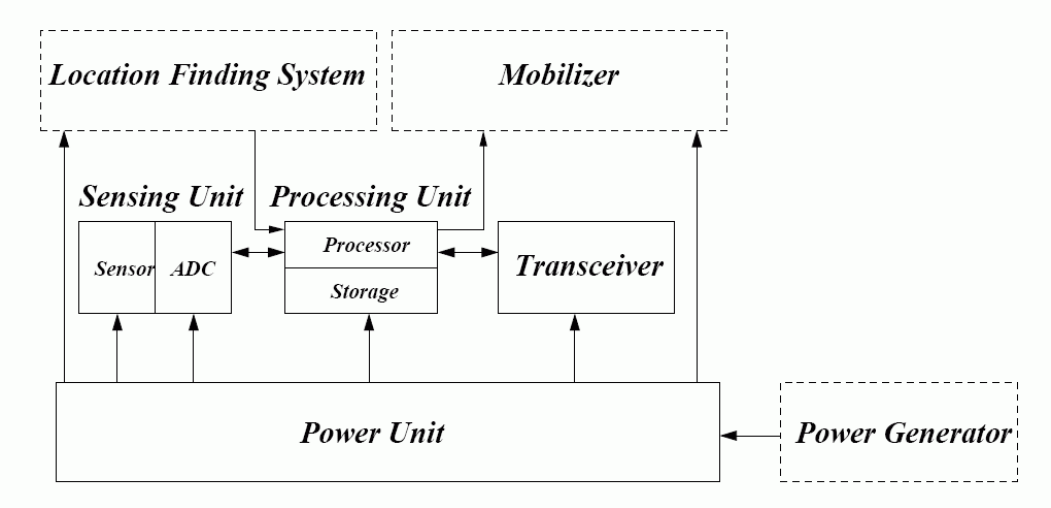

Figure-3 Block diagram of a sensor node [1]

The sensing unit is composed of an analog sensor and an analog to digital converter (ADC). This unit is used to sense the desired phenomena and deliver it to the processing unit. The processing unit is usually a processor with a small storage. The algorithms and procedures operating in different layers of the network are processed here. The transceiver unit handles the communication of the nodes using the wireless media. Most commonly an RF device is used, but it can also be a passive or active optical device. The power unit is mostly small sized batteries.

The battery of a sensor node is the critical factor for the life of an individual node and the overall network. There are many studies on sensor networks trying to minimize energy consumption in various phases of operation. The main energy consuming device in a sensor node is the antenna, and it uses up the highest energy while transmitting data. Therefore, it is crucial for the nodes to minimize data transmission for higher energy efficiency.

Computation in a sensor node requires much less energy than data communication. Transmitting a 1Kb of data may require the same energy as executing 3 million instructions on the processor [1]. Therefore, in order to

minimize power consumption, local data processing at the nodes should be used as much as possible instead of making more wireless transmissions.

Data aggregation replaces transmission with data processing by aggregating the data received from many nodes into one summary result, called aggregate, and transmitting this aggregate instead of all the data produced by the nodes. Since the required data from a sensor network is usually an aggregate value, such as min, max or average, performing data aggregation inside the network nodes increases energy efficiency of the network greatly.

A sensor network might have hundreds to thousand of nodes and they are deployed densely, possibly as high as 20 nodes / m^3 [1]. Moreover, the network is subject to frequent topology changes because of the following reasons:

- Nodes may deplete their energy or break off.
- Nodes might change their position using mobilizers or due to the environmental effects.
- Nodes might lose communication ability due to obstacles, noise or jamming.
- Additional nodes could be deployed to the original network.

All of these reasons show that topology maintenance also has a high importance in sensor networks. In order the network to be more reliable, adaptable routing mechanisms are required.

2.2 NETWORK STACK ARCHITECTURE

Layered network stack architecture has been the preferred method for computer networks since it helps to organize the system into logically distinct modules, such that the service of one layer is based on the service of the lower layer. This modularity helps different protocols to work together, easier

maintenance and transparent protocol changes. However, the traditional TCP/IP stack is not very suitable to sensor networks because of the following reasons [8]:

- The traditional TCP/IP network stack is designed for end-to-end networking requirements. However, most sensor networks work hop-by-hop routing.
- TCP/IP stack has services that are generally not required in sensor networks, such as flow control at the transport layer, node fairness at MAC layer and error control at both of these layers.
- Sensor Networks introduce new service requirements such as location awareness, time synchronization or node addressing.
- Sensor networks needs to optimize energy at every layer, since energy is a very critical factor for sensor nodes.

There has been some work in the literature proposing new sensor network stack architectures, or making modifications to the original stack. These are summarized in the following paragraphs.

The sensor network protocol stack proposed by Akyıldız et.al [1] is similar to TCP/IP stack, however, there are additional planes which are responsible for power management, mobility management and task management functions (Figure-4). These additional planes are able to operate in all of the layers, thus they are called planes.

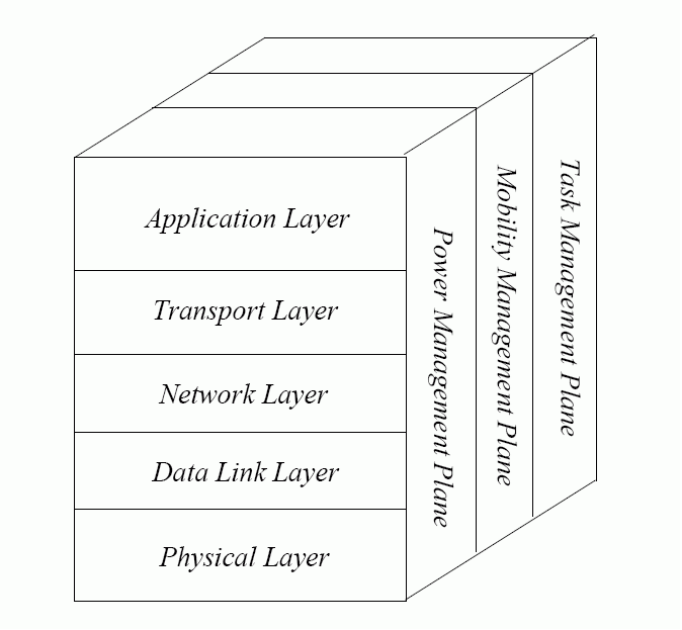

Figure-4 Sensor network protocol stack of Akyıldız et.al.[1]

The power management plane tries to increase power efficiency of a node. Since energy requirements are very strict for a sensor node, optimizing it at every layer would increase overall efficiency. Mobility management plane detects the movements of a node and keeps the location information of the node. Task management plane schedules and tries to balance tasks given to specific region.

Kumar et al. [8] introduces a new sensor network protocol stack, called SensorStack, which is designed considering the unique characteristics of sensor networks. Stack layout can be seen in Figure-5. The functions of these layers are explained in the following paragraphs.

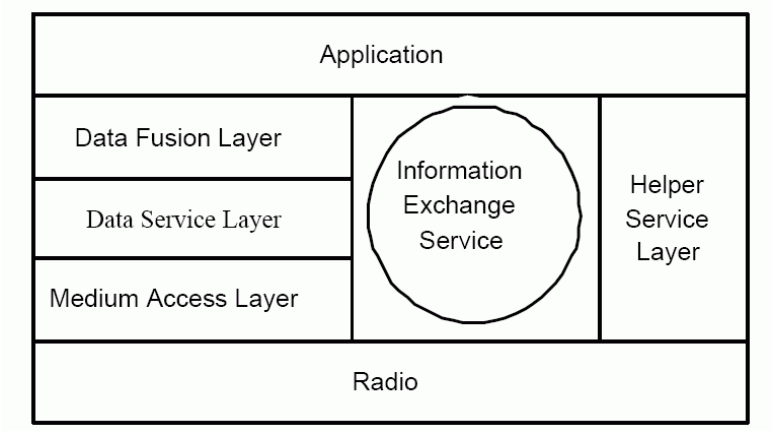

Figure-5 SensorStack layers [8]

Since radio is the mostly used physical medium for sensor networks, it is the physical layer of SensorStack. Medium access layer (MAC) is also similar to the TCP/IP such that it provides medium access for hop-to-hop data transfer and controls channel errors. MAC layer also controls the on-off schedule of antenna, thus, it can optimize energy efficiency by communicating with helper services and maintaining the node's wake-up/sleep schedule through the information exchange system (IES).

Data service layer provides dissemination of data to neighbors and reception of data packets, similar to network layer in TCP/IP. Moreover, this layer supports logical naming of the nodes instead of IP addresses. With logical naming, query or data packets will be filtered at this stage and unnecessary distribution will be blocked.

Data fusion layer is responsible for data aggregation inside the nodes. With this layer, data fusion becomes a part of the stack and packets will not have to reach up to application layer at every intermediate node. Moreover, providing the fusion logic as a separate module would make application development more modular, since same fusion code would be used for multiple applications. This layer can make fusion at every node or only at the selected nodes. The fusion

functions of this layer can be programmed by the application layer to be run in kernel space of the operating system.

IES service realizes communication of relevant information between different layers using a publish-subscribe functions. Using this service, cross layer optimization will be possible.

Application layer captures data and presents it at the sink node; programs fusion channels at the specified nodes and provides helper services for application specific demand.

Helper services layer is used to extend the stack by defining application specific services such as location awareness or time synchronization. These services can be executed either in user space or kernel space of the operating system.

SensorStack does not have a transport layer, but it is proposed that a middleware can be used for future applications that may need a transport service.

Yao and Gehrke [6] proposed the usage of a new query layer, which resides between application and network layers. This new layer provides cross-layer interaction between the routing protocol and distributed query protocol. The purpose of this layer is to abstract functionality of a large class of applications into a common interface of declarative queries.

There are also research projects that use modified versions of traditional stack. WINS [13] project discards transport layer, and uses other layers of TCP/IP stack. Smart Dust [14] project uses only the application, MAC and physical layers. TinyOS [15], which is the operating system for Berkeley motes, contains only the MAC and physical layer services.

2.3 SENSOR DATA MANAGEMENT SYSTEMS

Sensor networks generate a continuous information stream, and the applications are trying to retrieve data obtained using some part of this stream.

This resembles database systems, in which there is a huge data set and the required data is obtained using some part of this set.

Recently, it is proposed that database technology should be integrated with sensor network technology [2, 3, 4], so that the management of data produced by sensor networks would be improved. However, this integration is not simple, since traditional database technology deal with finite and static data, while sensor networks produce continuous and possibly infinite streams of data. Furthermore, traditional databases are stored in some persistent and always accessible repository, whereas the information in a sensor network is dispersed in all of the sensor nodes and changes in real-time. These factors show that new query languages, processing and optimization methods should be found for sensor network database systems.

Some recent publications define algorithms for sensor networks, and because of the methods they use, they can be classified as sensor data management systems [2,3,4,5,6,7,8]. These studies propose query languages to interact with the user; methods to request data from the sensor nodes and to collect the generated information from the nodes. Both of the querying and data collection phases involve routing and local data processing to be performed in sensor nodes. These systems usually define the algorithm as a whole and do not make exact distinctions of protocol layers.

The general attributes of sensor data management systems can be summarized as follows:

- The user of the system communicates with the system using the sink node, makes the query and retrieves the results through this node.
- The query is distributed throughout the network by routing the query hop-by-hop over the sensor nodes.
- Nodes receiving the query prepare and send the desired information to the sink node, again routing the data hop-by-hop over the nodes.

- If data aggregation is to be performed, all or some of the nodes along the routing path process and combine the results of previous and neighbor nodes; and transmit only the aggregated result.
- According to the query type, information flow might be continuous, time limited, periodic, or one-shot.

In the following subsections, the widely known algorithms in this area are explained briefly and compared according their query distribution methods, query types, data fusion and collection methods.

2.3.1 ALGORITHMS

Tiny Aggregation Service [2], abbreviated as TAG, is a generic aggregation service developed for TinyOS. TinyOS is the operating system for the sensor network nodes, also called motes, developed in Berkeley University. TAG is used to retrieve the aggregate value of the information generated by many nodes by making in-network aggregation at every node.

Aquisitional Query Processing System, also known as TinyDB, [3] is published by the same authors of TAG. It is based on the previous algorithm, but contains many enhancements in querying and data collection. TinyDB is also used to retrieve aggregates, but it can also make single node queries.

Directed Diffusion [4] is as a data-centric system for sensor networks. Data is generated and queried using attribute-value pairs. Directed Diffusion is used to retrieve information from one of the nodes which hold the information, and to select the best path to retrieve it.

Active Query Forwarding or Acquire [5] is also a data-centric querying mechanism, which is mostly suited for one shot, complex queries for replicated data.

Cougar [6] is a data management system which proposes a query processing layer between the application and network layers. All of the query

operations are done inside this layer. Cougar can make aggregate or single shot queries.

TinyDBD [7] is a system built by merging TinyDB with Directed Diffusion. The routing mechanism of Directed Diffusion and querying mechanism of TinyDB is used together. Therefore, TinyDBD shows the properties of both methods.

Synopsis Diffusion [9] and Approximate Aggregation [10] are two parallel studies, both of which are devised to overcome link errors in a tree-based sensor network topology. These solutions offer multi-path routing to reduce the errors introduced by node failures on the routing path. They define duplicate insensitive aggregation methods to get rid of duplicate information resulting from multi-path routing.

Tributary-Delta [11], builds on top of Synopsis Diffusion and proposes a new routing and aggregation method to decrease the message traffic in the overall network, and to achieve higher energy efficiency. This study suggests to use multi-path routing at the center of the network, and to use trees at the leaves of the network.

2.3.2 COMPARISON OF THE ALGORITHMS

2.3.2.1 QUERY DISTRIBUTION AND ROUTING

TAG algorithm uses a routing tree to distribute queries and collect results from the network. This routing tree can be set up separately from the query distribution, or they can be performed together. Routing tree construction is started by the sink node by broadcasting a message. Nodes hearing this message for the first time assign the sender node as their parent node, record their tree level and rebroadcast the message. Thus, the tree is constructed hierarchically. If this message is mixed with the query message, the tree would be refreshed for every query distribution. A sample routing tree is shown on Figure-6.

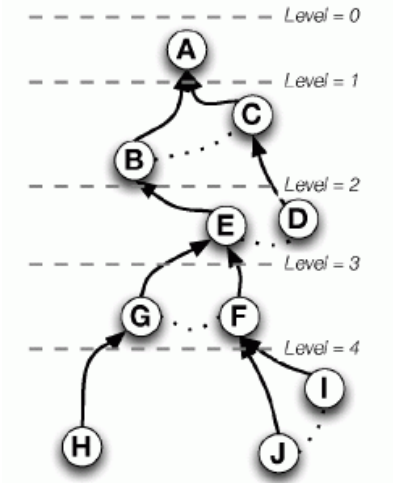

Figure-6 TAG routing tree [3]

TinyDB uses the same routing tree as TAG to distribute the queries. Moreover, TinyDB introduces Semantic Routing Trees (SRT) to achieve query informed routing. An SRT is a routing tree based on a specific sensor data. It is built similar to the actual routing tree; however, additionally every node keeps information about their children whether they have the required data or a path to it. Thus, unlike TAG, it would be possible to distribute a query submitted to the network only to the relevant nodes or paths to them.

Directed Diffusion constructs the routing topology for every new type of query, since the route is dynamically created while the query searches and finds a reply. In order to make a query, the sink node sends out an interest which contains the specifications of the requested data. (Figure-7a) This interest is called exploratory and it propagates in the network by flooding, constrained flooding or directional propagation. Nodes receiving this interest cache it along with the source node, called gradient of the interest. When a node has the requested data, it sends the data along the gradients up to the sink node. (Figure-7b) The sink node then reinforces one of the incoming replies; selection may be based on latency or consistency. Reinforcement is done by resending the same query along this selected neighbor with a higher data reception rate. The actual

routing path will be completed when the new query arrives to the node having the data. (Figure-7c)

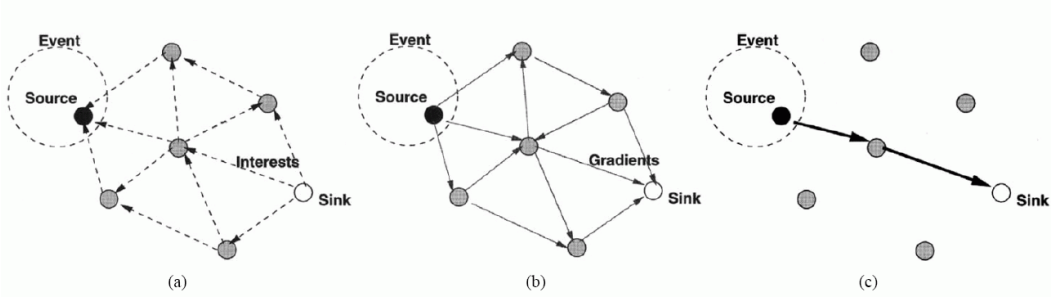

Figure-7 Directed Diffusion - (a) Interest propagation. (b) Initial gradients setup. (c) Data delivery along reinforced path. [4]

Similar to Directed Diffusion, Acquire also generates a routing path for each new query. It uses gossiping algorithm and flooding with a restricted radius to route the query. After the query packet is inserted into the network, it follows a random, predetermined or adaptive trajectory. When the query arrives at a node, that node obtains information from its neighbor nodes using flooding (Figure-8). The level of neighborhood is determined in the query by a number of hops, and it is called the look-ahead parameter. The node can also keep a cache of the neighbor nodes' values and it can use this cache until it becomes obsolete. If the query is solved at a particular node, the query becomes a completed response and it is returned back to the sink along the query path. If it is not solved, query continues its path.

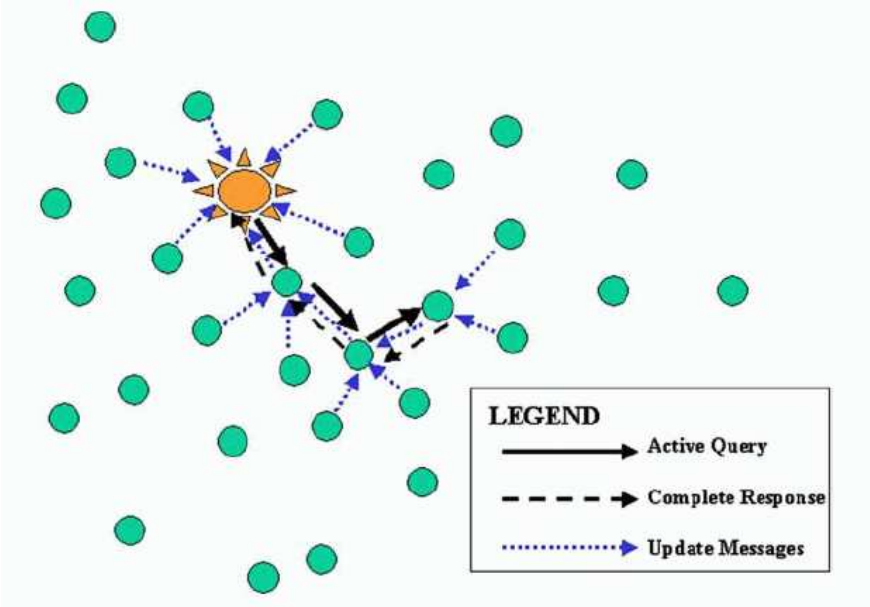

Figure-8 Acquire algorithm's routing diagram [5]

The size of the look-ahead hop number affects a trade-off between the information obtained and the cost of obtaining it. As the look-ahead hop number becomes larger, Acquire resembles flooding based queries. The downside of Acquire is that its average latency is much more than tree based or flooding based approaches.

Synopsis Diffusion and Approximate Aggregation techniques use a multi-path routing topology called rings. With this topology, every node will have multiple parents from the previous ring, and the wireless medium will be used optimally because nodes will process all of the messages they hear.

The rings topology is constructed while the query is distributed to the network. The sink node starts the query by broadcasting the query message. As seen in Figure-9, the sink node is in ring R0, and any node hearing the sink becomes in R1. The query is rebroadcast along the rings, and nodes hearing the query from ring i for the first time record their ring number as i+1.

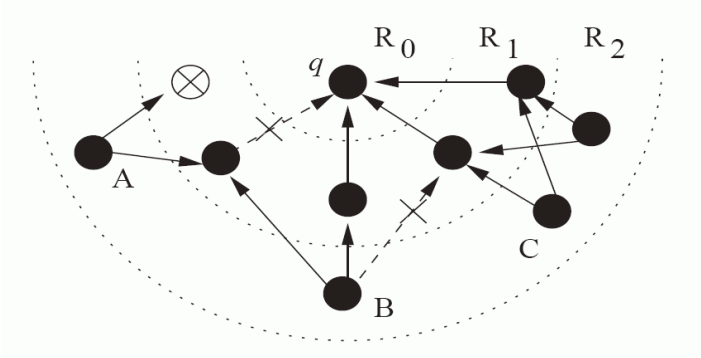

Figure-9 Rings topology [9]

Figure-9 also depicts link failures (crossed lines) and node failures (crossed circle) that can occur while data is collected from the nodes. The behavior of Synopsis Diffusion algorithm in these error conditions is explained in Section 2.3.2.4.

Multi-path routing with rings topology and duplicate insensitive aggregation used in Synopsis Diffusion and Approximate Aggregation has two main drawbacks. First, these techniques provide an approximation to the answer, thus they produce results with a bounded approximation error. Second, the message size is longer than the ordinary tree approach used in TAG or TinyDB, since the message in multi-path schemes contains duplicate insensitive aggregate data. With longer bytes, nodes will spend more energy to transmit it.

Tributary-Delta (TD) algorithm merges the tree topology with the multi-path topology. The tree topology is known for its low or zero approximation error, and its short message size. Multi-path approach is preferred when the loss rates become higher, since the multi-path routing has a loss-tolerant and robust nature. As seen in Figure-10, TD algorithm uses multi-path topology around the sink node, called Delta Region, where a link loss is very critical and would cause to lose the results of many lower tree levels. It uses tree topologies at the branches, called Tributary Region, since a link loss here in not as critical as the center.

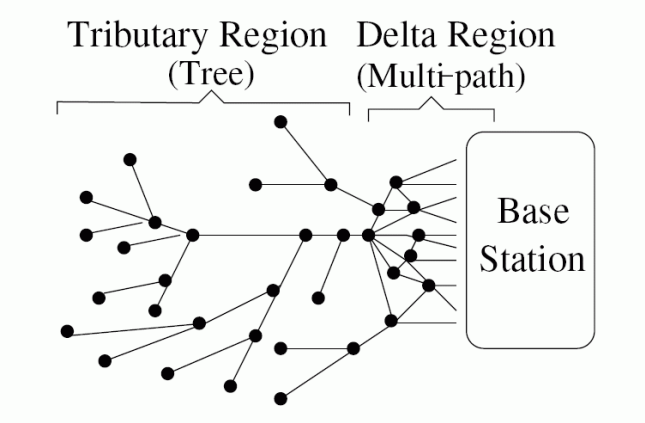

Figure-10 Tributary-Delta topology [11]

2.3.2.2 QUERY TYPES

TAG can work with aggregation queries only, and does not retrieve single queries for a particular node. Moreover, TAG distributes a query to all of the nodes in the network. Therefore, TAG is used most efficiently when all the sensor network nodes have same kind of sensors.

TinyDB can work with aggregation queries like TAG, but it can also work with events and single node queries. Events can be used to decrease network traffic and power consumption effectively by triggering the transmission of data upon the occurrence of an event.

Contrary to TAG, TinyDB sends the query to only those nodes that can participate in the query execution using Semantic Routing Trees. Moreover, TinyDB can make single node queries, whereas TAG cannot. Thus, TinyDB can be used in multi-purpose sensor networks, in which different kind of sensors are used in different nodes.

Directed Diffusion makes queries for a particular kind of data that can be answered by single nodes. It cannot make aggregate queries, whereas TAG and TinyDB can. Thus, Directed Diffusion can also be used efficiently in a multi-purpose sensor network.

In Directed Diffusion, more than one sink can make queries and receive data at the same time, which is not the case for other algorithms. Therefore, this algorithm can handle simultaneous and distinct queries inside a single network.

Similar to TinyDB, Directed Diffusion can make query aggregations in every sensor node and it can also work with events.

ACQUIRE is well suited to one shot and replicated data queries and can be found in many nodes. For example, it can query whether the current temperature is over 40°C. Answer to this query can be found in many nodes and it is not necessary to distribute the query to the whole network. On the contrary, Directed Diffusion is suited for continuous and non-replicated data queries; for example, number of vehicles detected in a region. Moreover, TAG and TinyDB are well suited for continuous and aggregate queries; e.g. maximum value of the temperature.

Synopsis Diffusion, Approximate Aggregation and Tributary-Delta algorithms are based on TAG, so they show similar properties to TAG in making aggregation queries.

2.3.2.3 DATA COLLECTION AND FUSION

In TAG and TinyDB; the epoch, or the period for a single aggregate produced by the network, is divided into the maximum level of the tree. All of the nodes are loosely synchronized with this epoch, and they allocate a time period in every epoch according to their tree levels. At this specified period of time the nodes at the same level wake up, take their sensor readings, receive their child nodes' results if available, aggregate them with own readings, send the aggregate result to their parents and go back to sleep. This process starts at the leaf nodes and when the epoch ends, the aggregation result would arrive at the sink node. Since the epoch is divided into the tree levels, and there should be a minimum time interval for every node level, the maximum data rate that can be obtained

from the network is limited by the tree level and minimum time interval for a single node operation.

In Directed Diffusion, the query result is received from a single node along the gradients of the reinforced gradient path. Directed Diffusion does not define any data aggregation scheme, since it is specially designed for single node queries; contrary to TAG, TinyDB, SD and TD.

ACQUIRE method retrieves the query results from the return path of the query, similar to Directed Diffusion. Again, ACQUIRE does not define any data aggregation scheme.

In Synopsis Diffusion and Approximate Aggregation, the query aggregation period is divided into epochs and at each epoch one aggregate is provided, similar to TAG. These algorithms generate the same optimal number of messages compared to the tree based approaches like TAG and TinyDB, but robustness is greatly enhanced.

Synopsis Diffusion and Approximate Aggregation algorithms use a bitmap of data at every node instead of the actual partial results. This bitmap is called synopsis by Synopsis Diffusion, and sketch by Approximate Aggregation, and it is based on the Flajolet and Martin's Probabilistic Count algorithm [12]. These synopses are used for duplicate insensitivity, so that even if duplicate readings arrive at a node, it will not change the value of the synopsis. A more detailed description of this algorithm is given below in Section 3.4.2.2.

At the beginning of each epoch, each node in the outermost ring generates its synopsis and broadcasts it. The nodes in ring R_i wake up at its specified time, generate local synopsis and receives from all the neighbors heard in ring R_{i+1}. It updates its local synopsis and at the end of its allocated time, broadcasts the updated result.

It is important to note that using synopses often introduce approximation errors in the exact aggregate result. This approximation error is the result of representing too much information in a limited size synopsis. However, for realistic packet loss rates, i.e. 5-30%, the errors from missing nodes are far more

than approximation errors. Furthermore, by increasing the length of the synopsis message, the relative error can be decreased.

Tributary-Delta approach merges tree and multi-path schemes together, and therefore the standard aggregation technique of TAG is used for the tree topologies and Synopsis Diffusion technique is used for the multi-path topology. However, a conversion function is required to convert a partial result generated by the tree algorithm to a synopsis that can be used by the multi-path algorithm. This conversion function will be used at the boundary nodes, which are the root nodes of the tree topology, and the leaf nodes of the delta-region.

2.3.2.4 FAULT TOLERANCE

In standard tree-based systems such as TAG, when a single node on the routing path fails, the entire sub-tree of values is lost; thereby introducing a large error to the final result.

TAG proposes to split the accumulated aggregation value to more than one parent. If the aggregate value is v, every node will send v/k value to each of its k number of parents. While this approach reduces the error for a link failure from v to v/k, the expected aggregation error is still the same as ordinary tree approach [9].

In TAG and TinyDB algorithms, the routing tree used can be refreshed frequently in order to adapt the network changes. Moreover, lost nodes can get a parent node to themselves, by snooping over other nodes' messages.

TinyDB can maintain the SRT in the case of lost nodes or link quality changes. SRT performance depends on the quality of the clustering of children beneath parents.

In Directed Diffusion, the initial interest contains a low data rate, and the actual query contains the desired data rate. However, the initial query message still causes other nodes to send data in a low frequency, and these alternative gradient paths are also kept at the nodes for reliability. When the main link fails,

the data transfer might continue from one of the alternative gradient paths, by reinforcing the new path and suppressing the old one.

Approximate Aggregation and Synopsis Diffusion are fault-tolerant algorithms that use duplicate insensitive synopses and multi-path routing to overcome the high error rates resulting from path failures in tree topologies. It is shown that using these algorithms results in enhanced robustness and significantly low error rates compared to TAG. However, these synopses are larger in size compared to TAG partial results and they require more energy to be transferred. Even considering the approximation errors caused by using synopses, it is shown that these algorithms result in enhanced robustness and significantly low error rates compared to TAG.

Figure-9 shows link and node failure examples for a sensor network operating with Synopsis Diffusion algorithm. As seen in this figure, node A has two parents, one of which has a node failure (crossed circle) and the other node has a link failure with sink node (crossed line). Since all the paths from node A to sink is blocked, node A's data cannot reach to sink. In the same figure, node B has three parents, two of which face link failures. However, there is still a path from node B to the sink, thus data of node B can be transmitted to the sink.

Tributary-Delta algorithm uses synopses at the delta region, where the error rate is critical. This way, the approximation error and overall transmitted data will be reduced, while keeping the link errors under control. Tributary-Delta also executes dynamic adaptation algorithms to adjust the boundary point between the tributary and delta regions, according to the current message loss rates in various regions.

CHAPTER III

DEFINITION AND IMPLEMENTATION

3.1 THE PROPOSED SENSOR NETWORK STACK ARCHITECTURE

The proposed stack architecture consists of the following layers: application, query, aggregation, network, MAC and physical (Figure-11). In addition, there is a notification board service, which is responsible for cross layer communication. This stack will be implemented in all the nodes of a sensor network.

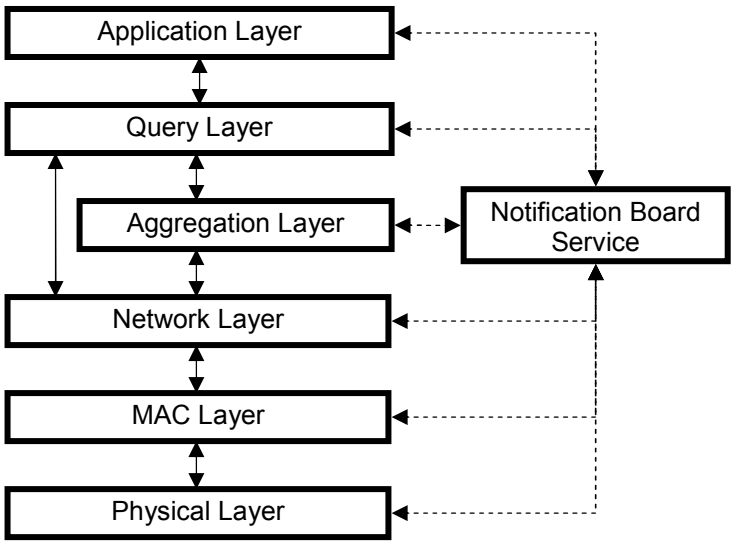

Figure-11 The new sensor network protocol stack

Sensor networks have similarities with distributed database systems, and there are many database oriented algorithms introduced for sensor networks [2..11] as explained in section 2.3. This stack architecture also builds on top of this idea. The two crucial layers of this stack, which are query and aggregation layers, try to utilize the distributed database notion for sensor networks.

Basically; query layer will request data from the sensor network, and aggregation layer will collect and aggregate data inside the network. Network layer will deal with routing and addressing; and application layer will perform higher level application operations. Mostly, the core of the sensor network algorithm will be executed by query, aggregation and network layers. In the following paragraphs, the operations of these layers will be explained in detail.

Application layer will communicate with the control center and receive the details of the sensor network tasks. Afterwards, this layer will prepare query requests and send them down to query layer. After receiving the results of a query, this layer will send them back to the control center. Other high level application related tasks can also be executed here.

Query layer is responsible for all query related operations. First of all, this layer will receive query requests from the application layer and prepare the query packets accordingly. If required, it will make query optimizations or merge queries with previous ones. Then, it will start a query by distributing a query packet inside the network. Furthermore, it will filter incoming queries, and receive query results from the aggregation layer whenever the response is available. This layer is also responsible to define events, such that when one of these events occurs, the node receiving this event will initiate a data transfer to the sink node. Network-wide query adaptations will also be directed and performed via this layer.

Query layer requires a connection with the network layer, since it will define the network route of a query and has to learn network address of an incoming query. After setting up the address of the outgoing query packet, it will be sent to network layer to be transferred to its destination. Moreover, this layer should also have a connection with the aggregation layer, in order to send

information about the incoming query, to receive results of a query, or for the case of an event definition.

Aggregation layer has to perform data response and aggregation processes inside the nodes for the whole sensor network. This layer will start functioning whenever a query has arrived or a predefined event has occurred. Depending on the algorithm, it will define data transmission and reception periods so that sensor node will be able to sleep for the rest of time. This layer will acquire the reading of its own sensor and store it. Moreover, data coming from neighbor nodes will also be received and stored by this layer. Data aggregation is the essential task of this layer, thus, the gathered data will be aggregated here. Query layer will be informed of this aggregate result. Finally, the results will be transmitted to the network.

Aggregation layer has to have a connection with the query layer in order to receive information about incoming queries, or to send back the query response. Similar to query layer, this layer also requires a connection to the network layer, because aggregation layer will define the route of an outgoing packet, and it might require the address of an incoming packet.

Network layer of this stack will have the same functions as the network layer of TCP/IP. However, since the addressing is confined for a particular network, there is no need to use IP addresses. The indexes of the nodes can also be used as their addresses. Nevertheless, better addressing schemes for sensor networks can be applied inside this layer. Network layer will have connections to both query and aggregation layers and will route both data and query packets to their appropriate addresses. This layer can realize different routing algorithms, such as geographical routing or a hierarchical routing. It is also possible to execute query informed routing, by receiving the query information from the query layer.

In this structure, query layer and aggregation layer will be able to keep information about next-hop neighbors. For example, they can keep the address of the parent query node, or the previous data sending node. By the help of the network layer, this next hop can also be a far away node which will be reached

after some routes. Thus, sensor network would be scalable. Moreover, using query informed routing; this layer will increase energy conservation, since the packets will only be sent to appropriate nodes instead of being broadcast to the entire network.

The MAC layer of our stack arranges the access to the wireless medium and tries to avoid collisions. It can be realized by using one of the MAC layer algorithms defined for wireless networks. In our simulations for this thesis, we used IEEE 802.11 MAC layer protocol.

The physical layer is used to actually transmit data between different nodes. Since sensor networks operate on wireless medium, this layer is mostly radio, as the case for our simulations.

3.2 SIMULATION STUDY

In this thesis, four different sensor network algorithms will be implemented using the stack architecture defined in 3.1. These algorithms are Tiny Aggregation [2], Synopsis Diffusion [9], Tributary-Delta [11] and Directed Diffusion [4].

TAG and Directed Diffusion are widely known algorithms of sensor network systems and they are referenced in many studies. They are also physically implemented and tested algorithms for real life sensor networks. Moreover, these two algorithms have a critical difference; TAG is designed to retrieve data aggregates from the whole network, whereas Directed Diffusion is used to retrieve data from single nodes. Simulating these two algorithms would show that our stack is reliable and can be used in a wide range.

Synopsis Diffusion and Tributary Delta are new algorithms, and they have both similarities and differences to TAG. To sum up, they are designed to increase the efficiency of TAG when a link failure occurs. Synopsis Diffusion

makes in-network aggregation similar to TAG, but does not use tree based routing. Instead, it tries to overcome the problem of link failure by using multi-path routing and duplicate insensitive data, called synopses. Tributary-Delta algorithm merges Synopsis Diffusion and TAG by using these two algorithms in different parts of the network and adapts the regions dynamically. In order to show the modularity and extensibility of the stack, it is helpful to simulate these algorithms.

The graphical results obtained from the simulations will be compared to the original results of these algorithms obtained in their respective studies. With this comparison, it will be possible to show that algorithms implemented with the proposed stack behave the same and do not cause performance degradation. As a second benefit, implementing these four algorithms would move the stack from concept into reality and will show that the stack is applicable. Last of all, by examining the implementation details of these algorithms, it will be possible to show the modularity and extensibility of the proposed architecture.

3.3 SIMULATION ENVIRONMENT

The simulations are done using OMNeT++ simulation environment [16,17] and INET framework [18,19] developed for OMNeT++.

OMNeT++ is an object oriented discrete event network simulator. It is freely available for academic and non-profit use. The simulator, user interfaces and the tools are portable on different operating systems; they can work with Windows or Unix platforms using various C++ compilers. [17]

OMNeT++ simulations have different execution types for different purposes; it can be used for debugging, fast execution or demonstration. The user interface helps to debug and demonstrate the simulation, by visually showing how the model works, or letting the user modify the variables inside the model.

An OMNeT++ model consists of hierarchically nested modules that can have unlimited depth. These modules have their own parameters which are used

to customize module behavior. Module communication is accomplished by passing messages between two modules, using the predefined gates or a sending the message directly to the module. The messages can contain any object that is known by both modules. The gates are used to transmit one way messages between two modules. These gates can be defined as network communication lines, or just an instantaneous communication path.

OMNeT++ uses NED files to define the simulation topology. A NED file is a textual declaration that defines the modules, parameters, gates, and module hierarchy. NED files are written is a specific language called NED language.

Every module in a simulation is either a simple module or a compound module. Simple modules are at the lowest level of the hierarchy, and they cannot encapsulate another module. Simple modules are programmed in C++ as a class derived from the cSimpleModule base class of OMNeT++. The actions required for the simulation are defined in these classes. Every simple module has also a NED file associated with it. In this NED file, the parameters and gates of the module are defined. The module class can use these parameters and gates defined in the NED file with the help of OMNeT++ simulation library.

Compound modules may contain simple modules or other compound modules. These modules do not have an associated C++ class; they are only defined in a NED file. This NED file contains definitions for the nested modules, own module parameters, and gates. Using this file, the parameters of a nested module are set or associated with the parameters of the compound module. Moreover, the gate connections between the nested modules and the compound module are defined.

A sample wireless sensor network simulation interface of 50 nodes in a 400x400 region is shown in Figure-12. The nodes are shown as square icons having blue circles inside. The wide circles around the nodes show the wireless communication region. The sun icon is the channel control module, which serves as the controller for the wireless communication.

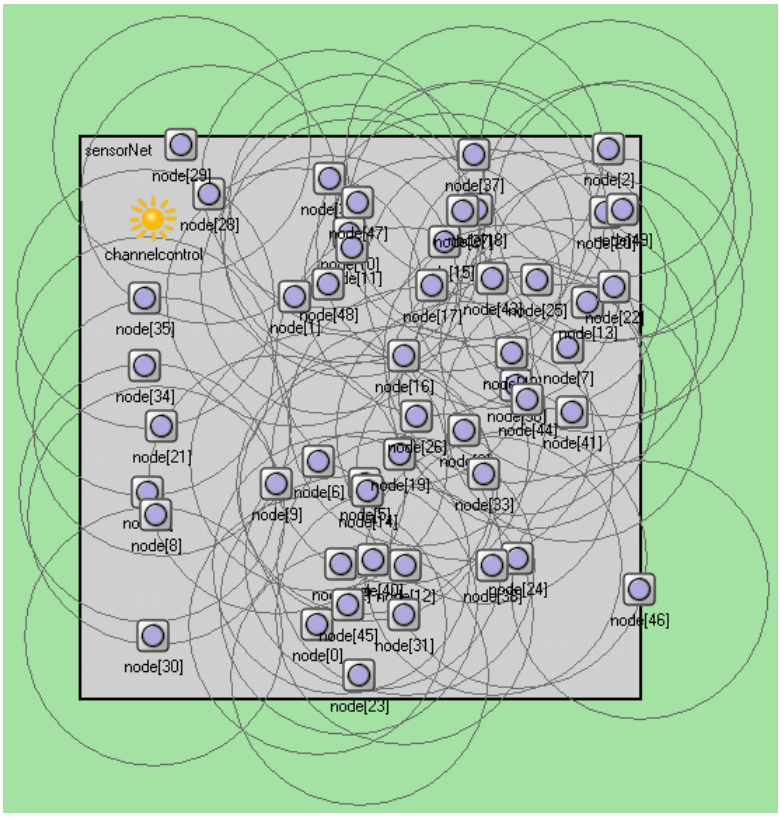

Figure-12 OMNeT++ sensor network simulation example

The simulation model of this thesis uses modules borrowed from INET framework. The INET framework is an open-source communication networks simulation package, written for the OMNeT++ simulation system. The INET framework contains models for several Internet protocols like TCP, UDP, IP, Ethernet or 802.11 [19]. The modules that are extracted from the INET framework are as follows:

- **ChannelControl:** This simple module gets location and movement information about the nodes of the network. Then, it determines which nodes are within communication or interference range of another. This information is used by radio interfaces, like Nic80211 module.

- **NotificationBoard:** Notification board simple module enables the modules to communicate directly using event firing and subscription to these events. This module is used to realize a cross-layer communication framework in this thesis.
- **InterfaceTable:** This simple module contains the network interface table for a sensor node. This module is required for Nic80211 module to work properly.
- **Nic80211:** This compound module implements a network interface card that uses 802.11 wireless MAC layer protocol. It is used as the MAC layer and physical layer combination. It contains MAC80211, Decider80211 and SnrEval80211 modules.
- **Mac80211:** This simple module implements 802.11 MAC layer protocol. This module supports ad-hoc mode only, that is, it does not generate or handle management frames.
- **Decider80211:** This simple module is the decider module for the physical layer implementation of 802.11 network interface card.
- **SnrEval80211:** This simple module is the signal to noise ratio evaluator for the physical layer implementation of 802.11 network interface card. Along with Decider80211 module, this module makes up the physical layer.
- **BasicMobility:** This simple module is a prototype module for mobility modules, and it is required for channel control module to determine the locations of the nodes. It does not contain any mobility algorithm.

The following modules have been developed within the scope of this study for the sensor network simulation model:

- **SensorNet:** This compound module is the top level module and it models the simulated sensor network. It contains the channel control

module from INET framework, and the sensor node array as seen in Figure-13.

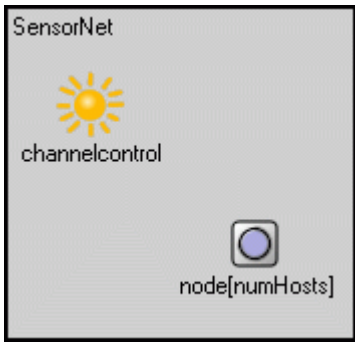

Figure-13 SensorNet module

- **SensorNode:** This compound module models a sensor node. It contains the QueryLayer, AggLayer and NetworkLayer modules; as well as NotificationBoard, InterfaceTable, BasicMobility, and Nic80211 modules from INET framework. The model is shown in Figure-14.

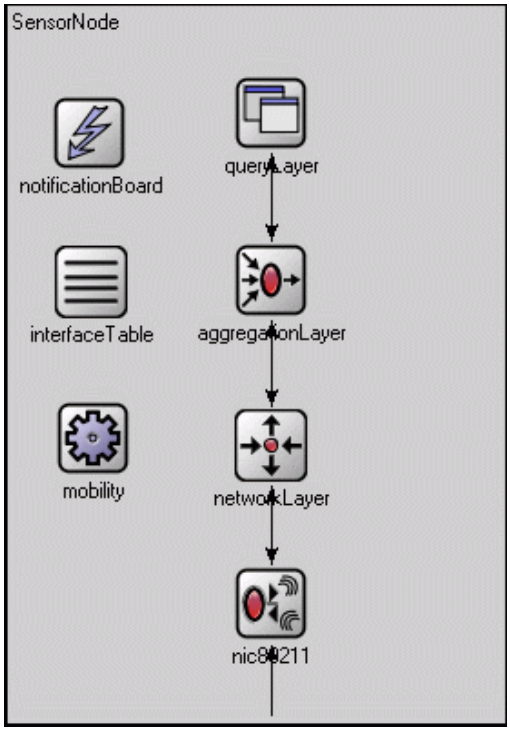

Figure-14 SensorNode module

- **QueryLayer:** This simple module defines and implements the query layer of our protocol stack.
- **AggregationLayer:** This simple module defines and implements the aggregation layer of our protocol stack.
- **NetworkLayer:** This simple module defines and implements the network layer of our protocol stack.

3.4 PROTOCOL IMPLEMENTATIONS

In the following subsections, the protocol implementations of query, aggregation and network layers for TAG, Synopsis Diffusion, Tributary Delta and Directed Diffusion algorithms will be explained. In these simulations, simple queries are executed using the query layer of the sink directly, without using an application layer.

3.4.1 TAG

3.4.1.1 QUERY LAYER

Query layer of TAG uses a packet structure to communicate with the query layer of another node. This packet is distributed through the entire network by broadcasting. The contents of this packet are shown in Figure-15.

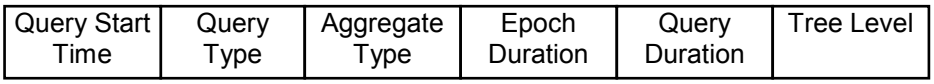

Figure-15 Query layer packet structure for TAG

In this packet, query start time field is the starting time of a query set at the sink node. This will be used as an identification of a particular query inside the whole network. Query type defines the type of information requested from the network. This can be the number of birds detected, temperature, or node count. Aggregate type defines the type of aggregation that will be applied at every node inside the network. Typical examples are average, sum, min and max.

Epoch duration is the period of a single aggregate result. At every epoch, the sink node will receive a single aggregate result from the network .Inside an epoch, the nodes at a specific tree level will awake, make their sensor readings, receive the results from the previous level nodes, aggregate the data, send their results and go back to sleep for the next epoch. Because of the time required to perform these actions, epoch duration has a lower limit and cannot go below that value for correct network operation.

Query duration is the total time of the query requested. For a particular query, there will be (query duration / epoch duration) times of aggregate result will be generated by the network and consumed at the sink node. Tree level

defines the distance of a node from the sink node defined as the number of hops. This is updated at every node that rebroadcasts the query.

The following pseudocode shows the query layer operation for TAG algorithm.

```
if (sink node)
{
      prepare query packet;
      broadcast it using network layer;
}

if (query packet arrived)
{
      if (not received before)
      {
            record query information;
            update tree level and parent node;
            Send query notification to aggregation layer;
            prepare new query packet;
            broadcast it using network layer;
      }
      else
            discard query packet;
}

if (query result notification arrived)
      if (sink node)
            record query result;
```

Figure-16 Query layer operation of TAG

The sink node will set the values required for the query packet at the start of a query. Tree level will be zero and the parent node will be null. This packet will be broadcasted to the network by sending it to the network layer.

Query layers of other nodes that hear this broadcast will first check whether they already received this query before, using the query start time field of the received query packet. The packet will be dropped if the query already arrived to that node. Otherwise, the query will be processed as follows.

The contents of the query packet will be extracted and stored inside the query layer. The tree level of the query will be updated by increasing the tree level by one. The parent node of this query will also be stored, using the network address of the node that sent the incoming query. This address will be taken from the network control packet of network layer. Afterwards, query layer will inform the aggregation layer about the incoming query using the notification board. Query layer will send the received query packet contents and parent node address. Finally, the query will be broadcasted to the network by preparing a new query packet using the updated information.

3.4.1.2 AGGREGATION LAYER

Aggregation layer of TAG uses the following packet structure to communicate with another aggregation layer.

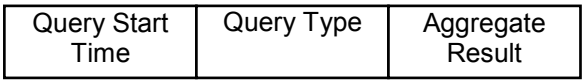

Figure-17 Aggregation layer packet structure for TAG

Similar to query layer, query start time and query type is used to identify the query that is being replied. Aggregate result is the obtained result after the aggregation is performed over the received results.

Aggregation layer implementation for the TAG algorithm is summarized in the following pseudocode.

```
if (query notification arrived)
{
      store query information;
      set next epoch_start, epoch_end, query_end timers;
      query_available = true;
}

if (epoch_start timer)
{
      read and store own sensor data;
      inside_epoch = true;
}


if (packet arrived from lower layer)
{
      if (not an aggregate packet)
            send it to query layer;
      else
      {
            if (inside_epoch)
                  store aggregate result;
            else
                  discard packet;
      }
}

if (epoch_end timer)
{
      inside_epoch = false;
      aggregate all stored results;
      send aggregate result to query layer;

      if (query_available)
            set next epoch_start, epoch_end timers;

      if (not sink node)
      {
            prepare aggregation packet;
            send it to the parent node;
      }
}

if (query_end timer)
{
      query_available = false;
}
```

Figure-18 Aggregation layer operation of TAG

The aggregation layer protocol starts operation whenever query layer informs about a new query by sending the query information. Aggregation layer stores the contents of this query information. According to the epoch duration, query duration and tree level information of this query, aggregation layer sets the epoch_start, epoch_end and query_end timers and lets the node sleep until the epoch start timer fires.

When an epoch starts, aggregation layer gets its own node's sensor reading. Then it starts to listen for aggregation packets that come from other nodes. Whenever a packet is received from a child node, the packet is checked for the query start time and query type. If they match, the aggregate result is added to the results array. As soon as the epoch ends, the node stops listening for other aggregation packets.

Afterwards, aggregation layer performs aggregation by running the requested aggregation function over the received results and its own reading, and obtains an aggregate result. This result is sent to the query layer by firing an event through the notification board. If the node is not the sink node, this aggregate result is sent to the parent node by setting the address in the network control packet.

If the query duration has ended, the aggregation layer will suspend the node operation and let it sleep. If not, the timers will be set for the next epoch and the operation will restart.

3.4.1.3 NETWORK LAYER

Network layer protocol implementation is the same for all of the algorithms. It uses the following packet structure to communicate with another network layer.

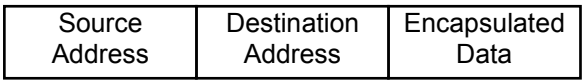

Figure-20 Network layer packet structure

Source address is the address of the node that sends the packet. It is set at the network layer. Destination address is the address of the node that will receive the packet. The data sent by query and aggregation layers are encapsulated and inserted into this packet.

The Network control packet is used to communicate the network layer and query and aggregation layers. The structure is the same as network packet structure.

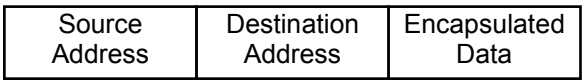

Figure-21 Network control packet structure

The query and aggregation layers use this packet to send data to the network. They set the destination address and encapsulate data, then send it to the network layer. This packet is also used to send data coming from the lower MAC layer to upper aggregation layer. The network packet received from the network is copied to a network control packet and sent to upper layer. The operation of this layer is explained as follows.

```
if (packet arrived from upper layer)
{
      prepare a network packet;
      broadcast it using MAC layer;
}

if (packet arrived from lower layer)
{
      if (destination == own address)
      {
            prepare a network control packet;
            send it to aggregation layer;
      }
      else
            discard packet;
}
```

Figure-22 Network layer operation

Query and aggregation layers send network control packets to network layer. When they arrive at network layer, the contents are copied to a network packet and sent to MAC layer for broadcasting. Whenever a network packet arrives from the MAC layer, first, the destination address field is checked against the address of the node. If it is the same, a network control packet is prepared according to the address and data fields and sent to aggregation layer. If it is not the destination node, the packet is discarded.

3.4.2 SYNOPSIS DIFFUSION

3.4.2.1 QUERY LAYER

Query layer implementation of Synopsis Diffusion is the same as TAG, explained in section 3.4.1.1.

3.4.2.2 AGGREGATION LAYER

Aggregation layer of SD uses the following packet structure to communicate with another aggregation layer.

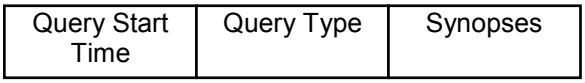

Figure-23 Aggregation layer packet structure for SD

It is similar to the aggregation packet of TAG except that the aggregate result field contains the synopses instead of the raw aggregate obtained. These synopses are stored inside a 32 bit integer array having 20 elements.

The operation of aggregation layer protocol for Synopsis Diffusion is summarized in Figure-24. It is similar to TAG but has some modifications as follows.

When an epoch starts, TAG obtains its own sensor reading and stores it directly. However, SD has to convert this value to a synopsis using a synopsis generation function. This function produces specified number of synopses from the sensor reading and stores them in an array. This function works briefly as follows. At first, all bits of the synopsis is set to 0. In order to insert the value “1” to the synopsis, a probability of 0.5 is evaluated repeatedly in a loop using a random number generator. If the probability evaluation is true, the bit in turn is set to 1 and the loop is broken. The final value of the synopsis is the result of this function. This operation is repeated n times in a loop in order to insert the value of n to the synopsis, moreover, a higher level loop is used to produce more than one synopses. This operation can be seen in the pseudocode given in Figure-24.

```
for each (synopsis in synopses_array)
{
      set all bits of synopsis to 0;
      randomize;

      while (i < number_to_insert)
      {
            while (j < 32) // max. 32 bit synopsis
            {
                  generate random number x;
                  if (x > 0.5)
                  {
                        set bit i of synopsis to 1;
                        break;
                  }
            }
      }
}
```

Figure-24 Synopses generation function of SD

The number of synopses directly affects the approximate error of SD such that increasing the number of synopses decreases the approximate error. The synopsis generation function is implemented according to the definitions given in [9], [10] and [12]. The probabilistic operations in this function are implemented using the random number generators of C++ and OMNeT++.

The second difference is that, when data received from another node, TAG stores it directly to the received data array. However, SD does not store it, and it directly aggregates the result to its own synopsis array using a binary sum operation. Therefore, the aggregate result is always up to date.

At the sink node, instead of sending the raw aggregate result directly to the query layer, the synopses are evaluated and converted to the real aggregate data and this value is sent to query layer. The synopsis evaluation function is also implemented according to the definitions given in [9], [10] and [12].

```
if (query notification arrived)
{
      store query information;
      set next epoch_start, epoch_end, query_end timers;
      query_available = true;
}

if (epoch_start timer)
{
      read own sensor data;
      generate synopses;
      inside_epoch = true;
}


if (packet arrived from lower layer)
{
      if (not an aggregate packet)
            send it to query layer;
      else
      {
            if (inside_epoch)
                  aggregate synopses;
            else
                  discard packet;
      }
}

if (epoch_end timer)
{
      inside_epoch = false;
      evaluate synopses;
      send aggregation result to query layer;

      if (query_available)
            set next epoch_start, epoch_end timers;

      if (not sink node)
      {
            prepare aggregation packet using synopes;
            broadcast it using network layer;
      }
}

if (query_end timer)
{
      query_available = false;
}
```

Figure-25 Aggregation layer operation of SD

3.4.2.3 NETWORK LAYER

Network layer implementation of SD is the same as TAG, explained in 3.4.1.3.

3.4.3 TRIBUTARY DELTA

Two different versions of the Tributary-Delta algorithm are implemented in this thesis. They are TD-Coarse and TD-Fine. TD-Coarse algorithm is implemented as defined in [11]. It adapts the network when link errors occur by broadcasting a network-wide message. Using this message, TD-Coarse algorithm varies the tributary and delta regions of the entire network, assuming a network-wide packet loss.

In [11], along with TD-Coarse, another version of Tributary Delta is defined and it is named TD. Contrary to TD-Coarse, this version makes adaptations to the network only in the regions where link errors occur. In order to achieve this, every node at the boundary of two regions sends the number of child nodes that didn't contribute up to the sink. These boundary nodes are the top level nodes of the TAG sub-trees, thus they can define the erroneous regions. According to these data, sink node broadcasts a message which will make the required adaptations in erroneous regions of the network. This message will switch the boundary nodes to TAG or the child nodes of the boundary nodes to SD.

In this thesis, as an extension to TD algorithm of [11], another version of Tributary Delta is developed and named as TD-Fine. Just like TD, TD-Fine makes adaptations only in the regions where the errors occur. However, contrary to TD, node contribution information is directly processed by the boundary nodes instead of sending them to the centralized sink. Moreover, adaptation is not performed using a network-wide message broadcast. Instead, the boundary nodes

send a message to their first level child nodes to switch them to SD or they directly switch themselves to TAG.

TD-Coarse algorithm is simple and can deal with global errors. TD-Fine is more complex; however, it achieves better results with regional errors. The only changing layer for these two algorithms is the query layer. Both of their query layers are explained in the following sections.

3.4.3.1 QUERY LAYER OF TD-COARSE

This algorithm adds a new field, called TD boundary, to the query packet structure of TAG as seen in Figure-26. This TD boundary field will determine the tree level boundary between the delta region, in which SD algorithm is used, and the tributary region, in which TAG algorithm is used. This information will be distributed to the network along with the query.

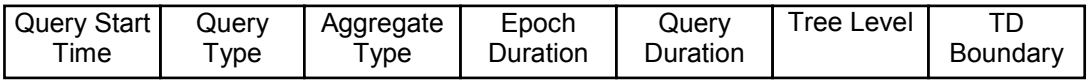

Figure-26 Query packet structure for TD

For TD-Coarse algorithm, query layer at the sink will make network adaptations by changing the TD boundary according to the number of nodes participating to the query. The boundary change will be distributed to the network by using the boundary change packet having the following structure.

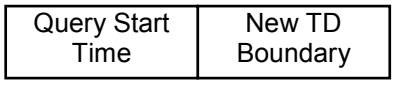

Figure-27 Boundary change packet of TD

Query start time field is be used to identify the query. New TD boundary field is used to announce the new boundary of the previously distributed query. This new boundary will be determined by sink node by measuring and comparing the node participation ratio to some predefined value.

This algorithm tries to keep the number of participants to a query over a threshold. Moreover, it also tries to increase TAG regions if the error rate is low. Hence, if the measured participation ratio goes down below the lower a limit, SD region will be extended so that more nodes will participate to the query. If the participation ratio goes over the upper limit, TAG region will be extended so that fewer packets will be transferred and approximate error will be reduced. Changes will be reflected by increasing or decreasing the current TD boundary value and distributing this new value to the network using boundary change packet. The measurement will be performed by making a count query with a predefined frequency.

The operation of the query layer of TD-Coarse algorithm is explained in the following pseudocode.

```
if (sink node)
{
      prepare measurement query packet;
      broadcast it using network layer;

      prepare actual query packet;
      broadcast it using network layer;
}
if (query packet arrived)
{
      if (not received before)
      {
            record query information;
            update tree level and parent node;
            Send query notification to aggregation layer;
            prepare new query packet;
            broadcast it using network layer;
      }
      else
            discard query packet;
}

if (query result notification arrived)
{
      if (sink node)
      {
            if (measurement query)
            {
                  compare contribution ratio;
                  if (below lower limit)
                  {
                        increase td boundary by one;
                        notify aggregation layer;
                        broadcast new boundary to network;
                  }
                  if (over upper limit)
                  {
                        Decrease td boundary by one;
                        notify aggregation layer;
                        broadcast new boundary to network;
                  }
            }
            else
                  record query result;
      }
}
```

Figure-28 Query layer operation of TD-Coarse

```
if (td boundary packet arrived)
{
      if (query start times match)
      {
            change boundary of the present query;
            notify aggregation layer;
            rebroadcast boundary change packet;
      }
}
```

Figure-28 Cont'd

The measurement of the contribution ratio is performed by distributing a count query before the actual query is started. The query duration will be same as the real query but epoch duration will be different. This epoch duration will determine the network adaptation frequency. When the result of this count query arrives, the query layer will determine whether an adaptation will be performed or not, by comparing it to the limits. If it is beyond the limits, a boundary change message will be prepared and broadcasted to the network.

Other nodes that receive this boundary change message will first check the start time field of the incoming message and the current query's start time. If they match, they will check whether the new boundary is different than the current boundary. If it is different, they will set their TD boundary information with the new one and inform aggregation layer about this change using the notification board. Afterwards, they will rebroadcast the boundary change message.

3.4.3.2 QUERY LAYER OF TD-FINE

TD-Fine algorithm uses the same query packet as TD-Coarse. However, the boundary change packet of TD-Fine has only the query start time field, since the tree level is not used to determine the boundary.

The key difference of this algorithm from TD-Coarse is that, every node residing at the boundary of tributary (TAG) and delta (SD) regions has to make adaptations to their sub nodes. These nodes can be thought as sinks of sub-trees. They collect the TAG aggregate of their sub-trees, translate it into a synopsis value, and send it to the sink using SD. According to the aggregate results, they decide whether to switch their child nodes to SD by sending a message, or switch themselves to TAG.

The query starts with every node working in the delta region, since the initial measurement is used to evaluate future measurements. In the regions where there are little or no link errors, the nodes would start to switch tributary region. The operation of this layer is summarized in the following pseudocode.

```
if (sink node)
{
      prepare measurement query packet;
      broadcast it using network layer;
      prepare actual query packet;
      broadcast it using network layer;
}
if (query packet arrived)
{
      if (not received before)
      {
            record query information;
            update tree level and parent node;
            Send query notification to aggregation layer;
            prepare new query packet;
            broadcast it using network layer;
      }
      else
            discard query packet;
}
if (query result notification arrived)
{
      if ((boundary node) and (measurement query))
      {
            if (first measurement)
                  store contribution ratio;
            else
            {
                  compare contrib. ratio to previous value;
                  if (below lower limit)
                        broadcast boundary packet;

                  if (over upper limit)
                  {
                        switch itself to TAG;
                        notify aggregation layer;
                  }
            }
      }
      if ((sink node) and (not measurement query))
            record query results;
}
if (boundary packet arrived)
{
      if (query start times match)
      {
            switch itself to SD;
            notify aggregation layer;
      }
}
```

Figure-29 Query layer operation of TD-Fine

Similar to TD-Coarse, the network adaptation will be performed at every epoch of the measurement query initiated from the sink node. The results of this query will be evaluated at every boundary node. These nodes will compare the count results of their sub-trees with the previous results. If the contribution rate has decreased below the lower limit, they will broadcast a Boundary Change message. The nodes hearing this message and working with TAG algorithm will switch from TAG to SD, and they will become the new boundary nodes. The message will not be broadcasted again. If the contribution rate has increased beyond the upper limit, the nodes at the boundary will not send a message; they will just switch themselves from SD to TAG, and start sending the aggregate values to their previous parents.

3.4.3.3 AGGREGATION LAYER

TD algorithm uses both of the aggregation layer packets of SD and TAG. TAG packets will be used in the tributary regions and SD packets will be used in delta regions. Since this algorithm is a combination of SD and TAG algorithms, the operation is also a mix of two. However, there are two additions to these algorithms as follows.

The node that resides at the boundary, that is, nodes whose tree level is the same as TD boundary will receive TAG packets but will broadcast SD packets. Therefore, these nodes have to convert the TAG aggregate result to SD synopses at the end of each epoch.

Second addition is that, when the query layer informs about a boundary change, aggregation layer will act accordingly, and will update its boundary information.

The operation of TD aggregation layer is summarized in the following pseudocode.

```
if (query notification arrived)
{
      store query information;
      set next epoch_start, epoch_end, query_end timers;
      query_available = true;
}

if (epoch_start timer)
{
      read own sensor data;

      if (inside SD region)
            generate synopses;
      else
            store own sensor data;

      inside_epoch = true;
}


if (packet arrived from lower layer)
{
      if (not an aggregate packet)
            send it to query layer;
      else
      {
            if (inside_epoch)
            {
                  if (inside SD region)
                        aggregate synopses;
                  else
                        store aggregate value;
            }
            else
                  discard packet;
      }
}
```

Figure-30 Aggregation layer operation of TD

```
if (epoch_end timer)
{
      inside_epoch = false;
      if (inside SD region)
            evaluate synopses;
      else
            aggregate all stored readings;

      send aggregation result to query layer;

      if (query_available)
            set next epoch_start, epoch_end timers;

      if (not sink node)
      {
            if (at the boundary)
                  generate new synopses;

            if ((inside SD region) or (at the boundary))
            {
                  prepare aggr. packet using synopses;
                  broadcast it using network layer;
            }
            else
            {
                  prepare aggr. packet using the result;
                  send it to parent node;
            }
      }
}

if (query_end timer)
{
      query_available = false;
}
```

Figure-30 Cont'd

3.4.3.4 NETWORK LAYER

Network layer implementation of TD is the same as TAG, explained in 3.4.1.3.

3.4.4 DIRECTED DIFFUSION

3.4.4.1 QUERY LAYER

Directed Diffusion algorithm uses a query packet similar to TAG to distribute the query into the network. The following figure shows the structure of this packet.

Query Start Time	Sink Node	Query Type	Aggregate Type	Epoch Duration	Query Duration	Tree Level

Figure-31 Query layer packet structure for Directed Diffusion

The difference of this packet from TAG's query packet is the sink node field. This field contains the address of the sink node that makes the query. It is important for Directed Diffusion algorithm, since there can be more than one sink working in the same sensor network.

The following pseudocode shows the query layer operation of Directed Diffusion.

```
if (sink node)
{
      prepare query packet;
      broadcast it using network layer;
}

if (query packet arrived from aggregation layer)
{
      if (not received before)
      {
            update parent node;
            send query notification to aggregation layer;
            rebroadcast query packet;
      }
      else
      {
            if (epoch and duration are different)
            {
                  update query information;
                  send reinforcement notification to aggr. layer;

                  if (not source node)
                        resend the packet to prev. response node;
            }
      }
}

if (response notification arrived)
{
      record address of the previous response node;

      if (sink node)
      {
            if (own query answered)
            {
                  if (initial query)
                  {
                        prepare the reinforced query packet;
                        send it to the previous response node;
                  }
                  else
                  {
                        record query answer;
                  }
            }
      }
}
```

Figure-32 Query layer operation of Directed Diffusion

Every sink node starts their initial query by broadcasting a query message. Query layers of the nodes that hear this message rebroadcast it and send query information to their aggregation layers using the notification board. Whenever this query reaches a node that has the requested information, the aggregation layer of that node starts sending the required data back to the sink node. As the query reply routes back to the sink via aggregation layers, query layers of these nodes are also informed of the incoming reply and the address of the sender. When the first response reaches sink node, it will reinforce this path by sending a reinforced query packet, which contains increased epoch time and duration compared to the initial query. Since every query layer keeps track of the previous responses, this packet will travel along the reverse path of the incoming reply via query layers of the nodes on the path.

3.4.4.2 AGGREGATION LAYER

Directed Diffusion uses the following query response packet to deliver a query reply to the sink node.

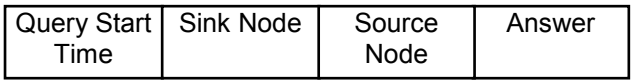

Figure-33 Aggregation layer packet structure for Directed Diffusion

Query start time and sink node fields are used to identify the query that is answered. Source node field is used to identify the query response while it is being reinforced.

The aggregation layer operation of Directed Diffusion is summarized as follows.

```
if (query notification arrived)
{
      record query information;
      if (data available)
            setup response timer;
}

if (response timer)
{
      read sensor value;
      prepare query response packet;
      send it to parent query node;
      setup next response timer;
}

if (packet arrived from lower layer)
{
      if (response packet)
      {
            send response notification to query layer;

            if (not sink node)
                  resend the packet to previous query node;
      }
      else
            send to upper layer;
}

if (reinforcement notification arrived)
{
      update query epoch and duration;
}
```

Figure-34 Aggregation layer operation of Directed Diffusion

Query layer of a node sends the notification of incoming query to the aggregation layer. Aggregation layer records the query information, and if it has the requested data, it sets up the response timer according to the query epoch and duration. When the timer starts, aggregation layer reads the sensor value, prepares the response packet and sends it to the parent node that sent the query. Afterwards, it sets up the next response timer. The node that receives this packet notifies its query layer about the incoming data and the sender address. If the node is not the sink, the packet is routed to the previous query sender. Thus, response packet reaches to the sink node using the reverse path of the query. Lastly, whenever the node is reinforced, the new query values are updated by the help of the reinforcement notification.

3.4.4.3 NETWORK LAYER

Network layer implementation of Directed Diffusion is the same as TAG, explained in 3.4.1.3.

3.4.5 SIMULATION RESULTS

In order to test and verify the proposed sensor network architecture, several sensor network algorithms are simulated with this approach. The simulated algorithms are TAG, Synopsis Diffusion, Tributary-Delta Coarse, Tributary-Delta Fine and Directed Diffusion. The simulations are done using OMNeT++ [16] as explained in section 3.3. Detailed explanations of the simulated algorithms are given in section 3.4.

The first simulation compares TAG and SD algorithms according to their network contribution percentages for different error probabilities. This contribution percentage is measured as follows. Sink node sends a count query, and the other nodes reply this query by generating 1 and summing all the results. Finally, the result reaching to the sink node is divided to actual number of nodes

in the network. The simulation results are compared with Figure-7a of the study on Synopsis Diffusion by Nath et.al. [9]. In order to have a similar environment, the simulation parameters are adjusted according to [9]. Considering one pixel as one centimeter, the simulation parameters are selected as shown in the following table.

Table-1. Simulation parameters for node contribution rate comparison of TAG and SD

	Synopsis Diffusion Study [9]	**This Simulation**
Number of nodes	600	600
Simulation region	20ft x 20ft	610 x 610 pixels
SD Synopses	20 synopses, 32 bit	20 synopses, 32 bit
Transmission Range	(*)	69 pixels
Number of readings for every error level	500	1000

The link error probability is the probability for a node to drop an incoming packet, or link failure probability explained in 2.3.2.4. For every error probability, SD and TAG simulations are executed for 1000 times, and the average of all the readings is taken. The results of this simulation are shown in Figure-35 along with the simulation results of [9] for comparison. As seen in this figure, our simulations results are compatible with the original simulation results of SD and TAG.

(*) In [9], a realistic channel model which consists of 6 different error probabilities for different transmission ranges is defined. However, instead of this realistic model, they have used a simpler model for simulations, and the transmission range of this simple model is not specified. In this simulation, the average range of that realistic model (2.28ft = 69 cm) is used as the transmission range.

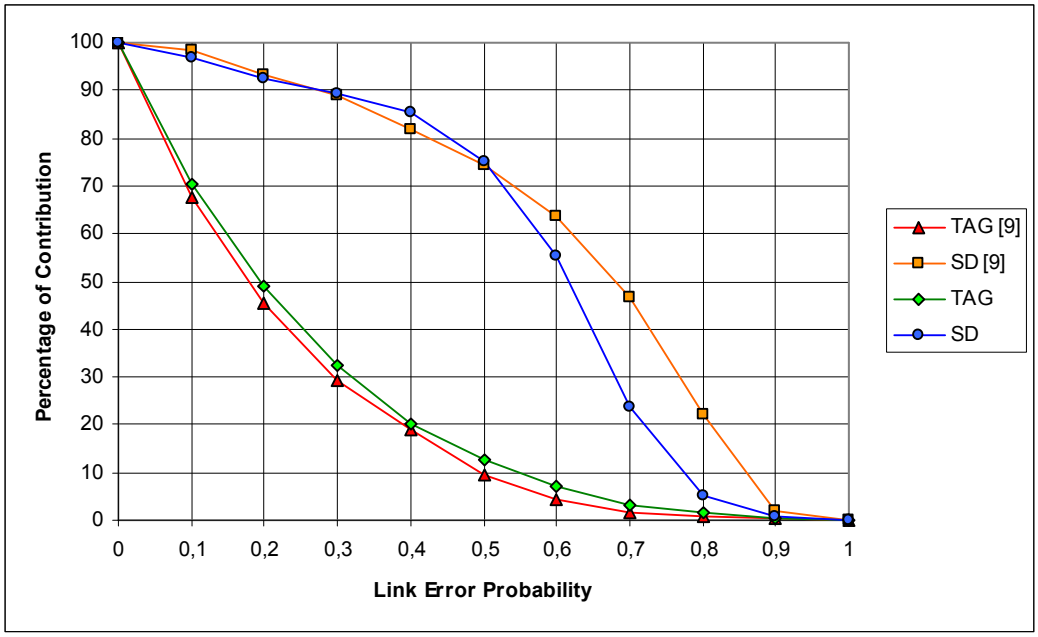

Figure-35 Node contribution rates of TAG and SD for different error probabilities

The second simulation is performed using the same simulation parameters as the previous case. This time, the RMS errors produced by TAG and SD are compared using the sum aggregate. The results are compared with Figure-7b in [9].

For both of the algorithms, every node generates a constant reading value of 20. The summation of these values, that is, sum aggregate is queried by the sink node. The RMS error is calculated using the following formula:

$$\frac{1}{V}\sqrt{\sum_{t=1}^{T}(V_t - V)^2 / T} \qquad (1)$$

In this formula, V is the actual value, V_t values are the individual epoch readings, and T is the total number of epochs, that is, 1000. In each epoch, nodes take a single reading and transmit it; hence, a single aggregate result is generated

from the sensor network. The results of this simulation and the results of obtained in [9] are shown in Figure-36. This figure also shows that our SD and TAG simulations match well with the results of [9].

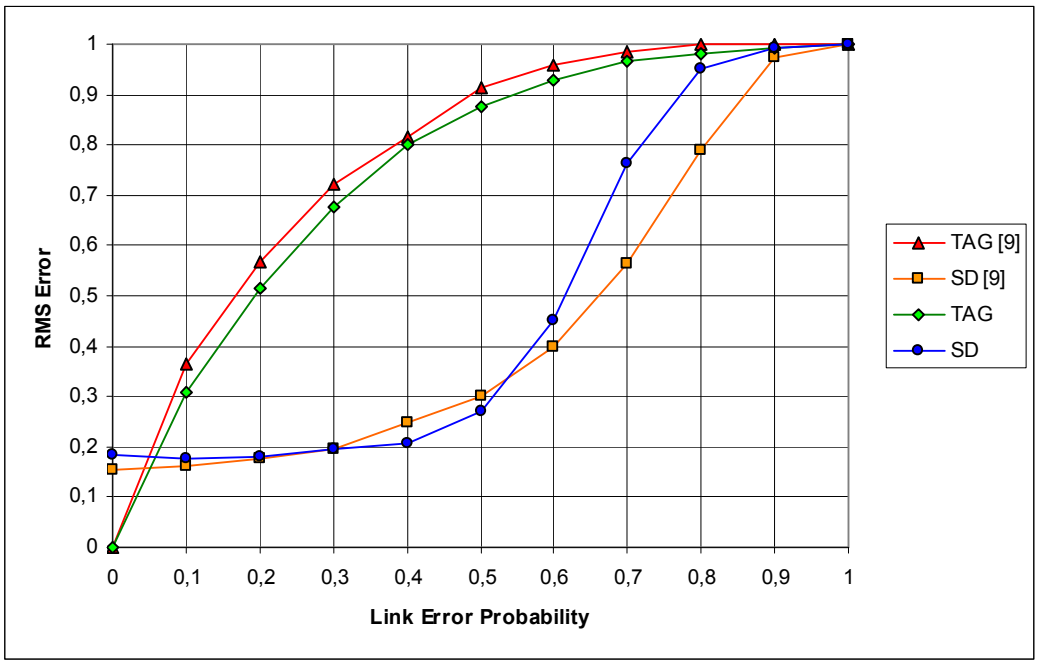

Figure-36 RMS error rate comparison of TAG and SD for different error probabilities

The next simulation performs RMS error comparison for TAG, SD and TD-Coarse algorithms. The aim is to achieve a graph similar to Figure-2 and Figure-5a of the study on Tributary-Delta by Manjhi et.al. [11], where TD-Coarse algorithm is applied for a global error scenario. The simulation parameters are matched to the simulation of [11] as follows:

Table-2. Simulation parameters for RMS Error comparison of TAG, SD and TD

	Tributary-Delta Study [11]	**This Simulation**
Number of nodes	600	600
Simulation region	20ft x 20ft	610 x 610 pixels
SD Synopses	40 synopses, 32 bit	40 synopses, 32 bit
Number of readings for every error level	100	1000

The simulation results of TAG and SD are shown in Figure-37, and TD-Coarse is shown in Figure-38. Both of these graphs include the results of [11].

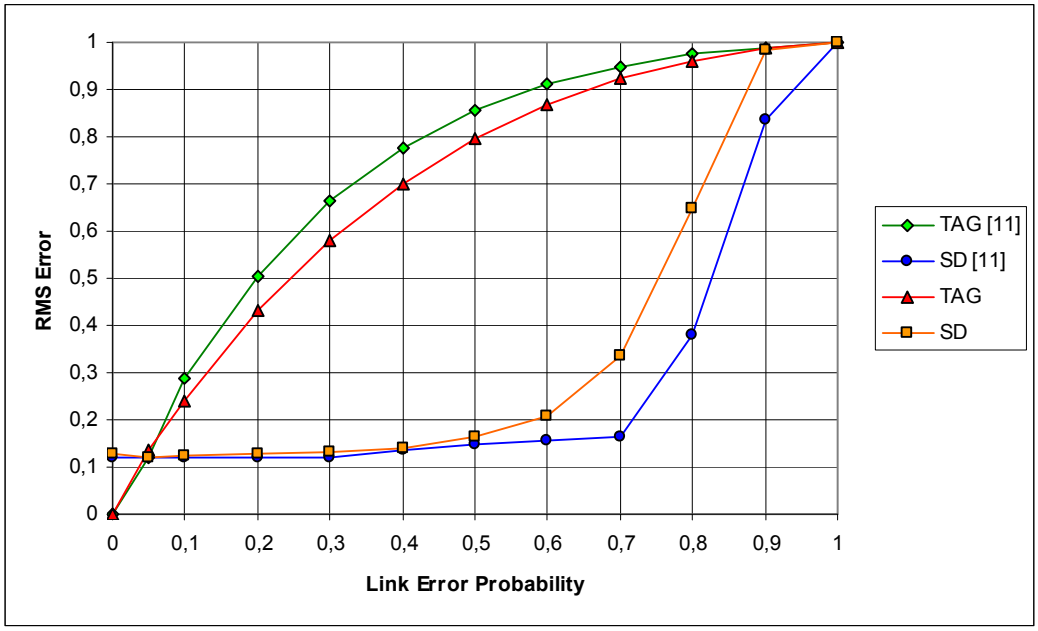

Figure-37 RMS error comparison for TAG and SD

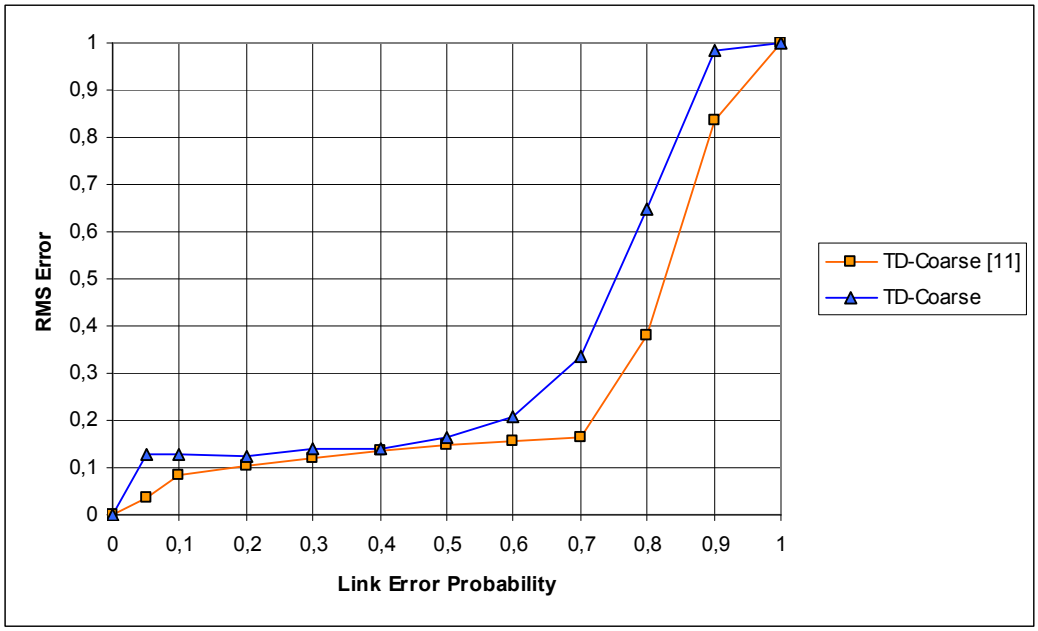

Figure-38 RMS error comparison for TD-Coarse

TAG, SD and TD-Coarse results in Figure-37 and Figure-38 are similar to their counterparts in [11]. TAG and SD were already verified in the previous simulation. The differences in the graphs are caused by using different random number generators and using different topologies. Since there is no specific random number generator defined for the simulations in [11], random number generators of C and OMNeT++ are used in this study. Moreover, the simulation topologies are not defined in [11] either, thus they are selected randomly for our simulations.

The last TD simulation compares the TD-Fine with SD for regional errors. The regional error is produced in the lower left quarter of the simulation region (0;0 – 305;305), while the other parts of the network are error free. The sink node is placed at the center (305;305). The other parameters are selected the same as in Table-2. As seen in Figure 39, TD-Fine shows better performance to SD for all regional error ranges, since TD-Fine switches only the erroneous region to SD, but error-free regions still work as TAG. This graph does not have a counterpart in Tributary-Delta study [11].

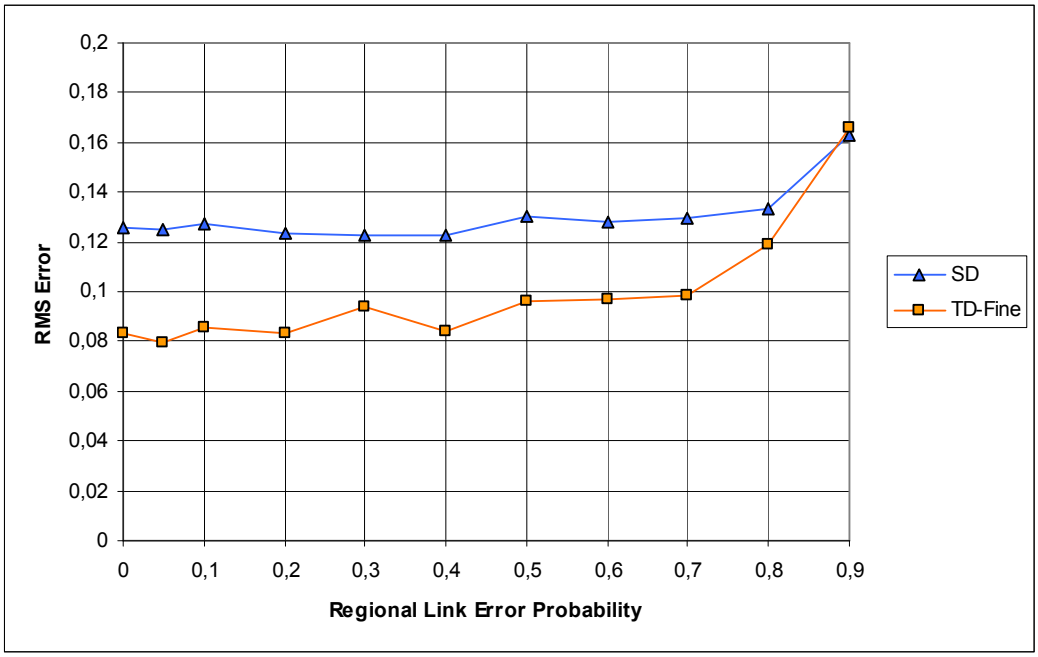

Figure-39 RMS error comparison for SD and TD-Fine for regional errors, no global loss probability

The last simulation is the Directed Diffusion implementation. In this simulation, the data delivery cost for all the nodes is calculated according to network size. Data delivery cost is calculated by summing all of the packets transmitted and received by the MAC layer for all the nodes in the network [4]. The simulation parameters are selected to match with the simulations of [4].

5 sinks and 5 sources are used in all of the simulations. The sinks are randomly selected inside the lower 100 pixel rectangle of the simulation region, and the sources are randomly selected from the top 100 pixel rectangle. The density of the nodes is kept constant at 3200 px^2/node for all network sizes. The exploratory query duration is for one epoch only, and it is distributed to the network through flooding. The first incoming data is reinforced and data coming from other nodes is discarded. The reinforced query duration is for 9 epochs. The results are obtained by taking the mean of 10 different simulations with different topologies and using a 95% confidence interval.

The first graph of this simulation shown in Figure-40 is drawn by omitting the initial query flooding and reinforcement phase, so the data delivery paths are already established. This graph shows that, as the network size increases, the cost of retrieving a query value per network size decreases exponentially. This is the expected result of directed diffusion, since, the network size increases geometrically while the path that delivers information from source to the sink increases linearly. This graph is consistent with the analytical results obtained in the Directed Diffusion study by Intanagonwiwat et.al. [4]. The y-axis values for data delivery cost are comparable but they don't have the same values as Figure-6c of [4]; since their results are analytical, they are using an ideal square-grid topology, and they consider data delivery cost per distinct data unit. Our simulation uses total data delivery cost in the network, moreover, nodes are distributed randomly.

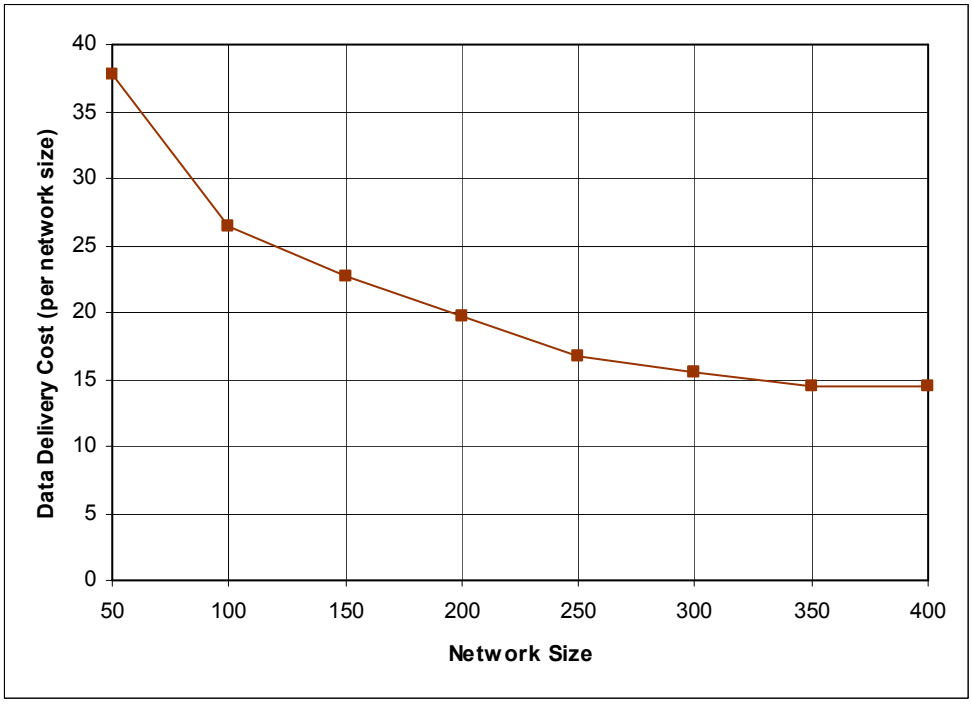

Figure-40 Directed Diffusion data delivery cost vs network size, without initial query flood

The graph in Figure-41 also shows the data delivery cost per network size; but it includes all of the cost generated by the simulation. This graph can be thought as a measure of average dissipated energy per node, since most of the energy is used for transmission and reception. In Directed Diffusion study [4], they performed simulations for average energy dissipation for different configurations. Since we do not apply suppression in the network, our results are consistent with the average dissipated energy without suppression graph shown in Figure 9-b of [4]. The simulations of [4] calculates the energy loss by using real power values for the antenna transmission, reception and idle states, so our y-axis values are different from their graph.

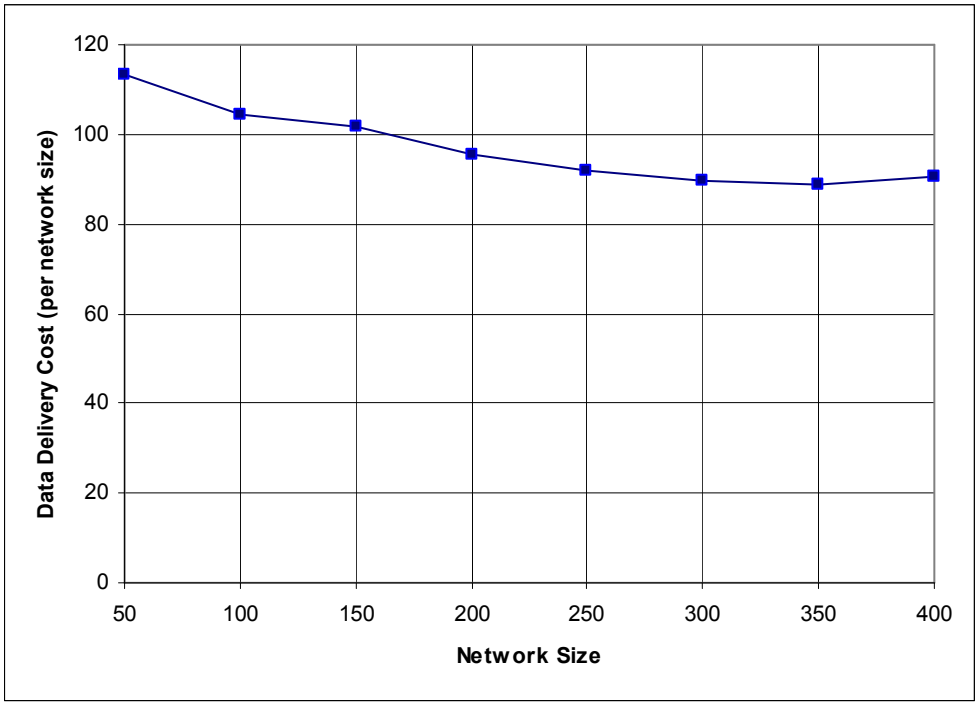

Figure-41 Directed Diffusion data delivery cost vs network size

CHAPTER IV

DISCUSSION AND CONCLUSION

In this thesis, a modular sensor network protocol stack is presented, defined and implemented using several sensor network algorithms. The implementations are realized using OMNeT++ simulation environment. Simulation results are compared with the original results of these algorithms.

The simulation results presented in Section 3.4.5 proves that the protocol stack suggested in this study is applicable to sensor networks. It is observed that the new layer stack proposed in this study has a similar performance to previous work and does not result in performance degradation. TAG and SD simulation results were similar to the results in [9] and TD result was similar to [11]. Directed Diffusion simulation results were as expected and similar to the results in [4].

As explained in Section 2.2, Akyıldız et.al. [1] and Kumar at.al. [8] have also proposed layered protocol stacks for sensor networks; yet applications of these stacks to real sensor network algorithms are not widely and generally available. In this thesis, along with proposing a new stack architecture for sensor networks, various sensor network algorithms are also implemented using the suggested stack. Furthermore, this architecture offers the following features; modularity, reusability and the ability to be used as a framework for sensor networks.

In the following section, these benefits are discussed in detail. In section 4.2, shortcomings and possible future work of this thesis are given.

4.1 BENEFITS OF THE PROPOSED ARCHITECTURE

The definition of modularity is the property of system modules being cohesive inside and less coupled with each other. Both the TCP/IP and OSI reference layered stack architectures aim to increase modularity by encapsulating related tasks in one layer. With the modularity of layers, upper level layers will be able to access lower-level functions transparently. Moreover, every layer can be supported by different vendors, and can be implemented in different hardware platforms. Modularity also leads to simplicity: every layer concentrates on its specific task and the complexities of lower layers are hidden from it.

The most important advancement of the stack proposed in this study is its modular architecture designed specifically for sensor networks. With this stack design, algorithms can be directly and easily divided into query and aggregation layers, without interfering with the tasks of application and network layers. These layers are coupled minimally; the required communication between query and aggregation layers is basically the notification of the incoming query downwards and the aggregate result upwards. They are also cohesive; such that query layer is concentrated with query optimization and distribution, while aggregation layer focuses on data collection and aggregation.

The previous stack suggestions in [1] and [8] do not define the implementation details for a particular algorithm. Consequently, the comparison of our stack with the previous stacks is based on the implementations we performed in this thesis. In the following paragraphs, TCP/IP, other stack proposals and our approach are compared briefly.

If the traditional TCP/IP stack is to be used for the data-centric algorithms implemented in this thesis, the actual algorithm designs will have to be placed inside the application layer. On top of its normal assignment,

implementing the whole algorithm in one layer would make the design more complex and less modular. Moreover, transport layer is not required for these algorithms, thus this layer would become essentially null.

The stack proposed by Akyıldız et.al. [1] has three vertical planes defined over the TCP/IP stack. These planes are *power*, *mobility* and *task management* planes. The simulated algorithms do not require mobility management, since they are oriented for stationary operations. The power management is intertwined with the query and data collection algorithms, such as sleep-wake up schedules of TAG and SD, or path enforcement of Directed Diffusion. Thus, it is not easy to separate these procedures to place inside another layer. The task management plane schedules and balances tasks for a specific region. Scheduling algorithm could be applied to TAG; however, it would again be difficult to separate from the original algorithm.

The stack suggested by Kumar et.al. [8] has a data fusion layer and it can be used for the aggregation tasks of TAG, SD and TD algorithms. Nevertheless, query and data collection operations would again have to be performed by the application layer, just like TCP/IP. Moreover, algorithms that do not make data fusion, e.g. Directed Diffusion, would always work in the application layer.

On the other hand, if our protocol stack is used, application and network layers would continue to perform their usual tasks, and for all algorithms, the core sensor network operations will be performed in query and aggregation layers. With the help of these layers, the implementation of the algorithms would be modular. In comparison to the structures in [1] and [8], our stack seems to provide simplicity of implementation.

In Table-3, the sensor network algorithms simulated in this thesis are classified according to the operations performed in the different layers of the proposed stack structure. By examining the implementation details for these algorithms, the modularity of this stack architecture can be observed.

Table-3. Algorithm comparison based on the operations performed in the different layers of the proposed stack

	TAG	**SD**	**TD-Coarse**	**TD-Fine**	**Directed Diffusion**
Query Layer	**Sink Node:** • Prepare query packet • Broadcast query • Receive and evaluate query results **Other Nodes:** • Receive query • Update query info • Inform aggregation layer • Rebroadcast query	Same as TAG	TAG and: **Sink Node:** • Send count query • Receive and evaluate count response • If required, broadcast a TD adaptation message to the network	TAG and: **Sink Node:** • Send count query **Nodes at TD Boundary:** • Receive and evaluate the count response • If required, send TD adaptation message to child nodes	TAG and: **Sink Node:** • Reinforce first arriving response by sending an updated query along that path
Aggregation Layer	• Receive query information from query layer • Wait for own epoch • Read own sensor data • Receive responses from child nodes • Aggregate all incoming and own reading • Send aggregated value to parent node	• Receive query information • Wait for own epoch • Read own sensor data • Generate synopsis • Receive and merge incoming responses from the outer ring levels • Send resulting synopsis to the inner ring level	• If the node is in Tributary section, act similar to TAG • If the node is in Delta section, act similar to SD • If the node is at the boundary, • Collect and aggregate data as TAG • Generate synopsis and send to inner ring	Same as TD-Coarse	• Receive query information • If data is available, send own data to the previous hop along the query path • Send the data of other nodes to previous hop along the query path
Network Layer	• Send a network packet to a destined node using node address • Filter incoming packets according to own address • Interface with MAC	Same as TAG	Same as TAG	Same as TAG	Same as TAG

Another advantage of this stack is reusability. It is possible that different algorithms use the same layer design, while altering other layers slightly or completely. As seen in Table-3, TAG and SD algorithms can use the same query layer, and they have different aggregation function implementations in aggregation layer. TD algorithm combines the aggregation layers of TAG and SD, and it adds adaptation logic into their query layer. Directed Diffusion also extends the query layer of TAG by including a query enforcement process.

In addition to reusability, this modular sensor stack leads to simplified refactoring. Since the algorithm is divided into separate functional parts, it will be easier to refactor the design of a single layer, and implement it transparently without disturbing other layers.

The last benefit of this architecture is that it can be used as a framework for sensor network algorithms. This framework will be useful in classifying and comparing different algorithms based on the operations performed on different layers. It will be possible to point out differences and similarities of sensor network algorithms; understanding and analyzing an algorithm will also be less complicated.

Table-3 shows that our stack architecture is quite convenient to classify and compare algorithms. For these implementations; network layers of all algorithms, query layers of TAG and SD, and aggregation layers of TD-Coarse and TD-Fine are completely the same. The difference between TAG and SD is at the aggregation layer, the way they process and send data. The difference of TD-Coarse and TD-Fine is at the query layer, the first one makes network adaptations initiated from the sink node, while the second one makes adaptations initiated from the nodes at the TD-boundary. Aggregation layers of TD algorithms combine TAG and SD aggregation layers, and include a data merging procedure for the nodes at the boundary. Directed Diffusion has extensions to the query layer of TAG; it contains procedures to enforce the minimum delay data path. Contrary to TAG, aggregation layer of Directed Diffusion does not wait to send data, or make data aggregation; it just sends its own data, or routes received data to previous query node.

4.2 SHORTCOMINGS AND FUTURE WORK

The other major stack structures proposed by Akyıldız et.al. [1] and Kumar et.al. [8] have not been used in implementing the algorithms considered within the scope of this study. This is because, it has not been possible to access specific implementation details based on those structures in the literature. Therefore, in this study, it has not been possible to claim that the structure proposed here is in any way superior to those others. It remains as a future work to examine whether different structures lead to concrete advantages over one another in terms of efficiency and effectiveness.

The layered stack structure explained in this thesis is the basis for a generalized, modular and applicable sensor network protocol architecture. In order to improve this structure and to strengthen the claim of being a generalized stack for sensor networks, more sensor network systems should be implemented and tested with this protocol stack. Likewise, the interface definitions of the layers are specified considering the simulated algorithms. These interfaces should be improved by generalizing them and making them applicable to any sensor network system.

This study has concentrated mostly on query and aggregation layer operations. The simulations did not consist of complex routing tasks for network layer. As a future work, sensor network systems that require intense routing operations should be investigated. In this sense, semantic routing trees of TinyDB [3] explained in Section 2.3.2.1 could be implemented using the network layer more effectively. Another subject could be to examine and implement query informed routing using the network and query layers in conjunction.

Query layer and aggregation layer keep the addresses of one hop neighbors, since they are directly related to query and data response tasks. It should be examined whether keeping the addresses inside network layer and feeding the data using query and aggregation layers will improve modularity or not.

REFERENCES

[1] I. F. Akyıldız, M.C. Vuran, Ö.B. Akan, W. Su, "Wireless Sensor Networks: A Survey Revisited", to Appear in Computer Networks (Elsevier) Journal, 2006.

[2] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "TAG: A Tiny Aggregation Service for Ad-hoc Sensor Networks", ACM SIGOPS Operating Systems Review, Vol. 36, No. SI, pp. 131–146, 2002.

[3] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "An Aquisitional Query Processing System for Sensor Networks", ACM Transactions on Database Systems, Vol. 30, No. 1, pp. 122–173, March 2005.

[4] C. Intanagonwiwat, R. Govindan, D. Estrin and J. Heidemann, "Directed Diffusion for Wireless Sensor Networking", IEEE/ACM Transactions on Networking, Vol. 11, No. 1, pp. 2–16, February 2003.

[5] N. Sadagopan, B. Krishnamachari and A. Helmy, "Active Query Forwarding in Sensor Networks", Computer Networks (Elsevier) Journal, Vol. 3, No. 1, pp. 91–113, January 2005.

[6] Y. Yao and J. Gehrke, "Query Processing in Sensor Networks", Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR), pp. 233–244, January 2003.

[7] O. Gnawali, R. Govindan and J. Heidemann, “Implementing a Sensor Database System Using a Generic Data Dissemination Mechanism”, IEEE Data Engineering Bulletin, Vol. 28, No. 1, pp. 70–75, March, 2005.

[8] R. Kumar, S. PalChaudhuri, D.Johnson and U. Ramachandran, “Network Stack Architecture for Future Sensors”, Rice Computer Science Technical Report, TR-04-447, 2004.

[9] S. Nath, H.Yu, P. B. Gibbons, S. Seshan, “Synopsis Diffusion for Robust Aggregation in Sensor Networks”, Intel Research Technical Report, IRP-TR-04-13, April 2004.

[10] J. Considine, F. Lee, G. Kollios, and J. Byers, “Approximate Aggregation Techniques for Sensor Databases”, Proceedings of the 20th International Conference on Data Engineering (ICDE), p. 449, 2004.

[11] A. Manjhi, S. Nath an P. B. Gibbons, “Tributaries and Deltas: Efficient and Robust Aggregation in Sensor Network Streams”, Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data, pp. 287–298, June 2005

[12] P. Flajolet and G. N. Martin, “Probabilistic Counting Algorithms for Database Applications”, Journal of Computer and System Sciences, Vol. 31, pp. 182–209, 1985.

[13] S. Vardhan, M. Wilczynski, G. Pottie, and W. J. Kaiser, “Wireless Integrated Network Sensors (WINS): Distributed in Situ Sensing for Mission and Flight Systems,” IEEE Aerospace Conference, Vol. 7, pp. 459–463, March 2000.

[14] B. Warneke, B. Liebowitz, and K. S. J. Pister, “Smart Dust: Communicating with a Cubic-Millimeter Computer,” IEEE Computer Magazine, Vol.34, No.1, pp. 44–51, January 2001.

[15] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J.Pister, “System Architecture Directions for Networked Sensors”, Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 93–104, November 2000.

[16] OMNeT++ Discrete Event Simulator, http://www.omnetpp.org (last access: August 2006).

[17] A. Varga, “OMNeT++ 3.2 User Manual”, http://www.omnetpp.org/doc/manual/usman.html (last access: August 2006).

[18] INET Framework for OMNeT++, http://www.ctieware.eng.monash.edu.au/twiki/bin/view/Simulation (last access: August 2006).

[19] A. Varga, “INET Framework Documentation”, http://www.omnetpp.org /doc/INET (last access: August 2006).

[20] B.Krishnamachari, D.Estrin and S.Wicker, “The Impact of Data Aggregation in Wireless Sensor Networks”, Proceedings of the 22nd International Conference on Distributed Computing System Workshops, pp. 575–578, July 2002.

[21] I. F. Akyıldız, W. Su, Y. Sankarasubramaniam, E. Cayirci,"Wireless Sensor Networks: A Survey", Computer Networks (Elsevier) Journal, Vol. 38, No. 4, pp. 393–422, March 2002.

[22] D. Estrin, L. Girod, G. Pottie, M. Srivastava, “Instrumenting the World with Wireless Sensor Networks”, Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP 2001), Vol. 4, pp. 20–33, May 2001

APPENDIX

SOURCE CODES AND EXECUTABLE FILES OF THE SIMULATIONS

The source codes and executable files of the simulations performed in this study are given in the CD attached at the back cover of this thesis.