Dynamic network layer for data query in sensor network

Zusheng Zhang¹², Liang Chen², Zhiqi Wang² and Fengqi Yu²

¹Dongguan University of Technology, No. 1 University Road, Songshan Lake Science & Technology Industry Park, Dongguan, Guangdong, China
²Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences/The Chinese University of Hong Kong, 1068 Xueyuan Avenue, Shenzhen University Town, Nanshan District, Shenzhen, China

ABSTRACT

Query processing systems in wireless sensor networks usually support tasks such as data acquisition, data aggregation, and event-based query. The performances of query processing with these tasks are greatly varied according to different routing protocols. Most existing data query systems usually use one routing protocol to deal with all kinds of queries. This work demonstrates that proper selection of routing protocols can improve the performance of query processing. We propose a dynamic routing layer that makes protocol selection on the basis of query tasks and can automatically switch between different routing protocols. Simulation results show that dynamic routing scheme is more energy efficient than single routing protocol.

Copyright © 2012 John Wiley & Sons, Ltd.

Received 1 April 2012; Revised 26 August 2012; Accepted 28 October 2012

KEY WORDS: wireless sensor network; data query; dynamic routing

1. INTRODUCTION

The data query processing systems such as TinyDB 1, Cougar 2, and SwissQM 3 are promising for wireless sensor networks. With these systems, a user can inject Structured Query Language (SQL) style queries into a network through a PC. The networked sensor nodes then work together to process the queries and to send results back to the PC. The performance of these sensor systems is greatly affected by routing protocol, because routing protocol is responsible for the query dissemination and result gathering. Some query processing tasks 4 and their matched routing protocols are presented as follows.

Query dissemination: User injects query commands to base station (BS), and then BS disseminates commands to the whole network. Some dissemination protocols are proposed for sensor network, such as Gossip-based Scalable Directed Diffusion (GSDD) 5 and Dissemination Protocol (DIP) 6. They are aimed to reliably disseminate new programs or commands for every node in the network.

Data acquisition: A node does not aggregate data packets from neighbors, and all data packets are to be forwarded to BS through multi-hop. Tree-based routing protocols are energy efficient for acquisition. MintRoute 7, Collection Tree Protocol (CTP) 8, and other collection protocols 9 build minimum-cost trees to BS to pull data out of a network.

*Correspondence to: Zusheng Zhang, Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences/The Chinese University of Hong Kong, 1068 Xueyuan Avenue, Shenzhen University Town, Nanshan District, Shenzhen, China.
†E-mail: zushengzhang@gmail.com

Copyright © 2012 John Wiley & Sons, Ltd.
Data aggregation: A node aggregates its sensed data and data packets received from neighbors into one packet. Then it is forwarded to the next hop. The clustering technique is effective in achieving scalability and load balance, and it is energy efficient for data aggregation. Many clustering protocols have been proposed for sensor networks, such as Low Energy Adaptive Clustering Hierarchy (LEACH) 10, Hybrid Energy-Efficient Distributed (HEED) 11, and Enhanced Power-Controlled Routing (EPCR) 12–14.

Each routing protocol in wireless sensor network is optimum for special task and network condition. However, existing data query systems only use one routing protocol to deal with all kinds of queries, such as TinyDB and Cougar, where a fixed tree-based network topology is assumed 15. So we propose the dynamic routing scheme for query processing, where multiple routing protocols coexist. These protocols can be switched according to query tasks. The dynamic routing scheme has the following attractive features.

- We adopt an overall modular design of the network layer. Unlike traditional monolithic way, we decompose the network layer into shared modules and private modules. The design increases code reuse and run-time sharing. Several routing protocols coexist in a node without burdensome memory requirement.
- The dynamic routing layer provides a unifying interface for Application and medium access control (MAC) layers. Small changes of the code inside of modules of routing protocols are required, whereas the code at other layers does not need not to be changed.
- BS selects the routing protocol according to query tasks. The protocol switching information is attached to the query, which is disseminated to the whole network. The protocol switching does not need additional energy consumption for control messages.

The remainder of this paper is organized as follows. Section 2 provides modular design of the network layer. Section 3 proposes the design of dynamic routing layer. Section 4 gives an example of implementation. Section 5 presents the simulation results. Section 6 describes the related works. Finally, Section 7 gives a brief conclusion.

2. MODULAR DESIGN

The routing protocols in sensor network are usually designed for special task and network condition, for example, MintRoute 7, CTP 8, LEACH 10, and HEED 11. Across these designs, there is a general lack of consistency in terms of the functionalities implemented in the modules as well as their interfaces, resulting in unnecessary coupling between modules. Coexisting protocols with overlapping functionalities unnecessarily consume more resources in terms of memory and energy, and therefore reduce the lifetime of a sensor network.

An overall modular design can increase code reuse and run-time sharing. Code reuse will foster more rapid protocol and application development. Run-time sharing refers to the sharing of code and resources such as memory and radio, and will allow several protocols to coexist. On the basis of [8,16], we decompose the network layer into separate four parts: forwarding engine (FE), message pool (MP), routing engine (RE), and neighbor management (NM), as shown in Figure 1. Upon the arrival of a packet from Application or MAC layer, the FE module needs to obtain the next hop for packet sending or forwarding. If multiple packets need to be forwarded at the same time, the packets have to be scheduled. The MP module implements buffer management and packet scheduling. The RE module is the main body of a routing protocol, which is responsible for discovering and maintaining the network topologies. The NM module is responsible for neighbors’ management. The MP and NM are shared modules for each routing protocol, whereas the FE and RE modules are private because different routing protocols have different algorithms being implemented in these modules.
2.1. **Forwarding engine**

The main function of an FE module is to obtain the next hop to which the packet is to be forwarded. This is achieved by having the FE query the corresponding RE through an interface. Subsequently, FE sends the packet to the MP. Additional functions include cycle detection, network level retransmission, and duplicate packet elimination.

2.1.1. **Cycle detection.** For many-to-one routing over a relatively stationary sensor network, the routing protocol usually uses simple mechanisms to avoid loop formation and to break cycles when they are detected. There are many cycle detection algorithms proposed for wireless communication. For example, cycles can be detected when a node in a loop originates a packet and sees its return. Once a cycle is detected, the FE module signals the corresponding RE module to change the route to a new one.

2.1.2. **Duplicate packet.** Duplicate packets can be created upon retransmission when the ACK is lost. Without duplicate packet elimination, a duplicate packet will be forwarded and possibly causes more retransmissions and more contention, leading to energy waste. One way to avoid duplicate packet is that the sender ID and a sequence number are appended in the routing header at originating node. Each receiver retains the most recent originator’s ID and sequence number in the neighbor table. When a packet is received, the node detects if it is duplicated by comparing the sequence number.

2.2. **Routing engine**

The RE is responsible for creating and maintaining basic communication abstractions. The RE exchanges traffic control message with its neighbor REs to determine and maintain network topology. Examples of communication topologies are trees, geographic coordinates, or clusters. RE includes algorithms such as link estimation, and route selection.

2.1.1. **Link estimation.** Routing protocols use link estimation to build routing structures. The stability and agility of link estimation can directly affect the stability of a route and the rate of route adaptation, especially when the estimations are combined to form a distance metric describing a path. In MintRoute 7 and CTP 8, individual nodes estimate link quality by observing packet success and loss events. The implementation of CTP 8 uses two mechanisms to estimate the quality of a link: periodic beacon packets and data packets. For data packets, it is a direct measure of expected transmission count (ETX) 22. Whenever the node transmits a data packet, it tells the link estimator the destination and whether it was successfully acknowledged. The estimator produces an ETX estimate every five such transmissions.
2.1.2. Route selection. Using different cost metrics, many algorithms are implemented for route selection. The cost of delivering a packet to destination can be the number of hops, expected number of transmissions, or some other estimate of energy required to reach the BS. The routing algorithm accesses the neighbor table and determines its next hop with the least cost. A node may switch to a new route if the cost of the new route is sufficiently smaller than that of the current route. It may also switch to a new route if the link quality of the current route drops below a threshold or if a cycle is detected.

2.3. Neighbor management

Routing protocols in wireless sensor network usually use neighbor table instead of routing table, and routing selection is based on neighbor table information. An entry in the neighbor table includes information of the address of the neighbor, link quality, next hop (or parent), and so on. Neighbor tables of MintRoute 7 and HEED 11 are shown in Figure 2. Both of them have many same attributes in table entry. In our design, multiple routing protocols share a neighbor table. For flexibility, the table is extensible, that is, each routing protocol can add itself specific attributes into the table. The maximum size of neighbor table usually is less than 20. For example, in the TinyOS system, the table size of MintRoute is 16. So a sensor node has enough memory space to store neighbor table for multiple routing protocols that coexist.

The NM module manages information about node’s neighbors, which include the neighbor table and table’s management functions. As sensor networks mature and multiple network protocols coexist, it becomes increasingly attractive to share the NM module among various network protocols, rather than require each of them to maintain its own table.

2.4. Message pool

All outgoing packets must be sent to an MP module, which acts like a data buffer. It supports packet scheduling for all network protocols. A basic MP module may implement a simple priority scheduling, which provides simple priority scheduling and queue management functionality. The module transmits a packet of high priority before any packet of lower priority. The determination of a packet’s priority is dependent on the network protocol. For instance, some protocols may require their control packets to be sent as soon as possible and preferably not dropped when the queue is full, for example, for the packets with high priority.

```c
typedef struct TableEntry
{
    uint16_t id;  // Node Address
    uint16_t parent;
    uint16_t missed;
    uint16_t received;
    int16_t lastSeqno;
    uint8_t flags;
    uint8_t liveliness;
    uint8_t hop;
    uint8_t receiveEst;
    uint8_t sendEst;
    #ifdef TREE
    uint16_t cost;
    uint8_t childLiveliness;
    #endif
} MintRoute_Table;

typedef struct TableEntry
{
    uint16_t id;  // Node Address
    uint16_t parent;
    uint16_t missed;
    uint16_t received;
    int16_t lastSeqno;
    uint8_t flags;
    uint8_t liveliness;
    uint8_t hop;
    uint8_t receiveEst;
    uint8_t sendEst;
    #ifdef CLUSTERING_ON
    uint8_t CH_type;
    uint16_t my_CH;
    uint16_t my_CH_Cost;
    #endif
} HEED_Table;
```

Figure 2. Neighbor tables of MintRoute and HEED.
3. DYNAMIC ROUTING DESIGN

Traditionally, in wireless sensor network, each routing protocol is designed, developed, and evaluated separately, and the network layer uses a single routing protocol to deal with all kinds of queries. However, a routing protocol is usually optimal for special task and network condition. We propose a dynamic routing scheme, letting the network layer coexist with several kinds of routing protocols, and these protocols can be switched according to query task in the run time.

3.1. Protocol selection

Routing protocols in wireless sensor network usually are responsible for establishing paths from sensors to BS (data concentrator). For initialization, only BS is in the network, and it has a default routing protocol. BS starts the process of routing construction. It broadcasts request messages to nodes in the whole network. Upon receipt of the route request messages, and through exchange of beacon messages between neighbor nodes, every node can find an optimum route to the BS. The detail process of routing construction can refer to CTP 8.

We adopt a centralized control mode for routing protocol selection. BS maintains a protocol selection table, as shown in Table I. User defines the table entries, that is, routing switching criteria, according to the overall knowledge of the network performance by theoretical analysis, simulation, and experiment. The procedural at BS is as follows.

(a) Upon receiving a query from a user, BS matches the query command to the corresponding task. For SQL language, BS checks the query task according to SQL keywords. If the query contains keywords such as ‘sum’, ‘average’, and ‘total’, it is an aggregation query. If the keyword is ‘WHEN’, it is an event-based query. On the other hand, it is an acquisition query.

(b) BS selects routing protocol according to query task. Then it attaches the routing information on the query command, such as ‘select temperature from sensors using MintRoute’, BS disseminates the query to the whole network and informs the network switching to MintRoute for the query.

3.2. Protocol switching

There are two design principles of the dynamic routing layer: (i) it provides unifying extern interfaces. The Application and MAC layers access the routing layer in a transparent way. (ii) Small changes of the code inside the modules of routing protocols are required to achieve compatibility in the routing layer.

The modular design of the dynamic network layer is shown in Figure 3. Because FE and RE are private modules of each routing protocol, multiple instances of FE and REs simultaneously exist on the single node. So correctly switching between FE and RE modules is the key issue of the dynamic routing scheme. We add three modules: HighDispatcher, LowDispatcher, and TopoControl in the dynamic network layer for routing protocol switching.

The TopoControl module controls the routing topology switching. It maintains a protocol table with an entry for each routing protocol running on the node. When a node receives a query, it decodes the routing protocol identifier (RoutID), which is carried by the query command and starts the corresponding application program (different queries have different application programs). The Application Layer informs the TopoControl model which routing protocol should be selected.

<table>
<thead>
<tr>
<th>Key words</th>
<th>Task</th>
<th>Routing protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUM, AVE, MAX, MIN</td>
<td>Aggregation</td>
<td>HEED</td>
</tr>
<tr>
<td>WHEN</td>
<td>Event</td>
<td>CTP</td>
</tr>
<tr>
<td>Others</td>
<td>Acquisition</td>
<td>MintRoute</td>
</tr>
</tbody>
</table>

Table I. Policies of protocol selection.
according to the RoutID. TopoControl module matches queries with REs. When the query task starts, TopoControl module triggers the RoutID’s RE module. When the query task finishes, TopoControl stops the RoutID’s RE module.

3.3. Data packet processing

The HighDispatcher and LowDispatcher modules hide the differences of routing protocols for Application and MAC layers. As shown in Figure 3, the dynamic network layer is responsible for the switching between FEs by the HighDispatcher and LowDispatcher modules. The network header of a data packet is shown in Figure 4. The dynamic scheme adds two fields: Low header and High header. The Low header is added and explained by the HighDispatcher module. The HighDispatcher module provides a bridge for data packets correctly flowing between application programs and FEs. The LowDispatcher manages the Low header field in the packet header, and it provides a bridge for data packets correctly flowing between MAC Layer and FEs. As shown in Figure 3, the numbered cycles show the procedure of a source node sending a data packet:

1. The application layer senses the environment and fills the Payload field in the packet header, and then it sends the packet to HighDispatcher with two parameters: RoutID and application identifier (AppID).
2. HighDispatcher fills the AppID field in the High header field of the packet header, as shown in Figure 4, and it sends the packet to the FE according to the RoutID.
3. FE obtains the next hop by querying the RE and fills the Routing header field in the packet.
4. Then FE sends the packet to the LowDispatcher module.
5. LowDispatcher fills the RoutID field in the Low header and sends it to the MsgPool.
6. The data packet is scheduled for transmission to MAC layer.

When a router receives a data packet from the network, the data flow process is shown in Figure 5:

![Diagram](image-url)

Figure 3. The dynamic network layer decomposition.

![Diagram](image-url)

Figure 4. Network packet header format.
The MAC layer signals the LowDispatcher to receive a data packet.

The LowDispatcher explains the RoutID field in the Low header field of packet header and informs the corresponding FE to receive the packet by RoutID.

An FE deals with the data packet and signals the HighDispatcher to receive the packet.

The HighDispatcher explains the AppID field in the High header field of packet header and checks whether an application has registered to intercept the packet. In the case of data aggregation, the received packet must be aggregated with its own packets of the node. Therefore, the packet is handled by the corresponding application program. In the case of data collection, unless the node is BS, the application layer does not register to intercept data packet. So the HighDispatcher module signals the FE to forward the data packet to next hop.

FE obtains the next hop by querying the RE.

FE forwards the packet to LowDispatcher.

LowDispatcher sends the packet to MsgPool. The data packet is scheduled for transmission to MAC layer.

4. IMPLEMENTATION

TinyOS is selected to implement our proposed dynamic scheme. Two routing protocols, MintRoute 7 and HEED 11, are used for our implementation. The system architecture for each sensor node is shown in Figure 6. In this architecture, the Energy Manager module keeps track of a node’s energy. The MAC is B-MAC protocol 17, which has a user-friendly interface that allows users to have flexibility for sleep interval control on the basis of workload. We can adaptively select a scheduling model according to the queries’ workload to reserve energy consumption. For example, if the query is to collect temperature from sensors at the period of 5 min, we can use mode 4 of B-MAC, and each node wakes up every 100 ms to detect channel activity. If the query is to collect temperature from sensors at the period of 5 s, we can use mode 1 of B-MAC, and each node wakes up every 10 ms to detect a channel activity. The routing layer provides routing services to the application layer and supports dynamically switching between two routing protocols: HEED and MintRoute. We take data query, which currently consists of acquisition and aggregation, as an example to discuss our implementation.
4.1. Topology switching

In our previous work 18, we compared MintRoute and HEED routing protocols according to different tasks and network conditions and have the following guidelines:

(a) For aggregation, HEED is 40–50% more energy efficient than MintRoute.
(b) For acquisition, MintRoute is 20–30% more energy efficient than HEED.

On the basis of the guidelines, we have the following dynamic schemes. Upon receiving a query from a user, BS selects routing protocol according to query task: for aggregation, BS chooses HEED as routing protocol; for acquisition, it chooses MintRoute. BS attaches the routing protocol information on the query command and sends it out using flooding algorithm.

A node receives a query and decodes it. If the query contains routing switch information, the node executes the switch. Figure 7 describes the module design of REs. The data acquisition and data aggregation modules inform the TopoControl module when to start and stop the corresponding REs according to the RoutID.

In HEED, member nodes communicate with their cluster heads over a single hop, as shown in Figure 8(a), and cluster heads communicate with BS over multi-hop, as shown in Figure 8(b). HEED adopts MintRoute to construct a routing tree among cluster heads. In our design, HEED and MintRoute share tree construction algorithm and neighbor table, as shown in Figure 7. The module of TopoControl is a switch between Clustering module and TreeConstruction module on the basis of application. Using the module design, we can obtain two kinds of routing topologies.
The first one is MintRoute topology, as shown in Figure 8(c). It can be obtained by stopping the Clustering module and starting the TreeConstruction module.

The second one is HEED topology, which is shown in Figure 8(b), where both Clustering module and TreeConstruction module are needed. When a member node joins a cluster, the Clustering module sets its cluster head as its next hop. The TreeConstruction only selects parent nodes for cluster heads. If a node is a member node, it does not execute the TreeConstruction module.

4.2. Code size

One of the main objectives of creating a dynamic network layer for sensor networks is to increase code reuse, thereby allowing multiple protocols to coexist efficiently on a single node. Compared with the monolithic implementations, Figure 9 compares code size among different implementations. The Modular is the implementation of MintRoute and HEED with code sharing, and the Dynamic refers to a dynamic network layer based on coexisting offer switching between MintRoute and HEED. Figure 9 shows that the Dynamic can save up 40% code compared with the code summation of MintRoute and HEED.

5. SIMULATION

We adopt the first-order radio model given in 10 for our simulation. The energy consumption for sending $l$-bit data with distance $d$ is $E_{Tx}$, and energy consumption for receiving $l$-bit data is $E_{Rx}$. The energy model can be expressed as follows:

\[ E_{\text{Rx}}(l) = lE_{\text{elec}} \]  

\[ E_{\text{Tx}}(l, d) = \begin{cases} 
    lE_{\text{elec}} + l\xi_\text{fs}d^2, & d < d_0 \\
    lE_{\text{elec}} + l\xi_\text{mp}d^4, & d > d_0
  \end{cases} \]  

where \( E_{\text{elec}} \) is the circuitry power consumption, which depends on digital coding, modulation, filtering, and so on. \( \xi_\text{fs}d^2 \) is the transmission power, which depends on the distance to the receiver and acceptable bit-error rate.

Dynamic routing scheme, HEED, and MintRoute for data query are simulated for 100 nodes randomly distributed in a 100 m \( \times \) 100 m area, node communication range is 50 m, the cluster radius is 25 m, and the energy consumption for aggregation is \( E_{\text{fusion}} = 5 \) nJ/bit. Simulation parameters are shown in Table II.

\[ \text{Table II. Simulation parameters.} \]

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Network size</td>
<td>100 m ( \times ) 100 m</td>
</tr>
<tr>
<td>BS position</td>
<td>(0 m, 0 m)</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>100</td>
</tr>
<tr>
<td>Communication radius</td>
<td>50 m</td>
</tr>
<tr>
<td>Cluster radius</td>
<td>25 m</td>
</tr>
<tr>
<td>Data packet size</td>
<td>500 bytes</td>
</tr>
<tr>
<td>Packet header size</td>
<td>25 bytes</td>
</tr>
<tr>
<td>Initial energy</td>
<td>2 J</td>
</tr>
<tr>
<td>( E_{\text{elec}} )</td>
<td>50 nJ/bit</td>
</tr>
<tr>
<td>( \xi_\text{fs} )</td>
<td>10 pJ/bit</td>
</tr>
<tr>
<td>( d_0 )</td>
<td>75 m</td>
</tr>
<tr>
<td>( E_{\text{fusion}} )</td>
<td>5 nJ/bit</td>
</tr>
<tr>
<td>( \xi_\text{mp} )</td>
<td>0.0013 pJ/bit</td>
</tr>
</tbody>
</table>

5.1. Single query

We simulate HEED, MintRoute, and Dynamic routing in the case of aggregation, respectively. For HEED, cluster members send data packets to cluster heads, and cluster heads aggregate these data packets then send the fused packet to next hop. For MintRoute, child nodes send data to parents, and parents process the aggregation. In the case of aggregation, dynamic routing scheme chooses HEED as routing protocol. The total energy dissipation per data packet received at BS is illustrated in Figure 10. Using HEED as routing protocol delivers 50% more data per unit energy than that
using MintRoute. Figure 11 shows the number of nodes alive per amount of data sent to the BS. The query using HEED is more efficient in load balance; all of its nodes almost run out of power at the same time. However, when routing protocol is MintRoute, nodes in the first level consume power faster than others.

We simulate HEED, MintRoute, and Dynamic routing in the case of data acquisition, respectively. According to the protocol selection table in BS, dynamic routing scheme chooses MintRoute as routing protocol. Figure 12 shows the total energy dissipation as a function of data packets received at BS, where MintRoute achieves about 25% more data packets per unit energy than HEED.

5.2. Multi-queries

To test the dynamic scheme, BS sends an aggregation query to the network at first. When the aggregation query finishes, BS sends an acquisition query to the network continuously. The sample period of each query is 10 s, and the duration is 30 min. In the situation of HEED and MintRoute, the network layer adopts a single routing protocol to deal with two kinds of query. They cannot achieve the optimal performance for these queries. The query system adopting a dynamic scheme can switch routing protocol according to query task. As shown in Figure 13, we record the number of data packets received at BS and energy dissipation. When the number of data packets is between 0 and 16,000, the network is running aggregation query, where HEED is more energy efficient than MintRoute. Therefore, the dynamic routing layer selects HEED as routing protocol. When the number of data packets is over 16,000, the aggregation query is finished. When the number
of data packets is between 16,000 and 32,000, the acquisition is executed, where the performance of MintRoute is better than that of HEED. From Figure 13, we can see that the slope of MintRoute is less than that of HEED between 16,000 and 32,000. The dynamic routing scheme has the best performance by dynamically switching between HEED and MintRoute. So it is more energy efficient than both MintRoute and HEED.

5.3. Random query

A number of queries are randomly generated. Query type is randomly selected from aggregation and acquisition, and query duration (the number of sampling) is a random integer between 1 and 30. Using these constraints, we generate 16 aggregation queries and 13 acquisition queries. BS sends queries continuously, that is, queries are injected into a network one by one, and there is only a single query running in the network at a time. For 100 nodes distributed in a 100 m × 100 m area, we simulate MintRoute, HEED, and dynamic scheme by using the aforementioned setup. As shown in Figure 14, the dynamic routing scheme is more energy efficient than HEED and MintRoute.

5.4. Power dissipation of control messages

The power dissipation for routing control messages in HEED mainly includes energy consumption for exchanging tentative cluster head declaration message, cluster head declaration message, cluster join message, routing tree construction message, and scheduling message. The energy consumption of MintRoute mainly includes the energy consumption for exchanging beacon message, child join message, and scheduling message. We have run the simulations of HEED, MintRoute, and
Dynamic routing, respectively, for five rounds. As shown in Figure 15, the average energy consumption of exchanging control messages is almost the same for both HEED and MintRoute. The average energy dissipation of dynamic routing layer for exchanging control messages in the case of routing protocol switching is slightly greater than that of HEED and MintRoute.

6. RELATED WORKS

There are some prior works that attempted to adjust routing protocols according to applications in ad hoc and sensor networks. Jun et al. 19 demonstrated that an optimal routing protocol can be selected for a particular application in ad hoc networks. Given environment characteristics and application requirements for performance metrics such as average end-to-end delay and average throughput, they showed that the selection of routing protocol for particular application is possible. But this work cannot be directly applied to sensor networks because of its energy inefficiency.

He et al. 20 described a framework to build programmable routing services for sensor networks. This framework includes a parameter space that identifies both common and different properties among routing services. With the parameter space, different routing services can be obtained with less programming effort. A programmable routing architecture was proposed to maintain this parameter space and support energy-efficient service.

Filter-based architecture in sensor networks 21 specifies a software structure that contains a list of filter handlers, each of which is executed when its attributes match incoming packets. This architecture provides flexibility to add an application-specific code into sensor nodes in the form of filters. But the filter architecture only considers diffusion algorithms.
Lin and Levis proposed DIP 6, which is a hybrid data detection and dissemination protocol. It dynamically uses a combination of searching and scanning on the basis of network and version metadata conditions. They indicate that a scalable dissemination protocol can obtain the best of both advantages by using a hybrid approach on the basis of run-time conditions. The idea of DIP is to integrate two algorithms into a protocol. In contrast, our scheme is a dynamic network scheme, which is aimed at switching among different routing protocols.

7. CONCLUSION AND FUTURE WORK

Many sensor query processing systems, which view sensor data as a continuous stream, have been developed to acquire, process, and aggregate data from wireless sensor networks. We decompose the network layer into sharing modules and private modules, and let several routing protocols coexist in a sensor node. BS decides which routing protocols should be selected according to query task. The dynamic network layer in sensor node can switch between routing protocols. Simulation results show that the dynamic routing scheme is more energy efficient than the single routing protocol.

In this paper, the dynamic scheme decides protocols switching only on the basis of query tasks. The performance of routing protocol is also significantly affected by network conditions. The dynamic scheme needs adaptive optimal matching applications with routing protocols. As future work, we intend to let BS decide protocol switching on the basis of automatic learning.

REFERENCES

DYNAMIC NETWORK LAYER


AUTHORS’ BIOGRAPHIES

Zusheng Zhang received his PhD from the Institute of Computing Technology, Chinese Academy of Sciences. Currently, he is an assistant researcher at Dongguan University of Technology and Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interest is in wireless sensor networks.

Liang Chen received his bachelor’s degree from Tsinghua University, Beijing, China. In 2009, he received his master’s degree from the Institute of Computing Technology, Chinese Academy of Sciences. Currently, he is an engineer at Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences. His research interest is in wireless sensor networks.

Zhiqi Wang received his BS degree from the Nanjing University of Science and Technology, Nanjing, China in 2008. He is currently working toward his PhD degree at the Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences.

Fengqi Yu received his PhD degree in Integrated Circuits and Systems Lab at the University of California, Los Angeles (UCLA). In 2006, he joined the Shenzhen Institutes of Advanced Technology (SIAT), Chinese Academy of Sciences, as a full professor and director of the Center for Integrated Electronics. Before joining SIAT, he worked at Rockwell Science, Intel, Teradyne, Valence Semiconductor, and Suzhou CAS IC Design Center. His interests include CMOS RF integrated circuit design, CMOS sensor design, wireless sensor networks, RFID, and wireless communications.