Wireless Sensor Network, 2011, 3, 167-173
doi:10.4236/wsn.2011.35019 Published Online May 2011 (http://www.SciRP.org/journal/wsn)

Implementation Study of a Centralized Routing Protocol for Data Acquisition in Wireless Sensor Networks

Trung Hieu Pham[1], Xue Jun Li[1], Wai Yee Leong[2], Peter Han Joo Chong[1]
[1]*School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore City, Singapore*
[2]*Singapore Institute of Manufacturing Technology, Singapore City, Singapore*
E-mail: *ehjchong@ntu.edu.sg*
Received January 17, 2011; *revised February* 27, 2011; *accepted April* 6, 2011

Abstract

Wireless Sensor Networks (WSNs) attract considerable amount of research efforts from both industry and academia. With limited power and computational capability available on a sensor node, robustness and efficiency are major concerns when designing a routing protocol for WSNs with low complexity. There are various existing design approaches, such as data-centric approach, hierarchical approach and location-based approach, which were designed for a particular application with specific requirements. In this paper, we study the design and implementation of a routing protocol for data acquisition in WSNs. The designed routing protocol is named Centralized Sensor Protocol for Information via Negotiation (CSPIN), which essentially combines the advertise-request-transfer process and a routing distribution mechanism. Implementation is realized and demonstrated with the Crossbow MicaZ hardware using nesC/TinyOS. It was our intention to provide a hand-on study of implementation of centralized routing protocol for WSNs.

Keywords: Wireless Sensor Network, Three-Way Communication, Centralized Sensor Protocols for Information Negotiation

1. Introduction

Thanks to recent advances in wireless communications [1] and electronics technologies, wireless devices with low price and high performance are now available in the market. Wireless communications can be infrastructure-based, e.g., multihop cellular networks [2], as well as infrastructureless, e.g., ad hoc networks. Recently, one type of ad hoc network, namely wireless sensor network (WSN) [3], becomes increasingly popular due to its wide applications, high flexibility and low cost. During the implementation of a WSN, adoption of small or even tiny devices is the first priority and the devices are usually referred to as sensor nodes. Unlike the traditionally large-size wireless devices, sensor nodes have many constraints such as scarce memory, limited power and low computational capacity. Furthermore, these sensor nodes are more prone to failures than other wireless communication devices. In addition, WSN is normally composed of a large number of sensor nodes. Therefore, as compared to conventional cellular networks, different approaches in designing wireless networking protocols are required for a WSN [4].

Many existing sophisticated networking protocols and algorithms that generally support point-to-point communications in traditional networks are not suited to the requirements of WSNs. Moreover, most of sensor nodes are usually unable to establish one-hop connections directly with the base stations or fixed access points due to their limited power, which unfortunately limits their transmission range. Consequently, they have to rely on multihop transmissions in order to communicate with the desired nodes, through data forwarding with the aid of their neighbors.

Limited computational capacity limits the application of multihop routing among sensor nodes and poses a big challenge because it causes difficulty to set up and maintain a WSN. Besides, there are other challenges related to hardware and software of a sensor node, which is going to be discussed in this paper. That is why robust and flexible routing protocols are desired for WSNs. Furthermore, the constraints on each WSN might not be the same and trade-offs should be considered. These issues should be considered dependent on the specific category of applications. As a result, various approaches have been studied during the design of routing protocols

for WSN [5].

In this paper, we design and implement a data-centric centralized routing protocol, namely Centralized Sensor Protocol for Information via Negotiation (CSPIN), which combines advertise-request-transfer process in SPIN [6] and the routing distribution mechanism. Two main objectives are to provide centralized control to the base station and to reduce unnecessary transmissions over the network.

The rest of this paper is organized as follows. Section 2 analyzes the challenges and requirements for the design of an efficient routing protocol for WSNs. Section 3 presents the design concept using a general routing model and Section 4 describes the CSPIN implementation with Crossbow MicaZ hardware using nesC/TinyOS. Then the operation of CSPIN is explained in detail in Section 5. Finally, Section VI discusses and concludes about the work.

2. WSN Routing Protocol Design Challenges

This section provides the list of all the important and common challenges that are encountered by aforementioned applications. With these technical challenges in mind, we can be able to focus the critical issues during the design and implementation of a routing protocol for WSNs.

2.1. Hardware Challenges

1) *Node Failure*: since the sizes of sensor nodes are normally small, they are more prone to failure due to lack of power, physical damages or environmental interference [3]. Thus, the entire network may be suspended or interrupted because WSN relies on multihop transmissions from each other. As a result, an adaptive routing protocol needs to be used to ensure reliability of the network if case of these failures.

2) *Node Density*: since the number of sensor nodes in WSN may be in the order of hundreds or even thousands, depending on the requirement of a particular application [3], the network requires robust and flexible routing schemes to efficiently utilize a single node's processing capacity.

3) *Power Limitation*: due to their limited battery capacity, sensor nodes have a very short transmission range. Besides, the distance between a particular source-destination pair may be very long so that multihop transmission should be adopted due to out of communication range between them. In other word, the source node needs to rely on its neighboring sensor nodes to forward data towards to the destination node.

4) *Computational Capability*: usually a small or even tiny sensor node needs to include all the required function blocks, such as sensing block, data processing block and wireless transceiver. These requirements obviously result in many limitations on the capability of a sensor node, such as limited power, slow processing speed and scarce memory.

5) *Sensing Purposes*: WSNs will provide different kind of services to have different type of sensor nodes such as sound sensors and temperature sensors for acoustic surveillance and volcanic monitoring, respectively. Consequently, communication links between different sensor nodes may not be the same type due to their particular hardware platforms and abilities.

6) *Autonomous Capability*: WSN must have its autonomous capability such that each sensor node has to collect and process data independently without any human intervention for a long period of time. Thus, it is good to have central stations to query for data or send other commands over the network because central stations are usually under manual control.

2.2. Design Challenges

The design of a new protocol requires the consistency of functionalities implemented in modules and interfaces. Minimizing unnecessary coupling between modules results in the reduction in effort to combine and implement new functionalities. Moreover, it is highly desirable that the protocol designed for one particular application can be reused in other applications. Since co-existing protocols with modules to implement overlapping functionalities unnecessarily occupy extra resources in terms of memory and energy, the life time of resources constrained sensor nodes can be prolonged by maximizing code reuse code portability [7].

3. Design Framework and Layout

We are enlightened by the pioneer work presented in [7], which not only provided a good starting point to implement our proposed routing protocol, but also outlined general rules for us to use in this work to make sure that the result discussed in this paper is generic enough to be reused. Similarly, it is also of our interest to develop some function modules that might be useful for other researcher during their implementation of networking protocols for WSNs.

A modular network layer for sensor networks was discussed in [7], which mentioned the main objective is to "ease the implementation of new protocols, by increasing code reuse, and enable co-existing protocols to share and reduce code and resources consumed at run-time". Subsequently, a representative set of various

protocols for sensor networks was examined in order to identify their common parts. Fortunately, this inspection results in a possible division of the protocol into several function modules, among which certain function modules can be shared by all or some of the protocol. Consequently, this methodology was then used to design a general layout of components that provides a framework for implementing the routing protocol [8]. The working mechanism of the layout is illustrated in **Figure 1**. In particular, the layout is divided into two parts—the data plane and the control plane. Implementing the control plane is not surprisingly much more complicated, since it is fully responsible to accommodate the routing protocol.

As shown in **Figure 1**, headers of packets coming from the lower or upper layer are examined by the *Dispatcher* in order to determine the protocol to which the packet belongs, and then these packets will be passed to an appropriate protocol service, which may include a *Forwarding Engine*, a *Routing Engine* and a *Topology Engine*. These protocol services are revisited as follows:

1) *Forwarding Engine—Forwarding Engine* is not aware of the protocol format and algorithms, though it is part of a protocol service. Its function simply include: 1) before forwarding a packet, *Forwarding Engine* will request the *Routing Engine* to fill the routing header; 2) when the packet has reached its destination, *Forwarding Engine* will deliver the packet to the upper layer. *Forwarding Engine* belongs to the protocol service because it may perform those tasks dependent on the protocol, such as packet aggregation or scheduling.

2) *Routing Engine and Topology Engine*—They are the core components of a protocol. In brief, the *Routing Engine* generates and processes control packets. Next, according to the data reported by the *Routing Engine*, the *Topology Engine* computes and stores the necessary information about the network topology.

3) *Output Queue*—It handles the packets to be sent from all the protocols running on the node, which can schedule them according to the node policy due to the fact that all packets must go through this component.

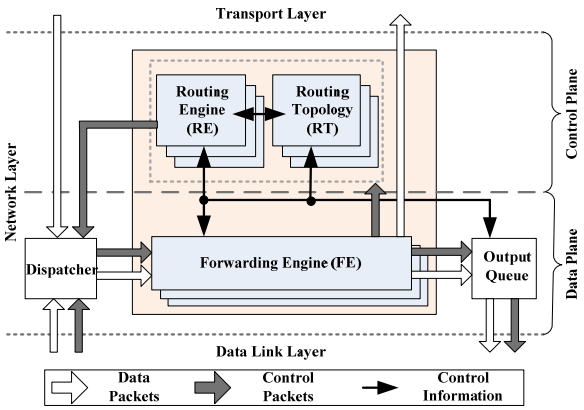

Figure 1. Network layer decomposition with flow of packet and control information [8].

4. Design and Implementation of CSPIN

As show in **Figure 2**, the implementation of CSPIN was realized by a configuration component, namely *CSpinNetworkC*, whose objective is to transparently send and receive data in a multihop WSN. *CSpinNetworkC* provides the wiring illustrated in **Figure 2**, which should certainly include a transport protocol to transport data through multihop route because CSPIN is a routing protocol. Any transport protocol using a 16-bit address can be adopted. In this paper, we refer to the transport protocol as Multihopping (MH). Through *CSpinNetworkC* via the *SubReceive* interface, *CSPIN* can possibly inspect all the multihop data packets that flow through this node and decide where the next hop is.

Next, the *SplitControl* interface initiates the network layer and *SplitControl* is implemented by a dedicated module, *NetControlM*, while the link layer is implemented by *ActiveMessageC* module. Before letting the application use the network layer, *NetControlM* waits for

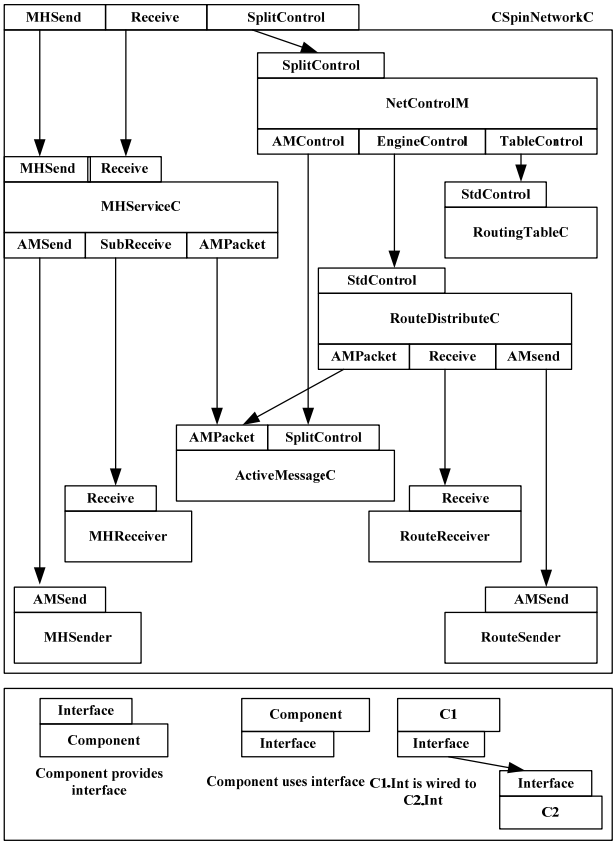

Figure 2. Simplified layout of the CSpinNetworkC component.

all other components to start. After that, multihop packets, including adverting, requesting and responding packets, can be then sent and received by the application and these packets may be manipulated with the *MHServiceC* component.

Consequently, the *RouteDistributeC* and *MHServiceC* components are responsible for all the processing and packet manipulations related to their respective protocol since they are the protocol services. In particular, each service has its own sending and receiving queue, an instance of *AMSenderC* and *AMReceiverC*. Although this is not depicted in **Figure 2** for simplicity reasons, it does not break the single *Output Queue* principle and these queues rely on *ActiveMessageC* to exchange packets within the radio chip. *ActiveMessageC* actually plays the role of the *Output Queue*, which utilizes the parameterized wiring feature of nesC [9] in order to process the packets to the *AMReceiverC* and from the *AMSenderC* components. As a consequence, a *Dispatcher* component is not necessary. The *ActiveMessageC* component provides the link-layer feedback by possibly request hardware acknowledgements for each packet sent, and thus determine if the packet has been received by a neighbor.

Interactions between components are shown in **Figure 3** and **Figure 4** when the application sends and forwards a packet to the base station, respectively. Before sending a packet, the application must wait for a routing message to compute its routing tables. After that, it sends the packet as if it was a single-hop packet. The packet is passed to the *MHServiceC* component (as illustrated in **Figure 5**), which is aware of how to obtain a route, though it does not know how the routing protocol operates. The route is obtained from the *RoutingTableC*,

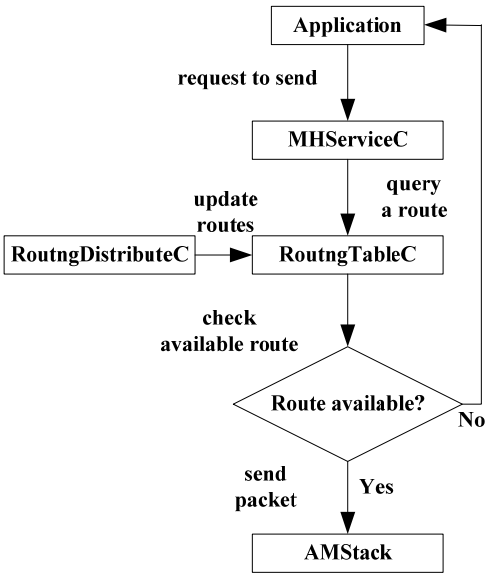

Figure 3. Flow chart of a node sending a message.

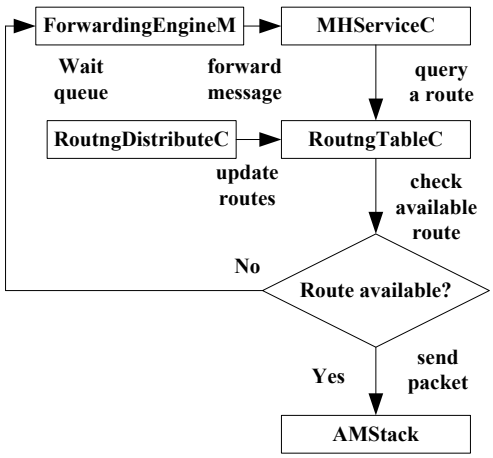

Figure 4. Flow chart of a node forwarding a message.

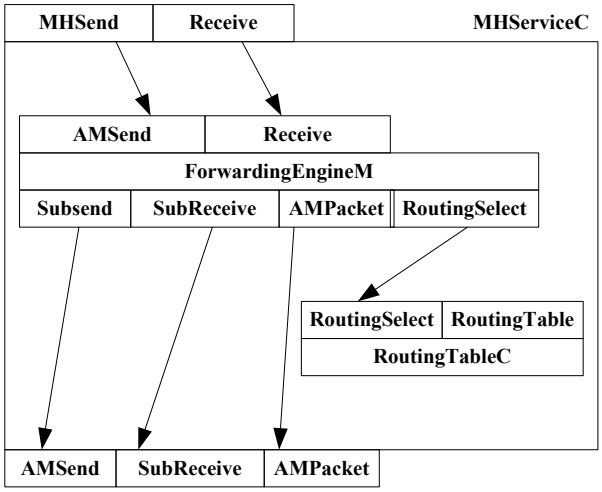

Figure 5. Modified MHService Component [8].

which is shared by both of the protocol services. Therefore, when the routing table is updated and signals to the application, MH service will be able to deliver the packet to the next hop along the route. Then, the application is signaled by the *sendDone* event and it may reuse the packet buffer.

The *RouteDistributeC* actually has a *Routing Engine* (the *RoutingEngineM* component) and a *Topology Engine* (the *RoutingTableC* component), but no *Forwarding Engine*. Thus, it does not exactly follow the modular layout presented in Section III because an exact implementation would have caused useless complexity. The delivering functionality of the *Forwarding Engine* is not needed as a result of the fact that upper layers are not interested in routing packets. Moreover, the *Forwarding Engine* would not need to request the *RoutingEngineM* to select a route because it would have already been selected by the *RoutingEngineM*, which is the only component that sends routing packets. Therefore, the *Rout-*

ingEngineM handles the received routing packets through direct connections to the *AMSend* and *Receive* interfaces. Processing a packet may take a long time and it thus is implemented as a split-phase operation. A routing packet is passed to the *RoutingTableM* module upon reception, which returns immediately by posting a task to read the packet. The information contained in a packet will be judged whether it is useful or not, and then decided if it should be propagated accordingly.

The route determined by the CSPIN protocol necessitates the usage of a multihop transport protocol because CSPIN services rely on it to fully function, which was unfortunately available in the TinyOS 2.0 distribution. Nevertheless, we have designed and implemented one, which allows for a directly usable multihop network layer to applications. Different from the CSPIN service, the MH service does have a *Forwarding Engine*.

As shown in **Figure 5**, the *Forwarding Engine* requests *ForwardingEngineM* to fill the AM fields in order to put the packet on the route toward the base station when a MH packet is received from the AM layer or sent by the application. The *Forwarding Engine* will not discard the packet if no route is immediately available because of its reactive nature. Next, the *Forwarding Engine* was made as generic as possible and does not rely on any MH-specific interface because it does not have any functionality specific to the MH protocol. As a result, it may therefore be used by other protocol services.

5. CSPIN Working Mechanism

As a data-centric protocol, CSPIN explicitly stores the routing path to the base station. More precisely, routing information periodically propagates from node to node. Starting from the central node, a node will compute the shortest route towards the desired destination and update their routing tables accordingly upon receiving the routing information. Since routing tables are constantly maintained, little processing is needed when a node wants to send or forward a packet, which reduces communication latency.

5.1. Routing Distribution Mechanism

As shown in **Figure 6**, CSPIN routing distribution makes it more centralized because the routing information is distributed from the central node [5].

Routing Advertisement: The advertisement (*RouteADV*) message is broadcast by base station periodically. All the nodes within its transmission range first receive the message. Then they build their routing tables based on the distributed routing information. Meanwhile, each of them will separately update the *RouteADV* message in order to encapsulate new routing information accordingly.

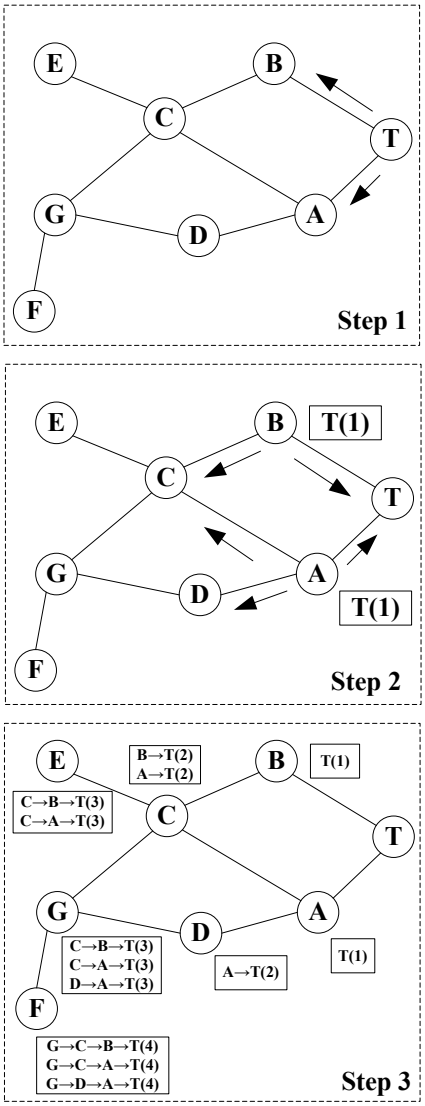

Figure 6. Three steps of route distribution.

Routing Information Propagation: The routing information is propagated throughout the network by broadcasting the new RouteADV messages. Then, the neighbors receive, process and update it, so on and so forth.

5.2. Routing Logic Update

When nodes receive a routing message, they look at all the included routing information. Then the nodes decide whether the routing information is better than what they already know based on sequence numbers and distances counted in hops from the routing information. If necessary, the nodes update their routing tables. If the received

routing advertisement is a new distribution sent from the base station or it has a smaller route sequence number, the receiving node will definitely update its routing table without any further process. Otherwise, the received routing advertisement will be further processed with the second condition of the number of counted hops. **Table 1** shows the logical steps how a node update its routing table.

During the processing, the content of the message is updated. Sensor nodes add information about themselves, which will be useful for surrounding nodes in determining the routing path. Moreover, hop counts are incremented and routing information that was considered unuseful is removed from the message so that it is not propagated further.

Routing Information Updating: Nodes only update their routing tables when they receive a new *RouteADV* or the *RouteADV* which has a smaller hop count than the local value.

Practically, if a new node joins the network, it will remain silent until a *RouteADV* message is received. Upon completing the computation of the shortest path, it knows to which address to send or forward a packet in order to reach the base station. Moreover, instead of storing global knowledge of a full path, each routing table contains only one-hop closer addresses, which reduces the complexity of architecture of the protocol and computational processing.

5.3. Three-Way Communication Procedure

As shown in **Figure 7**, the three-way communication helps CSPIN reduce unnecessary transmissions in the network. It also gives more control ability to the central node over the network.

1) *Adverstise*: Joined nodes regularly advertise themselves to the central node using node advertisement (*nodeADV*) messages. These messages include the identification, number of hops and full path information; upon processing all of them will provide the global knowledge of network topology. In other words, the base station has the list of available nodes, as well as the specific path to reach each of them.

2) *Request*: In order to obtain the data of a particular node in a specific interested region, the base station issues a request (REQ) message to this node.

Table 1. Logical condition for update.

new_route = rcv_newRoute or
(rcv_routeSeq > local_routeSeq) or
((local_routeSeq – rcv_ routeSeq) > MAX_SEQ/2) or
((rcv_routeSeq = local_routeSeq) and (rcv_hopCount < local_hopCount))

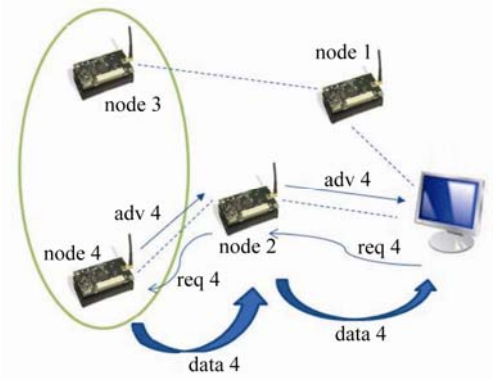

Figure 7. Three-way communication of data acquisition.

3) *Transfer*: The requested node in response starts sensing and sends its sensed data back to the sink. Since the data acquisition goes through three steps of advertising, requesting and responding, it is therefore called three- way process. This process ensures that only interested nodes and their corresponding regions are focused on.

6. Performance Evaluation

The proposed protocol was tested in TinyOS SIMulator (TOSSIM) [10] before hardware implementation so as to ensure its correctness and compliance with the proposed specifications. TOSSIM is a simulator provided with the TinyOS distribution, which runs codes written in nesC. It is thus possible to benefit from the advantages of a simulator without having to rewrite the application in a specific language. TOSSIM provides an interface to the execution of the TinyOS application, and the user writes a program in C, C++ or Python to control the execution of the simulation.

A testing component that provides and uses all the interfaces that the tested component uses and provides was written and wired to the tested component (see **Figure 8**). The features of TOSSIM were then used to provide information about the execution and to check that the component behaves as expected. During the testing of the CSPIN route distribution and multihop services (as shown in **Figure 9**), a testing component will wrap the whole service in order to control all the incoming and outgoing calls. The testing component creates a *MH packet*, passes it to the *MHserviceC* via its *AMSend* interface or its underlying Receive interface. The packet is then processed by the *MHserviceC*, during which various information about the process are displayed. The packet is eventually given to the upper or lower layer via the appropriate command or event, which are both implemented by the testing component so that it can check and display the content of the packet. As shown in **Figure 8**, this ensures that packets were processed correctly at lo-

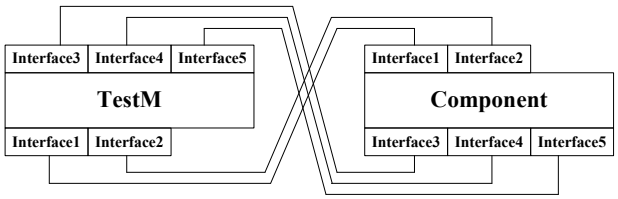

Figure 8.The generic wiring of a test program.

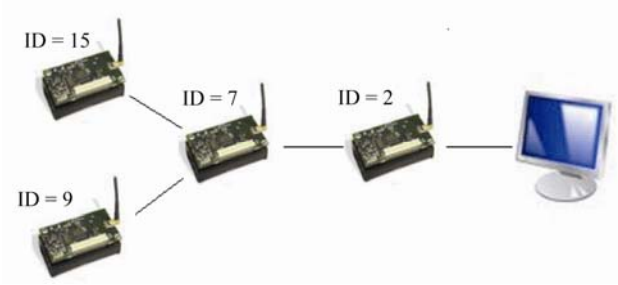

Figure 9. Four nodes topology with multihop routing.

cal nodes. After simulation study by TOSSIM, we have implemented CSPIN into CrossBow nodes to demonstrate its functionality. Experimental results from CrossBow nodes show that it works well.

7. Discussions and Conclusions

The goal of this paper is to design and implement a data-centric multihop routing protocol that is suitable for data acquisition in wireless sensor networks. The results obtained by CSPIN in a small WSN meet the design requirements (centralized control and reduced redundant transmission). However, performance of the CSPIN is yet fully evaluated in terms of scalability, energy consumption and comparison with other routing protocols due to technical issues. CSPIN still has much room for improvements. Currently, it is only a simple disseminated routing protocol that supports multihop. There is no reliable communication over links, which heavily impacts on the network performance when the size of the network gets bigger. As a result, providing a reliable transmission over communication links is necessary. This will make sure the requests reach the requested nodes and collected data arrive at the base station without any loss.

Next, another issue is the scalability of the WSN. CSPIN can operate properly in small network because the number of hops is small. If the network grows larger, certain nodes have to handle more processes, which will shorten the battery life of these nodes, and subsequently cause data transmission interruptions. Furthermore, overflow control is therefore needed to manage these situations. Nodes are able to switch over limited transmissions to the other nodes that are idle or have light computational load.

Finally, energy-saving operation is a certain requirement in the future implementation. Nodes when operating in energy-saving are able to alternate between sleep and wake-up states. They would be in the sleep mode most of the time and wake up when it is needed. Specifically, if there is no transmission or reception, each node turns off unnecessary activities to save power. When it has a message to send or there is a message coming, it turns on these activities on to carry out the process. Performance study about the effect of the proposed CSPIN on other system parameters, such as power dissipation, throughput, bit error rate and end-to-end delay, were left for future work.

8. References

[1] T. S. Rappaport, "Wireless Communications: Principles and Practice," 2nd Edition, Prentice Hall, New Jersey, 2002.

[2] X. J. Li, B.-C. Seet and P. H. J. Chong, "Multihop Cellular Networks: Technology and Economics," *Computer Networks*, Vol. 52, No. 9, 2008, pp. 1825-1837. doi:10.1016/j.comnet.2008.01.019

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, Vol. 38, No. 4, 2002, pp. 393-422. doi:10.1016/S1389-1286(01)00302-4

[4] H. Karl and A. Willig, "Protocols and Architectures for Wireless Sensor Networks," Wiley, New Jersey, 2005. doi:10.1002/0470095121

[5] K. Akkaya and M. Younis, "A Survey on Routing Protocols for Wireless Sensor Networks," *Ad Hoc Networks*, Vol. 3, No. 3, 2005, pp. 325-349. doi:10.1016/j.adhoc.2003.09.010

[6] J. Kulik, W. Heinzelman and H. Balakrishnan, "Negotiation-Based Protocols for Disseminating Information in Wireless Sensor Networks," *Wireless Networks*, Vol. 8, No. 2-3, 2002, pp. 169-185. doi:10.1023/A:1013715909417

[7] T. E. Cheng, R. Fonseca, S. Kim, D. Moon, A. Tavakoli, D. Culler, S. Shenker and I. Stoica, "A Modular Network Layer for Sensornets," *Proceedings of ACM ODSI*'06, Seattle, 6-8 November 2006, pp. 249-262.

[8] "TinyOS Documentation Wiki," http://docs.tinyos.net/index.php/Tymo.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer and D. Culler, "The NesC Language: A Holistic Approach to Networked Embedded Systems," *Proceedings of ACM SIGPLAN*'03, San Diego, 11-13 June 2003, pp. 1-11.

[10] P. Levis and N. Lee, "TOSSIM: A Simulator for TinyOS Networks," *Proceedings of the ACM SIGPLAN* 2003 *Conference on Programming Language Design and Implementation*, Vol. 38, No. 5, 2003, pp. 1-11. doi:10.1145/781131.781133

