Design and Implementation of a Zigbee-based Communication Substrate for Wireless Sensor Networks

無線感測網路之 Zigbee 通訊平台的設計與實作

Wei-kou Li* dimi@os.nctu.edu.tw

Chih-Hung Chou* robertchou@os.nctu.edu.tw

Zhi-Feng Lin* ttom@os.nctu.ed.tw

Da-Wei Chang* david@os.nctu.edu.tw

Chung-Chou Shen# ccshen@itri.org.tw

*Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan

#Internet Embedded System Division, Computer & Communications Research Laboratories, Industrial Technology Research Institute, Hsinchu, Taiwan

摘要

一個大型的無線感測網路可能包含許多由不同廠商製造的感測結點。為了讓不同廠商所製造的感測結點能互相溝通，必須要有一個標準化的通訊機制。而 Zigbee 因為其低成本，低功耗的特性，已經成為未來無線感測網路頗具希望的通訊標準之一。

在此論文中，我們將描述 Zigbee 網路層在設計與實作上的一些議題與細節。此外，我們也提出一個 beacon 排程的演算法以避免在一個 Zigbee 網路中 beacon 的相互碰撞。最後，我們提出一個基於 Zigbee 的快速物件追蹤方法，可用來追蹤移動的感測結點。

我們的 Zigbee 網路層是實作在工研院的 SCAN 感測結點上。網路層本身是實作成一個 Linux 核心模組。經過測試，資料傳輸數度可達 88 Kbps，足夠應付大部分的感測網路應用。

關鍵詞： 無線感測網路，Zigbee，IEEE 802.15.4

Abstract

A Wireless Sensor Network (WSN) consists of a large number of sensor nodes, which may not be produced by the same vendor. To enable communication among sensor nodes from different vendors, a standard communication link is needed.

Zigbee has features such as low cost and low power consumption so that it has been considered as a promising candidate for the communication technology of WSN.

In this paper, we describe the design issues and implementation details of the Zigbee network layer, which serves as a communication substrate for WSN. We also propose a beacon scheduling algorithm to avoid beacon collision, and a fast object-tracking technique that uses the functionality provided by Zigbee/IEEE 802.15.4 to track moveable sensor nodes.

We have implemented the Zigbee network layer on the SCAN sensor board, which is developed by Industrial Technology Research Institute (ITRI). The network layer is implemented as a Linux kernel module. The

performance result shows that the throughput of the network layer can reach 88 Kbps, which is sufficient for most WSNs.

Keywords: Wireless Sensor Network, Zigbee, IEEE 802.15.4.

1. Introduction

Wireless Sensor Network (WSN) [1, 3, 13] has received much attention in recent years. A WSN consists of a large number of sensor nodes that can communicate and cooperate with each other to accomplish a specific task. One of the most important requirements of a WSN is that sensor nodes should be low power, both in computation and communication, so that long battery life is allowed.

Zigbee [15] is a wireless communication technology for wireless personal area network (WPAN). It is especially suitable for low-power sensing and control applications. The Zigbee protocol stack, which consists of the network and application layers, sits on top of the MAC and PHY layers defined by the IEEE 802.15.4 [11] standard. Due to the low power and low cost features of a node, Zigbee is becoming an emerging standard for the communication technology of WSNs. For example, the MICA sensor series contains a product MICAz [5] that is equipped with a 2.4 GHz IEEE 802.15.4-compliant RF transceiver. For another example, there are also efforts [12] that implement Zigbee stack on TinyOS[14], the most popular operating system for WSNs.

Although Zigbee has become a promising candidate for the communication technology of WSNs, the implementation details are still not mentioned in the literature. In this paper, we describe the design issues and implementation details of the Zigbee network layer, which plays the roles of network formation, joining, leaving, and packet routing, and it can act as a communication substrate of a WSN.

In a Zigbee-based WSN, multiple devices may send beacons periodically, and hence a beacon scheduling algorithm is needed to avoid beacon collision. In this paper, we propose a beacon scheduling algorithm, which has been embedded into our Zigbee network layer implementation.

In addition to the Zigbee network layer, we also developed an object-tracking application based on this layer. The application can be integrated into a health-caring system for watching aged people in a community or a student-tracking system that tracks students when they are in the path between their homes and the school. In order to support fast object tracking, we propose a technique that take advantage of the functionality provided by Zigbee/IEEE 802.15.4 network. We also implemented the technique and demonstrate its feasibility and performance.

We have implemented the Zigbee network layer as a Linux kernel module on the SCAN sensor board, which is developed by Industrial Technology Research Institute (ITRI). According to the performance results, the proposed object tracking approach is efficient and the data throughput of the network layer (i.e., 88 Kbps) is sufficient for most WSNs.

The sensor board together with the Zigbee protocol stack can serve as a research platform for WSN. Different from the TinyOS platform, a sensor node application on our platform is a typical Linux application program. Thus,

researchers can implement their techniques with less effort.

The remainder of the paper is organized as follows. Section 2 describes the related efforts. Section 3 presents the design issues and implementation details. The performance results are given in Section 4, and we conclude in Section 5.

2. Related Work

2.1 Wireless Sensor Network

Wireless Sensor Network (WSN) [1, 3, 13] has received a great attention in recent years, both from academy and industry. Typically, a sensor node consist of a sensor unit (e.g., a temperature sensing module), a less-powerful computation unit (e.g., ARM, 8051), a small storage unit (e.g., Flash memory), and a low-power communication unit. Due to the large number and tight resource and power constraints of sensor nodes, WSN put challenge on solving the problems such as deployment, energy-efficient routing, reprogramming, and etc.

Traditionally, sensor nodes communicate with each other via proprietary RF links. To enable communication among nodes from different vendors, a standard communication link is necessary. Zigbee has features such as low cost and low power consumption so that it has been considered as the promising candidate for the communication technology of WSN.

2.2 IEEE 802.15.4

As mentioned above, Zigbee chooses IEEE 802.15.4. as the bottom layers of the protocol stack. The specification of IEEE 802.15.4 defines the MAC and PHY layers for a low data rate WPAN. It uses carrier sense multiple access with collision avoidance (CSMA-CA) as the medium access mechanism, and it can achieve 250kbps on the 2.4 GHz ISM band (16 channels).

Due to the low power feature and standardization, some companies such as Crossbow[4] and Freescale[9] have developed IEEE 802.15.4-compliant sensor boards.

2.3 Zigbee

Zigbee [15] is a low data rate wireless communication technology for wireless personal area network (WPAN). It is defined by the Zigbee Alliance, and is especially suitable for low-power sensing and control applications. As mentioned above, the Zigbee Alliance defines the network and application layer protocols, and it chooses IEEE 802.15.4 as the bottom layers.

A Zigbee node can be a full function device (FFD) or a reduced function device (RFD). The former is equipped with more resources. It can send beacons, form an IEEE 802.15.4 network, and route packets. The latter has fewer resources and usually can not perform the above tasks. The nodes in a Zigbee network can be classified into three roles: coordinator, router, and end device. A FFD can play any one of the roles, while a RFD can only play the end device role. In Zigbee, the network topology can be star, mesh, or cluster tree. Figure 1 shows some possible topologies of a Zigbee network.

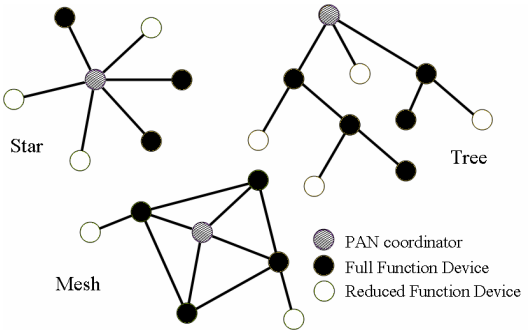

Figure 1. Possible Topologies of a Zigbee Network

Although companies such as Ember [7] and Figure 8 Wireless [8] have implemented a full Zigbee stack, little implementation details were mentioned. In this paper, we describe the design issues and implementation details of the Zigbee network protocol, which serves as the communication substrate of a WSN.

3. Design and Implementation

The Zigbee network layer is implemented as a state machine, which is described in Section 3.1. Section 3.2 shows the interface provided by the network layer to the other layers. Section 3.3 mentions some important data structures used in the network layer. The beacon scheduling algorithm and the fast object-tracking technique are described in Section 3.4 and 3.5, respectively. Finally, we mention the implementation of the applications.

3.1 State Machine

As we mentioned above, a Zigbee node plays one of the three roles: coordinator, router, or end device, and we implemented a state machine for each role. Figure 2 shows the state machine of a Zigbee coordinator. The states colored in gray indicate whether the node is in a Zigbee network or not, and the role of that node. For example, the *Out NWK* state is the initial state and indicates that the node is not in a network. The *In as Coordinator* state indicates that the node is currently in a Zigbee network and acts as the network coordinator.

From Figure 2 we can see that, a coordinator transfer from the initial state (i.e., the *Out NWK* state) to the *In as Coordinator* state via four steps. First, it receives the network formation request from the application layer, enters into the energy detection (ED) scan state, and performs the ED scan procedure. When receiving a network formation request, the node tries to start a WPAN. Therefore, it performs ED scan first to find out the channel with the least signal energy among all the available channels. The node will try to form a WPAN on that channel. Second, when an ED scan confirmation is received from the MAC layer (i.e., indicates that the ED scan procedure is done), the node enters the passive scan state and performs the passive scan procedure. Through the procedure, the node can learn the PAN identifiers (PAN IDs) that are currently used by the other WPANs on the target channel, and then the node can select an available PAN ID. Third, when the passive scan confirmation is received from the MAC, the node enters the start request state, and sends a start request to the MAC so that periodic beacons will be scheduled for transmission by the MAC. Fourth, when the start request confirmation is received, the node enters into the *In as Coordinator* state.

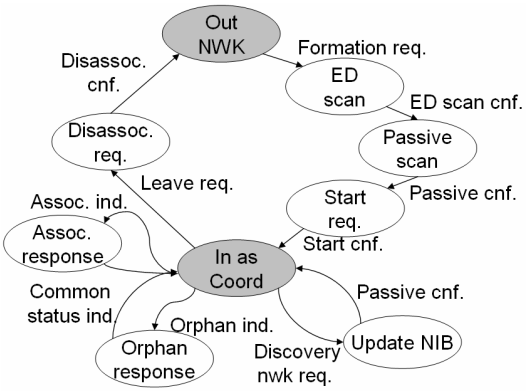

Figure 2. State Transition in a Coordinator

It is worth mentioning that the state transition happens according to the management flow, not the data flow. In other words, data packet transmission/reception does not lead to state transition. Data transmission/reception can happen at anytime as long as the node is in the network (i.e., the node is in one of the *In as Coordinator*, *Association Response*, *Update NIB*, and *Orphan Response* states in Figure 2).

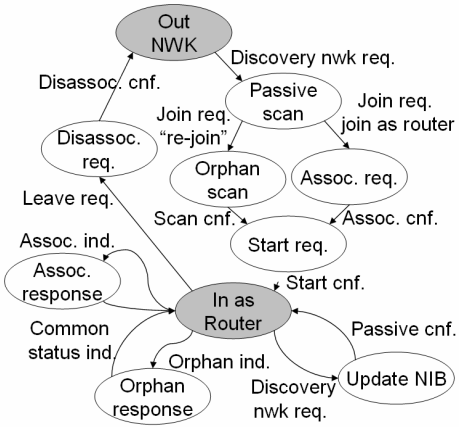

Figure 3. State Transition in a Router

Figure 3 shows the state transition of a Zigbee router. During the initialization of a router, the application layer issues a *network discovery* request to the network layer, which triggers a passive scan procedure. Slightly different from the passive scan procedure mentioned in Figure 2, the passive scan procedure of a router scans all the available channels and returns the PAN information (such as PAN ID, channel number) on these channels. According to the information, the application layer chooses a PAN that it wishes to join into and issues a join request (that specifies the *join as router* option) to the network layer. When receiving the join request, the network layer selects a node with the strongest signal strength in the target PAN and associates with it by issuing an association request to the MAC layer. Then, the *start router* request is issued and the router enters into the *In as Router* state. Note that the router may send periodic beacons or not, according to the parameters specified in the *start router* request.

In addition to the normal join operation, Zigbee also contains a re-join operation, which can be used to re-associate with a coordinator/router that the node had already associated with but the link was once broken. The re-join operation triggers the orphan scan procedure of the MAC, which is typically much faster than the normal association procedure. We use this feature to enable fast object tracking, which is described more clearly in Section 3.5.

When a coordinator/router enters into a network, it can allow others to join with it. When receiving an association indication from the MAC, the coordinator/router checks whether or not to allow the remote device to join into the network. If the device is allowed, the network layer will assign a network address for the device. When an orphan notification is received, the network layer checks whether or not the remote device had already associated with the local node. If it had, the network layer assigns

the same network address for the remote device so that the remote device can enter the network again.

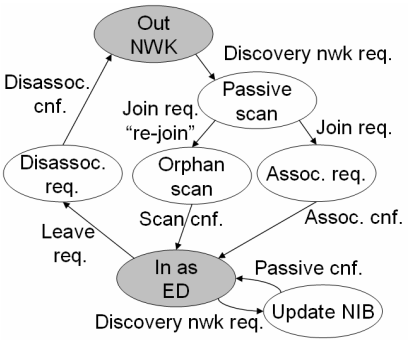

Figure 4. State Transition in an End Device

Figure 4 shows the state transition of a Zigbee end device. The states are a subset of those shown in Figure 3. During the initialization of an end device, the application layer also issues a *network discovery* request to the network layer, which triggers a passive scan procedure and therefore the node can associate with a coordinator/router it chooses.

Whatever the role a node plays, an in-network node can leave the network when it receives a leave request form the application layer. The request triggers the disassociation procedure of the MAC to leave the network.

3.2 Inter-Layer Communication

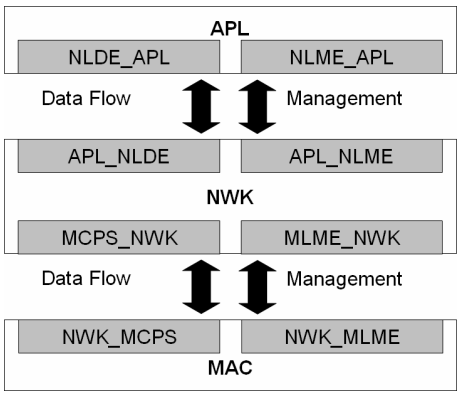

Figure 5. The Inter-Layer Interface

As mentioned above, Zigbee network layer sits between the application layer and the MAC layer. The inter-layer communication is via messages. Figure 5 shows the inter-layer interface, which is defined by [10]. The interface is divided into data and management parts, and the network layer provides four APIs to the other layers: APL_NLDE/APL_NLME for the application layer, and MCPS_NWK/MLME_NWK for the MAC layer. These APIs are used by the application and MAC layers to transfer messages to the network layer. When a message is received, the network layer inserts the message into the corresponding queue to allow the state machine to process the message.

The network layer checks the message queues periodically to see if there are any pending messages. If there are, it removes the first message from the queue and uses the message as the input of the state machine. The state machine will process the message, and perform state transition when necessary. Then, the memory used by the message will be freed and the network layer will check the next pending message.

Figure 6 shows the structure of the message transferred from the application layer to the Network Layer Management Entity (NLME). Basically, each message contains message type and message data. Other kinds of messages are similar to that shown in Figure 6 so we do not describe them in the paper.

```
typedef struct aplToNlmeMessage_tag {
        primAplToNlme_t msgType;
        union {
                nlmeNwkDiscvReq_t       nwkDiscvReq;
                nlmeNwkFormReq_t        nwkFormReq;
                nlmePermitJoinReq_t     permitJoinReq;
                nlmeStartRtrReq_t       startRtrReq;
                nlmeJoinReq_t           joinReq;
                nlmeDirectJoinReq_t     directJoinReq;
                nlmeLeaveReq_t          leaveReq;
                nlmeResetReq_t          resetReq;
                nlmeSyncReq_t           syncReq;
                nlmeGetReq_t            getReq;
                nlmeSetReq_t            setReq;
        } msgData;
} aplToNlmeMessage_t;
```

Figure 6. Application-to-NLME Message

3.3 Important Data Structures

In addition to the message structure, the most important data structure is the network layer information base (NIB), which stores the status of the network, the routing table, and the state of the neighbor nodes.

Figure 7 shows the data structure of NIB. It consists of the information such as the maximum depth of the network, the maximum number of the router, a neighbor table, and the capability of the node.

```
typedef struct
{
  uint8_t nwkBCSN;
  uint8_t nwkPassiveAckTimeout;
  uint8_t nwkMaxBroadcastRetries;
  uint8_t nwkMaxChildren;
  uint8_t nwkMaxDepth;
  uint8_t nwkMaxRouters;
  nwkNeighborTableEntry_t nwkNeighborTable[MaxNTable];
  uint8_t nwkNetworkBroadcastDeliveryTime;
  uint8_t nwkReportConstantCost;
  uint8_t nwkRouteDiscoveryRetriesPermitted;
  nwkRouteTableEntry_t nwkRouteTable[MaxRTable];
  uint8_t nwkSecureAllFrames;
  uint8_t nwkSecurityLevel;
  bool_t  nwkSymLink;
  uint8_t nwkCapabilityInformation;
}NIB_t;
```

Figure 7. The NIB

The structure of a neighbor table entry is shown in Figure 8. It consists of PAN ID, address information, device type (coordinator, router, or end device), relationship between the local and the remote nodes, the signal strength, and etc.

```
typedef struct
{
  uint8_t   PAN_Id[2];
  uint8_t   Extended_Addr[8];
  uint8_t   Network_Addr[2];
  uint8_t   Device_Type;
  bool_t    RxOnWhenIdle;
  uint8_t   RelationShip;
  uint8_t   Depth;
  uint8_t   Beacon_Order;
  bool_t    Permit_Joining;
  uint8_t   Transmit_Failure;
  uint8_t   Potential_Parent;
  uint8_t   LQI;
  uint8_t   Logical_Channel;
  uint8_t   Incoming_Beacon_Timestamp[3];
  uint8_t   Beacon_Transmission_Time_Offset[3];
}nwkNeighborTableEntry_t;
```

Figure 8. Neighbor Table Entry

3.4 Beacon Scheduling

As mentioned before, multiple routers (and the coordinator) may send periodic beacons. A beacon scheduling algorithm can be used to select the starting time of the beacons so that beacon collision can be avoided.

The most important parameters of a beacon scheduling algorithms are *beacon order* and *superframe order*. The former represents the time interval between two successive beacons, while the latter stands for the actual time period managed by the router (i.e., the router sends a beacon at the start of this period, and routers and end devices under this router can send/receive packets to/from the router in this period).

Note that, in IEEE 802.15.4, the beacon order time is 2's power times to the supreframe order time. For example, if the value of beacon order is 5 and the value of supreframe order is 2, there can be 2^{5-2} (i.e., 8) superframes in the beacon order time. Thus, we can regard the superframe order time as a time segment, and therefore the goal of a beacon scheduling algorithm is to allocate an unused segment (from all the available segments in the beacon order time) to the current router. When a segment is allocated to a router, the router can send a beacon at the starting time of the segment.

To find an unused segment, the router must first collect all of the beacons it can receive, and find out the segments that are used by these beacons. However, only considering the beacons the router can receive may suffer from the hidden node problem. As shown in Figure 9, if the white node is a router that wants to join with the gray node, it may receive beacons from the gray node and select a segment which is different from that used by the gray node. However, the segment may be overlaps with that of the black node, which is a hidden node of the white node, so that collisions may happen when both the black node and the white node wants to send beacon/data frames to the gray node.

To avoid this problem, the Zigbee network layer embeds *TxOffset* values in the payload of a beacon, which indicate the timing differences between the beacon and the neighboring beacons. By considering the TxOffset values, a beacon scheduling algorithm can avoid the problem of hidden nodes. Take Figure 9 as an example, the beacon of the gray node embeds a TxOffset value, which indicates the timing difference between the beacon of the gray node and the beacon of the black node. When receiving the beacon (from the gray node), the white node can learn the time segments of both the gray and the black nodes according to the receive time of the beacon and the TxOffset value. Therefore, the white node can choose a time segment that is different from those of the black and gray nodes.

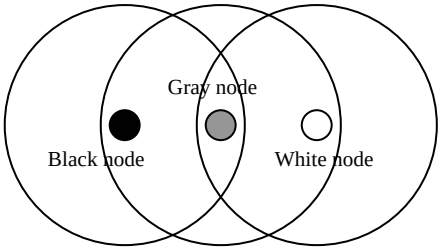

Figure 9. The Hidden Node Problem

```
c: current interval
b: beacon order interval
B: an array of segments in the beacon
      interval

collect the beacon information (including the
TxOffset information)
mark all used beacon segments in B
let c = b / 2          /* half of the beacon
                          interval*/
while(c >= 1)
{
    check all segments with segment number
    i*c in B ( where 0 <= i <= b/c -1 )
        if an unused segment is found
                return the segment number
        else
                c = c/2
    }
/* all segments are used */
return “no available segment”
```

Figure 10. Beacon Scheduling Algorithm

Figure 10 shows the proposed beacon scheduling algorithm. We take an example to make the readers understand the algorithm more easily. Assuming that the there are 8 (i.e., 2^3) segments in a beacon order time. This algorithm first check segment 0 and 4 (i.e., multiple of 2^2),

then segment 2 and 6 (multiple of 2^1), and finally segment 1, 3, 5, 7 (multiple of 2^0). The algorithm returns the unused segment number that it finds first.

3.5 Fast Object Tracking

Some WSN applications require tracking the location of specific moveable sensor nodes. In this section, we describe the approach that we used to track a Zigbee-based sensor node (i.e., object).

When a Zigbee-based object moves from the radio range of a node to that of another node, the object usually has to perform association with the new node. However, association requires a long time, so we use orphan join operation instead. As shown in Figure 11, the object will broadcast orphan request command, which will be received by router R1, R3, and R4 since they are in the radio range of the object. The object will join into one of the routers (i.e., become the child of the router), and all the above routers can send packets to the coordinator to specify that the object is in the radio ranges of the routers. Each router can also record the signal strength of the received packet and pass the information to the coordinator so that more precise positioning is possible.

We will demonstrate that the orphan join operation outperforms the association operation in time in Section 4.3.

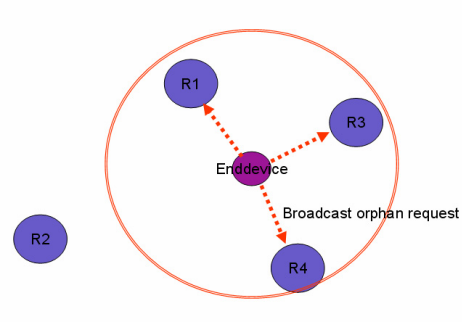

Figure11. Orphan Join

3.6 Application Implementation

In the initial implementation, the application is linked together with the Zigbee network layer as a Linux kernel module. Therefore, the communication between the application and the Zigbee network layer is through function calls.

To make application development easier, we currently allow the applications to be executed in user mode. We have wrapped the Zigbee network layer as a character device so that an application can perform communication via Zigbee by simply accessing the character device file. Specifically, an application can send commands/data to the Zigbee network layer by performing *ioctl()* calls. On the other hand, the Zigbee network layer notifies the application by using Linux signals. We reserve two signal numbers for Zigbee applications. SIGUSR1 is used by the management flow, and SIGUSR2 is used by the data flow.

4. Evaluation

4.1 Experimental Environment

We use the SCAN ZB V1.0B sensor boards developed by Computer and Communication Research Laboratory (CCL) at Industrial Technology Research Institute (ITRI) as our sensor devices. Each sensor board is

equipped with a Hynix HMS30C7202 CPU (70MHz), 16-MByte Flash, 16-MByte DRAM, and a Chipcon CC2420 RF Transceiver. The operating system is Linux, version 2.4.21.

The MAC layer, which we use in the following experiments, is developed by Computer Systems Laboratory at CSIE Department of Chang Gung University. In addition to this MAC, our Zigbee network implementation was also ported to the D18 MAC, the IEE 802.15.4-compliant MAC developed by Freescale Semiconductor Inc.

Besides the sensor nodes, we also use a packet sniffer CC2420EB [2] developed by Chipcon Inc. in some of the following experiments.

4.2 Object Tracking

To demonstrate the feasibility of the object tracking, we build up a backbone network that consists of three nodes: one coordinator and two routers. As shown in Figure 12, device 0000 is the coordinator, while devices 0001 and 0002 are the routers. This figure is presented by the packet sniffer with the Network Sensor Analyzer software [6] developed by Daintree Netwrok Inc., which is used to visualize the network topology. In addition to the backbone, there are three moveable objects that need to be tracked. Originally, the three objects join with the coordinator. During the experiment, we move the objects from the original position to the radio range of router 0001, and then to the radio range of router 0002. Figure 13 shows the result. From the figure we can see that, the objects finally join with router 0002.

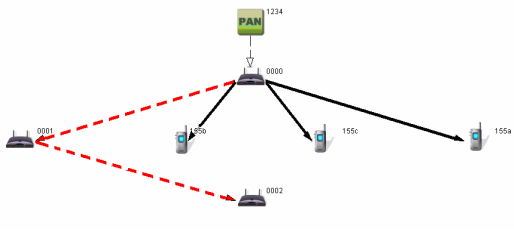

Figure 12. Object Tracking (Initial Condition)

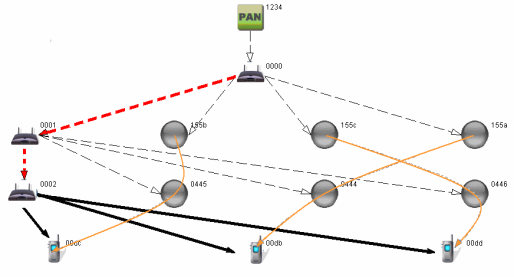

Figure 13. Object Tracking (Final Condition)

4.3 Performance of Orphan Join

In this section, we measure the performance of orphan join, and compare it with the performance of association.

4.3.1 Orphan Join v.s. Association

To measure the performance, we have an end device associate (or orphan join) with the coordinator and measure the time of the whole procedure on the end device. Figure 14 shows the results. From the figure we can see that the association time is about 9 times more than the orphan join time. This proves that the orphan join outperforms association in performance.

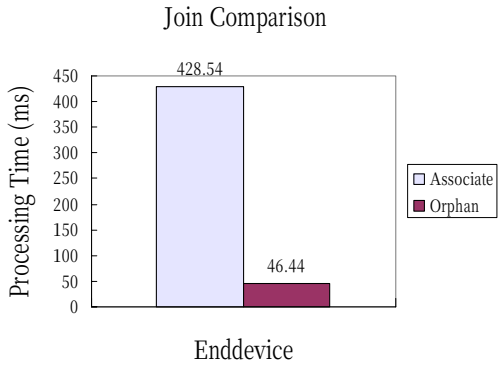

Figure 14. Performance Comparison between Orphan Join and Association

Note that most of the time an end device spends in an orphan join procedure is to wait for the response of the coordinator. In current implementation, we use a fixed waiting time of 5 jiffies (on Linux) which equals to 45ms in average. In the future, we will let an end device terminate the procedure once it has received a response from any coordinator or routers. This reduces the time of the procedure further.

4.3.2 Orphan Join with Multiple Nodes

In this section, we measure the performance of orphan join when multiple nodes join into the same coordinator at the same time. Figure 15 shows the results. The x-axis indicates the number of nodes, while the y-axis represents the average time of the procedure measured on the end devices. From the figure we can see that about 46ms is required when there is a single end device join with the coordinator, and only about 50 ms is required when there are 14 devices. This demonstrates that the performance does not degrade much when there is a moderate number of end devices join with the same coordinator/router. We did not measure the performance for more devices due to the unavailability of more device hardware. In the future, we will perform this experiment again for more devices.

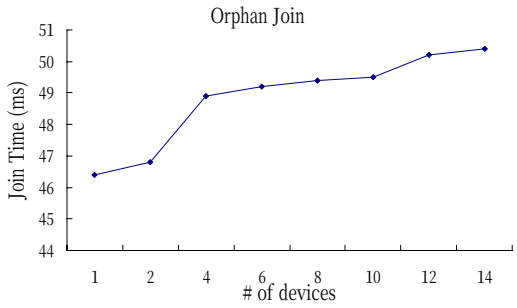

Figure 15. Performance of Orphan Join with Multiple Nodes

4.3.3 Timing Breakdown

Table 1. Orphan Join Timing Breakdown

Primitive	Time(us)
NWK_TX	60
MAC_TX	341
Wait Time	45937
MAC_RX	10
NWK_RX	53
System	53

In this section, we provide a timing breakdown of the orphan join procedure, which includes sending a request to and receiving a response from the coordinator. Table 1 shows the result. As mentioned above, most of the time is spent on waiting for the response. The real processing time (including the network layer, the MAC layer and the operating system) is quite small.

4.4 Data Throughput

In this section, we measure the data throughput with the presence of the network layer. Figure 16 shows the results. As shown in the figure, the data throughput increases as the size of the payload grows, and the maximum

through reaches 88 kbps.

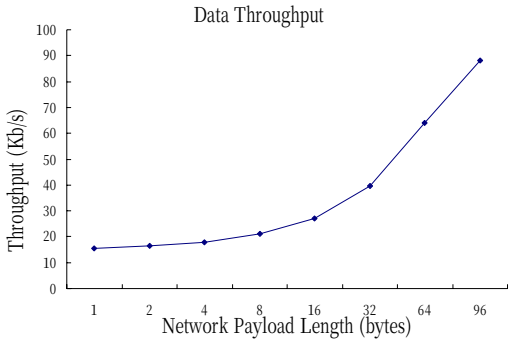

Figure 16. Data Through under Different Payload Sizes

In the last experiment, we measure the throughput degradation due to the adding of the Zigbee network layer implementation. The right bar of Figure 17 indicates the maximum performance of the MAC, while the left bar represents the maximum performance under the situation that the Zigbee network layer implementation is added on top of the MAC. As shown in the figure, the throughput degradation is about 9 kbps, which comes mainly from queuing delay and network layer header processing.

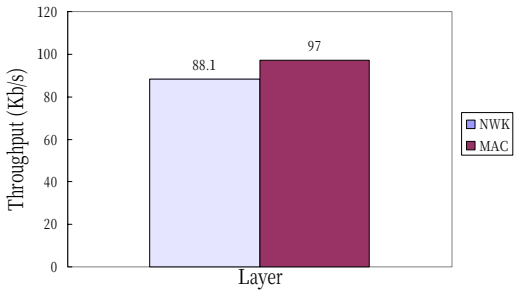

Figure 17. Performance Impact of Adding the Zigbee Network Layer

5. Conclusions

In this paper, we have described the design and implementation of the Zigbee network layer, which is responsible for network formation/joining/leaving and packet routing. In addition, we also proposed a beacon scheduling algorithm for avoiding the beacon collision problem. Finally, we proposed using the orphan join operation to achieve fast object tracking in a Zigbee WSN.

We have implemented the Zigbee network layer as a Linux kernel module. According to the performance results, the proposed object tracking approach is efficient and the data throughput of the network layer (i.e., 88 Kbps) is sufficient for most WSNs.

The sensor board together with the Zigbee implementation can serve as a research platform for WSN. Different from the TinyOS platform, a sensor node application on our platform is a typical Linux application program. Thus, researchers can implement their techniques with less effort.

References

[1] L. F. Akyildiz, W. Su, Y. Sankarasubramaniam and E. Cayirci, “A Survey on Sensor Networks”, IEEE Communication Magazine, Vol. 40, No. 8, pp. 102-114, August 2002.

[2] Chipcon Inc., “CC2420 Zigbee DK Development Kit”, available at http://www.chipcon.com/index.cfm?kat_id=2&subkat_id=12&dok_id=176, 2005.

[3] C. Y. Chong and S. P. Kumar, “Sensor Networks: Evolution, Opportunities, and Challenges”, Proceedings of the IEEE, Vol. 91, No. 8, pp. 1247-1256, August 2003.

[4] Crossbow Technology Inc., the Company Web Site of Crossbow Technology Inc., available at http://www.xbow.com/, 2005.

[5] Crossbow Technology Inc., MICAz datasheet, available at http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf, 2005.

[6] Daintree Networks, Inc., “Daintree Network Analyzer Software”, available at http://www.daintree. net/products/sna.htm, 2005.

[7] Ember Corporation, "EmberZNet? 2.0 - Fact Sheet ", available at http://www.ember.com/products/software/zigbee2.html, August 2005.

[8] Figure 8 Wireless, Inc., "ZigBee Software Development Suite", available at http://www.figure8 wireless.com/DataSheet.pdf, 2005.

[9] Freescale Semiconductor Inc., the Company Web Site of Freescale Semiconductor Inc., http://www. freescale.com, 2005.

[10] Freescale Semiconductor Inc., “802.15.4 MAC/PHY Software Reference Manual”, available at http://www.freescale.com/files/rf_if/doc/ref_manual/802154MPSRM.pdf, May 2005.

[11] IEEE, IEEE standard for information technology - telecommunications and information exchange between systems - local and metropolitan area networks specific requirements part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs), 2003.

[12] Luxoft Labs., The Zigbee Stack Page, available at http://www. luxoftlabs.com/tiki-index.php?page=ZigBee%20Stack, 2005.

[13] V. Rajaravivarma, Y. Yang and T. Yang, "An overview of Wireless Sensor Network and applications", IEEE Proceedings of the 35th Southeastern Symposium on System Theory, pp.432-436, 2003.

[14] TinyOS Web Site, available at http://www.tinyos.net/, 2005.

[15] Zigbee Alliance, available at http://www.zigbee.org/, 2005.