Study of Reliable Data Communication in Wireless Sensor Networks

Purushotham BV, Prakasha S, and Dr. K Ganesan

***Abstract*— the technology of wireless sensor networks (WSNs) is in the vanguard of the investigation of the computer networks and could be the next technologic market of a huge sum of money. An advance in mote technology is "mesh of network" special software that lets each device wake up during a fraction of a second when events occur and data has to be sent and forwarded to neighbors. Sensor networks are designed to carry out a group of tasks about information processing like detection, search or classification. Applications of these networks have a wide range. An example of such an application is sensors with RFID readers mounted on them to read tag information from the objects in a factory warehouse. Here, the tag information recorded by the RFID reader is a critical piece of information, which may not be available at a later point of time and hence has to be reliably transported to the sink. We study the various issues and analyze the design choices proposed in literature in addressing the challenge of sensors-to-sink reliable data communication in such applications. A cross layer based protocol with MAC layer retransmissions and NACK (Negative Acknowledgment) based rerouting of data packets is developed to overcome link failures and provide reliability. The protocol is implemented on TinyOS and the performance of NACK based rerouting protocol in terms of percentage successful message reception is compared with NACK based retransmission protocol by running simulations on TOSSIM. The NACK based rerouting protocol provides greater reliability under different metrics like varying network size, network traffic and percentage of failed links in the network.**

***Index Terms*—** Convergecast, MAC, NACK, RFID, TinyOS.

I. Introduction

Overview of Sensor Networks

A sensor network consists of several sensing devices deployed in a given geographical area for collaboratively gathering/sensing specific information in the environment for later analysis at a central base station. The sensor nodes self organize after deployment to establish radio communication paths to the sink. The sensing devices are low power devices consisting of a microcontroller for information processing, a microchip and antenna for radio communication and a sensor for sensing environmental parameters like temperature, humidity, light intensity etc [6].

Manuscript received October 9, 2007. (8th January 2008 and paper number:ICICWS_52).

Purushotham BV, is a Research Scholar, School of Computing Sciences, Vellore University, TN, INDIA (corresponding author to provide phone: 91-94805-14894; e-mail: utham74@gmail.com).

Prakasha S is with the Dayananda Sagar College of Engineering, Bangalore 560078. (e-mail: sprakashjpg@yahoo.co.in).

Dr. K Ganesan is with the School of Computing Sciences, Vellore University, Vellore, TN, INDIA (-mail: kganesan@vit.ac.in).

Motivation

Current research in the areas of wireless communications, micro-electromechanical systems and low power design is progressively leading to the development of cost effective, energy efficient, multifunctional sensor nodes. Sensing, communication, processing and battery units are the primary components of a sensor node. Individual sensors have the capacity to detect events occurring in their area of deployment.

Reliable data transport is an important facet of dependability and quality of service in several applications of wireless sensor networks. Different applications have different reliability requirements, for example an application to collect environmental parameters like temperature, humidity etc periodically can ignore an occasional loss of a value from a particular sensor but for an application in which the data collected by every sensor is a critical piece of information then end-to-end reliability has to be guaranteed for every individual packet [1][19].

An example for an application that requires guaranteed end-to-end reliability is an integration of Radio Frequency Identification (RFID) and wireless sensor network for automated inventory management and tracking [24]. In this application setup the sensor devices called motes [25] are attached with RFID readers to record RFID tag information on the objects. These sensor motes have a critical piece of information to be sent to the sink. Therefore reliable sensor-to-sink communication has to be guaranteed for such applications. This is the main motivation behind studying the various issues and strategies of reliable communication in this paper.

II. Cross-Layer Optimizations for Single Packet Reliable Transfer

Need for Cross-Layer Approach

We study a cross-layer based mechanisms for meeting the reliability requirements in applications where the data collection/generation rate by the sensors is not periodic i.e. random and the data being critical requiring guaranteed delivery to the sink. The specific application of Radio Frequency Identification (RFID) integration with wireless sensor network to automate inventory management and tracking applications [23][24] is considered as an example for studying the various cross-layer (MAC and Routing layers) optimization options for increasing the reliability of data transfer. We have seen some of the approaches like MAC-layer retransmission and end-to-end retransmission for achieving reliable single packet data transfer. These approaches cannot by themselves guarantee reliable transfer of data from the sensor to sink as cases of broken routing paths is not considered [4]. MAC-layer retransmissions overcome the issue of temporary failure of links which could be due o packet collisions or when the receiver is temporarily unable to receive messages but for cases where a particular routing path gets permanently broken then the retransmission attempts will be unsuccessful in delivering the packet to the destination. End-to-end retransmission of packets is similar to the functionality provided by the transport layer in the TCP/IP protocol stack for the internet [2].

We develop a cross-layer based approach by experimentally studying and evaluating the MAC layer retransmissions under different physical conditions on a real test bed and also propose and evaluate a routing layer mechanism for routing path rediscovery and packet recovery in the event of routing path breaks.

Experimental Test Bed

The experimental test bed consists of 5 – 8 Mica2 motes running TinyOS operating system.
Mica2 motes: Mica2 motes are the third generation wireless sensor network devices offered by Crossbow Inc [25]. They have the following characteristics:
Program Flash Memory: 128k bytes; Battery: 2x AA batteries; User Interface: 3 LEDs; Size (in): 2.25 x 1.25 x 0.25; Weight (oz): 0.7; Multi-Channel Tranceiver: 315, 433, or 868/916 MHz

nesC: Network embedded system C (nesC) is an open source programming language that is specialized for sensor networks [29]. It is an extension of C, which is a language that is supported by many microcontrollers and includes the necessary features to interface with hardware. nesC defines a component based model in order to make it possible to split applications into separate parts which communicates with each other using bidirectional interfaces[27].

TinyOS: TinyOS is an event driven operating system designed for sensor networks, where demands on concurrency and low power consumption are high but the hardware resources are limited [7]. TinyOS is written in nesC and much of the design of nesC was actually done in a way to increase the performance and utilization of TinyOS. Tiny provides a number of system components that can be reused in many applications. The components are wired together to the final application by using implementation independent wiring specification. The event-based concurrency model TinyOS uses has a close relation to the concurrency model that nesC uses. TinyOS uses two types of concurrency, tasks and events. Tasks are run to completion and cannot preempt each other [28]. They are to be used for computation processes where timing requirements are not strict. The tasks can be posted by the components.

Events also run to completion but can preempt other events and tasks. They can be used to handle time critical operations and hardware interrupts. The simple concurrency model that TinyOS uses offers relatively high concurrency but with low overhead in contrast to threaded concurrency, which requires a lot of overhead. The data races that can occur when using concurrency are detected by the compile time analysis that nesC compiler offers [18][20].

TOSSIM: TOSSIM is a bit-level simulator for TinyOS wireless sensor networks [28]. It has the following salient features:

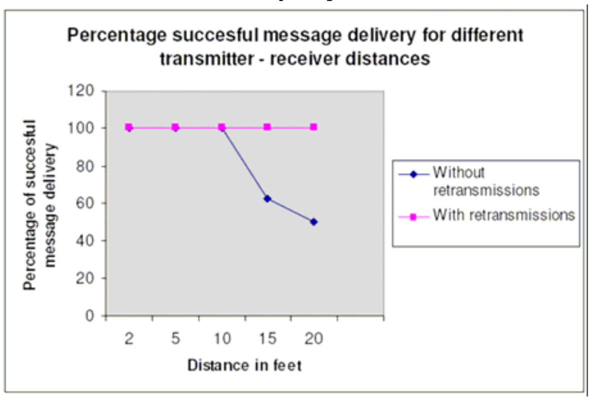
Completeness: The simulation covers as many system behaviors as possible; Fidelity. The simulator is able to capture the behavior of the nodes in detail;

Scalability: It has the capability to simulate a large number of nodes simultaneously; else it would be impossible to simulate an entire network;

Bridging: Errors often occur due to an incorrect implementation of a proper algorithm. The simulator uses the same code that is used to program the hardware, which means that the errors in the implementation will be detected [14].

MAC Layer Protocol

We experimentally study and evaluate the CSMA with random back-off MAC layer protocol [13] for reliably



transmitting message on a single hop. An implementation of the protocol is supplied as part of the TinyOS source code distribution.

Evaluating Reliability in One Hop Unicast Message Delivery

An experiment is conducted by setting up a test bed of 5 mica2 motes which includes one mote (base station or the sink) connected to a PC running and other 4 motes programmed to send messages to the sink on the radio transceiver.

Figure 2.1: Percentage successful message delivery for different transmitter-receiver distances

The goal of this experiment is to study the effect of transmitting distance on the reception rate when the nodes are interfering with one another's transmission. The sink broadcasts a message to all the 4 nodes in the listening range and the nodes on receiving the message start the timer to fire after 5 seconds so that all of them are triggered to send a unicast message to the sink at the same time [3].

Figure 2.1 shows the effect of MAC level retransmission in successfully receiving packets at the receiver. MAC level retransmissions are enabled at the different nodes by using a Boolean variable to track whether a "send complete" event has been triggered and if not the call to send is made again when the timer fires. The timer is set to fire repeatedly until the send is successful. Though the underlying CSMA with random-back off protocol resolves the contention to the access of the radio channel and establishes guarantee for successful message reception using ACK, it may not make enough attempts to successfully send the message. As can be seen from Figure 2.1, when retransmissions is enforced by repeated calls to send until send is complete, successful one-hop message delivery is guaranteed [11][12][15][16].

Network Layer – NACK Based Route Rediscovery

The reliable multi-hop routing scheme that we saw in section 2 used link quality estimation and good neighborhood table management techniques to determine the parent in the routing tree. We propose to add packet tracking in routing nodes and NACK based scheme to reconstruct routing paths during network partitions due to routing path failures [9]. Routing path failures occur when a forwarding node (intermediate node) in the routing tree loses its link with its parent requiring to re-do the parent selection process and as a result the packet to be forwarded is dropped [10]. Moreover, when there is no status tracking for the forwarded packets, if the forwarding node is not successful in establishing a parent then the packets that the node's children had sent to be forwarded gets lost and the senders have no way to know that the messages are lost. A negative acknowledge message (NACK) sent to the children is the only way to let the child nodes know of the status of the routing path and hence the status of their messages. The originating node or the node at one level below the current node will take suitable action like resending the packet to a different node that it chooses as a parent. The details of the working of the algorithm is as follows –

- Convergecast routing tree rooted at the sink is established in the network. Each node identifies its parent in the routing tree by considering either hop count to the sink or the link quality as the metric [3].
- Any node on sensing an event, like for example the RFID reader detecting/reading tag information of an object, prepares to send it to the sink by packaging the information into a packet, P. Every node maintains sequence umber for the messages it generates [21][22].
- Any node on receiving a data packet P from its child node saves a copy of it in the buffer B and forwards the packet to its parent. If the node loses the link to its parent due to parent node failure or drop in link quality, it triggers the parent selection process. If the node is not able to successfully find a parent to forward the packet P, it sends a NACK packet to the child node, which sent the packet indicating that it is not successful in forwarding the packet, and hence the child node needs to resend the packet to any other node it chooses as parent [8][9][10].

A node on receiving a NACK packet from the parent triggers the parent selection process so that some other node in the neighborhood (maintained as a neighborhood table [3]) is chosen as the parent. To reiterate, a NACK packet from a parent node means that the parent node is unsuccessful in forwarding the packet sent. The distributed algorithm that runs on every node in the network is listed below:

Algorithm 1 Modified routing algorithm to improve reliable transfer of single message by using NACK based route rediscovery for every processor pi

```
Initially parent = p, children = C, seq_no = 0, no de_id = i, Buf = 0
 1:  upon sensing an event E :
 2:  begin
 3:      create a data message M=(seq_no, node_id, data) with the event information E;
 4:      Add a copy of M to message buffer
 5:      call SendMsg (M, p);
 6:      seq_no ++;
 7:  end
 8:  procedure SendMsg(M, p)
 9:  begin
10:      send M using radio transceiver;
11:  end
12:  upon receiving a message:
13:  begin
14:      if message is M then
15:          call SendMsg(M, p);
16:      if message is NACK then
17:          look for the copy of the message in the buffer Buf ;
18:    if message is not found in Buf
19:  call Look ForMsgInChild ( ) to find the message in the child's buffer;
20:    else
21:      M = msg in Buf;
22:      call ParentSelection ( ) to find a new parent;
23:          call SendMsg ( M )
24:  end
25:  procedure LookForMsgInChild (src, seq_no)
26:  begin
27:      call SendMsg (NACK) to send message to child
28:  end
```

III. Implementation on TinyOS

The NACK based rerouting protocol is implemented on TinyOS using the nesC [29] programming language. A complete application utilizing the library components of TinyOS is developed to test the protocol. The CSMA with random backoff protocol available in the TinyOS distribution is used as the MAC layer protocol for the application. A simple hop-count metric based shortest path protocol is used to construct the routing tree with the sink as the root. Algorithm listing for constructing the routing tree is as follows:

Algorithm 2 Shortest path routing tree construction using hop-count as the metric for every processor pi

```
    Initially parent = 0, neighborTable = 0, node_id = i, hop_count = 9999, isRouteUpdateSent = 0
 1:  upon timer fired event E :
 2:  begin
 3:      if node_id is equal to BASESTATION_ADDRESS then
 4:      begin
 5:  create ROUTE_UPDATE message M with hop_count = 1
 6:  call SendBroadcastMsg (M);
 7:      end
 8:  end
 9:  upon receiving a message:
10:  begin
11:      if message M is a ROUTE_UPDATE message then
12:      begin
13:          if M(hop_count) is less than hop_count then
14:          begin
15:              hop_count = M(hop_count);
16:              create new ROUTE_UPDATE message M with hop_count value;
17:              call SendBroadcastMsg (M);
```

```
18:        end
19:  end
```

Application Configuration

The application uses the following TinyOS library components –

1. Main – The application begins its execution by running the StdControl interface of this component.
2. GenericComm – This TinyOS library component is used for radio communication. The SendMsg and ReceiveMsg interfaces are used for sending and receiving messages. These interfaces abstract the low level details of radio communication.

TimerC – This library component is used for generating timer events. It can be set to fire timer events periodically or only once.

Message Structure and Types

The structure o f the message used in the implementation is as given in Figure 3.1.

0	1	3	5	7 16
Messag e Type	Origin Addres s	Source Addres s	Sequenc e No.	Data

Figure 3.1: Message Structure

The numbers in Figure 3.1 indicates bytes used by different fields. The total size of the message is 16 bytes. The following message types are used in the implementation –
(i) ROUTE_UPDATE; (ii) DATA_MSG; (iii) NACK_MSG; (iv) NEW_PARENT_FOUND

ROUTE_UPDATE: Messages of this type are used for constructing the routing tree rooted at the sink. The sink initiates the transfer of this message type by broadcasting this message to all its one-hop neighbors with the value of the hop_count field set to 1. The value 1 indicates that the receiver is 1 hop away from the sink. Every node on receiving a ROUTE_UPDATE message updates the Neighborhood table by including the sender of the message and the hop count value in the message. Also, every node once broadcasts the ROUTE_UPDATE message with its hop count value.

DATA_MSG: Messages of this type is used by the nodes to send, receive and forward data. On receiving this message the node adds it to the data buffer.

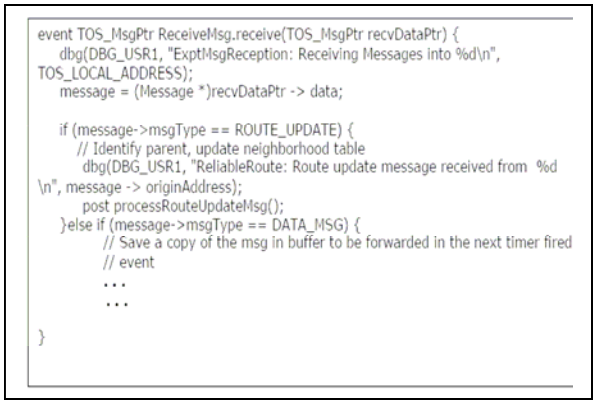
NACK_MSG: A node sends NACK messages when it is unable to forward the data messages it received from its neighbors. It sends a NACK message for every message stored in the data buffer to source node o f that message.

NEW_PARENT_FOUND: a node sends this message to its new parent after it chooses the new parent from the neighborhood table on receiving a NACK message from its old parent. This message type is useful in preventing loop formation in the routing structure. For example, if node A and node B have chosen a node C as its parent during the initial routing path establishment and if node C fails by losing its link to its parent or is unable to forward the messages collected in its data buffer then it sends a NACK message to both node A and node B.

Event Handling

The application needs to provide event handlers for the following events –

1. Send done event: This event is generated when the underlying MAC layer successfully sends the message. **sendDone** event handler declaration is part of the **SendMsg** interface. On successfully sending a ROUTE_UPDATE message, the Boolean state variable routeUpdateSent is set to 1. If the message type is DATA_MSG and NACK_MSG, the corresponding message in the Data Buffer is marked as sent.
2. Receive event: This event is generated when a node receives a message on its radio. Figure 3.2 shows a section of the Receive event handler source code. On receiving a ROUTE_UPDATE message a task processRouteUpdateMsg() is posted. Tasks are used to perform longer processing operation, which can be interrupted by a hardware event handler like receiving a message. This task performs the parent selection. It selects the source node of the message as its new parent if the hop count to the sink is lesser than the hop count using the current parent. On receiving a DATA_MSG, the message is included in the data buffer. On receiving a NACK_MSG the source node of the message is blacklisted, so that it is not in consideration to be chosen as a parent and a parent selection process is trigger ed to find a new parent [14][26].
3. Timer Fired event: The timer is set to fire every 10 seconds by calling the start command of the Timer interface. This call to the start command is made in the implementation of start command of the StdControl interface and hence the timer is started on every mote when the application starts up. Figure 3.3 shows the source code for the Timer Fired event handler. The event handler contains the implementation for sending Data messages, NACK messages and Route update messages. Designating certain nodes whose node ids fall in a given range as failed nodes simulates failure condition for the nodes. On every timer-fired event a check is made for unsent messages in the databuffer and one of it is sent. The checkUnSentMsgs() method implements this[17].

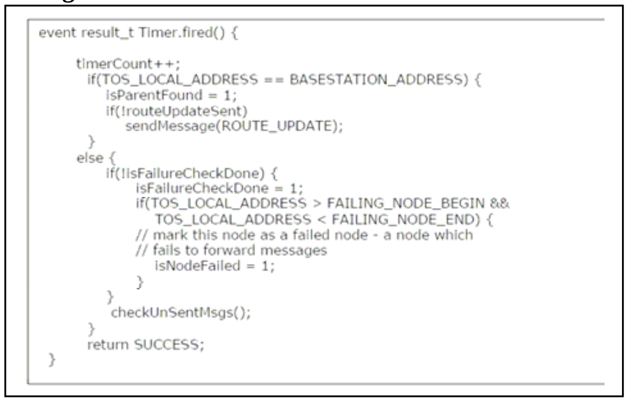


```
event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr recvDataPtr) {
    dbg(DBG_USR1, "ExptMsgReception: Receiving Messages into %d\n",
TOS_LOCAL_ADDRESS);
    message = (Message *)recvDataPtr -> data;

    if (message->msgType == ROUTE_UPDATE) {
       // Identify parent, update neighborhood table
        dbg(DBG_USR1, "ReliableRoute: Route update message received from  %d
\n", message -> originAddress);
        post processRouteUpdateMsg();
    }else if (message->msgType == DATA_MSG) {
           // Save a copy of the msg in buffer to be forwarded in the next timer fired
           // event
           ...
           ...

}
```

Data Message Buffer Management

Every node maintains a data buffer containing the data messages received from its children in the routing tree and the data messages generated by it. The message buffer is searched for the first buffer entry whose message is sent and the message originating address is not the address of the current node. If a buffer entry is found, then this entry no longer needs to be saved and hence removed from the buffer and the new message is added into its position. If no such buffer entry is found then the message is dropped.

Figure 3.3: Timer Fired Event Handler



```
event result_t Timer.fired() {

      timerCount++;
        if(TOS_LOCAL_ADDRESS == BASESTATION_ADDRESS) {
           isParentFound = 1;
           if(!routeUpdateSent)
              sendMessage(ROUTE_UPDATE);
        }
      else {
           if(!isFailureCheckDone) {
                isFailureCheckDone = 1;
                if(TOS_LOCAL_ADDRESS > FAILING_NODE_BEGIN &&
                   TOS_LOCAL_ADDRESS < FAILING_NODE_END) {
                // mark this node as a failed node - a node which
                // fails to forward messages
                   isNodeFailed = 1;
                }
           }
            checkUnSentMsgs();
        }
        return SUCCESS;
  }
```

IV. Conclusion and Future Work

Conclusion

In this thesis we have developed a cross-layer based mechanism with MAC layer retransmissions and NACK based rerouting of data packets for providing guaranteed reliability of sensors-to-sink single packet delivery. The implementation of the protocol as an application on TinyOS is described. Extensive simulations of the protocol are done on TOSSIM to compare the performance in terms of successful packet reception at the sink with NACK based retransmission scheme for the metrics varying network sizes, network traffic and percentage of link failures. The NACK based rerouting protocol developed provides much better reliability than NACK based retransmission protocol.

Future Work

This work can be extended in the following way: A realistic experimental scenario can be established to study the reliable transfer of RFID tag messages by using RFID readers mounted on sensing nodes and considering different rates of data generation that depends on the actual factory warehouse setting.

References

[1] C. Wan, A. Campbell, L. Krishnahmurthy. PSFQ: A Reliable Transport Mechanism for Wireless Sensor Networks. ACM International Workshop on Wireless Sensor Networks and Applications, Atlanta, Georgia. Sept 2002.
[2] Y. Sankarasubramaniam, O. B. Akan, and I. F. Akyildiz, “ESRT: Event-to-sink reliable transport in wireless sensor networks,” presented at the ACM Mobile Ad hoc, Annapolis, MD, Jun. 2003.
[3] A. Woo, T. Tong, D. Culler. “Taming the Underlying Challenges of Reliable Multi hop Routing in Sensor Networks”, Senys ’03, Los Angeles, California, USA.
[4] R. Stann and J. Heidemann, “RMST: Reliable data transport in sensor networks,” in Proc. 1st IEEE Int. Workshop Sensor Net Protocols Applications. (SNPA), Anchorage, AK, May 2003, pp. 102–112.
[5] S. Madden. M. Franklin and J. Hellentein. TAG a Tiny Agregation Service for Ad - Hoc Sensor Networks. OSDI. Dec 2002.
[6] M. Marin-Perianu, P. Havinga. “Experiments with Reliable Data Delivery in Wireless Sensor Networks” in Proc. Intelligent Sensors, Sensor Networks and information Processing Conference, 2005, pp. 109 - 114.
[7] Tiny OS Homepage [Online]. Available: http://webs.cs.berkeley.edu/tos/
[8] J. Hill, R. Szewczyk, A.Woo, S. Hollar, D. Culler and K. Pister, “System architecture directions for network sensors,” in Proc. 9th Int. Conf. Arch. Support Program Languages & OS, Nov. 2000, pp. 93–104.
[9] Gnawali, O.; Yarvis, M.; Heidemann, J.; Govindan, R. “Interaction of retransmission, blacklisting, and routing metrics for reliability in sensor network routing”. Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. Page(s):34 – 43.
[10] D. Tian and N. D. Georgan as, “Energy efficient routing with guaranteed deliver y in wireless sensor networks,” in Proc. IEEE (WCNC’03), Institute of Electrical and Electronics Engineers. New Orleans, USA: IEEE Press, Mar. 2003.
[11] A. Woo and D. Culler, “A transmission control scheme for media access in sensor networks,” in Proc. ACM/IEEE International. Conf. Mobile Computing and Networking, Rome, Italy, July 2001, pp. 221–235.
[12] Wei Ye, John Heidemann and Deborah Estrin, "Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks," IEEE/ACM Transactions on Networking, Vol. 12, No. 3, June 2004.
[13] B. Deb, S. Bhatnagar, and B. Nath, “Information assurance in sensor networks,” in Proc. 2nd ACM Intl. Workshop on Wireless Sensor Networks and Applications (WSNA), San Diego, CA, Sept. 2003.
[14] D. Ganesan, R. Govindan, S. Shenker, and D. Estrin, “Highly-Resilient, Energy Efficient Multipath Routing in Wireless Sensor Networks,” Mobile Computing and Communications Review (MC2R), vol. 1, no. 2, 2002.
[15] D. Tian and N. D. Georganas, “Energy efficient routing with guaranteed delivery in wireless sensor networks,” Institute of Electrical and Electronics Engineers. New Orleans, USA: IEEE Press, Mar. 2003.
[16] B. Deb, S. Bhatnagar, and B. Nath, “Re inform Reliable information forwarding using multiple paths in sensor networks,” in Proc. 28th Annual IEEE Conference on Local Computer Networks (LCN 2003), 20-24 Oct. 2003 Page(s):406 - 415.
[17] B. Deb, S. Bhatnagar, and B. Nath, “Re inform Reliable information forwarding using multiple paths in sensor networks,” in Proc. 28th Annual IEEE Conference on Local Computer Networks (LCN 2003), 20-24 Oct. 2003 Page(s):406 - 415.
[18] S.-J. Park, R. Vedantham, R. Sivakumar, and I. F. Akyildiz, “A scalable approach or reliable downstream data deliver y in wir eless sensor networks,” in Proc. Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc’01), Tokyo, Japan, May 2004.
[19] S.-J. Park and R. Sivakumar, “Poster: Sink-to-sensors reliability in sensor networks,” in Proc. 4th ACM Intl. Symp. OnMobile Ad Hoc Networking and Computing (MOBIHOC), Annapolis, MD, June 2003.
[20] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, “Directed diffusion for wireless sensor networks,” IEEE/ACM Transactions on Networking, vol. 11, no. 1, pp. 2–16, Feb. 2003.
[21] C. Englund and H. Wallin. RFID in wireless sensor network. Master's Thesis, Gotenborg, Sweden, April 2004.
[22] McKelvin, M. L., Williams, M. L., and Berry, N. M. 2005. Integrated radio frequency identification and wireless sensor network architecture for automated inventory management and tracking applications. In Proceedings of the 2005 Conference on Diversity in Computing, New York, NY, 44-47.
[23] Crossbow Inc Homepage [Online]. Available: www.xbow.com
[24] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, “The esC language: A holistic approach to networked embedded systems,” in SIGPLAN Conference on Programming Language Design and Implementation (PLDI’03), June 2003.
[25] A. Woo and D. Culler. Evaluation of efficient link reliability estimators for own power wireless networks. Technical Report UCB//CSD-03-1270, U.C. Berkeley, Computer Science Division, September 2003.
[26] E. D. Demaine, A. Lopez-Ortiz and J.I.Munro. Frequency estimation of internet packet streams with limited space. In Proceedings of the 10th Annual European Symposium on Algorithms ESA 2002, pages 348–360, September 2002.
[27] David Gay, Philip Levis, David Culler, and Eric Brewer. necC 1.1 language reference manual. http://nescc.sourceforge.net/papers/nesc-ref.pdf , May 2003.
[28] Philip Levis, Nelson Lee, Matt Welsh, and David Culler. TOSSIM: Accurate and scalable simulation of entire tinyos applications. Proceedings of the ACM Symposium on Networked Embedded Systems, Nov 2003.
[29] Mainwaring, Alan. Polastre, Joseph. Szewczyk, Robert. Culler, David. Anderson, John. Wireless Sensor Networks for Habitat Monitoring. First ACM Workshop on Wireless Sensor Networks and Applications. September 28, 2002. Atlanta, GA, USA.