A Medium Access Control Protocol for Wireless Sensor Networks

Alexandru Copoţ
Computer Science and Engineering Department
University POLITEHNICA of Bucharest
Email: alex.mihai.c@gmail.com

Florin Donţu
Computer Science and Engineering Department
University POLITEHNICA of Bucharest
Email: florin1288@gmail.com

***Abstract*—Consistent research is engaged in the area of wireless sensor networks. One of the main reasons is the wide range of application potential in areas such as target detection and tracking, environmental monitoring, industrial process monitoring, and tactical systems. An important problem arises because of the low level of sensing; the method to counteract this limitation is to use dense sensor networks. Now, here comes the reason why a medium-access protocol is needed: it should be distributed, sensitive to network changes, it should avoid collisions and so on. First, this paper outlines some sensor network properties that are crucial for the design of MAC layer protocols and then, it shall present and describe a MAC protocol implementation for sensor networks, using energy saving mechanisms.**

I. Introduction

Improvements in hardware technology have resulted in low-cost sensor nodes, which are composed of a single chip embedded with memory, a processor, and a transceiver. Low-power capacities lead to limited coverage and communication range for sensor nodes compared to other mobile devices. Hence, for example, in target tracking and border surveillance applications, sensor networks must include a large number of nodes in order to cover the target area successfully. Unlike other wireless networks, it is generally difficult or impractical to charge/replace exhausted batteries. That is why the primary objective in wireless sensor networks design is maximizing node/network lifetime, leaving the other performance metrics as secondary objectives. Since the communication of sensor nodes will be more energy consuming than their computation, it is a primary concern to minimize communication while achieving the desired network operation. However, the medium-access decision within a dense network composed of nodes with low duty-cycles is a challenging problem that must be solved in an energy-efficient manner. Keeping this in mind, we first emphasize the peculiar features of sensor networks, including reasons for potential energy waste at medium-access communication. Then we give brief definitions for the key medium-access control(MAC) protocols proposed for sensor networks, listing their advantages and disadvantages. Moreover, protocols that propose the integration of MAC layer with other layers are also investigated. Finally, the survey of MAC protocols is concluded with a comparison of investigated protocols and future directions are provided for researchers with regard to open issues that have not been studied thoroughly.

II. Wireless Sensor Network Simulator

Avrora[7] is a set of tools for programs that run on the AVR series of microcontrollers produced by Atmel. It contains a flexible framework for simulating, analyzing, and optimizing assembly programs, and provides a clean Java API and infrastructure for experimentation, profiling, and analysis. The event-queue model for cycle-accurate simulation of device and communication behavior allows improved interpreter performance and enables an essential sleep optimization. Avrora allows date-/time-dependent properties of large-scale networks to be validated. A highly accurate energy model is available, enabling power profiling and lifetime prediction of sensor networks. Distance attenuation for multiple-hop scenarios is also modeled. However, Avrora does not yet model node mobility. Another phenomenon that is not modeled is clock drift, which takes into account that nodes may run at slightly different clock frequencies over time due to manufacturing tolerances, temperature, and battery performance.

A. Simulation of a platform

Avrora does not contain a disassembler, so it cannot load directly the machine code. In order to load program, you need to compile the source code and then use the avr-objdump tool for the binary file to obtain assembler code that can be interpreted by Avrora. The steps to simulate a simple program:

```
$ avr-gcc -o simple.elf -mmcu=atmega128 simple.c
$ avr-objdump -zhD simple.elf > simple.od
$ avrora -throughput=true simple.od
```

B. Monitoring and profiling

Avrora provides mechanisms to profile your application, to monitor the number and types of interrupts triggered during runtime, visualize memory accesses, routine calls, processor state and the most important thing, you can precisely estimate the energy consumption of your platform during runtime.

- Energy monitoring
- Interrupts monitoring
- Serial Interface monitoring

Another feature of Avrora that we used for testing and debugging our application is the possibility to redirect the serial ports of the nodes to the terminal.

III. Sensors system on node Sparrow v3

The Sparrow sensor node contains 3 types of sensors: temperature, humidity and ambient luminosity. The light sensor is analogical and provides a voltage directly proportional with the level of ambient light. This voltage can be read by ATMega128RFA1 microcontroller from PF2(Port F, Pin 2). Also, on PF0 you can read power supply voltage of the sensor node.

The transceiver from the ATMega128RFA1 microcontroller allows different transmission bands in 2.4GHz. It allows transmission speeds of 250 Kbps, 1Mbps and 2Mbps. It has 2 operating modes: Basic and Extended; the Extended mode includes the functionalities usually provided at medium access control level.

A. *Interface with ATMega128RFA1*

The interface is needed to command the transceiver from the application and to obtain/send data to it. In RFA1 case, having both the transceiver and microcontroller on the same chip is a big advantage, since you can do very fast transfers. RFA1 provides the following interface:

- Command registers: are treated as any other I/O register and are used to send commands to transceiver or obtain punctual information about its state. E.g. TRX_STATE, TRX_STATUS, PHY_RSSI
- Frame buffer: 128 bytes memory area where packets are sent/received. Accessible from microcontroller as 128 bytes I/O register area, beginning with TRXFBST and ending with TRXFBEND.
- TRXPR register: has 2 flags that can affect the transceiver even if is in sleep mode. The flags are TRXRST - resets transceiver anytime and SLPTR - controls the sleep state and the start of packet transmission.
- Interrupts: The transceiver can generate multiple interrupts to the microcontroller. E.g. TRX24_RX_START - begin receipt of packet; ready to read the frame buffer. TRX24_TX_END - end of transmission.

B. *State machine*

The transceiver from RFA1 uses a state machine to know the current operating mode. The states could be:

SLEEP	- Hibernation state, no energy consumption
RESET	- Just restarted
TRX_OFF	- Standby state, no packet listening
RX_ON	- Listening for packets
BUSY_RX	- Packet receiving
PLL_ON	- Transmission active and ready
BUSY_TX	- Packet transmitting

The diagram in Figure 1 shows the entire state machine and all the possible transitions.

States transition procedure: (a) Set your state in TRX_STATE. (b) Wait until TRX_STATUS does not contain TRANSITION_IN_PROGRESS in TRX_STATUS 4..0 bit field. Packet transmission procedure: (a) Enter PLL_ON state.

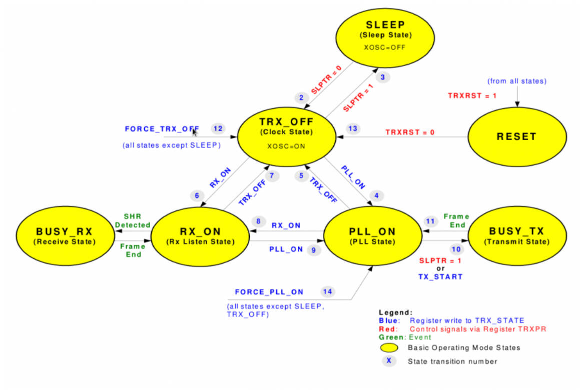

Figure 1. Basic State Machine for the RFA1 transceiver

This guarantees that a package reception does not begin over the frame buffer, between the moment of writing into the buffer and the moment when transmission begins. (b) Copy data into buffer: first byte represents the packet size and then the 127 bytes of data. (c) Start transmission - setting SLPTR bit or through TX_START command. Packet reception procedure: (a) Enter RX_ON state. (b) Receive RX_START or RX_END interrupt. (c) Begin reading the frame buffer: packet contains only data, from the first byte to the last byte. The packet size can be read from TST_RX_LENGTH.

IV. Implementation of a MAC protocol for WSN

A. *Overview*

When designing a good MAC protocol for wireless sensor networks, we must take into consideration the following problems:

Energy efficiency Most sensor nodes are battery-equipped, so reducing the energy consumption to prolong the service lifetime of the nodes becomes a critical issue. To solve the energy problem, we identified the sources of energy waste: - idle listening. This is a dominant factor of energy waste, especially when the traffic load on the network is light. - collision or corruption. Normally, collision may occur when neighboring nodes contend for free medium. When this happens, corrupted packets should be re-transmitted, which increases energy consumption. - overhearing. This happens when a node receives some packets that are destined to other nodes. - control packet overhead. Exchanging control packets between sender and receiver also consumes some energy.

Scalability and self-configuration For a wireless sensor network, its topology and size may change over time. So a good MAC protocol should accommodate to such changes.

Latency, throughput, bandwidth utilization, fairness There are common attributes for most of MAC protocols. For most of sensor network applications, the speed of changes on physical objects sensed by sensor nodes is much slower than the network speed.

One general issue comes from the fact that nodes might start to loose synchronization between each other. If the listening periods of neighboring nodes do not overlap for the right amount of time, the network may loose connectivity in time. Increasing the listening periods is also not an option because it increases power consumption. One solution that was implemented by our protocol is to synchronize the wake-up times of all the nodes in the network.

B. Protocol

The sensor nodes will follow the same program of wireless activity:

1) In the initial phase, the base station(node with address 0) initiates the broadcast procedure. The other nodes are waiting for the broadcast message sent by node 0. If the message reaches its destination, the node's routing table is updated, but if the message reaches only a hop, the node will forward the broadcast message.
2) After the broadcast phase, the node enters in sleep mode.
3) Now, the communication phase starts. Before sending a packet to the destination, we search for a valid route; then we start a collision detection mechanism to check if the channel is ready to be used. Moreover, prior to sending the actual data, we use the RTS-CTS mechanism in order to avoid the hidden station problem. When receiving a packet, each node repeats this process until the packet reaches the destination node.

Another possible communication phase is used for time synchronization. Special time synchronization packets are broadcasted by each node. The neighboring nodes use the information in the received packets in order to calculate the next wake-up time. An average of the wake-up times of the neighboring nodes is used to determine the next time each node will wake-up in the next phase. This can also be applied to synchronize the actual real time clock of each node.

The synchronization of the wake-up times may be a good option for nodes that are processing data before sending it on the network. Depending on the application, this processing can take various amounts of time that can be different from node to node. We can use this aspect to dynamically power down nodes depending on how much work they need to do. However, we can't let the nodes choose an individual sleep schedule because they will start to loose sync. We can enforce limits for the shortest active period and shortest sleep period and then synchronize the wake-up times between neighboring nodes. This is one technique that we investigated and implemented in our protocol.

The previous synchronization method works for relative wake-up times. Each node would schedule a real time clock interrupt to wake it up in the future. However, this is relative to the absolute time of each node. If the clock will get too much out of sync, it might not be possible for the nodes to communicate. Another solution is to synchronize the absolute values of the real time clocks of neighboring nodes. This resembles the use of Network Time Protocol (NTP) for synchronizing the clocks of computing devices.

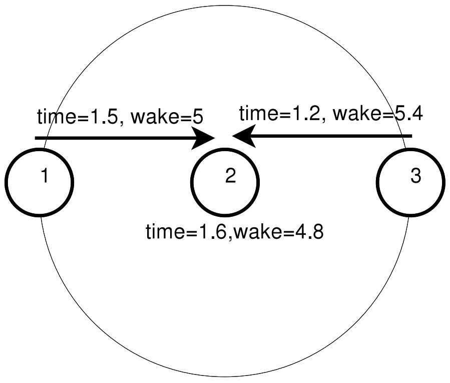

Figure 2. Topology for wake-up synchronization

To calculate the next wake-up time, each node does the following calculations:

$$S = \sum_{n \in neighbours} (n.local\_time + n.wakeup - n.sync\_rcv\_time)$$

$$next\_sleep = \frac{old\_sleep + S}{N + 1}$$

The $local\_time$ is the local time of the node when it sent the message. We must also take into account the processing time and transit time of each message so the receiver will also record $sync\_rcv\_time$ as the local time when the time synchronization message was received.

This is how the algorithm would work for the example topology in Figure 2. Node 2 receives time synchronization messages from his neighbors. We will assume that the transit time of each message is negligible, although our implementation takes it into consideration. With this assumption, the $sync\_rcv\_time$ from the previous equations is equal to the local time of each node.

$$S = 1.5 + 5 - 1.5 + 1.2 + 5.4 - 1.2 = 10.4$$

$$next\_sleep = \frac{4.8 + 10.4}{3} \simeq 5.07$$

These results show that node 2 will have to sleep more than the last time. We have to note that without an absolute clock synchronized between the nodes, errors will affect the resulted wake-up times.

C. Implementation

Node addresses are implemented as 8bit unsigned numbers. There is a special broadcast address of 0xFF used mostly for the routing protocol and time synchronization. This leaves 255 useable addresses for the entire network. However, if an application requires an increased number of nodes, changes to the protocol will be very little to support it. The only drawback of larger addresses is the decrease of available payload space in the data packets.

Listing 1. Frame header
```
struct frame {
  uint8_t ll_src;
  uint8_t ll_dst;
  uint8_t src;
  uint8_t dst;
  uint8_t flags;
  uint8_t ttl;
  uint8_t data[];
};
```

The format of the frame header is described by the C structure in Listing 1. Two types of addresses are used: a link layer address and a network layer address. The link layer address is only valid for communication between local neighboring nodes. At this level, a destination address of 0xFF does not make sense because all the nodes can receive the packet anyway without a need for a broadcast. However, when the intent is to force all the nodes to process the packet, this address can be used. The control packets for time synchronization are an example that use broadcast on the link layer.

Control packets are identified by a non-zero flags field in the frame. Possible flags include `FLAG_RTS`, `FLAG_CTS` and `FLAG_TIME`. They are mutually exclusive and correspond to a single control packet.

The network layer addresses are used to identify the endpoints of a communication. This is where routing takes places. A node that is not addressed by a packet will try to find a next hop towards the final destination. All the information regarding neighboring nodes, routing and time synchronization are kept in a single table. An entry in this table is shown in Listing 2.

Listing 2. Neighbour Entry
```
struct route {
  uint8_t  dst;
  uint8_t  next;
  uint8_t  hops;
  uint32_t this_time;
  uint32_t dst_time;
  uint16_t dst_sleep;
  uint8_t  used;
};
```

We chose to use a static table whose total capacity can be configured before compilation. This increases the the performance of adding and removing operations by not relying on the dynamic memory allocator. We also use less RAM this way. If the size of the future network may became large enough, a custom allocator or even the default malloc() allocator could be used. Currently, each entry in the table is marked by a used flag when in contains valid information. Allocation operations need to go over the table and manipulate this flag.

For the purpose of timing, the ATMega128RFA1 offers a symbol counter that can operate even in deep sleep by using a RTC. We use this counter as local clock for each node and this gives us millisecond resolution for all the time calculations. The time synchronization packets have a payload described by the structure in Listing 3. These values are also millisecond representations of time.

Table I. Clock Drift When Using Only Wake-up Synchronization

Topology	Initial Clock Drift (ms)	Final Clock Drift (ms)
Linear	460	2440
Cross	470	6700

Table II. Clock Drift When Using Global Clock Synchronization

Topology	Initial Clock Drift (ms)	Final Clock Drift (ms)
Linear	483	153
Cross	511	109

Listing 3. Time Synchronization Packet Payload
```
struct time {
  uint16_t dst_sleep;
  uint32_t dst_time;
};
```

The timestamp dst_time is taken immediately before the packet is sent, just like in TPSN[3], in order to decrease the possibility of errors. There are still interrupt and packet processing delays, but we consider these to be negligible.

V. Results

Initially, we have had some issues with the simulation using Avrora. The interrupts for receiving a packet and for waking up from sleep were delivered to the wrong ISR. More specifically, the interrupt executed the next ISR numerically following the correct one in the interrupt vector table.

The wake-up time synchronization algorithm was first tested using a Python simulation script. This allowed us to execute many iterations while introducing random clock drifts. The results of this simulation showed that in time, the clocks of neighboring nodes would not drift more than the maximum artificial errors introduced, provided that the clocks where initially synchronized.

For more accurate results, we ran the simulation using Avrora. We tested the communication and clock synchronization using a linear and a cross-shaped topology made of 5 nodes. The results in Table I show that when only synchronizing relative wake-up times, initial clock drifts are an important source of error. Eventually, nodes will start to loose sync between active periods and messages will get lost. Clock drift is thus amplified in this way.

However, if we first synchronize the absolute clock counters of neighboring nodes we could then apply wake-up time synchronization. As Table II shows, we are able to decrease the initial clock drift. All the values are maximum between each pair of nodes.

VI. Related Work

In [1], various medium access control protocols for wireless sensor networks where compared. The conclusion is that there is no universally accepted or better protocol. It all depends on the final application that uses it. Knowing that, we have made our protocol based on a mix of features. Like S-MAC[4], we use CSMA/CA with RTS/CTS packets only for unicast messages. We also provide clock synchronization between neighbors. However, one drawback of S-MAC is that the listening and sleeping periods are fixed[1]. We propose a separate time synchronization algorithm that allows for dynamic active and sleep periods. There are various other protocols with very specific use, as we shall describe next. The WiseMAC[1] protocol performs better under variable traffic conditions. Thus, the main drawback of WiseMAC is that decentralized sleep–listen scheduling results in different sleep and wake-up times for each neighbor of a node. The TRAMA(Traffic-Adaptive MAC)[1] protocol proposes to increase the utilization of classical TDMA in an energy-efficient manner. The drawback of TRAMA is that all nodes are defined to be either in receive or transmit states during the random-access period for schedule exchanges. This means that without considering the transmissions and receptions, the duty cycle is at least 12.5 percent, which is a considerably high value. Sift[5] is a MAC protocol proposed for event-driven sensor network environments. Very low latency is achieved for many traffic sources, since the energy consumption is a traded-off for latency. One of the main drawbacks is increased idle listening caused by listening to all slots before sending. The second drawback is increased overhearing. When there is an ongoing transmission, nodes must listen until the end in order to contend for the next transmission, which causes overhearing. The principal aim of DMAC[6] is to achieve very low latency for convergecast communications, but still be energy efficient. The drawback is that Collision avoidance methods are not utilized; hence, when a number of nodes that have the same schedule(the same level in the tree) try to send to the same node, collisions will occur.

VII. Conclusion

Wireless sensor networks continue to evolve as new technologies emerge. However, the limited resources still require energy-efficient communication protocols. We have investigated and implemented a MAC protocol with time synchronization among neighboring nodes. We have shown that having a relatively small clock drift can be further used to implement dynamic active periods.

It is difficult, if not impossible, to say which is the best MAC protocol. However, depending on the application, we can choose one that is better suited to our objectives. The implementation presented here focuses more on obtaining energy efficient communication, but it can still be improved. A queuing system could be integrated so that packets that don't reach the destination in time to be resent afterwards from the last node that received them. Also, security is an other area that was not taken into consideration. A malicious node could subvert communications in large areas of the network.

References

[1] Ilker Demirkol, Cem Ersoy, and Fatih Alagöz, Bogazici University. MAC Protocols for Wireless Sensor Networks: A Survey. IEEE Communications Magazine. April 2006.

[2] Ana-Belén García-Hernando, José-Fernán Martínez-Ortega, Juan-Manuel López-Navarro, Aggeliki Prayati, Luis Redondo-López. Problem Solving for Wireless Sensor Networks. Springer-Verlag London Limited 2008.

[3] Holger Karl, Andreas Willig Protocols and Architectures for wireless sensor networks

[4] Wei Ye, John Heidemann, Deborah Estrin An Energy-Efficient MAC Protocol for Wireless Sensor Networks

[5] Jamieson, Kyle and Balakrishnan, Hari and Tay, Y.C. Sift: A MAC Protocol for Event-Driven Wireless Sensor Networks

[6] Lu, G.; Krishnamachari, B.; Raghavendra, C.S. An adaptive energy-efficient and low-latency MAC for data gathering in wireless sensor networks

[7] Avrora: The AVR Simulation and Analysis Framework, http://compilers.cs.ucla.edu/avrora/, Jan 2014