

SMAC Implementation for ETRI SSN-based Sensor Networks¹

Kiran Tatapudi and Chansu Yu

Department of Electrical and Computer Engineering
Cleveland State University
2121 Euclid Avenue, SH 332, Cleveland, OH 44115

SMAC is medium access control protocol which aims at reducing the energy consumption of wireless nodes, thus increasing the overall network lifetime. The protocol identifies collisions, overhearing, control packet overhead and idle listening as the major sources of energy wastage and tries to minimize them by following a duty-cycle based operation. The authors argue that although this may lead to a reduction in the per-hop fairness and latency, the overall performance of the network will not decrease significantly due to the nature of most wireless sensor applications.

This document presents the implementation details of the above mentioned protocol on the ETRI-SSN sensor hardware. These nodes are based on the ATMEL Atmega128 microprocessor and use the Chipcon CC2420 RF transceiver for radio communication.

¹ This study was in part supported by Electronics and Telecommunications Research Institute (ETRI).

TABLE OF CONTENTS

| | |
|--|-----------|
| 1. INTRODUCTION..... | 4 |
| 2. MAC LAYER CORE COMPONENTS | 4 |
| 2.1 SMAC TIMER | 4 |
| 2.2 SCHEDULER..... | 6 |
| 2.3 MAC DATA HANDLER..... | 6 |
| 2.4 APPLICATION INTERFACE FUNCTIONS | 7 |
| 2.4.1 <i>void smac_init(.....)</i> | 7 |
| 2.4.2 <i>smac_send (...)</i> | 7 |
| 2.4.3 <i>smac_getpacket(...)</i> | 8 |
| 2.4.4 <i>extern void smac_sendResult (...)</i> | 8 |
| 3. MAC INTERNALS | 8 |
| 3.1 OVERVIEW | 8 |
| 3.2 MAC LAYER TASKS | 9 |
| 3.2.1 <i>smacNbrScanTask</i> | 9 |
| 3.2.2 <i>smacSyncSendTask()</i> | 9 |
| 3.2.3 <i>smacSendTask()</i> | 10 |
| 3.2.4 <i>smacSendBcastTask()</i> | 10 |
| 3.2.5 <i>smacTransmitTask()</i> | 10 |
| 3.2.6 <i>smacPacketProcessTask()</i> | 10 |
| 4. SMAC SOFTWARE PACKAGE | 11 |
| 4.1 SMAC CODE..... | 11 |
| 4.2 TEST APPLICATIONS | 12 |
| 5. CHANGES..... | 13 |
| 5.1 CHANGES IN VERSION 3.1 | 13 |
| 5.2 CHANGES IN VERSION 3.02 | 13 |
| 6. REFERENCES..... | 16 |

1. Introduction

SMAC [1] is medium access control protocol which aims at reducing the energy consumption of wireless nodes, thus increasing the overall network lifetime. The protocol identifies collisions, overhearing, control packet overhead and idle listening as the major sources of energy wastage and tries to minimize them by following a duty-cycle based operation. The authors argue that although this may lead to a reduction in the per-hop fairness and latency, the overall performance of the network will not decrease significantly due to the nature of most wireless sensor applications.

This document presents the implementation details of the above mentioned protocol on the ETRI-SSN sensor hardware. These nodes are based on the ATMEL Atmega128 microprocessor and use the Chipcon CC2420 RF transceiver for radio communication.

2. MAC Layer core components

2.1 SMAC Timer

The Atmega128 Timer module provides a 16-bit timer/counter [2] module with pre-scaling options. The MAC layer uses this timer module for:

- Duty-cycle operation
- Sync Message broadcast
- Call-backs
- Scheduler

The TIMER1 comparator module ‘A’ is configured to generate interrupts every 500usecs and this defines the basic “slot” period used by the MAC layer for its operation. The slot is the primary timing unit for all the functions MAC functions and is used for defining the protocol parameters such as duty-cycle, SIFS, DIFS periods etc.

Duty-cycle operation

In accordance with the SMAC protocol, the timer operates the radio based on a duty-cycle in the range 1-99. The radio “On” (listening period) is defined as the sum of “smacSyncCW” and “smacDataCW” parameters. The current implementation uses 100 and 900 slots respectively for these parameters and hence has an effective listen period of 500msecs. The sleep period is calculated based on the duty cycle specified and maximum sleep period is 50secs (100,000 slots) at 1% duty-cycle. As a result of the duty-cycle operation, an SMAC node can save considerable amount of energy, For example, a node operating on a 50% duty-cycle can save as high as 50% of its energy.

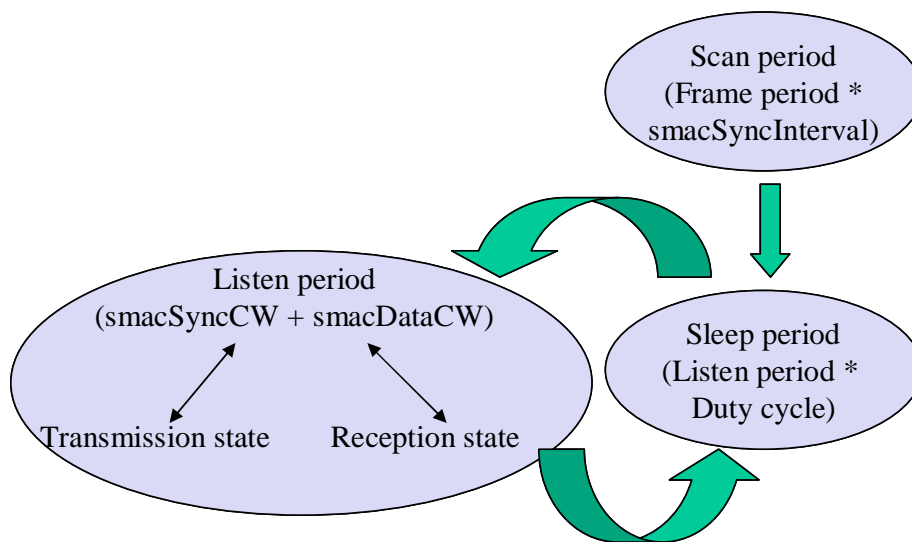


Figure 1: State transition for sleep schedule.

Sync Message broadcast

The SMAC timer also keeps the smac “frame” count and tries to broadcast the node “schedule” at regular intervals as defined by the “smacSYNCINTERVAL” parameter.

Call-backs

The Call-back interface provides the MAC layer functions with a convenient interface to perform time based operations. This allows the functions to register call-back functions which are called after the specified numbers of slots have elapsed. They are executed sequentially in the order of registration. The call-back functions are required to return quickly and are usually used to perform very small operations.

Scheduler

The internal operations of the MAC layer are performed, while allowing the application program to run. This is achieved by implementing a scheduler which periodically checks if the MAC layer needs to perform any tasks. The smac timer starts the scheduler and allows the scheduler to execute any pending tasks.

2.2 Scheduler

The smac scheduler helps the MAC layer to perform its functions while allowing the application program to run. The scheduler implements a FIFO task queue which is prioritized into three levels: Normal, Medium and High. Each of these levels has a queue length of SMAC_MAX_TASKS. The value of this parameter has to be chosen carefully based on the hardware memory constraints. In the current implementation this value is defined to be “5”.

The FIFO task queue is checked for tasks once in every “slot” (500 usecs) and pending tasks are executed based on their priority levels. The tasks once started run to completion and hence cannot be preempted by another task. The completion of a task is signaled by it by setting its status variable to the value SMAC_TASK_DONE. Once, the task status is set to this value, the scheduler in its next cycle removes the task from the queue and executes the next task.

2.3 Mac Data Handler

The “macDataHandler” function handles the data reception over the radio and is called when the FIFOP interrupt from the RF transceiver is active. The function reads the data received over the radio from the CC2420 RXFIFO [3] and creates a task to process it. The current implementation of the function discards a packet if the destination address field and destination PAN address in the MAC header does not match node’s address and pan address respectively.

The function does not wait for the entire MAC frame to be received before reading it from the RXFIFO and thus tries to avoid FIFO overflow under heavy traffic conditions.

2.4 Application Interface Functions

All the application interface functions provided by the MAC follow the following naming convention: “smac” followed by a “_” and the function name. The current implementation provides the following interfaces to the application program:

2.4.1 *void smac_init(.....)*

Must be called before any other functions. This function initializes the various MAC layer parameters, initializes the scheduler and starts the neighbor discovery tasks.

2.4.2 *smac_send (....)*

*UINT8 smac_send (UINT16 pDAddr, UINT16 pDPan, UINT8 pLen, UINT8 *pData);*

This function is used to request the MAC layer to send data to the target node. The function is a non-blocking function and returns immediately. The function is successful if the return value of the function is below SMAC_SENDERR_ERROR. The returned value is the packet id of the requested data. If the value returned by the function is above SMAC_SENDERR_ERROR, the request has failed and the reason for failure can be known by the value returned. These values are defined in the file smac.h as follows:

| | | |
|--------------------------|---|------|
| SMAC_SENDERR_ERROR | - | 0xF0 |
| SMAC_SENDERR_NODEUNKNOWN | - | 0xF1 |
| SMAC_SENDERR_QUEUEFULL | - | 0xF2 |
| SMAC_SENDERR_POOLFULL | - | 0xF3 |
| SMAC_SENDERR_DATATOOBIG | - | 0xF4 |

A success returned by this function however does not guarantee the transmission of data to the destination node. The result of such an attempt is signaled by the MAC layer by the function “smac_sendResult”.

Parameters:

pDAddr: 16-bit smac address of the destination node
pDPan: 16-bit smac Pan-id of the destination node
pLen: size of the data being passed to the function
pData: Pointer to the data

2.4.3 *smac_getpacket(...)*

*BOOL smac_getpacket (SMAC_PACKET *pPkt);*

This function gets a packet received over the radio. The function returns TRUE if a valid packet is available in the buffer. It returns FALSE if the receive buffer is currently empty.

2.4.4 extern void *smac_sendResult (...)*

This function has to be implemented by the application program. It is called by the MAC layer to signal the result of a transmission requested by using the “smac_send()” function.

3. MAC Internals

3.1 Overview

The current MAC layer implementation operates the node based on a duty-cycle as specified by SMAC [1]. The radio is turned ON at the beginning of a new frame and is turned off at the end of the listen period. If a data send is in progress at the end of listen period, the radio switch off is deferred till the end of current transmission. The MAC layer can transmit packets to nodes which follow a wakeup/sleep schedule different to it. For such nodes, the radio is turned on when required and is turned off as soon as the transmission ends. The nodes always follow a single scheduler. However, if other schedules are detected, the nodes transmit their SYNC packets at the beginning of all schedules. Furthermore, for broadcast packets, the data is transmitted during all the schedules.

The MAC layer performs all its functions within tasks, which are executed by the smac scheduler. For example, the `smac_send()` function which is called by the application program to send data to a remote node registers a `smacSendTask` with the scheduler. When this task is executed by the scheduler, it performs the various steps necessary for data transmission over the radio. Similarly, the SYNC messages to be transmitted as specified by SMAC [1] are sent by the `smacSyncSendTask`.

3.2 MAC Layer Tasks

- `smacNbrScanTask()`
- `smacSyncSendTask()`
- `smacSendTask()`
- `smacSendBcastTask()`
- `smacTransmitTask()`
- `smacPacketProcessTask()`

3.2.1 *smacNbrScanTask*

The `smacNbrScanTask` is the first task executed by the scheduler. During the smac initialization, this task is registered with the scheduler. The task keeps the radio “ON” and listens for an entire “`smacSYNCINTERVAL`” period. At the end of the scan period, the task checks if any SYNC messages have been received. The first SYNC message received is used to synchronize with the network and the source of that message becomes the synchronizer for this node. In the absence of any SYNC messages, the node continues its current sleep / wakeup schedule.

3.2.2 *smacSyncSendTask()*

This task is posted by the timer module once every “`smacSYNCINTERVAL`” period. The task broadcasts a SYNC message with its own schedule. If multiple schedules have been detected in the network, the task transmits a SYNC message during the SYNC period of each of the schedules.

3.2.3 smacSendTask()

The smacSendTask() performs the various steps involved in the transmission of data to a destination node. When the task starts, it checks if the destination of the data is a neighbor and waits for the destination node to wakeup. Once, the time has come (based on the destination node's schedule), the task optionally performs (can specified in the packet options) the CSMA/CA procedure. If the successful, the smacTransmit() task is scheduled for the actual transmission of data. Once, the data has been transmitted, the task waits for an acknowledgement and if the packet is not ACK'ed by the destination node within a "smacACKTIMEOUT" period, it repeats the process again. After three such retries, the packet is dropped and the application is notified about the failure. On a successful ACK, the packet is removed from the TX Queue and the application is notified about the success of the transmission.

3.2.4 smacSendBcastTask()

Broadcast messages are destined to all the nodes in the network and hence, if the nodes in the network follow different schedules, the source node is required to transmit the packet during all the schedules. This task handles this special task of transmitting the packet during multiple schedules and performs the same steps as mentioned in the smacSendTask() during each schedule.

3.2.5 smacTransmitTask()

The smacTransmitTask() is responsible to fill the TXFIFO of the CC2420 RF transceiver and start the transmission over the radio. However, this task expects the radio to be switched on before it is called. It can also optionally time stamp the transmitted packet with the slot number and TCNT1 values at the start of SFD.

3.2.6 smacPacketProcessTask()

This task is responsible to process the packets which have been received over the radio by the macDataHandler(). Once, the packet is received, this task is automatically started. On the complete reception of the packet (after the address match), this task examines the packet header and takes specific actions for each type of packet. These can be listed as:

DATA: The packet is checked for duplicity and is put into the RX buffer. If the packet needs to be acknowledged, a call-back is registered which transmits an ACK after SIFS duration.

ACK: Checks if a smacSendTask() is in progress and is waiting for an ACK and sets the for that packet appropriately

SYNC: Updates the neighbor and schedule tables based on the information provided in the packet.

4. SMAC Software Package

4.1 SMAC Code

SMAC code (smac_v3.1.zip) is based on the CSMA principle but enforces nodes to repeat a specified duty cycle (e.g. 60%) of sleep and wakeup periods (frame time of 2666 slots, wakeup during 1000 slots and sleep during 1666 slots). Each node periodically (every 10 frame periods or 26660 slots) transmits its own schedule to its neighbors. When a node starts its execution, it wakes up for 26660 slots so that it can collect information about its neighbors and their sleep schedules. The node then determines its own schedule. Based on the original SMAC design, a node should synchronize its schedule with its neighbors as much as possible so that packet delay is reduced while saving energy.

The smac package (smac_v3.1.zip) has the following structure. The directory smac/ includes all the files implementing the SMAC protocol as described above. The directories node1/, node2/, node3/, node4/ implement test programs.

```
smac/  
  mac/  
    smac.h, smac.c, smac_scheduler.h, smac_scheduler.c  
  hal/  
    etri_ssn.h, hal_atmega128.h, hal_atmega128.c, hal_cc2420.h,  
    hal_cc2420.c
```

node1/
node1.c, makefile
node2/
node2.c, makefile
node3/
node3.c, makefile
node4/
node4.c, makefile

4.2 Test Applications

Node1 and node2 test programs are identical and transmit data to node3 and node4 respectively. They transmit sequential numbers to the destination nodes, i.e., node3 and node4. The node3.c and node4.c are identical and act as sink nodes. They constantly check for any incoming packets and display the incoming data on the serial port. Note that there is no routing protocol is employed because it is simply beyond the scope of MAC functionalities.

Red / Green / Orange LEDs are used to verify the operation of the SMAC protocols. Red LED is to verify the transmission, Green LED is used to verify the reception and Orange LED is used to verify the sleep schedule. The three LEDs are ON and OFF simultaneously at the startup time notifying the scan procedure. Serial ports are also used to verify the data transmission between nodes. Node3 and node4 output the packet numbers for the acknowledged packet from node1 and node2, respectively.

- Red LED: ON when a node transmits a packet and OFF when it confirms its transmission (smacTransmit() in smac.c)
- Green LED: ON when a node receives a header of a packet and OFF when it completely received the packet (macDataHandler() in smac.c)
- Orange LED: ON when a node wakes up and OFF when it sleeps (smacProtoHandler() in smac.c)

Serial ports are also used to verify the data transmission between nodes. Connect node3 and node4 to the development PC via serial cable. Node3 outputs the packet numbers for the acknowledged packet from node1. Node4 outputs the packet numbers for the acknowledged packet from node2. Note that duplicate packet filtering is implemented in this version of SMAC protocol. As in IEEE 802.11 and most of the other CSMA-based MAC algorithms, a node may receive multiple copies of the same packet and can be managed.

5. Changes

5.1 Changes in Version 3.1

- Restructured the smac code
- macDataHandler modified to receive packets in multiple steps
- The smac scheduler has been modified to provide priority levels to tasks. Also, the scheduler now does not require the tasks to return before the slot completion.
- The slot duration has been changed to 500usecs from 320usecs
- The smacReceiveTask() has been removed. The functionality is now implemented in the smacPacketProcessTask()
- Interrupt handling has been improved.
- The test applications have been re-written. They now transmit packets at full speed.

5.2 Changes in Version 3.02

We have a list of features that need to be implemented in a future version of SMAC implementation. They are: Duplicate packet filtering, Receive/Transmit priority, Backoff after its own transmission, Schedule synchronization and Parameter tuning. Features implemented in version 3.02 are noted accordingly.

- Duplicate packet filtering mechanism: In wireless communication, it is possible that a receiver receives the same packet more than once, which is called duplicate packets. For example, when node *A* transmits a packet to node *B*, node *B* successfully receives the packet and replies with ACK to node *A*. In this case, if the ACK packet doesn't go through, node *A* will retransmit the packet again assuming there was a packet drop. Therefore, node *B* will receive the same packet twice. Such duplicate packets can be filtered out within the receiver MAC similarly specified in IEEE 802.11 MAC. It defines a duplicate packet filtering mechanism, that matches the sender address (*Addr2* in Fig. 2) and the sender-generated *sequence control number* (*SC*) of a new packet against those of previously-received ones. If there is a match, the receiver transmits ACK but does not forward the packets. In IEEE 802.11, the *SC* is a 12-bit field which we think is more than enough in most of SMAC applications.

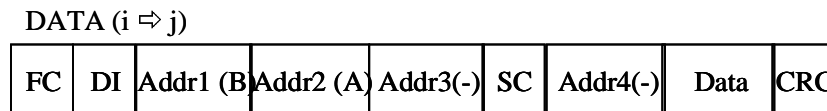


Figure 2: Format of MPDU frames in the IEEE 802.11 MAC (MPDU: MAC protocol data unit, FC: Frame control, DI: Duration/ Connection ID, SC: Sequence control).

- Receive/Transmit priority: In the current SMAC implementation, a node ignores an incoming packet if it is in the middle of preparing to transmit its own packet. This is to ensure that a node does not starve and has a chance to process its own packets even in the case of cascade of incoming packets. In other words, packet transmission has a higher priority than packet reception when both of them occur simultaneously. However, since a node cannot utilize the medium anyway when it sees an on-going communication, it seems not a good idea from the throughput point of view. We'll make a node to receive packets and acknowledge them even if it has packets to transmit. Additionally, fairness issue should be considered. ➔ The SMAC code has been modified to give a higher priority to an incoming packet. The frozen-backoff mechanism addresses the fairness problem.

- Backoff after its own transmission: When a node sees a free medium, it waits for DIFS and transmits its packet. However, when a node sees a busy medium, it waits for the current communication to be done, waits for DIFS, and randomized backoff before transmitting its packet. What if a node just finishes transmitting its own packet and has another packet to transmit? Does it wait for DIFS or DIFS + backoff? The current version implements the former but it should be the latter, which will be corrected in a later version. → The code has been modified to make a random backoff in the case of consecutive packet transmissions.

- Schedule synchronization: Every node is supposed to broadcast its sleep schedule every SCAN interval (255 slots). When a node starts its execution, it wakes up for the SCAN interval so that it can collect information about its neighbors and their sleep schedules. The node then determines its own schedule by synchronizing it with its neighbors as much as possible. In the current implementation, each node goes through the scan procedure but it does not periodically broadcast its schedule. → All the nodes periodically (Every 4th cycle) transmit their schedules in a “SYNC” packet. Each “SYNC” packet contains the following information:
 - The node Address
 - The number of slots for which the node is going to be awake
 - The initial sequence number

A node on hearing the SYNC packet updates its schedule table with the information. This information along with the time of arrival of the packet is used to synchronize with a particular node. A node automatically synchronizes with the first schedule it hears during the scan phase. And nodes wake up at a non-scheduled time to transmit a packet to a node which has a different schedule. However, data transmission at very high rate for a long period of time will cause a very large time skew which might lead to loss of synchronization.

- Parameter tuning: MAC protocol implementation requires decisions on many parameters such as slot time (320usec), DIFS (3 slots), SIFS (1 slot), backoff window

(30 slots), sleep-wakeup time (25 and 25 slots), SCAN (255 slots), etc. We need to fine tune those parameters to optimize the performance.

- Features that are not considered: RTS/CTS handshake and exponential backoff are not considered to be implemented at this time of writing.

Modifications in test applications in version 3.02:

- The nodes now use the `smacGetPacket()` function to check if a packet has been received and retrieve it. (The `smac_dataReceived()` call-back function is still called by the MAC layer, however the data can be retrieved only by calling this new function.)
- Node1 transmits data packets (Full Payload length) at an interval of two `halWait(50000)`'s.

6. References

[1] An Energy-Efficient MAC Protocol for Wireless Sensor Networks., W Ye, JS Heidemann, D Estrin - INFOCOM, 2002 - ieeexplore.ieee.org

[2] ATmega128(L) Preliminary Complete Datasheet

[3] Chipcon CC2420 Data Sheet