Cleveland State University
Department of Electrical and Computer Engineering

Bulk Synchronous Medium Access (BSMA) for Mote-based Sensor Networks

Electrical Engineering Technical Report
TR – 05 - 101

Sridhar Kalubandi and Chansu Yu

January 24, 2005

2121 Euclid Avenue, Stilwell Hall 332
Cleveland, Ohio 44115-2425
http://www.csuohio.edu/electrical_engineering/

Bulk Synchronous Medium Access (BSMA) for Mote-based Sensor Networks[1]

Sridhar Kalubandi and Chansu Yu

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

1. Research overview

Energy efficiency is the chief concern in sensor networks as they have limited power and it often requires tradeoffs with throughput and latency when designing a *medium access control* (MAC) protocol. The goal of this research is to maximize energy savings based on *sleep mode* operation and *Transmit Power Control* (TPC) to save power in Mote-based sensor networks. Two key objectives are (i) to review and evaluate existing MAC protocols developed for Mote such as *CSMA MAC* (default in Mote), *TDMA MAC* (T-MAC, Univ. Virginia), and *Sensor-MAC* (S-MAC, USC) based on IEEE 802.11 concept with sleep/wake and (ii) to develop an energy efficient MAC solution, called *Bulk Synchronous Medium Access* (BSMA). Planned future work include expanding BSMA protocol based on TPC (using PA_POW register) and RSSI (*received signal strength*).

2. Related Work

Common methods for MAC protocols in sensor networks are random access and TDMA. Prominent sources for energy inefficiency at *medium access control* (MAC) are collisions, idle listening, overhearing, and control packet overhead [1].

- Collisions: the packets involved and control overhead needs to be retransmitted
- Idle listening: the radio is in listening mode in anticipation of being solicited instead of being in sleep mode.
- Overhearing: the radio listens promiscuously on the channel receiving packets addressed to other nodes when it could be in sleep mode.

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

- Control packet overhead: the headers for the packets and acknowledgement packet, which is used by most MAC schemes, contribute to the overhead.

They are quite common in random access MAC protocols proposed for ad hoc networks. Control overhead is the only significant factor in TDMA-based MAC schemes, as it requires setting up and maintaining schedules. TDMA works best when there is no severe variation in traffic intensity. Also, packets may be delayed until the node's turn to transmit to the receiver comes up. Since energy efficiency is crucial for sensor networks, the main approach is to maintain a very low duty cycle with less control overhead.

Power save mode in IEEE 802.11

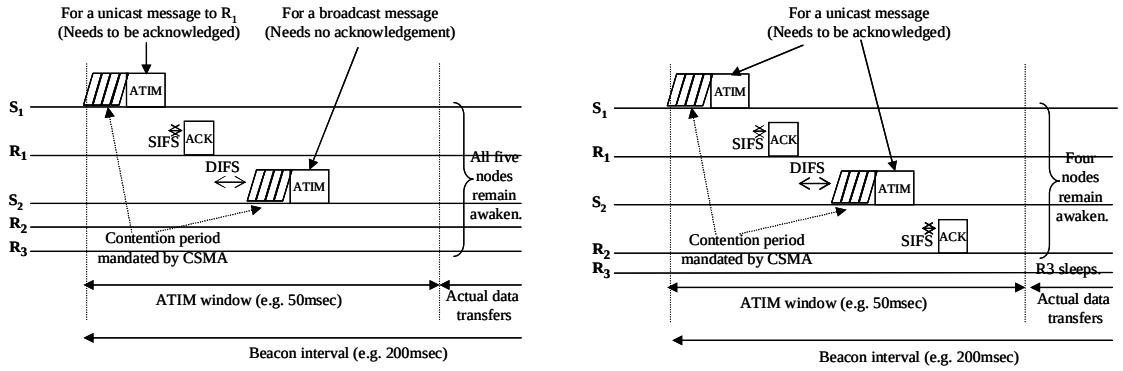
According to IEEE 802.11 standard [10], there are two modes of operation depending on the existence of an AP. They are referred to as the *Distribution Coordination Function* (DCF) and the *Point Coordination Function* (PCF). The DCF uses a contention algorithm to provide access to all traffic based on the *Carrier Sense Multiple Access with Collision Avoidance* (CSMA/CA) and delay known as *InterFrame Space* (IFS). The PCF is an optional access method implemented on top of the DCF and provides a contention-free service coordinated by an AP. In the PCF, the AP has higher access priority than all other mobile nodes, and thus, it not only sends downlink packets but also controls uplink packets by polling stations [9].

Power saving in the PCF mode is simply achieved by the coordination of the AP. Every mobile node switches its radio subsystem off whenever possible. The AP periodically sends beacon signals to synchronize with other mobile nodes and informs them whether they are addressed or not using *Traffic Indication Map* (TIM), which is included in the beacon in the form of a bitmap vector. The DCF mode of operation achieves the synchronization among the nodes in a distributed way[2]. A beacon interval consists of actual data transmission period followed by the advertisement, called *Ad hoc TIM* (ATIM) *window*, and packets are buffered at the sender node and are directly transmitted to the receiver if its advertisement was successfully acknowledged. Data transmission in PS mode saves energy but it takes longer time due to the synchronization overhead and the TIM/ATIM window size [8]. Note that both PCF and DCF assume that every node is within every other's radio transmission range and are not directly applicable in multihop mobile networks.

Fig. 1 shows the activities during the ATIM window with an example mobile network of five nodes, S_1, R_1, S_2, R_2, and R_3. In Fig. 1(a), node S_1 has a unicast packet to node R_1 and

[2] Tseng et al. [13] and Huang and Lai [11] studied the clock synchronization problem but we do not discuss this issue in detail in this paper and assume all mobile devices operate in synchrony using one such algorithm.

advertises it during the ATIM window. Node S_2 has a broadcast packet and advertises it during the same ATIM window. Note that advertisements are made based on competition using the CSMA/CA principle with the backoff procedure and IFS period as specified in IEEE 802.11 standard. Also note that the ATIM announcement from node S_1 needs acknowledgement from the intended receiver, for example node R_1. However, the broadcast announcement from S_2 does not need to be acknowledged. In this scenario, all five nodes remain awaken during the entire beacon interval in order to receive the unicast and/or broadcast packets. In Fig. 1(b), node S_2 intends to send a unicast packet to R_2, and thus nodes S_1, R_1, S_2, and R_2 must be awaken but node R_3 can return to sleep immediately after the ATIM window because it does not have any packet to receive. It is important to note that node R_3 should remain awaken if overhearing is mandatory.



(a) One unicast and one broadcast packet (all five nodes remain awaken during the entire beacon interval). (b) Two unicast packets (all nodes except node R_3 should remain awaken during the beacon interval).

Figure 1: PS mechanism in IEEE 802.11 (SIFS: Short IFS, DIFS: DCF IFS).

Recently, the abovementioned PS mode-based power saving mechanism has been studied not only to investigate the tradeoff between the energy and network performance but also to apply it in multihop environment. Woesner et al. evaluated the PS mechanism in one-hop ad-hoc scenario [8]. Since ATIM window may become wasted if only a few packets need to be advertised, they concentrated on discovering the optimal size of ATIM window for a given beacon interval. They suggested that the ratio between ATIM window and beacon interval should be around ¼ and that the ATIM window size should be dynamically adjusted depending on network traffic while the standard states it should be a constant [8]. Krashinsky and Balakrishnan proposed the *Bounded Slowdown* (BSD) protocol that dynamically adapts the beacon interval based on network condition in an AP-based infrastructured wireless networks [14]. They showed that the BSD reduces the delay of a TCP-based application while

simultaneously reducing energy consumption. Similarly, Jung and Vaidya presented the *Dynamic Power Saving Mechanism* (DPSM) that dynamically adjusts the ATIM window based on network traffic in a one-hop network [15].

On the other hand, there have been research efforts to exploit the PS mode in multihop networks. SPAN [12] is a unique approach in that it mandates a set of nodes to be in AM mode while the rest of the nodes stay in PS mode. Nodes in AM mode offer the routing backbone so that any neighboring node can transmit to one of them without waiting for the next beacon interval. Two major drawbacks of this scheme is that it degenerates to all-AM mode scenario in sparse networks allowing no energy savings regardless the traffic pattern and the effect of routing overhead is not considered by employing Geographic routing with the assumption of availability of location information. Zeng and Kravets suggested a similar approach, called *On-Demand Power Management* (ODPM), in which a node switches between AM and PS mode based on communication events and event-induced timeout values [16]. For example, when a node receives a RREP packet, it is better to stay in AM for more than one beacon interval (timeout) hoping that there will be data packets to be delivered in the near future. This scheme asks for the MAC layer software to understand the semantics of routing layer messages and a completely different set of timeout values must be used for a different routing algorithm. In addition, they should be determined based on traffic patter, which is lacking in ODPM. For instance, a small timeout value in sporadic traffic condition brings no benefit because a node already went back to PS mode after a small timeout when the next packet arrives. Therefore, the MAC layer design should be well-tuned as well as well-combined with the underlying routing layer protocol.

In summary, PSM defined in IEEE 802.11, BSD and DPSM perform well in one-hop networks but they are not directly applicable in multihop networks. Span and ODPM are targeted for multihop networks but they either do not consider the routing complexity or lack the adaptive capability in determining the protocol parameters, which limit their application in real scenarios.

Power Aware Multi-Access protocol with Signaling (PAMAS)

The PAMAS [17] is an energy efficient media access control protocol for ad hoc networks. This protocol uses a separate *signaling channel* apart from the channel used for data transmission. The RTS/CTS messages are transmitted using this separate channel.

Data transmission protocol works as follows. When a node has a packet to transmit, it sends a RTS packet and waits for the CTS packet to be received. If the node gets the CTS packet, it starts transmitting the data packets. Otherwise, it goes into exponential backoff. The receiving node waits for the data packets after transmitting the CTS packet. If the packet does not arrive

within a particular amount of time, it goes to the idle state. If the packet starts arriving, the node transmits a busy tone over the signaling channel and receives the packet. When the receiving node hears a transmission of RTS message over the signaling channel, it transmits the busy tone of duration equal to double that of CTS message. Thus, the CTS message the neighbor is expecting will collide with the busy tone and is lost. This is how the receiving node makes sure that no other transmission interrupts its data reception.

PAMAS achieves the goal of energy efficiency by turning the nodes' power off by themselves whenever they need to. A node can do so under two situations: (i) if it has no packets to transmit and one of its neighbors starts transmitting to somebody else, or (ii) if one of its neighbors is receiving. The node can know that one of its neighbors is transmitting by hearing the transmissions over the data channel. It can also know that a neighbor is receiving a data packet by listening to the busy tone transmitted over the signaling channel. But the duration for which the node should be powered off is decided by the *probe protocol*, where the node sends query messages (*t_probe*) to transmitters over the signaling channel to estimate the duration [17].

Prototype Embedded Network (PEN) Protocol

As in SPAN, the PEN protocol [18] exploits the low duty cycle of communication activities and powers down the radio device when it is idle. However, unlike SPAN, nodes interact "*asynchronously*" without master nodes and thus, costly master selection procedure as well as the master overloading problem can be avoided. But in order for nodes to communicate without a central coordinator, each node has to periodically wake up, advertises its presence by broadcasting beacons, and listens briefly for any communication request before powering down again. A transmitting source node waits until it hears a beacon signal from the intended receiver or server node. Then, it informs its intention of communication during the listening period of the server and starts the communication. Fig. 2 shows those source and server activities along a time chart.

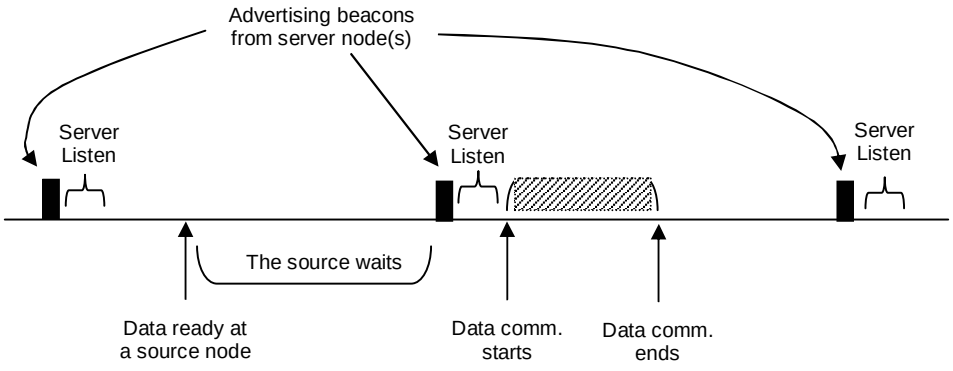


Figure 2: Source and server node activities in PEN.

Route discovery and route maintenance procedures are similar to those in AODV, i.e., on-demand route search and routing table exchange between neighbor nodes. Due to its asynchronous operation, the PEN protocol minimizes the amount of active time and thus saves substantial energy. However, the PEN protocol is effective only when the rate of interaction is fairly low. It is thus more suited for applications involving simple command traffic rather than large data traffic.

Sensor-MAC (S-MAC)

S-MAC [1] follows a random access model similar to IEEE 802.11 by having RTS-CTS-DATA-ACK sequence. For energy savings, it sets the radio off during the transmissions of other nodes as in PAMAS. However, unlike PAMAS, it only uses in-channel signaling instead of using an additional control channel. It reduces energy consumption by having each node sleep for some time and then wake up and listen to see if any other node wants to talk to it. If the corresponding durations are half second and half second (50% duty cycle) as depicted in Fig. 3, it can achieve close to 50% energy savings. Nodes do not need to synchronize among themselves because, for example, if node A wants to talk to node B, it just wait until B is listening assuming node A has information about node B's listen/sleep schedule. And, after they start data transmission, they do not follow their sleep schedules until they finish transmission.

All nodes are free to choose their own schedules and broadcast them in a SYNC message, indicating when they will go to sleep. However, it is preferable for neighboring nodes to have the same schedule in order not to wait and thus reduce packet latency. For this reason, if a node receives a schedule from a neighbor before choosing its own schedule, it follows that schedule by setting its schedule to be the same. In a multihop network, it is possible that a node receives a different schedule after it selects and broadcasts its own schedule. In this case, which is expected to be rare, the node adopts both schedules.

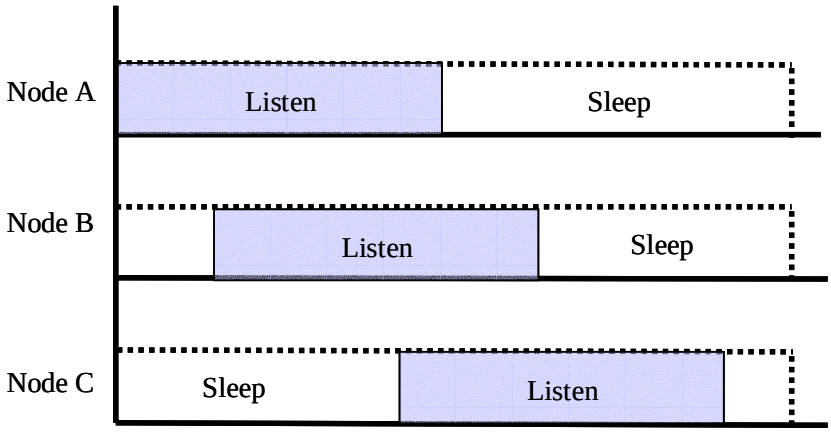


Figure 3: Period listen and sleep in S-MAC. (Nodes do not need to synchronize but try to do so among neighbors by transmitting and receiving SYNC messages containing the listen/sleep schedule.)

Selection of sleep and listen duration is based on application requirements and each node stores a schedule table that contains the schedules of its known neighbors. Adaptive listening is used to reduce latency. When a sensing event occurs, it is desirable to reach its destination with minimum delay. Hence when a node hears transmissions from its neighbors, it wakes up for a brief period after the transmission instead of waking up at its scheduled wake up period if it is the next hop. Collisions are avoided based on RTS and CTS as in IEEE 802.11. Energy consumption due to unnecessary overhearing can be avoided by having all immediate neighbors of sender and receiver go to sleep for the duration specified in *Network Allocation Vector* (NAV) included in RTS and CTS packets, which is also used in IEEE 802.11.

Timeout-MAC (T-MAC)

T-MAC [6] tries to improve over S-MAC (fixed duty cycle) as well as classic CSMA (no duty cycle) by introducing *dynamic duty cycle* to further reduce idle listening periods while efficiently handling load variations in time and location (see Fig. 4). The main idea of the T-MAC protocol is to reduce idle listening by transmitting all messages in bursts and to end the active listen period by timing out on hearing nothing. More specifically, an active period ends when no activation event has occurred during the current maximum possible active duration. An activation event is:

- the firing of a periodic frame timer;
- the reception of any data on the radio;
- the sensing of communication on the radio, e.g. during a collision;
- the end-of-transmission of a node's own data packet or acknowledgement;

- the knowledge, acquired through overhearing prior RTS and CTS packets, that a data exchange of a neighbor has ended.

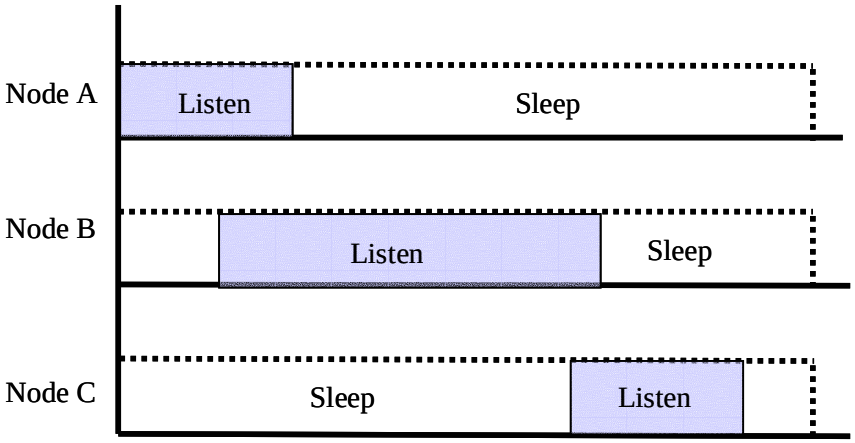


Figure 4: Dynamic duty cycle in T-MAC. (Nodes have different duty cycle depending on their communciation requirements.)

T-MAC also considers the *early sleeping problem*, which typically occurs in sensor networks, where most of communication traffic is unidirectional from nodes to sink. In other words, a node in the region of CTS of the winner of the contention is unable to send RTS to its destination as in Fig. 5(a) and thus the intended destination of the node duly goes into sleep, which makes the transmission delayed considerably. Two possible solutions have been suggested. The first solution is that the node in the CTS region sends a *Future RTS* (FRTS) to its destination after it hears CTS as shown in Fig. 5(b). Collision of data at the original receiver is prevented by having the sender send a small *Data-Send* packet, which is equal to the size of FRTS and will result in collision preventing the collision of data which is sent subsequently. This solution results in increase of throughput along with usage of energy. Another solution to the problem is to use *full buffer priority*, i.e. when a node's transmit/ routing buffers are full, it would prefer sending to receiving as in Fig. 5(c). This means that when a node receives an RTS destined for it, it immediately sends its own RTS to another node instead of replying with a CTS. The solution however becomes dangerous in a high load environment where chances of collision increase rapidly.

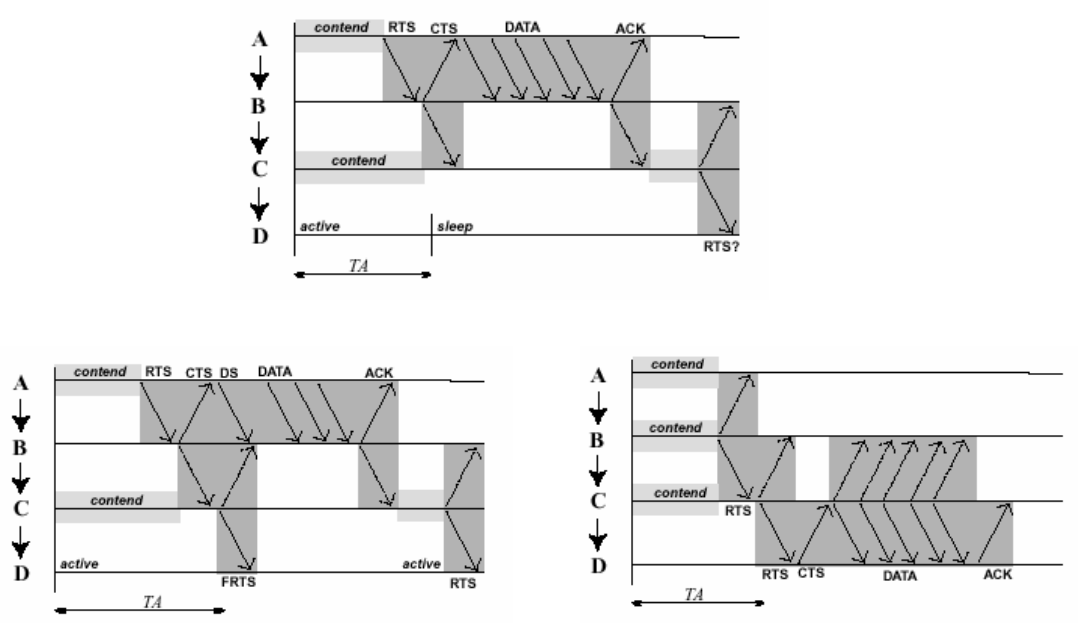


Figure 5: Early sleeping problem and two solutions [6]. (a) Early sleeping problem. (Node D goes to sleep before C can send an RTS to it.) (b) Future RTS. (FRTS keeps node D awake.) (c) Full priority buffer (Node D takes priority upon receiving RTS.)

Traffic-Adaptive Medium Access Protocol (TRAMA)

While all the previous schemes are contention-based, TRAMA [4] is a schedule-based TDMA algorithm which offers collision-free access to the medium. Energy efficiency is thus attained by avoiding unnecessary communication due to collisions and by having nodes switch to sleep state when there is no packets intended for those nodes. In TRAMA, time is organized as random-access periods and scheduled-access periods. During the random-access periods, all nodes must remain awaken and exchange neighbor information to obtain consistent two-hop topology information (*Neighborhood Protocol* or NP). Since nodes perform contention-based channel acquisition during this period, TRAMA is not entirely a collision-free protocol.

A scheduled-access period is divided into a number of *transmission slots*, which are used for collision-free data exchange as well as for schedule propagation. A node has to announce its transmission schedule, based on packet rate requirement produced by the higher level application and the corresponding packet interval, using *Schedule Exchange Protocol* (SEP) before starting actual transmissions. A node then pre-computes the number of transmission slots in the interval for which it has the highest priority among its two-hop neighbors. Here, the priority of node 'u' at time slot 't' is defined as the pseudo-random hash of the concatenation of the node's identity

and ‘t’. SEP maintains consistent schedule information across neighbors and updates the schedules periodically.

Since a selected node may give up its transmission slot if it does not have any packet to send, this slot could be used by another node. Nodes exchange current traffic information with their neighbors to make effective use of transmission slots. *Adaptive Election Algorithm* (AEA) is used to find the winner of the slot and reuse of unused slots. At any given transmission slot, a node is in transmit node if it has the highest priority among its contending set and it has data packet to send. It is in the receive mode if it is the intended receiver of the current transmitter. Otherwise, the node switches off to conserve power.

On-Demand TDMA Scheduling

Busy Tone On-demand Scheduling (BTODS) and *On-Demand Scheduling* (ODS) are two on-demand TDMA MAC protocols that schedule flows of sensor traffic in non-interfering slots [3]. The difference between the two is that BTODS uses non-interfering channels to transmit busy tones while ODS uses the same channel to inform the current traffic. Since it assumes that packets are sent from a sensor at a deterministic, period rate and the rate changes infrequently, it focuses on scheduling *flows* instead of individual packets where a flow is defined as a MAC level flow where data packets are sent periodically.

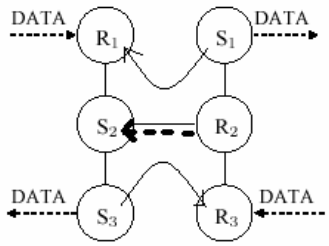


Figure 6: A new flow (S_2 to R_2) is scheduled in addition to two existing flows (S_1 to R_1 and S_3 to R_3) [3].

In BTODS and ODS, time is organized into advertisement period and actual data transmission period as in power saving mechanism of IEEE 802.11. When a node has a new flow to schedule, it must advertise it during the advertisement period by sending FLOW-ADV packet which must be acknowledged by FLOW-ACK again as in IEEE 802.11 (ATIM packet and the corresponding ACK). However, unlike IEEE 802.11, the data transmission period is slotted based on the pre-computed schedule agreed among the neighbors. In this sense, these schemes are similar to TRAMA but they do not require nodes to maintain consistent two-hop

neighborhood information, which can be a source of control overhead. In Fig. 6, a new flow (S_2 to R_2) is added. After S_2 and R_2 have completed their FLOW-ADV/FLOW-ACK exchange, both will remain awaken until they can find an open slot in which S_2 can send a packet and receive and ACK from R_2 without interfering with existing flows. BTODS requires the sender S2 to refrain from attempting to schedule and send a data packet in any slot in which it detects a busy signal. In ODS, since busy tones are not used, extra periods are added for nodes to indicate they will be busy sending or receiving in the current slot.

ZigBee (IEEE 802.15.4)

ZigBee [19] is poised to become the global control/sensor network standard and defines the network, security, and application framework for an IEEE 802.15.4-based system [5], which covers PHY and MAC layer protocols. The MAC layer mechanism defined in IEEE 802.15.4 is similar to IEEE 802.11 in that it basically supports CSMA/CA-based contention access but optionally provides contention-free access to the medium. In IEEE 802.11, a super-frame consists of contention section (DCF) and contention-free (PCF) sections. In IEEE 802.15.4, they are called *Contention Access Period* (CAP) and *Contention Free Period* (CFP), respectively, where *Guaranteed Time Slots* (GTSs) comprise the CFP as shown in Fig. 7. Time synchronization is achieved by using frame beacons which are emitted at super-frame duration intervals and the nodes that do not have anything to send/receive need to awake only during the beacon resulting in very low duty cycle.

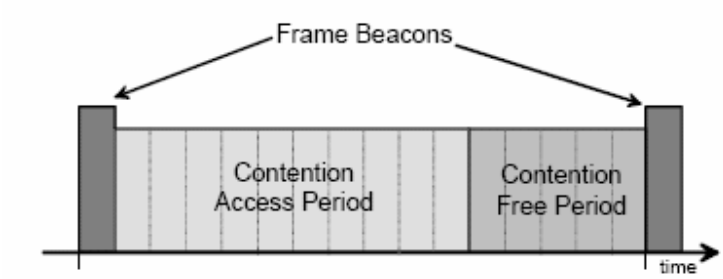


Figure 7: A super-frame with Contention Access Period (CAP) and Contention Free Period (CFP), where GTSs comprise the CFP [19].

IEEE 802.15.4 provides application support for creation of two wireless network topologies; Star for home networking and Peer-to- Peer (P2P) for industrial and commercial applications. Management of these networks is done in network layer. Two types of devices are identified, *Full functional device* (FFD) and *Reduced functional device* (RFD). In Star topology,

Personal Area Network (PAN) coordinator that sends controls the communication by sending beacons periodically for device synchronization. A FFD can establish its own network by becoming a PAN coordinator, it also sends beacons, selects network id and allows association. It also has a non-beacon mode in which beacons are for association purposes only. ZigBee networks are primarily intended for low duty cycle sensor networks with low power consumption, which comes from the ability to quickly attach information, detach, and go to deep sleep.

3. BSMA (Bulk Synchronous Medium Access): MAC Protocol for Sensor Networks

Based on the extensive survey, we found that CSMA-based MAC protocols such as IEEE 802.11, IEEE 802.15.4, PAMAS, S-MAC and T-MAC may be good at light load but incur a large overhead due to collision at moderate load. TDMA-based protocols such as TRAMA, BTODS and ODS are collision-free but require a large amount of scheduling overhead. This is the motivation of our research to develop BSMA that avoids energy consumption due to collisions, overhearing and idle listening by adopting a TDMA principle with a simple scheduling algorithm. A key idea is to assign time slots to sensor nodes, which is done not individually but in "bulks."

BMAC is a TDMA protocol, which takes advantage of spatial reuse. Nodes need not be bothered about transmissions beyond two-hop environment as they neither interfere nor are interfered by them. It takes a two-step approach for forming a schedule, first by identifying non-interfering concentric rings around the sink and assigning BIGSLOTs to them. These BIGSLOTs are in turn subdivided into multiple slots to avoid interference within the concentric region. In other words, a set of nodes listen (and receive) in BIGSLOT 0, a second set of nodes listen (and receive) in BIGSLOT 1, and so on. And, mutually exclusive sets are determined based on the hop count from a "token" node (sink), which periodically transmits beacons. Beacon period is determined based on traffic density but in sensor network, we assume that sensing activity happens at a known interval. Receiver acknowledges the received packet with ACK packet if it receives the packet correctly. Once the TDMA mechanism starts, if any node has much larger number of children nodes than its neighbors then it can initiate transfer of some of these nodes to its neighbors wherever possible. It helps in obtaining an even distribution of traffic resulting in longer network lifetime.

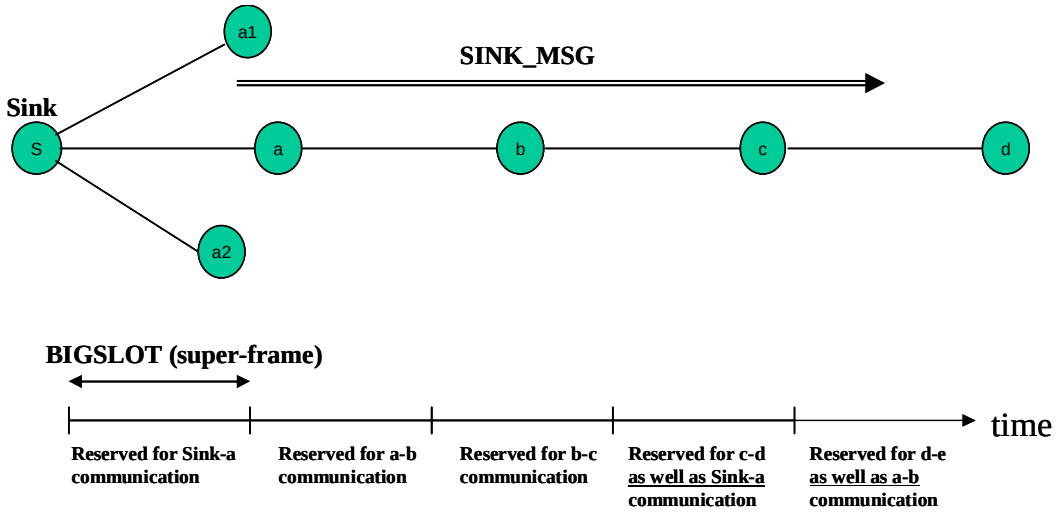


Figure 8: BSMA mechanism.

Bulk Synchronous Mechanism

The sensor nodes and the sink once deployed have to form a topology centered at the sink. The sink sends the initial beacon message, SINK_MSG packet, containing a timestamp that is forwarded towards the periphery by the sensor nodes. The communication between the nodes before the schedules are drawn up is through random access. Each node upon receiving the SINK_MSG notes the hop count and the time slot to find its own BIGSLOT period, which is the nearest super-frame the node can participate. Subsequent super-frames can be computed by adding 3×super-frame-size. It then increments the hop count and forwards the SINK_MSG. If a node receives same hop-count from two nodes, it chooses the one with higher signal strength. This ensures better quality link and more tolerance to noise. Refer Fig. 8 for details.

In Fig. 9 nodes A and H are beyond each other's range. Suppose node J receives SINK_MSG from node H first and does not consider signal strength, it chooses node H as its parent. It may receive later SINK_MSG from nodes A and C but with same hop-count. Ideally node J should be child node for A or C. Now node H may suffer from interference from node A and may interfere with transmissions to node A from its child nodes. Though relocating the slots will solve this problem, shorter and more tolerant links are desirable. Hence, signal strength is also taken into consideration while determining the parent node.

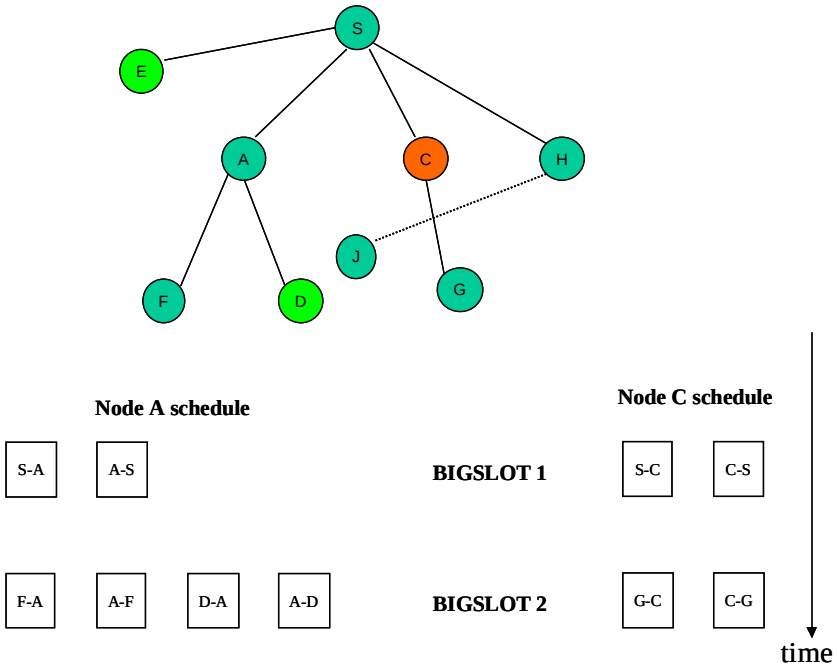


Figure 9: BIGSLOT allocation in BSMA.

Beacon period is determined based on traffic density but in sensor network, we assume that sensing activity happens at a known interval. A node waits for synchronization period during which it gets the SINK_MSG from the sink being forwarded by different nodes. It selects the node with least number of hops to the sink as its parent node and informs the selected node by PARENT_SEL_MSG. This forms stage 1 of the BSMA algorithm. The following figure describes how node d in Fig. 8, which is four hops away from the sink, calculates its next BIGSLOT period from the hop-count and original timestamp in the SINK_MSG.

Since we have a sink-oriented network, nodes only need to have active links from and to their parent node and child nodes. Once a node receives PARENT_SEL_MSG it adds the sender to its list of child nodes. Whenever the node receives SINK_MSG or PARENT_SEL_MSG it adds the sender to its neighbor list. Parent node schedules the links to its child nodes during its assigned BIGSLOT period. Hence, a node while scheduling can ignore links to its neighbors who are not its child nodes. The child nodes that are not leaf nodes repeat the process. The information gathered during the random access period regarding the neighbors is retained; it is useful during any future rearrangements.

Fig. 10 shows the pseudo code for implementing BSMA protocol. It consists of two stages: In stage 1, the nearest super-frame (or BIGSLOT) the node can participate as parent is determined. Each node forms a schedule for its child nodes based on the neighbor information,

SCHEDULE_PKT heard from neighbors and PARENT_SEL_MSG messages. Initially it finds if it has heard any SCHEDULE_PKT from its neighbors in the same level. If it has then it schedules its child nodes in the remaining part of the BIGSLOT. If no SCHEDULE_PKT has been heard, the node can choose to position its slots anywhere in its BIGSLOT period. Contiguous slots are desired for a parent node as it helps in reducing the costs of switching on and off multiple times. It can be done by obtaining *n* contiguous slots in the slot table, where *n* is at the minimum number_of_childnodes*2.

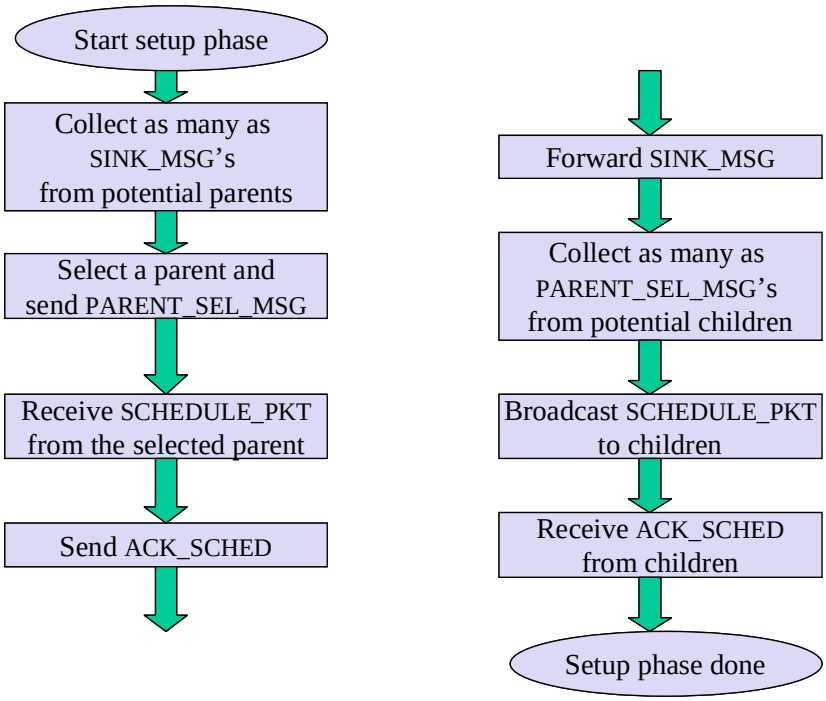
Finding such contiguous zones will not be difficult as the protocol aims at very low duty cycles, bandwidth available is larger than the traffic offered and the whole BIGSLOT duration is divided only within the two-hop environment beyond which it can be reused. Once the node decides on the schedule it advertises it in its neighborhood using SCHEDULE_PKT. If it receives a SCHEDULE_PKT from its neighbor which conflicts with its allocation, the node with lower address gives up and calculates and advertises a new schedule. It waits for acknowledgement from its child nodes in the form of ACK_SCHED. If a node receives a SCHEDULE_PKT from its parent node it replies with ACK_SCHED, other nodes overhearing can use the information present to form their own schedules, which do not conflict. If the node sending SCHEDULE__PKT times out, it re-advertises a few times. Eventually if no ACK_SCHED is received from one or few of the nodes it adjusts the schedule to reflect the changes and sends a new SCHEDULE_PKT. Once all ACK_SCHED are received, the nodes move into TDMA mode. In Fig. 9 nodes E, A, C and, H are in BIGSLOT2. Node A has F and D as child nodes and node C has G as its child node. Links to and from S to nodes A and C are scheduled in BIGSLOT1. Nodes A and C are responsible for sharing BIGSLOT2 among their neighbor. The figure shows the slots shared by them in the BIGSLOT2 on a mutually exclusive basis using stage 2 of BSMA.

```
/* Stage 1
In this stage the super-frames are assigned to regions demarcated using hop count */

void establish_super_frame (int hop_count, real original_timestamp)
{
        // to align according to multiples of super_frame_size
        base = present_time_stamp - (super_frame_size – (present_time_stamp % super_frame_size ) )
        immediate_frame_pos = (present_time_stamp – original_timestamp)/super_frame_size % 3;
        original_region = hop_count % 3;
        shift = find_shift ( original_region, immediate_frame_pos);
        next_frame_start = base + shift * super_frame_size;
}

int find_shift (original_region, immediate_frame_pos)
{
        switch (original_region – immediate_frame_pos)
        {
                case 0: return 0;
                case –2:
```

```
                        case 1: return 1;
                        case –1:
                        case 2 : return 2;
                }
}

/* Stage 2
In the region where the super-frame is allotted it is subdivided into finer fixed size slots, which are shared in the two-hop environment */

/* every parent node runs form_schedule to draw up a schedule with its child nodes and advertises it to its neighbors*/

void form_schedule (slot slot_table [] )
{
        slot_iter = 0;
      slot reserved_slots [num_slots];
        for ( i :1 to  number_of_childnodes)
        {
                while (slot_table [slot_iter] not empty)
                        slot_iter++;

                reserved_slots [slot_iter]. node = child_node [i] ;
                reserved_slots [slot_iter]. mode = receive ;
                slot_iter ++;
                while (slot_table [slot_iter] not empty)
                        slot_iter++;
                reserved_slots [slot_iter]. node = child_node [i] ;
                reserved_slots [slot_iter]. mode = send ;
                slot_iter ++;

        } //end for
        advertise (reserved_slots);
        update_slot_table (reserved_slots);
}

/* if the advertisement is received from neighbor node then update_slot_table is called to reflect the available slots */
```

Figure 10: Pseudo code for implementing BSMA protocol.

Fig. 11 shows the setup phase of BSMA. Fig. 11(a) shows the process of electing a node's parent and Fig. 11(b) shows the process of electing its children nodes in the tree structure rooted at the sink. Once the setup phase is completed, nodes shift to TDMA mode. They use the assigned slot for their own communication. If they experience successive collisions, they need to change the slot scheduling. Monitored data is periodically sent to the sink and forms majority of the packets directed towards the sink. The packets from the sink may be maintenance messages that are broadcast over the network and travel from sink to periphery. Sink originating messages tend to form a small fraction of the total messages sent. The Data packet can be of variable size. The slot size has to at minimum maximum packet size plus ACK packet size. Every data packet is immediately acknowledged with an ACK packet. Guard space between slots is present to tolerate some amount of clock drift before the nodes get synchronize again.



(a) Setting up a node's parent (b) Setting up a node's children

Figure 11: Setup phase of BSMA protocol.

Synchronization

Clock drift among the nodes can cause synchronization errors. Synchronization can be achieved by having sink node send synchronization messages periodically at a higher power level to cover the whole network. The nodes can also be synchronized when they update the clock using the timestamp from the parent node. This can happen, at best, once in 3 BIGSLOT periods. Guard space between individual slots can be calculated to tolerate certain amount of drift. It can be calculated based on the periodicity of the synchronization messages, duty cycle and the actual clock drift.

Handling Collisions

Collisions can happen due to drift in clock or if a new node enters the network. If it is due to drift it will be corrected when the node receives the synchronization message. If a parent node detects collision either as garbled reception or not receiving an ACK for the transmitted packet. If this happens for p consecutive times, it concludes that the slot suffers from collision and tries to relocate the slot. The child node also stays awake beyond the schedule in its parent's BIGSLOT period to facilitate the relocation of the slot. Both update their schedules accordingly. Since,

BIGSLOT is designed so that nodes have small duty cycle and that they are reused beyond two hops, the conflicting slot can be easily relocated. The other nodes having the schedule of the nodes involved in relocating the slot need not be informed as exchanging schedules is mainly useful during set up of the network. The accidental collision problem could be more remote if the neighboring node chose their schedules quite far from each other in their BIGSLOT period. This also facilitates relocating the slot near to its existing schedule if consecutive collisions happen due to extraneous reasons. Thus handling of collisions in above fashion provides more flexibility to the BSMA's TDMA phase.

Scalability

The network is sink-oriented and can be replicated with multiple sinks with a separate sink-to-sink protocol to build large-scale network. Individual network can be large as BSMA leverages spatial reuse along with as structured layout. However the nodes nearer to the sink could be loaded more resulting in their early demise and partitioning of network. This can be handled by having a higher density of the nodes near the sink, which would prolong the overall network lifetime. This also helps to develop almost a spoke pattern near the sink due to lesser node degree for nodes near the sink. The disadvantage being more nodes is needed. However, the network is useful only if the nodes have a path to the sink. Otherwise even if the nodes at the periphery survive the resulting network partition due to demise of nodes near the sink will render the network useless. The cost of few more nodes near the sink would not be considerable if the increase in overall network lifetime is taken into consideration. Since we are looking at very low duty cycles, unavailability of timeslots during TDMA phase will not be a concern. Once the TDMA mechanism starts, if any node has much larger number of children nodes than its neighbors then it can initiate transfer of some of these nodes to its neighbors wherever possible. It helps in obtaining an even distribution of traffic resulting in longer network lifetime.

Addressing Scheme and Routing

Since sink-oriented networks are considered the flow of messages is predominantly upstream (towards the sink) or downstream (towards the periphery) and does not require a very general routing scheme. The Sink node can allocate the addresses. This helps the sink to locally get a rough topology of the network. Multi-casting can be done efficiently. The sink divides the address space among its child nodes. The child node takes up the first address in the subdivision and last address is used to refer to the group. The child node in-turn divides the remaining among its child nodes. Hence, if the sink wants to send a message only to a branch along a child node, it

can do so by selecting the address of that subdivision. If a provision is made to inform the sink of further divisions, it can obtain a more detailed picture of the network. More fine multi-casting can be achieved. Intermediate nodes in the tree network can multi-cast based on their subdivisions. This kind of distribution is decentralized. The sink node distributes the address space among its neighbors. The nodes allocate addresses to the nodes further downstream. Since the address space is quite large there should be deficiency at any node. Another method is that the nodes use their device numbers during the formation of the network. The nodes, including the sink, then distribute the address space proportional to the number of nodes in each branch. The sink addresses can be universally known, since there is only one sink in each group. Nodes could embed the branch density as part of their first message. The sink may be informed about the allocation if it needs to have a picture of the whole topology.

4. Tiny OS Environment

BSMA is implemented on Motes using TinyOS operating system environment [20]. TinyOS provides a well-defined programming model with focus on modularity, efficiency and concurrency. Modularity is based on component model where each component implements a specific function. The key advantages are reusability and simplicity of design. Each component specifically declares in the interface (.comp file);

- The commands(services) it provides to the other components
- The commands it uses which are provided by other components
- The events handled
- The events signaled

The functionality is implemented in the .c file. Thus a clear separation of the implementation and interface is maintained providing flexibility to change the implementation as long as the interface is not changed. Modular design also enables faster development. Each component has its own Frame where the memory used is declared and a set of tasks. Since frames are statically allocated it gives the memory requirements of a component at compile time avoiding overhead associated with dynamic memory allocation. Tasks are primary unit of computation and in the two-level structure they can be pre-empted by events. Tasks run to completion unless pre-empted. They can call lower level commands, signal higher-level events and schedule other tasks in the component. Tasks simulate concurrency using events. Commands are non-blocking calls

to lower level components. They deposit the request parameters in component's frame and post a task for execution. They cannot signal events, this is done to avoid cycles. Event handlers deal with hardware events either directly or indirectly. They deposit the information in the frame and can post tasks, signal higher level events and call lower level commands.

The components are then wired together. Thus all the applications and OS components compile into a single executable. This is done for efficiency and to obtain a small footprint required for embedded devices. The wiring is done by connecting the interfaces by macros, i.e. one interface command is #defined as another. The wiring of the components is explicitly defined in .desc file. Thus we have the component definition in .comp file, the implementation in .c file and wiring information in .desc file. The whole wiring structure can be represented as a graph. Component graph below shows how components can be wired to form a single executable.

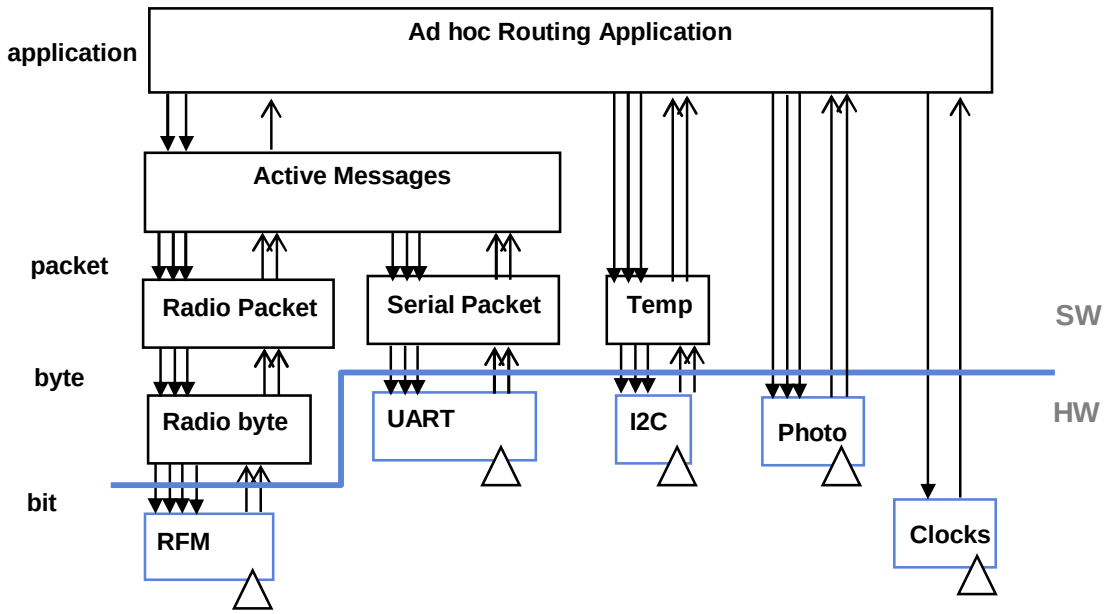


Figure 12: Component graph of TinyOS [21].

5. System Model

Sensor networks considered are multi-hop with following assumptions

- Network consists of many sensors and more than sinks. Sinks have superior power and processing availability.
- Time synchronization is achieved by beacons broadcast by sink or by nodes synchronizing with their parent nodes.

Nodes are organized rooted at a sink in a tree fashion. Tree is built by propagating an initialization message from the sink toward periphery. Hop-count is updated as the message propagates towards periphery. Nodes associate themselves as child nodes with the node from which they receive the message with highest signal strength with the least hop count from the sink.

Mote system has the following characteristics [7].

- MICA2 series hardware: MPR400 (916 MHz), MPR410 (433 MHz), and MPR420 (315 MHz)
- MICA2 utilizes a powerful Atmega128L microcontroller and a frequency tunable radio with extended range
- MICA2 radio
    - Consumes 8mA and 8~12mA in full operation but consumes 2uA in sleep mode
    - Can be adjusted for a range of output power levels via register called PA_POW: 5.3~26.7mA (-20~10dBm) (MPR 410/420)
    - Also provides a measurement of the received signal strength (RSSI) and is available to the software via ADC channel 0.

6. BSMA Protocol Interface

BSMA is a MAC and limited routing layer solution. Hence it provides commands to applications to send data and utilizes the services provided by the physical layer. BSMA is a two-stage protocol. In the first stage it builds up the information required for TDMA scheduling using random access and then transitions into TDMA. BSMA is implemented as a component. It provides an interface that can be used by applications to send and receive data. BSMA uses the commands provided by components PhyControl, RadioState, CarrierSense, PhyComm, Random, Clock, TimeStamp and PowerManagement. Here the Physical layer used is the one provided by ISI as part of the SMAC protocol [2]. BSMA replaces SMAC as the medium access protocol while retaining the same physical layer. Fig. 2 shows the BSMA interface followed by the discussion on each of the interface functions.

```
interface BSMAComm
{
   // Broadcast a message
   command result_t broadcastMsg(void* msg, uint8_t length);
   event result_t  broadcastDone(void* msg);

   // The following are unicast messages

   command result_t unicastMsgAddr(void* msg, uint8_t length, uint16_t toAddr ); //
Address is specified and direction is derived

   command result_t unicastMsgDirec(void* msg, uint8_t length, uint16_t direction ); //
just upstream or downstream is mentioned
// upstream for messages to sink and downstream for messages to child tree

   event result_t unicastAddrDone(void* msg);
   event result_t unicastDirecDone(void* msg);

   // msgClear clears all pending messages in queue
   command result_t msgClear ();

   // signal received message
   event void* recvDone(void* msg);
}
```

Figure 2: BSMA interface.

Commands

Commands included in the BSMA interface are used by higher level components to obtain services of BSMA. The main services provided by BSMA are

- **broadcastMsg**: This command is used to send broadcast message. The message is sent both upstream and downstream
- **unicastMsgAddr**: This command is used to send unicast message if the application knows the address of the destination node. At present it can be either the parent node or one of the child node.
- **unicastMsgDirec**: This is most commonly used command and it specifies whether the message is for sink or for distribution among the child nodes.

Signals

Signals included in the BSMA interface are as follows.

- **recvDone**: This signals higher level component that a message was received
- **broadcastDone**: This signals higher level component that the request to broadcast has been acted upon. The result is provided in result_t.
- **unicastAddrDone**: This signals higher level component that the request to unicast based on address has been acted upon. The result is provided in result_t.

- **unicastDirecDone**: This signals higher level component that the request to unicast based on direction has been acted upon. The result is provided in result_t.

7. BSMA Protocol Implementation

This implements the actual handling of the data at MAC and limited routing. The messages can be either sent upstream or downstream. From the functionality perspective it can be divided into timer handling, event handling, and dispatchers.

Timer handling

BSMA component uses the event generated by Clock component. The event is delivered periodically and all timers are updated upon reception of this event. The function called is ***event void Clock.fire()***. The individual timers are actually variables initialized to their timeout value. Upon entering this routing the timers are decremented and when they reach zero the respective action is taken. The clock can be signaled to higher components if needed using **signal.**

Event handlers

Clock is one type of event handler. Other important one is to handle reception of message and status message of transmissions. **event void* PhyComm.rxPktDone(void* packet, char error)** is signaled when the physical layer completes reception of a message. If the message is received successfully then the type of the packet is determined and respective handler is called

- Sink Message handler **handleSINK_MSG(packet):** This function is results in vertical spatial reuse by the dividing the whole network into three BIGSLOT periods. It the uses the timestamp present and the hop count to derive the BIGSLOT number of the node. It then increments the hop count and rebroadcasts the message.
- Data Packet handler **handleDATA_PKT(packet):** Processing of data packet and passing it on to the application module.
- Parent Select Message handler **handlePARENT_SEL_MSG(packet):** Upon receiving the PARENT_SEL_MSG the node adds the sending node as one of the child nodes. This information is needed to generate the TDMA Schedule.
- Schedule Packet handler **handleSCHEDULE_PKT(packet):** On receiving the SCHEDULE_PKT if the node is the intended child node it updates the information about which slot is allocated to it by the parent node. Else it node whether it is from a node in

same BIGSLOT as it is, if so this information is vital to determining if any conflicts result during drawing up the schedule for its child nodes.

- Acknowledgement Packet handler **handleACK(packet):** Handling of the acknowledgement for the data send. If ACK is lost then node updates the information required to determine if the link has gone bad.
- Ack Schedule Packet handler **handleACK_SCHED(packet):** All child nodes acknowledge the receipt of SCHEDULE_PKT, if this is not received the parent node may assume that the child node has a problem.
- Unknown Message type handler **handleErrPkt():** error handling.

Dispatchers

They are responsible for generation of different kinds of messages.

- **sendSINK_MSG:** This function is initiated by the sink node only. The other nodes forward it when they receive the sink message. Before forwarding the hop count is incremented.
- **sendPARENT_SEL_MSG:** Once the timeout for collecting sink messages is over, the node selects the node closest to the sink, determined by the hop count field, as the parent node. This information is conveyed to the node by means of a PARENT_SEL_MSG.
- **sendSCHEDULE_PKT:** Once the timeout for obtaining the PARENT_SEL_MSG is over, the parent node computes the schedule for the child nodes and broadcast them in the SCHEDULE_PKT. If it receives conflicting selections from the neighboring node then the node with the lower address will recomputed the schedule and rebroadcasts it. Since the BIGSLOT period is quite big and can be reused two hops away, schedules of the neighboring nodes can be accommodated. The Schedules overheard are also maintained if possible to help recovery procedures and also to compute the schedule in a non-conflicting manner.
- **sendACK_SCHED:** Upon reception of SCHEDULE_PKT the child nodes sends an ACK_SCHED to the parent node. Sending is randomized within a small duration so that all the child nodes don't send at the same time.
- **sendACK:** All Data packets are acknowledged.

Local information

Another important part of BSMA implementation is how to maintain local information. The local information such as states, schedule, and neighbor is maintained in the following data structures.

First, states provide vital information in both random access and TDMA periods. The different states possible are maintained as an enumeration. The present state is maintained in a state variable of the enumeration type. The state transition diagram provides the transitions between various states. Information regarding whether in Random Access or TDMA phase is also maintained. Once the building of schedules is done the nodes transition into TDMA phase. In this the operating state machines are that of the sender and the receiver.

Second, schedule is maintained in **Schedule schedTab[BSMA_MAX_NUM_SCHED]**, where Schedule is

```
typedef struct {
        uint16_t NodeAddr;
        uint8_t BSNum;
        uint16_t BSOffset;
        unit8_t NumSlots;
}__attribut__((packed)) Schedule;
```

The availability of the slots in the BIGSLOT is maintained using the following data

```
 // BigSlot availability
   typedef Struct {
   uint16_t startSlot;
   unit8_t numSlots;
   uint8_t sendRecv; // I guess not needed, just availability info
   } slotUsage;

   uint8_t myBS;
   uint8_t parentBS;
   uint8_t childBS;

   slotUsage usedSlots [BSMA_MAX_NUM_NEIGHB];
```

Hence finer details about the actual Schedule of the neighbors need not be maintained. Just the number of slots used by them in the BIGSLOT period and their location suffices. The information derived from sink message is used to determine the BIGSLOT number used by the node and is stored in *myBS*, the parent BIGSLOT number and child BIGSLOT number is stored in `parentBS, childBS` respectively.

Thirdly, neighbor information is derived from the Schedule packets broadcast by the nodes. The information in them is processed and stored in the following data structures.

```
slotUsage usedSlots [BSMA_MAX_NUM_NEIGHB];
```

```
slotUsage UsedSlotsInParentBS [BSMA_MAX_NUM_NEIGHB];
```

However, memory permitting, more detailed information can be stored in

```
uint8_t numNeighb;  // number of known neighbors
NeighbList neighbList[BSMA_MAX_NUM_NEIGHB]; // neighbor list
typedef struct {
      uint16_t neighbAddr;
      uint8_t relation;
      Schedule sched [15];
      unit8_t numActiveSchedules;
      uint8_t active; //flag indicating the node is active recently
   } NeighbList;
```

8. Source Structure of BSMA

BSMA is implemented using SMAC implementation [1] by ISI, which is again based on TinyOS implementation. It consists of three files: BSMAMsg.h, BSMAComm.nc, and BSMAM.nc. BAMAMsg.h contains the data structures used in BSMAM.c. BSMAComm.nc contains the interface provided by the BSMA component (see Fig. 2). BSMAM.nc implements the BSMA protocol.

BSMAMsg.h

```
// MAC header to be included by upper layer headers -- nested headers
typedef struct {
} __attribute__((packed)) MACHeader;

typedef struct {
}__attribut__((packed)) Schedule;

//Control packet
typedef struct {
} __attribute__((packed))  MACCtrlPkt;

typedef struct {
} RadioTime;
```

BSMAM.nc

```
module BSMAM
{
  provides {
    interface StdControl as MACControl;
    interface MACComm;
    /*
    interface MACTest;
```

```
    interface MACPerformance;
    */
  }
  uses {
    interface StdControl as PhyControl;
    interface RadioState;
    interface CarrierSense;
    interface PhyComm;
    interface Random;
    interface ClockBSMA as Clock;
    interface TimeStamp;
    interface PowerManagement;
  }
}

//Handle packet reception completion from physical layer
event void* PhyComm.rxPktDone (void* packet, char error) {}

// handle transmission complete signal from Physical layer
event result_t PhyComm.txPktDone (void* packet) {}

// Handle Clock fire event
// The timers implemented in BSMA are dependent on this function for updation
event void Clock.fire() {}

// Calculate the shift needed to position BIGSLOT
int findShift (unit8_t originalRegion, unit8_t nextFramePos) {}

// find the BIGSLOT of the node
void findBigSlot (MACCtrlPkt* pkt) {}

// Handle receipt of sink message, determine the BIGSLOt and
// forward it after updating the hop_count
void handleSINK_MSG (void* pkt) {}

//Hanlde PACKET_SEL_MSG, store the sender as a child node
void handlePARENT_SEL_MSG (void* pkt) {}

// Handle receipt of Data Packet
void* handleDATA_PKT (void* pkt) {}

// handle receipt of ACK
void handleACK (void* pkt)

//Upate the portion the BIGSLOT used
void updateUsedSlots (void* pkt) {}

// Handles SCHEDULE_PKT, note the slot in parent BIGSLOT
// period available to send and received data
void handleSCHEDULE_PKT (void* pkt)

// Checkt if channel is idle
event result_t CarrierSense.channelIdle() {}

// Compute the Schedule from the information gathered through
// PARENT_SEL_MSG and other SCHEDULE_PKT previously broadcast
```

```
// by neighbors
void form_schedule (slot slot_table [] )
```

9. Running BSMA

A simple application (BSMATest.nc and BSMATestM.nc) has been designed to test the basic functionality of BSMA. The default configuration is to use *n* motes. And each mote sends out 10 unicast messages. The node with ID 1 is selected as the sink node. In order to run the application, 1) Compile and install by 'make mica' and 'make install mica'. 2) You can either specify node ID (TOS_LOCAL_ADDRESS) in system/tos.h, or simply using 'make install.x mica', where x is the node ID. One of the node has to have node ID 1, which will be the sink node.

BSMATest.nc

```
implementation
{
  components Main, BSMATestM, BSMA;

  Main.StdControl -> BSMATestM;
  BSMATestM.MACControl -> BSMA;
  BSMATestM.MACComm -> BSMA;
  BSMATestM.MACTest -> BSMA;

}
```

BSMATestM.nc

```
module BSMATestM
{
  provides interface StdControl;
  uses {
    interface StdControl as MACControl;
    interface MACComm;
     }
}
```

References

[1] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," *IEEE Infocom*, pp. 1567–1576, 2002.

[2] IEEE, *Wireless LAN medium access control (MAC) and physical layer specifications, ANSI/IEEE Standard 802.11,1999 Edition*, 1999.

[3] M. J. Miller and N. H. Vaidya, "On-Demand TDMA Scheduling for Energy Conservation in

Sensor Networks," *Technical Report*, June 2004.

[4] V. Rajendran, K. Obraczka, and J.J. Garcia-Luna-Aceves, "Energy-Efficient, Collision-Free Medium Access Control for Wireless Sensor Networks," *The First ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, November 2003.

[5] Jose A. Gutierrez, Edgar H. Callaway, Jr., and Raymond L. Barrett, Jr., *Low-Rate Wireless Personal Area Networks: Enabling Wireless Sensors with IEEE 802.15.4*, IEEE Press, 2003.

[6] Tijs van Dam, Koen Langendoen, "An Adaptive Energy-Efficient MAC Protocol for Wireless Sensor Networks," *The First ACM Conference on Embedded Networked Sensor Systems (SenSys'03)*, November 2003.

[7] *MPR/MIB Mote Hardware Users Manual* (Rev. A, Document 7430-0021-05), December 2003.

[8] Hagen Woesner, Jean-Pierre Ebert, Morten Schlager, and Adam Wolisz, "Power-Saving Mechanisms in Emerging Standards for Wireless LANs: The MAC Level Perspective," *IEEE Personal Communications*, Vol. 5, Issue 3, pp. 40-48, Jun. 1998.

[9] W. Stallings, "IEEE 802.11 Wireless LAN Standard," Chapter 14, *Wireless Communications and Networks*, Prentice Hall, Inc., 2002.

[10] IEEE Std 802.11-1999, *Local and Metropolitan Area Network, Specific Requirements, Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, http://standards.ieee.org/getieee802/download/802.11-1999.pdf.

[11] L. Huang and T.-H. Lai, "On the scalability of IEEE 802.11 ad hoc networks," *The 3rd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc)*, pp. 173-182, 2002.

[12] Benjie Chen, Kyle Jamieson, Robert Morris, Hari Balakrishnan, "Span: An Energy-Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks," *7th ACM International Conference on Mobile Computing and Networking (MobiCom'01)*, July 2001.

[13] Y.-C. Tseng, C.-S. Hsu, and T.-Y. Hsieh, "Power-Saving Protocols for IEEE 802.11-Based Multi-Hop Ad Hoc Networks," *IEEE Infocom*, 2002.

[14] Ronny Krashinsky and Hari Balakrishnan, "Minimizing Energy for Wireless Web Access Using Bounded Slowdown," *8th ACM International Conference on Mobile Computing and Networking (MobiCom'02)*, September 2002.

[15] E. Jung and N. Vaidya, "An Energy Efficient MAC Protocol for Wireless LANs," *IEEE Infocom*, 2002.

[16] Rong Zheng, Robin Kravets, "On-demand Power Management for Ad Hoc Networks," *IEEE Infocom*, 2003.

[17] S. Singh and C.S. Raghavendra, "PAMAS: Power aware multiaccess protocol with signalling for ad hoc networks", *Computer Communication Review*, Vol. 28, No. 3, pp. 5–26, 1998.

[18] G. Girling, J. Wa, P. Osborn, R. Stefanova, "The Design and Implementation of a Low Power Ad Hoc Protocol Stack," *IEEE Wireless Communications and Networking Conference (WCNC)*, 2000.

[19] P. Kinney, "ZigBee Technology: Wireless Control that Simply Works," IEEE 802.15.4 Task Group, 2003.

[20] TinyOS, http://www.tinyos.net/

[21] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, “System architecture directions for network sensors,” *ASPLOS 2000*, Cambridge, November 2000.