

**A Wireless Sensor Network Monitored Air Conditioning System**

**Amr Elgendy**

**Abdallah Gabara**

**Mina Gamal**

**Aly Sultan**

# 

# INTRODUCTION:

The monitoring system consists of six nodes, five for sensing nodes in addition to a sink node for receiving the data of all sensors. The sink node collects and aggregates the data sent from the nodes for monitoring purposes. The system is to be designed to minimize collision, conserve power and adapt to any change in the topology if a new node is introduced or an existing node fail. The design proposed adopts a simple MAC protocol and hierarchical IP protocol to coordinate communication between nodes. The main functions used in coding the nodes is attached in the appendices.

# 

# 

# Protocols:

Once a packet is sent by a node, an 8 bytes header is concatenated to the packet. Therefore, when receiving a packet, nodes should these 8 bytes are to be accounted for and discarded to read the MAC header, IP header, and the data originally sent.

# 1- MAC Protocol:

A simple MAC Protocol is adopted for the application. The MAC header consists of the destination and source mac. In addition, a CRC part is added to the packet to verify its validity. Once a packet is validated by a node through the CRC part, the destination MAC address of the packet is checked. In case the destination MAC matches the MAC of the node or the predefined broadcast MAC of (0xFF,0xFF,0xFF,0xFF,0xFF,0xFF), an acknowledgement packet is sent by the node to the sender of the packet. The sender node, on the other hand resend the packet, in case an acknowledgement was not received for a predefined period.

# 2- IP Protocol:

A conventional IP scheme is adopted in this system. A subnet mask of 0.0.0.255 is used. Consequently, the first 3 bytes are fixed and the last one is to distinguish the different nodes in thes system. The sink node has an IP of 10.10.10.1. While all nodes are assigned an IP of 10.10.10.x where x is a unique number for each node. In the case of the discarding of a node, the IP it had becomes available to be assigned for new nodes. The IP scheme allows for a maximum of 254 nodes to join the system. However, if needed the scheme can allow for using more nodes by simply changing the subnet.

# Adding A new node:

Once a node is added to the system, it broadcasts a packet to be processed by all its neighbours. If the sink node receives the packet, the MAC address of the node is added in the memory of the sink and an IP address is assigned for the new node. The node receives the assigned IP from the sink through a packet with the destination MAC being the MAC of the new node. The IP address assigned in this case will have a depth byte of 1 as it connects directly to the sink. If the new node cannot communicate with the sink node, the broadcasted packet sent by this node will be read by all neighbouring nodes. Consequently, each node responds by assigning an IP address with a depth byte equal to the byte of the assigning node added to 1. The new node then picks the IP with the highest depth byte and sends and acknowledgment back to the assigning node informing it that it shall connect to it. The fact that a new node picks the highest depth is to guarantee a small branching factor in order to minimize the power consumption.

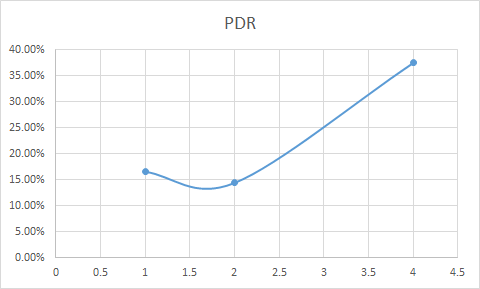
# Removing an idle node:

Each node will have a variable number corresponding to the number of milliseconds elapsed since the its last activity. Once that variable exceeds a predetermined value, the node is discarded and is considered to be dead. The sink node will keep count of for how long each node has been inactive

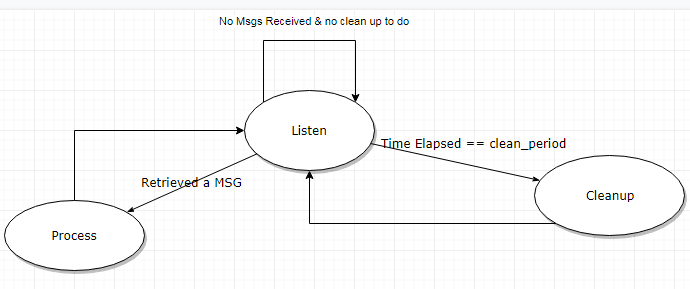
# 

# Experiments:

In order to test the quality communication, the packet loss ratio is plotted as a function of the delay between 2 consecutive sent packets. The experiments conducted were 1000 packets for delays of 1 and 2 seconds and 200 packets for a delay of 4 seconds (due to time limitation). All experiments were conducted with one sink node to receive and acknowledge received data and 2 nodes to send the data to the sink. All experiments were conducted with the use of acknowledgements packets.



# Finite State Machine - Sink:

The sink node is constantly functioning to capture the traffic potentially coming from any node in the system. The sink is modeled as a finite state machine with 3 states; listen, process, and cleanup. The sink starts in the listening state to receive the traffic from the nodes. Once a packet is received and verified, the sink processes the packet and transitions to the processing state. There are two types of packets the sink receives. The first packet type is a request packet in which case a new node requests an IP address from the sink. The sink respond with an IP address generated to the new node. Consequently, the node MAC address gets stored in the memory of the sink and it then can start sending data. The second is a data packet in which case the sink displays the received data on the terminal and send an acknowledgement back to the sending node. The sink transitions to the cleanup state every predetermined period defined as clean\_period. In this state the sink starts discarding the addresses of nodes that have not contacted the sink for more than a predetermined period defined as dying\_period.****

# Finite State Machine - Source:

# Appendices:

**Main Functions:**

1- **send1pac\_with\_sd**:

**Description**: sends data (char\* data) to a certain MAC address (char\*data) on a certain channel (channel) with the passed values of the rate (rate) and power (pwr) given a Socket (Soc).

static int send1pac\_with\_sd(int iSoc, int channel, int pwr, int rate, char\* dest, char\* data, int dataSize){ int i=0;

int lRetVal;

assert(dataSize+MAC\_ADDRESS\_HEADER\_OFFSET <= BUF\_LEN);

char s\_data[BUF\_LEN];

for(i = 0; i<BUF\_LEN; i++)

{

s\_data[i] = 0;

}

for(i = 0; i<MAC\_ADDRESS\_LEN; i++)

{

s\_data[i] = dest[i];

}

for(i = MAC\_ADDRESS\_LEN; i<MAC\_ADDRESS\_LEN\*2; i++)

{

s\_data[i] = macAddressVal[i-MAC\_ADDRESS\_LEN];

}

for (i=MAC\_ADDRESS\_LEN\*2; i < MAC\_ADDRESS\_LEN\*2 + dataSize; i++)

{

s\_data[i]=data[i-MAC\_ADDRESS\_LEN\*2];

}

lRetVal = sl\_Send(iSoc,s\_data,BUF\_LEN,\

SL\_RAW\_RF\_TX\_PARAMS(channel, rate, pwr, PREAMBLE));

if(lRetVal < 0)

{

ASSERT\_ON\_ERROR(lRetVal);

return -1;

}

return lRetVal;

}

**2- recv1pac:**

**Description**: recieves data from a certain socket (Soc) and stores the data in a passed array (data).

static int recv1pac\_with\_sd(int iSoc, char data[MAC\_DATA\_LEN], char srcmac[MAC\_ADDRESS\_LEN])

{

char acBuffer[BUF\_LEN];

long lRetVal = -1;

char dest[6]={};

lRetVal = sl\_Recv(iSoc,acBuffer,BUF\_LEN,0);

if(lRetVal < 0 && lRetVal != SL\_EAGAIN)

{

//error

ASSERT\_ON\_ERROR(sl\_Close(iSoc));

ASSERT\_ON\_ERROR(lRetVal);

LOOP\_FOREVER();

}

else

{

parse(acBuffer,dest, srcmac, data);

if(isMyMac(dest))

{

return MSG\_ME;

}

else if(isBroadcastMac(dest))

{

return MSG\_BROADCAST;

}

else

{

return MSG\_NONE;

}

}

static int recv1pac\_with\_sd(int iSoc, char data[MAC\_DATA\_LEN], char srcmac[MAC\_ADDRESS\_LEN])

{

char acBuffer[BUF\_LEN];

long lRetVal = -1;

char dest[6]={};

lRetVal = sl\_Recv(iSoc,acBuffer,BUF\_LEN,0);

if(lRetVal < 0 && lRetVal != SL\_EAGAIN)

{

//error

ASSERT\_ON\_ERROR(sl\_Close(iSoc));

ASSERT\_ON\_ERROR(lRetVal);

LOOP\_FOREVER();

}

else

{

parse(acBuffer,dest, srcmac, data);

if(isMyMac(dest))

{

return MSG\_ME;

}

else if(isBroadcastMac(dest))

{

return MSG\_BROADCAST;

}

else

{

return MSG\_NONE;

}

}

}

}

3- **sd\_setup**:

**Description**: Setup a socket on a passed channel (iChannel) and with a timeout value of (timeOut)

static int sd\_setup(\_u8 iChannel, SlSuseconds\_t timeOut)

{

int iSoc = sl\_Socket(SL\_AF\_RF,SL\_SOCK\_RAW,iChannel);

long lRetVal = -1;

ASSERT\_ON\_ERROR(iSoc);

struct SlTimeval\_t timeval;

timeval.tv\_sec = 0; // Seconds

timeval.tv\_usec = timeOut; // Microseconds.

// Enable receive timeout

lRetVal = sl\_SetSockOpt(iSoc,SL\_SOL\_SOCKET,SL\_SO\_RCVTIMEO, &timeval, \

sizeof(timeval));

ASSERT\_ON\_ERROR(lRetVal);

return iSoc;

}