www.coloradomicrodevices.com

SimpleMesh User's Guide

Colorado Micro Devices, Open Source Mesh Networking Stack

Colorado Micro Devices

SimpleMesh User's Guide

SimpleMesh is an open source simple mesh-networking protocol stack. Its goal is to balance the combination of elements of a mesh network topology with a common sense approach to the embedded processor on which it will run. Most common "marketing hyped" embedded mesh network standards are now so complex they require more than 128K to 256K bytes of FLASH memory to meet minimum stack requirements. Most commercial stack companies presume that the embedded developer will co-mingle the host application code with the networking stack code – A guaranteed nightmare! To minimize the headaches, often, most of these commercial stacks provide only libraries with a bare main loop and the comment

```
// Write your code here…
```

With a caveat to not touch this or that peripheral or change some #define and to only use prescribed function callbacks to execute application code… Essentially, the embedded developer is shackled by the constraints of the mesh-networking stack and his code, the APPLICATION code, is considered of tertiary importance. Yikes!

Thus SimpleMesh endeavors to change that artificial constraint. How? By accepting the logic that networking stuff should be processed on a networking processor and host stuff should be processed on a host processor! Some embedded folks may say, "but I am so smart I can do it all in one processor." Fine, do it! We encourage you! Especially if you are working on a hobby project with no deadlines and lot's of money to burn… But, if you need to "absolutely, positively get a product delivered fast", then this stack is for you. Are you still arguing with us? OK, here's a challenge, open up your PC – Do you see a graphics card or an embedded graphics processor? Let's not discuss the NIC or embedded Ethernet processor or wireless system and while we're at it we won't discuss the north and south bridges… Finally, you did see your big fat multi-gigahertz CPU, right?! Well, what in the world are all those extra processors doing in your PC? Their specific functions minimize stuff your CPU needs to process although just about everything could get processed in the CPU. Let's take a page out of a successful play book and consider that running SimpleMesh in a small, cheap, FCC certified platform, that plugs in to your host system, might just allow you to do what you do best and allow SimpleMesh to just take care of your mesh networking for you. Still not convinced? Let's take the cover off of your cell phone and start counting separate embedded processors…

But… At the end of the day we realized that, gulp, smart folks are probably smarter than we are. So we decided to make SimpleMesh open source so that the worldwide community of developers can have free and open access to the stack. Our hope is that the community makes SimpleMesh the best open source mesh-networking stack on the planet (if it isn't already).

SimpleMesh reduces or eliminates complexity while maintaining a simple, understandable code base BUT it need not be understood at all to use! It has minimal commissioning requirements, which, by the way, are modeled after your 802.11 wireless access point. All of these thoughts make SimpleMesh

"**Simply Useful!**"

A Peek Inside SimpleMesh

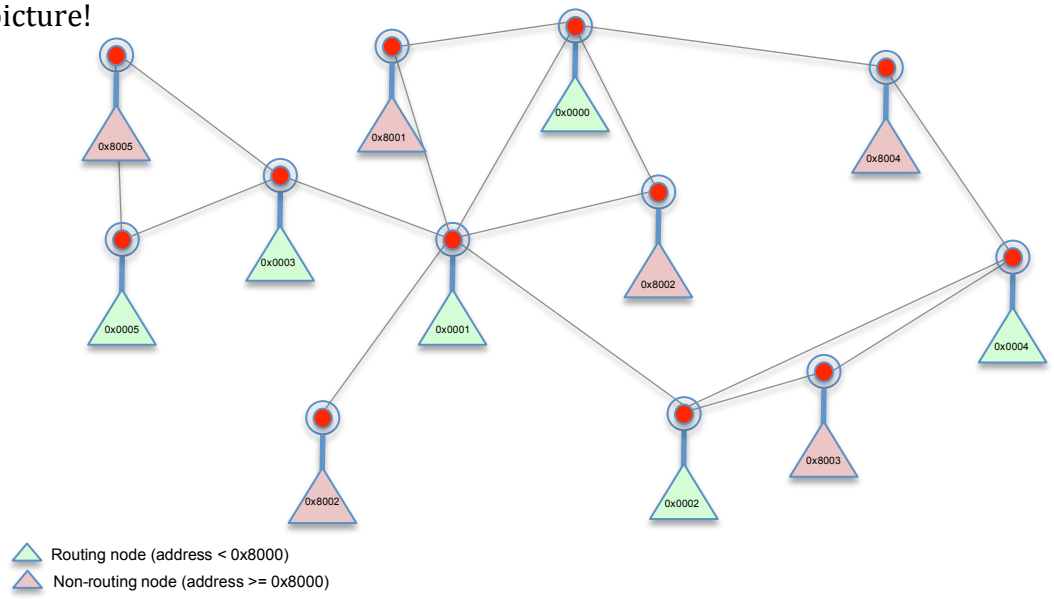
Even though SimpleMesh is designed and operates as a mesh network it is not limited to only mesh networking. Let's have a very brief overview of common network topologies:

1. **Point to point** – This network consists of two nodes that send data back and forth to each other.
2. **Star** – This network has one node at it heart or center and every node communicates to and through the central node. If a node X wants to send a message to node Y, it can do so by first sending to the central node and the central node will forward the message.
3. **Mesh** – In this network, any node can send a message to and through any other node. A message may pass though several nodes before reaching its destination and the path, or route, through the mesh may change from, one message to the next.

If you have only two nodes, voilà, SimpleMesh will work like a point-to-point network. If you would like to operate as a star, voilà, SimpleMesh will do that too. Finally, it is, of course, a mesh network …

How does SimpleMesh do that?

Here, a picture is worth a thousand words. To start, let's look at SimpleMesh's picture!



The Network Topology

The philosophy behind SimpleMesh is that "there is no central node" and every node is one of two types; a *routing* or *non-routing* node. When a SimpleMesh node is powered on and *commissioned* it is ready to use and can send and receive data without any special procedure. Commissioning will be covered later on, don't worry, its simple. If the route to an unknown destination occurs, route discovery will automatically happen and while non-routing nodes cannot be used as a next-hop, they can send and receive data from any other node. A node's *routing table* is automatically updated using data from frames that it receives and transmits. So, if signal strength or conditions change the internal routing table will adapt. Finally, the network may need to broadcast frames, if duplicate broadcast frames are detected they are rejected.

All this data is fully encapsulated in Colorado Micro Device's *RadioBlocks* platform. The UART protocol is fully specified in the document *SimpleMesh_Serial_Protocol.doc* that can be found at http://www.coloradomicrodevices.com/software/documents/

So, why all these details for a network one is supposed to just plug in and start sending and receiving? Because there are lots of folks who may need to modify the network for their particular purpose…

Commissioning SimpleMesh

Why? We know people are grumbling, "Why do I have to worry about network commissioning…" Maybe we are using too complex a term here, maybe not… First of all let's tell you what it is – It is just setting some particular parameters for your network in each node. The parameters are:

1. Channel (11-25)
2. Address (0x0000 through FFFD)
3. PANID (0x0000 through 0xFFFE)
4. Network key

That's it, that's all of them. Are you still saying why, why, why? Because, it is all in an effort to make SimpleMesh "simply useful" rather than so simple it is useless or so complex it is also useless. Hmm, still not convinced? Here's a reality check, go to someplace you've never been that offers free wireless service and prepare to go surfing. What's the first thing you do? Oh, you asked the folks at the establishment for the network name. Hey, wait, you asked for the name of the wireless network? You didn't cry out, "Why, why, why should I have to enter the name of the wireless network!" Let's not even talk about a security key and which flavor of security key… The point is, you are used to doing those things for a computer that has millions of times more power than your simple embedded system – Yet, you did it without wondering or complaining. Just do the same thing for SimpleMesh and you'll not even question it after a few times.

The SimpleMesh API

Let's have a look at the internals of how SimpleMesh works. We think the best way to introduce SimpleMesh is through its API. If you have downloaded the code base you'll find a couple of sample applications in the paths:

```
../simplemesh/source/apps/demo/src/demo.c
```
or
```
../simplemesh/source/apps/serial/src/serial.c
```

Both of these files are a great starting point to examine how an application is built. In the case of the "demo" application, we thought it would be fun to connect SimpleMesh to Atmel's WSN Demo application – You can find WSN Demo by searching the Atmel web site. The "serial" application is the ***fundamental*** application for SimpleMesh – It is this application that can be connected to your host embedded system and deployed.

There are three essential requirements for the SimpleMesh to operate – Starting it, sending and receiving data. Let's have a look at some examples. These examples will be fully illustrated using Colorado Micro Device's *RadioBlock* FCC certified platform – An IEEE 802.15.4 based 1" square embedded radio modem with UART interface!

Starting a SimpleMesh network

SimpleMesh is started and runs in none other than the *main* loop.

```
#define APP_DEFAULT_SECURITY_KEY "SimpleMesh_12345"

int main(void)
{
    NWK_Init();
    // ... other application initializations if necessary
    NWK_SetAddr(APP_ADDR);
    NWK_SetPanId(APP_PANID);
    NWK_SetChannel(APP_CHANNEL);
    PHY_SetRxState(true);
    while (1)
    {
      NWK_TaskHandler();
      // ... other application task handlers if necessary
    }
    return 0;
}
```

That's it! Make sure you set the network parameters defined earlier and that's all you need to do. Since it is assumed that you have connected your host system to communicate with a SimpleMesh enabled node via UART, you don't need to do anything at all here except what we've shown.

Sending a Frame

There are two essential things a network does; send and receive data. Let's discuss sending first:

```
NWK_DataReq_t    nwkDataReq;
uint8_t          data[10];

static void appSendData(void)
{
    nwkDataReq.dst = 0x0005;
    nwkDataReq.endpoint = 1;
    nwkDataReq.options = NWK_OPT_ACK_REQUEST;
    nwkDataReq.data = data;
    nwkDataReq.size = sizeof(data);
    nwkDataReq.confirm = appDataConf;
    NWK_DataReq(&nwkDataReq);
}

void appDataConf(NWK_DataReq_t *req)
{
    if (NWK_SUCCESS_STATUS == req->status)
        // success
    else
        // fail
}
```

When data is to be sent we would usually like to know it got sent and/or it was received. SimpleMesh will send the data and it will notify you of the results. Of course, since you are connected via UART to a SimpleMesh enabled node, you will send and receive data over your UART.

Receiving a Frame

Now we have a look at how SimpleMesh receives a frame:

```
bool NWK_DataInd(NWK_DataInd_t *ind)
{
    /*
    ind->src      - source address
    ind->dst      - destination address
    ind->endpoint - destination endpoint
    ind->data     - received data
    ind->size     - size of received data
    ind->lqi      - LQI as reported by RF-chip
    ind->rssi     - RSSI as reported by RF-chip
    */
    return true;  // "true" if frame should be acknowledged, or "false"
                  // otherwise
}
```

You can see that SimpleMesh gets a pointer to a memory location that has the format of the *MWK_DataInd_t* data type. Simple! What is received over the UART is slightly different. Refer to the aforementioned *SimpleMesh_Serial_Protocol.doc*.

These three APIs are all you need to get SimpleMesh running but that only applies to folks who are mucking around in the embedded code. Folks who just want to "use it" need only refer to the *SimpleMesh_Serial_Protocol.doc* document – No coding necessary!

The numbers in the top of the columns represent the number of bytes per item.

2	1	2	2	2	1	1	2	2	0-110	2
Frame Control	Sequence Number	PAN ID	Dest. Address	Src. Address	Frame Control	Sequence Number	Source Address	Dest. Address	Variable	CRC
Mac Header					Network Header				Payload	CRC

Table 1 - Network Frame Format

Details of the frame components, frame control, sequence number, etc. can be found in files *nwkPrivate.h*

The Routing Algorithm

A main component of SimpleMesh is its routing protocol and that protocol is based on the use of tables – So called, next-hop tables. Perhaps the best way to "talk" about the algorithm is to show some diagrams. Hang on, here we go…

Initial Condition

Given the starting conditions:

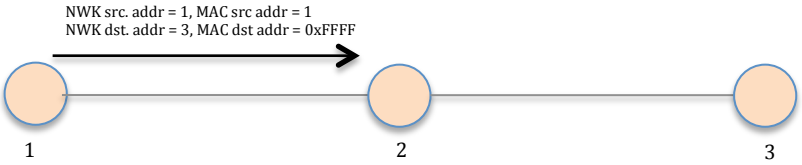
1. Node 1 needs to send data to node 3
2. Routing tables are empty
3. There is no direct path between node 1 and node 3

Destination	Next hop

Destination	Next hop

Destination	Next hop

Step two



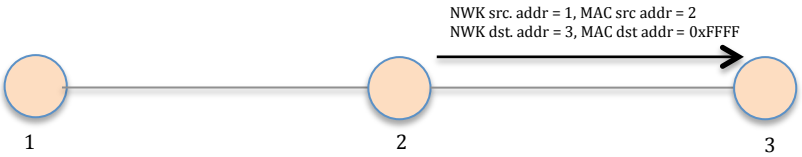
Destination	Next hop

Destination	Next hop
1	1

Destination	Next hop

1. Node 1 sends a frame with the Network (NWK) Destination (dst) Address (adr) set to 3 and MAC Destination Address set to 0xFFFF.
2. Node 2 receives this frame and adds a record for node 1 to its routing table.

Step 3



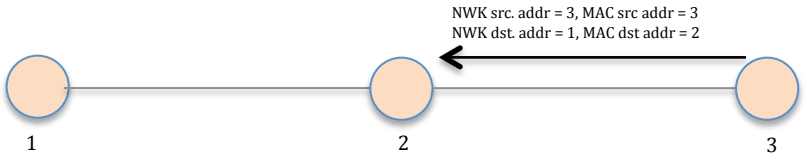
Destination	Next hop

Destination	Next hop
1	1

Destination	Next hop
2	2
1	2

3. Node 2 broadcasts the frame because the MAC destination address is set to 0xFFFF.
4. Node 3 receives this frame and adds a record for node 2 to its routing table.
5. Node 3 also adds a record for node 1 to its routing table, which it gleans from the NWK Source Address.
6. Note that when node 2 broadcasts the frame, node 1 receives it but discards it after interrogating the NWK header.

Step 4



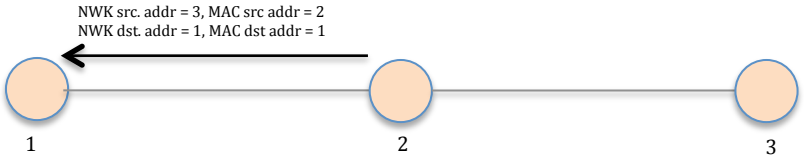
Destination	Next hop

Destination	Next hop
1	1

Destination	Next hop
2	2
1	2

7. Node 3 interrogates the frame and sends an Acknowledgement frame, even if one was not requested. This is done to establish a reverse route. Node 3 now knows the route to node 1 so a unicast frame is sent.

Step 5



Destination	Next hop
3	2

Destination	Next hop
1	1
3	3

Destination	Next hop
2	2
1	2

8. Node 2 receives the frame and adds a route to node 3 to its routing table.
9. Node 2 has a route record for node 1 so it routes the received frame to its final destination.
10. Node 1 receives the frame and adds a route to node3 to its routing table.

Now a route between node 1 and node 3 is established and can be used for future frame transmission between the new nodes. If the network hasn't changed (new nodes, better signaling through another node), nodes 1 and node 3 can communicate without the use of broadcast frames.

Outgoing Frame Routing Algorithm

Now that we have seen an illustration of the way routing works, we can outline the steps that occur to support it.

1. The NWK Destination Address is set as the final destination address.
2. If the next hop address is known (it is in the routing table), set the MAC destination address to the next hop address.
3. If the next hop address is not known, set the MAC destination address to 0xFFFF (broadcast). This will cause route discovery in the mesh network.

Incoming Frame Routing Algorithm

This part is only slightly more complicated but it is still *simple*.

1. An incoming frame will need routing if the NWK Destination Address of the frame is not equal to the node's own address.
2. If the next hop address is known, set the MAC Destination Address to the next hop address and send the frame.
3. If the next hop address is not known and not equal to 0xFFFF, discard the frame and send a Route Error command frame to indicate that one of the intermediate nodes in a route does not know the next hop. This will invalidate the route and case route discovery the next time a frame needs to be sent.
4. If the node is the final destination and the incoming frame was received with a MAC Destination Address set to 0xFFFF an Acknowledgement Command frame is sent even if no acknowledgement was requested. This will initiate discovery of the reverse route.
5. For any received frame the routing table is updated to tell that MAC Source Address is the next hop for the Network Source Address.

Brining it All Together

Now that you've got an overview of the SimpleMesh design and architecture we need to put it to use to actually experience it. How are we going to do that? Well we didn't design and implement SimpleMesh just for fun; we had our own reasons – To drive our own networking system. You can buy a kit from us to experiment or you can buy our FCC certified hardware to use in your own end products, experiments and hobbies.

Who are "we"

We are the authors and implementers of SimpleMesh. Each of us worked for a semiconductor company that couldn't market and sell wireless systems if you paid the customers to buy them… OK, that's a little harsh. The company was one of many who inhaled too many marketing hyped fumes and thought that the world would come running. Well, try as they might, no one was really buying into that over-hyped marketing buzzed network stack and we intrinsically knew real embedded folks wanted simple open source solutions that didn't take a years to get to market with and consume kilo bytes of FLASH that required processors that cost way too much…

So we combined our knowledge and produced some hardware and what has become SimpleMesh. Our hardware is called the *RadioBlocks* – Check it out at http://www.coloradomicrodevices.com And SimpleMesh? Check it out at: "git@git.assembla.com:SimpleMesh.git". Really, get over there right now and check it out – it's free…

Who are "you"

We are looking for talented embedded folks to join our open source team to continue to make SimpleMesh the best IEEE 802.15.4 based network on the planet. If you think you have the intellectual "guns" to code with us email us and we'll consider brining you on board our band of merry meshed hacksters…