



KoliadaES

EtherMESH – A mesh for low power, embedded systems

An Overview

Revision History

5/12/2001 - First Edition
2/13/2005 - Updates for BridgeLINK
5/3/2010 - Updates for CC2500
13/4/2012 - Updates for Win32
26/12/2013 - Updates for CC8051
26/4/2014 - Updates for CC8051

Table of Contents

<i>KoliadaES</i>	1
EtherMESH – A mesh for low power, embedded systems	1
An Overview	1
1. Introduction	4
1.1 Purpose	4
1.2 Scope	4
1.3 Acronyms	4
1.4 Reference Documents	4
2. EtherMESH	4
2.2 Mesh Profile	4
3. Addressing	5
3.2 Network address assignment	5
3.3 Addressing in EtherMESH	5
4. Routing	6
4.1 Overview	6
4.2 Neighbor Table	7
5 End-to-End Acknowledgements	7
6 Security	7
7 Miscellaneous	8
7.1 Channel Configuration	8
7.2 Configuring the mesh ID	8
7.3 Maximum Payload Size	8
7.4 Fragmentation	8
7.4 Memory Requirements	8

1. Introduction

1.1 Purpose

This document provides an overview some of the components of the KoliadaES EtherMESH stack and their functioning.

1.2 Scope

This document describes concepts and settings for the KoliadaES EtherMESH mesh network stack.

1.3 Acronyms

AF Application Framework

AES Advanced Encryption Standard

AIB APS Information Base

API Application Programming Interface

CCM* Enhanced counter with CBC-MAC mode of operation

MSG Message

NIB Network Information Base

NWK Network

OTA Over-The-Air

1.4 Reference Documents

[1] EtherMESH Specification.

[2] EtherMESH API

[3] EtherDATA

2. EtherMESH

An EtherMESH network is a multi-hop ‘meshed’ network for battery-powered devices. A meshed network is a type of network topology where each node must not only capture and disseminate its own data, but also serve as a *relay* for other nodes. Each node must collaborate and propagate the data in the network and any two devices that wish to exchange data in a EtherMESH network may have to depend on intermediate devices to be successful.

EtherMESH uses an optimized flood fill algorithm to propagate traffic that eliminates the need for specific node types and significantly reduces the need for configuration parameters. EtherMESH optimizations include the ability to significantly reduce redundant retransmissions and the ability to allow extended sleep times.

2.2 Mesh Profile

The set of mesh parameters that need to be configured to specific values is called a *mesh profile*. The parameters that comprise the mesh profile are defined in the EtherMESH Specification (currently v1.0). All devices in a network must conform to the same mesh

profile (i.e., all devices must have their mesh profile parameters configured to the same values). However, some of these values may be changed and shared dynamically. EtherMESH defines a standard mesh profile with the goal of promoting interoperability. All devices that use this mesh profile will be able to interoperate with any other devices that also use it. If application developers choose to change the settings for any of these parameters, they can do so with the caveat that those devices will no longer be able to interoperate with devices that use the standard mesh profile. The mesh profile identifier is shared as part of each EtherMESH exchange and is used to allow nodes to ignore messages from non-conforming mesh profiles.

3. Addressing

3.1 Address types

EtherMESH devices have two types of addresses - a 64 byte name and a 16-bit network address.

A device name may be statically or dynamically assigned but must be unique within the mesh and may also be designed to be globally unique. The 16-bit network address is assigned to a device when it joins a network and is intended for use while it is on the network. It is only unique within that network and is used for identifying devices and sending data within the network. The developer uses the EtherMESH APIs to manage names and addresses.

3.2 Network address assignment

EtherMESH uses a distributed addressing scheme for assigning the network addresses, which ensures that all assigned network addresses are unique throughout the whole network. Network addresses are defined and defended autonomously and typically at power up time. Once assigned, network addresses remain unchanged for the duration of mesh participation. The developer is not involved with the network address assignment except for the case where a device is unable to assign an address.

EtherMESH uses a stochastic (random) addressing scheme for assigning the network addresses. This process randomly assigns addresses to new devices, and then uses the rest of the devices in the network to ensure that there are no duplicate addresses. When a device joins a mesh, it self generates a new address. The new network node then ‘announces’ its new address and its name (if any) to the rest of the network. If there is another device with the same short address an address conflict message will be broadcast to the entire network and the device generating the conflicting short address will regenerate a new short address. When a device determines its new address is valid, it joins the mesh using the newly assigned network address.

3.3 Addressing in EtherMESH

EtherMESH frames can be unicast, multicast or broadcast where a unicast frame is sent to a single device, a multicast frame is sent to a group of devices and a broadcast frame is sent to all meshed devices.

3.3.1 Unicast Addressing

Used to send a frame to a single device whose network address is known simply use the network address. If the network address is not known, it can be found using the API. EtherMESH uses a cached name table to remember names and addresses. If the name is not in the cache, EtherMESH will force a name lookup request out to the network and may therefore be less efficient than knowing the address beforehand.

3.3.2 Group Addressing

Any address may also be a group address and any frame sent to a group address will be seen by all devices that are part of the group. A device may send a message to any group at any time using the relevant group address, but to receive particular group messages, each device must add itself to any groups it is part of.

3.3.3 Broadcast Addressing

Used to send a frame to all devices in the network. The destination address can be set to one of the following broadcast addresses:

BCAST (0xFFFF) message sent to all devices in the mesh. Sleeping devices will see the message when they next wake.

Broadcast messages are received by the standard, developer defined, callback handler.

4. Routing

4.1 Overview

A mesh network is described as a network in which the routing of messages is performed as a decentralized, cooperative process involving many peer devices routing on each other's behalf. EtherMESH message routing is completely transparent to the application, which simply sends data destined to another device via the EtherMESH API.

EtherMESH is specifically designed to facilitate dynamic routing, providing multiple simultaneous routes in the face of varying transmission quality or mobile nodes that appear and disappear. If a particular wireless link is down, EtherMESH simultaneously finds multiple parallel routes to any given node avoiding the broken link. A fundamental assumption within any mesh network is that there will be more than one route from any given node to any other.

EtherMESH uses an optimized flooding protocol where a message is passed simultaneously to any node within range and this process is repeated by each node until the message exists on every node. Optimizations allow for eliminating redundant transmits and for nodes to sleep for significant parts of their duty cycle.

Because EtherMESH uses a flood-fill algorithm, there are no routing tables. This allows message passing to scale dynamically to large numbers of nodes with very constrained resource requirements. Nodes may enter and leave the mesh rapidly without needing to

rebuild routes. EtherMESH facilitates an environment capable of supporting mobile nodes, link failures and frame losses.

4.2 Neighbor Table

Neighbor nodes are nodes that are within radio range of each other. Each node keeps track of their neighbors in a “neighbor table”, and that table is updated when the node receives any message from a neighboring node (unicast, broadcast or beacon). Each neighbor table entry contains the node address, some house keeping details, and the link status. The neighbor table size is implementation dependant (8 for the CC25xx).

5 End-to-End Acknowledgements

EtherMESH does *not* support either end-to-end or single-hop message acknowledgement. This is a specific design decision to more easily support simultaneous dynamic message routing and avoid the significant complexity and resource expense of message buffering. This does not degrade delivery reliability (for meshed nodes); it is very hard for a message to get lost as the flood fill algorithm ensures that all nodes receive each message. In general, and for the types of application for which EtherMESH has been designed, message acknowledgement is not a major requirement. For applications that need end-to-end acknowledgment, or to support nodes that drop out and then return ‘later’, it is generally a simple affair to arrange a suitable higher level communication paradigm. EtherDATA [3] one such example and is designed to maintain shared data in the face of extensive message loss.

6 Security

EtherMESH supports encryption but not authentication or authorization. When available, EtherMESH security is based on the underlying hardware. For the TI CC25xx MCUs, encryption uses AES block cipher and CCM* mode of operation as the underlying security primitive. AES/CCM* security algorithms were developed by external researchers and are used widely in other communication protocols.

AES/CCM* provides the following security features:

- Infrastructure security
- Network access control
- Application data security

6.1 Configuration

EtherMESH does not currently support key distribution and the default key must be provisioned on each device in the network either at compile time or runtime under manufacture or application control.

6.2 Network access control

In a secure EtherMESH network, each device uses the provisioned key to encrypt each frame. Because the EtherMESH uses flood-fill, there is very little side-channel information that may be gleaned from packet distribution.

6.3 Joining a Network

EtherMESH devices are provisioned for a particular mesh ID and encryption key. Joining the mesh is transparent to the application.

7 Miscellaneous

7.1 Channel Configuration

Dynamic channel hopping is implementation dependant. Currently the CC25xx implementation does not currently support channel hopping. These devices are currently provisioned for a particular channel. However, devices can change channel, and other mesh joining details (ID and encryption key) as part of their profile attributes, under application control.

7.2 Configuring the mesh ID

EtherMESH devices are generally provisioned for a particular mesh ID. However, devices can change mesh ID, and other mesh joining details (channel and encryption key), under application control.

7.3 Maximum Payload Size

The maximum payload size is implementation specific.

7.4 Fragmentation

EtherMESH does not currently support fragmentation. However, messages larger than the max payload size can be split by the application. Note that this is not generally a problem for the types of application EtherMESH is designed to support. In particular, EtherDATA [3], a data management layer that sits on top of EtherMESH, routinely manages data sets larger than the max payload size.

7.4 Memory Requirements

EtherMESH uses 1.2Kb of RAM and ~6.5Kb flash. Moving some tables and config items to flash may further reduce the ram footprint, however, this is at the expense of a slight performance penalty.