ONE NET

wireless control for everyone

ONE-NET MULTI-HOP AND ROUTING

Version 2.3.0

August 14, 2012

Table of Contents

1 Overview **4**

2 Multi-Hop Repeaters **4**

3 Device Tables **4**

4 Client-To-Client Communication **5**

4.1 Scenario 1 – Devices Are Within Range *5*

4.2 Scenario 2 – Devices Are Not Within Range *5*

4.2.1 Scenario 2a – Multi-Hop Not Attempted 6

4.2.2 Scenario 2b – Multi-Hop Succeeds With One Hop 6

4.2.3 Calculating the Response Timeout Time 7

4.2.4 Scenario 2c – Multi-Hop Succeeds With Two Hops 9

4.2.5 Scenario 2c – Multi-Hop Fails 11

4.3 Compile Time Constants *11*

5 Sending To Devices Which Are Already In The Device Table **12**

6 Overruling the Number Of Hops From The Application Code **12**

7 Routing And “Spoofing” Connectivity Using The “Range Test” Option **13**

8 Application-Level Code **13**

1 Overview

Multi-Hop and Routing in ONE-NET is covered in several other documents besides this one. In particular please see the **ONE-NET Specification** and **Block And Stream Messages In ONE-NET** documents. This document attempts to cover things that were not covered in those documents.

2 Multi-Hop Repeaters

ONE-NET devices cannot use Multi-Hop if there is not at least one Multi-Hop Repeater available. Multi-Hop Repeaters are always client devices. ONE-NET devices keep track of both the number of Multi-Hop-Capable and Multi-Hop Repeater devices in the network. Every time a device is added to or deleted from the network, the Master device updates all clients of the new device's DID. In addition, the Master informs all clients of the number of Multi-Hop-capable and Multi-Hop-Repeater-capable devices in the network. If two devices are not in range and there is at least one Multi-Hop Repeater device in the network (not including the source or destination devices!), Multi-Hop is attempted.

3 Device Tables

Devices "keep track" of other devices using device tables. A device table is an array of devices listing their Device IDs and things about them like their Features and settings. Masters keep track of all Client devices. All Client devices keep track of the Master. Two elements of this are the Features of the other device and the number of hops between the two devices. Thus, the master and Client will always know the other's Features and how many hops, if any, it takes to communicate. Client-to-Client communication can be more difficult. Clients will keep track of SOME other clients but not all. For those clients that are not kept track of (i.e. not on the table), some guesswork must take place. For those devices that are on the device table, a much more intelligent guess can be made.

4 Client-To-Client Communication

Consider the following scenario. Client Device 002 wants to communicate with Client Device 005. The two devices have never communicated before. Let’s assume that a unit of Device 002 is peered with a unit of Device 005 and a status change has taken place on that unit of Device 002. Device 002 wants to send its updated status to Device 005.

Device 002 has no idea whether device 005 is within range of it. It will make a guess that it is within range and send the message. It will send a non-Multi-Hop packet.

4.1 Scenario 1 – Devices Are Within Range

If Device 005 is in fact in range it will respond to the packet. Client 002 will have successfully communicated with Client 005. The two devices will add each other to their Device Tables. Two elements of that table are the Device’s Features and the number of hops it took to communicate. Each will be stored. The number of hops will be 0 since they are within range. Devices 002 and 005 achieved communication without using Multi-Hop.

4.2 Scenario 2 – Devices Are Not Within Range

Now assume that device 005 is NOT within range. Device 002 will send the non-Multi-Hop message as before. The message will not be responded to since device 005 cannot hear the message. Device 002 will try, the message will time out after 50 milliseconds with no response, 002 will re-send, it will time out again after 50 milliseconds, then device 002 will send again, not get a response again, re-send again, etc. It will try and fail a total of 8 times, waiting 50 milliseconds each time.

At this point, Device 002 knows that 005 is either not listening for messages or is out of range. It will either give up or try Multi-Hop.

4.2.1 Scenario 2a – Multi-Hop Not Attempted

How does it decide whether to try Multi-Hop? Device 002 will try Multi-Hop if all of the following are true.

1. Device 002 is Multi-Hop-capable
2. Device 005's features are either unknown or Device 002 is known to be Multi-Hop-capable.
3. There is at least one Multi-Hop-capable repeater in the network other than Devices 002 and 005.

In our particular scenario, device 005's features are unknown, so item 2 is true. Device 002 will definitely know if IT is Multi-Hop-capable. If it is not, clearly it cannot use Multi-Hop and the message attempt will terminate with a failure. All device's know the total number of Multi-Hop Repeaters. From the total, the device will subtract itself if it is a Multi-Hop Repeater. If the other device's features are known and it is known to be a Multi-Hop-Repeater, t is also subtracted from the total. If the total number of repeaters remaining is not a positive number, Multi-Hop cannot be attempted and the message will fail.

4.2.2 Scenario 2b – Multi-Hop Succeeds With One Hop

Assume that all three criteria from Scenario 2b are true. Multi-Hop will be attempted. A best guess of how many hops are needed is made. Since the first guess (0) did not work, 1 will be tried. As specified in the ONE-NET Specification document, Multi-Hop packets contain two values (Hops and Max Hops). In this case, the number of hops guessed will become the Max Hops field. The Hops field represents the number of hops SO FAR, so it always starts as 0. So device 002 tries again with a Multi-Hop packet where Hops is 0 and Max Hops is 1.

Recall that in the last attempt, it tried 8 times. Each time, it waited 50 milliseconds before assuming that there would be no response. In that case (no hops, all that needed to happen was that a data message needed to be sent, then an ACK returned). 50 milliseconds seemed like sufficient time for that to occur.

Now, however, we are assuming that there will be a repeater involved. At best, the message will be sent, the repeater will receive it, then resend it, then the recipient will receive it, then send an ACK, which the repeater will send the ACK. Finally device 002 will receive the ACK. There are twice as many messages being sent, plus one has to expect some processing time on the part of the Repeater. Hence device 002 will need to wait a little bit longer for the response. How long should it wait?

4.2.3 Calculating the Response Timeout Time

There are two compile-time defined constants that are relevant.

1. ONE_NET_RESPONSE_TIME_OUT (defined as 50 ms)
2. ONE_NET_MH_LATENCY (defined as 5 ms)

The 50 millisecond response timeout is the maximum time needed for the data packet and ACK to be transmitted, plus some time for the recipient device to process the packet and wait for a clear channel. 50 milliseconds is the amount of time needed for ONE device to do this. We now have two devices involved. A repeater and the recipient. So we multiply the number of devices involved times 50 milliseconds, then we add a 5 second "fudge factor" for each repeater. The formula for calculating the response timeout in milliseconds is thus as follows…

50 + (Max number of Hops x 55).

In other words, start with 50. Add 55 for each Hop / Repeater.

Max Hops	Response Timeout (ms)
0	50
1	105
2	160
3	215
4	270
5	325
6	380
7	435

Figure 4.1 –Response Timeouts

Hence for our example, with max Hops equal to 1, device 002 will wait 105 milliseconds for a response before trying again.

Let's assume that device 003 is a Multi-Hop-Repeater and that it is in range of both devices 002 and 005.

003 will receive the packet and look at the addresses. It will notice that it is not the source or the recipient. It will now check whether this is a Multi-Hop packet. If it is not a Multi-Hop packet, it will ignore it. In this case, it is a Multi-Hop packet. Figures 4.1 and 4.2 from the ONE-NET Specification document are combined here with only the relevant fields

RPTR DID	DST DID	SRC DID	PTYP	HOPS
RAW – 12 BITS	RAW – 12 BITS	RAW – 12 BITS	Determines Whether Multi-Hop	RAW – Hops – 3 bits Max Hops – 3 bits

Figure 4.2 – Multi-Hop Packet Information

The packet sent by device 002 and received by the Repeater 003 is as follows.

RPTR DID	DST DID	SRC DID	PTYP	HOPS
002	005	002	Multi-Hop = Yes	Hops = 0 Max Hops = 1

Figure 4.3 Data Packet

Device 003 has received this packet. It notices that it is not the sender or recipient. It notices that the packet is a Multi-Hop packet. Finally, it notices that the number of hops is less than the maximum number of hops. Therefore device 003 will repeat this packet. It will change two things: it will add one to the number of hops and it will replace the Repeater DID field with its own Device ID. The repeated packet is therefore this.

RPTR DID	DST DID	SRC DID	PTYP	HOPS
003	005	002	Multi-Hop = Yes	Hops = 1 Max Hops = 1

Figure 4.4 – Repeated Data Packet

Device 005 receives this packet. If device 005 DOES NOT have Multi-Hop capability, it will ignore the packet. Let's assume it does have that capability and sends back a response. As before, the initial number of hops is 0. The Max Hops will be the number of hops it took to get there, in this case 1.

RPTR DID	DST DID	SRC DID	PTYP	HOPS
005	002	005	Multi-Hop = Yes	Hops = 0 Max Hops = 1

Figure 4.5 – Response Packet

The process repeats itself. Device 003 will repeat the response packet.

RPTR DID	DST DID	SRC DID	PTYP	HOPS
003	002	005	Multi-Hop = Yes	Hops = 1 Max Hops = 1

Figure 4.6 – Repeated Response Packet

Devices 002 and 005 are now able to communicate with one hop. After the transaction is over, they will have each other's features and will store the other device in their device tables for further use. The number of hops will be 1.

4.2.4 Scenario 2c – Multi-Hop Succeeds With Two Hops

Again assume that device 002 is within range of device 003, but not device 005. Now let's also assume that device 003 is NOT within range of device 005. However, assume that device 004 is a repeater as well and within range of devices 003 and 005. Also assume that device 004 is not within range of 002. In other words, the route that the message must take is this:

002→003→004→005→004→003→002

As before, device 002 will attempt to communicate with 005 first with no hops, which will fail. Then it will try again with one hop. Again, 002 sends, 003 repeats. Now 004 hears 003's repeated packet.

RPTR DID	DST DID	SRC DID	PTYP	HOPS
002	005	002	Multi-Hop = Yes	Hops = 0 Max Hops = 1

Figure 4.7 Data Packet

RPTR DID	DST DID	SRC DID	PTYP	HOPS
003	005	002	Multi-Hop = Yes	Hops = 1 Max Hops = 1

Figure 4.8 – Repeated Data Packet Heard by 004

004 goes through the same process that 003 did. It confirms that it is neither the source or the recipient and that it is a Multi-Hop packet. Both are true. The last step is to make sure that the number of hops is fewer than the maximum number of hops. In this case, it is not. Both are 1. Hence 004 will NOT repeat this packet. Hence 005 will never hear it.

As before, 002 will re-attempt 8 times with no response. From the earlier chart, we can see that the time device 002 waits for a response is 105 milliseconds. This attempt will fail. Device 002 will now try two hops as the maximum number of hops. The time to wait for a response, from the chart, is therefore now 160 milliseconds

RPTR DID	DST DID	SRC DID	PTYP	HOPS
002	005	002	Multi-Hop = Yes	Hops = 0 Max Hops = 2

Figure 4.9 Data Packet

RPTR DID	DST DID	SRC DID	PTYP	HOPS
003	005	002	Multi-Hop = Yes	Hops = 1 Max Hops = 2

Figure 4.10 – Repeated Data Packet Heard by 004

This time, the number of hops is fewer than the Maximum number of hops, so device 004 will repeat.

RPTR DID	DST DID	SRC DID	PTYP	HOPS
004	005	002	Multi-Hop = Yes	Hops = 2 Max Hops = 2

Figure 4.11 –Data Packet Repeated by 004, heard By 005

The responses will be by the same method, only in reverse order.

The data message will get through, as will the response messages. As before, the devices will store each other’s features and the number of hops in their tables.

4.2.5 Scenario 2c – Multi-Hop Fails

Multi-Hop will be re-attempted with Max Hops steadily increasing until the Max Hops field equals the total number of Repeaters in the system or 7, whichever is less. At that point, if the message has not succeeded, failure is declared.

4.3 Compile Time Constants

Four compile-time constants can be changed and experimented with and developers are encouraged to do so in order to optimize their network. Two have been mentioned already.

1. ONE_NET_RESPONSE_TIME_OUT: Time to wait for a response to a non-Multi-Hop packet (defined as 50 ms).
2. ONE_NET_MH_LATENCY: Processing time for each repeater between receiving a message and repeating it (defined as 5 ms).
3. ON_MAX_RETRY: Maximum number of tries to retry for a give value of Max Hops (defined as 8).
4. ON_MAX_HOPS_LIMIT: The highest value that Max Hops can be (defined as 7). Note that this value MUST NEVER BE GIVEN A VALUE HIHER THAN 7 because it must be representable in three bits.

These are the general values defined by ONE-NET. They may not be a good fit for all networks. Changing them can increase efficiency drastically.

5 Sending To Devices Which Are Already In The Device Table

Sending to devices which are ALREADY in the device table is exactly the same as sending to devices which are not with one exception. Much of the guesswork has been removed. The device's features will already be known so devices won't accidentally send a Multi-Hop packet to a device that is not Multi-Hop-capable. Two, the number of hops are in the table. Chances are that if a message took two hops the last time, it will take two hops this time. Hence the code will skip the 16 failed attempts at sending with zero and one hops and go directly to trying two hops. Clearly this speeds things up!

6 Overruling the Number Of Hops From The Application Code

Sometimes the application code will know ahead of time how many hops are needed regardless of the table. The application code can overrule the maximum number of hops involved at any time. In particular, several application-level functions are passed an on_txn_t object as a parameter, which contains the number of hops and the maximum number of hops and can be changed. In particular, the one_net_adjust_hops() application code function is called whenever a transaction is setup. The number of hops can be changed in that function.

7 Routing And “Spoofing” Connectivity Using The “Range Test” Option

There are times, either when debugging or at other times, when devices that are actually in range of each other want to pretend they are not. This is an excellent way to test Multi-Hop without having to physically place devices hundreds of meters apart. In addition, the application code may want to specify a particular route. It can do that by having repeaters only repeat for certain devices. Devices using range testing store an array of Device IDs. All devices that are NOT on that list will be ignored. For more information on range testing, please see the **Debugging Tools And Techniques** document.

8 Application-Level Code

Developers implementing Multi-Hop must write the following application-level code. See the one_net_port_specific.h file for more details. See the Evaluation Board project code for examples.

1. one_net_adjust_hops: Called every time a transaction starts. Allows the application level code to change the number of hops to attempt when sending.