RDM discovery: How does it work?

BY SIMON NEWTON

The RDM discovery process contains a number of corner cases which, if not handled correctly, can cause controllers to crash.

WRITING AN EFFICIENT AND RELIABLE DISCOVERY ROUTINE is one of the more challenging tasks when building an RDM controller. The RDM discovery procedure may appear straightforward at first, but in the real world, responders do not always behave correctly, so controllers must handle abnormal behavior if they are to operate reliably.

RDM discovery overview

The ANSI E1.20 standard describes the RDM discovery process, and pseudo-code for an RDM discovery routine is provided in Appendix E. I'll provide a brief summary here. For further details you should obtain a copy of the standard from The ESTA Foundation.

Each RDM device, whether a controller or a responder, is assigned a six-byte address called a Unique Identifier (UID). The first two bytes of a UID are the ESTA Manufacturer ID and the last four bytes the Device ID. UIDs are represented as hexadecimal numbers, with a colon separating the Manufacturer ID and the Device ID, e.g. 4845:000000C2. The UID of FFFF:FFFFFFFF **i**s reserved for the *broadcast* UID, which is used to address all responders on an RDM link. Each manufacturer also has a *vendorcast* address, made up of the ESTA Manufacturer ID and a Device ID of FFFFFFFF. For example, the UID that represents all responders made by Howard Eaton Lighting is 4845:FFFFFFFF.

. . . in the real world, responders do not always behave correctly, so controllers must handle abnormal behavior if they are to operate reliably.

There are three types of RDM messages used during the discovery process, all of which use the DISCOVERY_COMMAND command class. They are:

- DISC_UNIQUE_BRANCH, which takes two parameters, a lower and an upper UID. Any non-muted responders with a UID within this range (inclusive) must respond.
- DISC_MUTE, which mutes a responder
- DISC_UN_MUTE, which un-mutes a responder

The end goal of the discovery process is for the controller to locate all responders on the RDM line. This is achieved by building a set of UIDs containing the UID of each responder present on the line. There are many different ways to implement the discovery process. One (simplified) method is as follows:

- **Step 1**: Clear the set of UIDs.
- **Step 2**: Send a DISC_UNMUTE to the broadcast UID to unmute all responders.
- **Step 3:** Set the *lower UID* to 0000:00000000 and the *upper UID* to FFFF:FFFFFFFE.
- **Step 4**: Send a DISC_UNIQUE_BRANCH with the range of [lower UID, upper UID].

This will result in one of the following:

No response, indicating there are no unmuted responders within this range (**Figure 1**).

A response with a valid checksum (**Figure 2**). The controller can then extract the UID from the response data and send a DISC_MUTE message addressed to the responder. If the controller receives an ACK, the UID represents a valid responder and can be added to the set of known UIDs.

A collision (**Figure 3**). In this case a controller should divide the range in two e.g. [0000:00000000, 7FFF:FFFFFFFF], [8000:00000000, FFFF:FFFFFFFE] and repeat Step 4 for each range to isolate where the collision is occurring.

In a perfect system, once the entire process is complete, all responders will be muted and sending a DISC_UNIQUE_BRANCH with a range of [0000:00000000, FFFF:FFFFFFFE] will elicit no further responses.

Simple discovery optimizations

One obvious optimization we can make to the discovery process is to store the list of responders found in previous runs and attempt to mute them before sending any DISC_UNIQUE_BRANCH messages. After Step 2 we can add:

- **Step 2.1**: For each previously discovered responder, send a DISC_MUTE message addressed to the corresponding UID. If a reply is received, the controller can add the UID to the set of discovered UIDs.

This process is often referred to as *incremental discovery*.

Unfortunately, what I've described here only works in well-behaved systems, where there are no lost messages and where all responders follow the standard correctly. If a controller uses the process above, the addition of a misbehaving responder to the system can result in other responders being undiscoverable, or, worse still, the controller looping indefinitely or crashing. The rest of this article outlines several situations that can occur, and provides some recommendations on how controllers can detect and handle such cases.

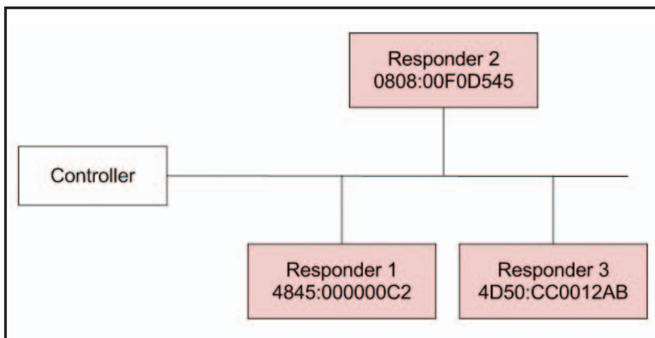

Figure 1 – Muted responders are shown with a pink background. No response is received when there are no unmuted responders within the specific range.

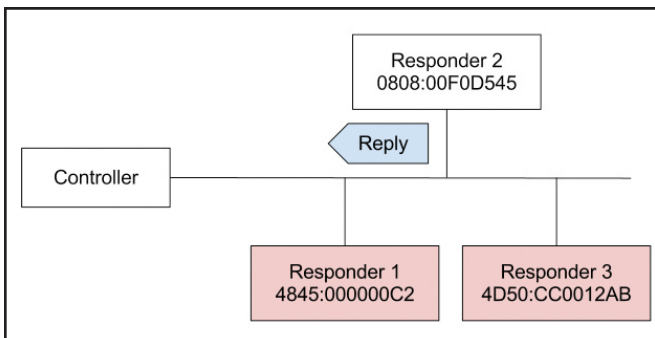

Figure 2 – When there is only one unmuted responder within the range, the controller will receive a single reply.

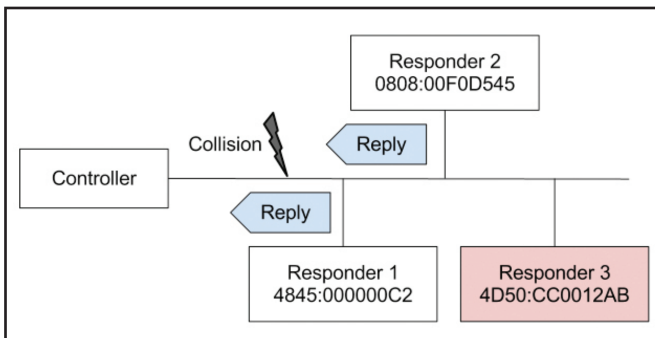

Figure 3 – If more than one responder replies at the same time, a collision will occur.

A dropped DISC_UNMUTE message

The RDM protocol does not guarantee that broadcast messages are received by any or all of the responders. The message may be corrupted by line noise, or a responder may be busy with another task. Upon receiving a broadcast SET message, many responders will have to update persistent storage, possibly causing the next RDM message to be missed. This means if a controller sends a broadcast set (e.g. SET IDENTIFY_DEVICE off) and then immediately follows with a SET DISC_UNMUTE message, some responders may not unmute correctly.

This is a specific example of a class of RDM timing problems that will be covered further in a separate article. For now, my recommendation is to ensure a delay between a broadcast SET message and the following RDM message and to consider sending the DISC_UNMUTE message twice.

> By understanding these cases and by using the suggestions discussed, manufacturers can build controllers that are more resilient to non-conforming responders and data loss.

A missed DISC_UNIQUE_BRANCH message

The timing problems that affect the broadcast DISC_UNMUTE message can also affect the DISC_UNIQUE_BRANCH message. A common one occurs when controllers send a DISC_UNMUTE message, followed immediately by a DISC_UNIQUE_BRANCH. Some responders will miss the DISC_UNIQUE_BRANCH message, causing them to fail to appear in the set of UIDs.

As before, the recommendation is to add a small delay between DISC_UNMUTE and DISC_UNIQUE_BRANCH messages, and to resend a DISC_UNIQUE_BRANCH for the interval [0000:00000000, FFFF:FFFFFFFE] once the controller believes all responders have been discovered. The latter will also help in the case of proxied responders, which are discussed below.

A responder that replies outside the given UID range

A responder may have bugs in the code that performs the UID matching, which can result in it replying to a DISC_UNIQUE_BRANCH message not addressed to it. One responder I have seen replied when only the Manufacturer ID portion of the UID stipulated in the DISC_UNIQUE_BRANCH message matched its own. Depending on the type of bug, the discovery process may sometimes proceed. Generally, the other responders on the line are found and muted first before the bad responder is located.

A pathological case is a responder that replies to every DISC_UNIQUE_BRANCH request, preventing the discovery of all other responders. There is nothing a controller can do to defend against this.

A responder that does not reply when within the UID range

Contrary to the case above, a responder may fail to respond to a DISC_UNIQUE_BRANCH request for an interval that encompasses its UID. This could be caused by either a corrupted DISC_UNIQUE_BRANCH message or by a bug in the responder code. A common example of this is the off-by-one case, where a responder fails to reply if its UID is at the endpoint of the range (remember the upper and lower limits of the DISC_UNIQUE_BRANCH message are inclusive). This can cause responders to disappear suddenly when an interval that previously triggered a collision is divided in two. Without correct detection, controllers may loop indefinitely as they alternate between the two branch levels, trying to locate the missing responders.

The end goal of the discovery process is for the controller to locate all responders on the RDM line.

A responder that does not acknowledge a mute command

A responder may not reply to the MUTE_DEVICE message. This behavior must be differentiated from the case where the MUTE_DEVICE message is corrupted or lost, so the responder can never reply. The pseudo-code in the E1.20 standard accounts for this by attempting to mute each responder up to ten times before continuing.

A responder that never mutes

Not to be confused with the scenario above, a responder may acknowledge the MUTE_DEVICE request but continue to act in an unmuted fashion and respond to subsequent discovery requests. If there is only one responder that behaves in this manner, controllers should still be able to locate all other responders but discovery will be slow, as additional DISC_UNIQUE_BRANCH messages will need to be sent. Discovery is unlikely to succeed if there are multiple responders that have this problem.

To guard against this case, controllers should track the number of DISC_UNIQUE_BRANCH messages sent for each UID range and abort discovery if they cannot make any forward progress after some number of messages have been sent.

A responder that mutes completely

Some responders may get carried away with the mute message and, once muted, halt all further responses. This causes the responder to be discovered, but subsequent GET and SET commands will time-out. This type of bug is usually detected very early in the development process and so is rarely seen in released products.

Discovering proxied responders

The E1.20 standard states that a proxy shall not return more than one DISC_UNIQUE_BRANCH response at any one time. This means that once a proxy has been located and muted, controllers must continue to send DISC_UNIQUE_BRANCH messages and mute responders one at a time until all proxied responders have been discovered. Alternatively, the PROXIED_DEVICES message can be used to obtain a list of all responders located behind a proxy.

The easiest way to deal with RDM proxies is to send an additional DISC_UNIQUE_BRANCH request for each sub-interval once the controller believes all responders in the interval have been located. This enables any responders that appear due to muting a proxy to be found.

Corner cases in responder message handling

Some responders may not behave correctly when unusual combinations of discovery commands and destination UIDs are used. Such commands are not normally used during the discovery process, so responder manufacturers may not necessarily test for them.

An example is responders that do not behave correctly when a

DISC_UNMUTE request is sent to a vendorcast address. This can cause problems if a controller attempts to perform discovery limited to responders from a particular manufacturer.

Another example is a DISC_UNIQUE_BRANCH request sent to a particular UID rather than the broadcast UID. Most controllers will not do this, so it is not a well-tested code path in the responders. Some responders have been observed to reply even when they are not the intended recipient of the request, while others crash entirely.

Due to the lack of support for these and other unusual commands / address pairs, my suggestion is to avoid them completely during the discovery process.

The RDM discovery process can be tricky to get right the first time, as there are many corner cases to consider. By understanding these cases and by using the suggestions discussed, manufacturers can build controllers that are more resilient to non-conforming responders and data loss.

I would like to thank Hamish Dumbreck and Eric Johnson for contributing ideas for this article, based on their experiences in debugging RDM discovery problems. ■

Simon Newton has been interested in lighting control systems since middle school and founded the Open Lighting Project in 2004 with the aim of accelerating the adoption of new control protocols by the industry. He is a member of the Control Protocols Working Group and contributes to the RDM and RDM over IP standards. His day job finds him designing and building the serving infrastructure for a large Internet company in Silicon Valley. Simon can be reached at nomis52@gmail.com.