RMR Route Table Updates

A route manager application may send complete route table updates to an RMR based application, or may send updates. Starting with version 1.0.24, RMR supports the ability for the route manger to send partial route table updates rather than requiring that the whole table be sent each time. Beginning with version 3.2.0, RMR supports the ability to connect to the Route Manager to request an initial table load. This page describes these update interfaces.

RMR Request For Update

Starting with version 3.2.0 (February 18, 2020) RMR supports the ability to create an RMR session back to Routing Manager and to both request table updates and to send the state of the received table. Even when requesting tables, RMR still accepts table updates as before.

To support using an RMR connection for communicating route tables, there are three message types which have been added. These are defined in rmr /RIC_message_types.h included with the development package.

RMRRM_TABLE_DATA	This message type is sent from Route Manager to RMR. The payload contains any valid record described below (e.g. route table entry). The message may contain multiple records; all records in a message must have a trailing newline.
RMRRM_REQ_TABLE	This message type is sent from RMR to the Route Manager to request that a new table be sent. RMR populates some information in the payload, but at the moment it is expected that the payload is ignored by Route Manager
RMRRM_TABLE_STATE	This message type is sent from RMR to Route Manager to communicate the state of the last transmitted table. The payload contains one of two states: • "OK <id>\n" Indicates that the table was received and is installed. <id> is the table ID which was sent on the start record (see the RMR Route Table wiki page for more information about table ID). • "ERR <id> <message-text>\n" Indicates that the table, identified by <id> was not received correctly and is not installed. The tokens following <id> are a brief message that might explain the problem.

Messages arriving with a type which is not listed above are silently dropped.

Request at RMR initialisation

When the `rmr_int()` function is called, the route table collector thread is started. This thread will determine which mode it must operate in (table request or push) and will initialise. When running in *table request mode* the thread will attempt to connect to the Route Manager and will send one `RMRRM_REQ_TABLE` per second. This allows the Route Manager to start after an xAPP, and to ignore any requests until it is ready to send table(s) to xAPPs. Currently RMR only sends a request for a table load when it starts. The Route Manager may still update the table as before provided that the update messages are sent on the RMR connection with the proper message type.

RMR route collector mode

The "mode" of the route collector depends on the state of one or two environment variables. When the (`RMR_RTG_SVC`) environment variable contains a host:port (or ip-addr:port), RMR will assume that it is in "table request mode" and will follow the previously described attempt to request a table. If the `RMR_RTG_SVC` variable contains just the port (as it does today), RMR assumes that it is in "push" mode and will open the listen port and assume that Route Manager will connect and push table information to RMR. This allows RMR to support an environment with an older Route Manager.

When using NNG as the transport library, the RMR_RTG_ISRAW environment variable is used to determine whether or not the Route Manager is sending "raw" NNG messages or if it is sending RMR messages. When RMR_RTG_ISRAW is set to 1, RMR will open the listen port (as defined by `RMR_RTG_SVC`) and expect NNG connections on that port, and plain ASCII messages. If the variable is set to 0, then RMR assumes that the communication will be via RMR messages. (When SI95 is the transport library, the only mechanism supported is RMR messages.)

Route Manager initialisation of RMR

A route manager must be implemented as an RMR application in order to send RMR messages, but it does not need to have the internal RMR route collector thread running. To prevent the collector thread from being started, the Route Manager application should add `RMRFL_NOTHREAD` as an option on the `rmr_init()` call. The Route Manager will be able to communicate directly to xAPPs via the RMR wormhole functions, but will not be able to route messages via a routing table. An example of this is:

```
mrc = rmr_init(  port, max_payload_sz, RMRFL_NOTHREAD );
```

Partial Table Update

RMR expects an update to consist of at least three new line separated records which may arrive in one or multiple messages from the Route Table Manager. The start and end records are similar to the newrt start and end records which are used when sending a complete table. The request tag on each is `updatert`; the exact syntax is shown below:

```
updatert | start | <table-id>
updatert | end | <count>
```

When RMR encounters a start update request in the stream from the Route Manager it will create a copy of the current table which will be updated with any rte/mse requests which are received before the end request is received. The end request carries one additional piece of information: the number of update requests that were sent between start and end. If RMR did not receive the indicated number when the end request is received, the table is discarded.

The <table-id> field is a string supplied by the Route Manager and used to respond with a table status message that RMR will send back to Route Manager.

Entry Updates

The mse and rte requests which are recognised as a part of a full table are used to change existing, and add new, entries. Entries in the current table which are not changed with new data remain the same.

Deleting Entries

One new request is recognised to delete an existing entry. The delete request, syntax below, is needed only if an entry is to be removed from the table; the Route Manger does not need to specifically delete an entry before updating it.

Replacement Order

All of the entries in a single update are applied to the current table atomically, such that if the entries in the update are related, they will all appear for use to the application at the same time. In other words, there will not be a point where only some of the updates have been applied.