White Rabbit Switch
Multiple VLAN Registration Protocol

Juan Luis Manas (Integrasys)
Miguel Baizán (Integrasys)

March 2012
White Rabbit Multiple VLAN Registration Protocol

INTRODUCTION

PROTOCOL OPERATION OVERVIEW

MVRP/MRP GENERAL ARCHITECTURE

DATA MODEL

MVRP OPERATION

IPC COMMUNICATION

SYSTEM TESTS

TEST PROCEDURE

TESTBED CONFIGURATION

TEST RESULTS
White Rabbit Multiple VLAN Registration Protocol

INTRODUCTION

The Multiple VLAN Registration Protocol (MVRP) reduces the cost of maintaining a consistent VLAN configuration in an overall network and permits automatic VLAN reconfiguration in the event of network failures. It is therefore not a surprise that it is in fact one of the VLAN-aware bridge requirements stated in Std 802.1Q Section 5.

This document describes both the MVRP and Multiple Registration Protocol (MRP) implemented for the White Rabbit Switch in accordance to amendment 7 to the IEEE Std 802.1Q-2005 and IEEE “Corrigendum 1: Corrections to the Multiple Registration Protocol”. It briefly outlines the protocol operation and presents the general architecture and main functionalities supported by these protocol implementations.

Although a detailed description of each of the MVRP/MRP components is presented in Std 802.1ak-2007, this document provides a brief overview of their relationships and operation to help getting the overall picture. Also note that informal terminology has been used when considered appropriate to aid the reader not familiarized with the peculiarities of this protocol to roughly understand how it works.

PROTOCOL OPERATION OVERVIEW

The MVRP protocol permits dynamically learning VLANs, conceptually following a similar approach to MAC learning: whenever the switch receives a VLAN announcement in one of its ports, it learns that such port is associated to the announced VLAN\(^1\). It then broadcasts the announcement on the other switch ports\(^2\) so other network switches can also learn the VLAN following the same procedure.

Also, similar to the MAC aging process, MVRP periodically requests participants to confirm that their VLAN announcements are still valid, and periodically cleans VLANs that are no longer in use in any of the switch ports.

---

\(^1\) The MVRP entity registers a dynamic VLAN entry in the VLAN table.

\(^2\) Only ports in the active topology (i.e. in forwarding state)
The protocol gets a little more complex mainly to make it resilient to message losses and to take into account network topology changes, but its main behavior is fairly simple.

Being this a common behavior for a family of high-layer protocols that require registering information in network switches (let it be a multicast address, a VLAN identifier, etc), a standard protocol known as Multiple Registration Protocol (MRP) was designed to handle such common behavior, making it reusable for different applications.

The general structure of an MRP application is shown in the following figure. An MRP application can be regarded as a set of application participants. Each participant is attached to one switch port, controlling protocol interactions that take place through it. Each application participant consists of a declaration component, which keeps track of overall state and handles information exchange, and an application component, which understands such information and knows what to do with it (e.g. dynamically creating a VLAN in the case of MVRP, or registering a Multicast MAC address in the case of MMRP). A propagation component controls internal information distribution among the participants within the switch.\(^3\)

---

\(^3\) The application component is specific for each application while the declaration and propagation components can be reused across applications.
Inside a switch, propagation of application-specific information (a.k.a. attributes) happens within propagation contexts. What a context really means depends on each application, but in general it determines the port and VLAN (if applicable) through which a given information should be sent. Propagation contexts are rapidly updated whenever network topology changes are detected.

Attribute declaration events are exchanged between switches by means of MRP protocol data units, which identify the attribute type, length and value, together with the event type. MRP events mainly signal attribute announcements and withdrawal of announcements, with variants used to also communicate registration state. The following figure shows the structure of MRPDUs.

The way to encode all this information is optimized so that it only takes one Ethernet frame to exchange declaration events for a complete set of up to 4094 VLANs: events that apply to consecutive attribute values (and in particular for MVRP, to consecutive VLANs) are compressed into a so called vector\(^4\).

The declaration component takes both applicant and registrar responsibilities: while as an applicant it announces declarations to other participants (sending MRP messages), as a registrar it records that an attribute value has been registered (it receives MRP messages but does not send them). Applicant and registrar state machines are used to

\(^4\) This way it is neither necessary to encode the attribute type and length nor the value, as it is implicitly indicated by the event relative position within the vector.
keep track of per-attribute per-port per-VLAN (if applicable) announcements and registrations. A per-participant LeaveAll state machine carries on the ‘aging’ process.

In the case of MVRP, VLAN declarations and withdrawal of declarations carried by MRPDUs received in one port are turned by the MVRP application component into dynamic VLAN registrations and deregistrations for that port in the RTU filtering database.

Additionally, static VLAN registrations/deregistrations taking place in the switch as a consequence of management actions are announced to the network by means of MVRP in a similar way.

Note once more that all these artifacts just support the learning process as described in the first few paragraphs of this section.
**MVRP/MRP GENERAL ARCHITECTURE**

The White Rabbit MRP conforms to the standard Full Participant service and can operate in both point-to-point and shared media. The following figure shows the main modules that implement the MRP and MVRP functionality.

The *mrp_com* module handles low-level data transmission and reception using both raw and datagram sockets. It is also in charge of properly tagging and untagging MRP PDUs when the application requires data units to be marked as BPDUs.

The *mrp_pdu* module handles MRP PDU encoding and decoding. It is also responsible for the delivery of parsed attribute events to the appropriate application participant.

The *mrp* module contains the applicant, registrar, LeaveAll and periodic transmission protocol state machines. It interprets MRP events that happen either as a consequence of receiving MRPDUs, timers expiration or management actions and

---

5 i.e. it supports both applicant and registrar functionality.
invokes MRP service primitives to register and deregister attributes appropriately. It also controls attribute propagation among application participants.

These three modules are bundled in the libwrmrp.so library.

The `mvrp` daemon uses the MRP Join.request and Leave.request service primitives to announce VLANs statically registered and deregistered at each switch port. It handles Join.indication and Leave.indication service primitives triggered by the MRP module.

Since `mrp` ignores attribute semantics, it is the application responsibility to maintain an attribute database and assign a per-attribute unique identifiers that the `mrp` is able to handle. Encoding and decoding attribute values is also application responsibility. This requirements translate into a series of callback functions that the application module should configure as part of the initial registration process.

**DATA MODEL**

The data model has been designed to support the following cases:

1. The application needs to maintain a propagation context per-VLAN and delivers tagged MRPDUs. Each participant is therefore attached to a single context. Multiple participants may be attached to the same port (one for each VLAN). The Multiple Multicast Registration Protocol follows this scheme.

![Diagram of data model](image)

2. The application keeps a propagation context per-port and delivers untagged MRPDUs that should be handled as BPDUs (Ports must be set up to accept this traffic even when untagged). The Multiple Vlan Registration Protocol is an example of this type of application.
3. The application keeps a propagation context per-VLAN but sends untagged PDUs (i.e. PDUs are delivered per-participant). This is the typical case of MVRP working over the Multi Spanning Tree Protocol.

Thus, to cope with the general case, the following guidelines apply: first, the application keeps a reference to the list of ports and propagation contexts. Each port has a reference to the list of application participants (one per context) attached to the port. Each participant keeps a list of propagation contexts where propagation should happen and at the same time, each propagation context stores the list of application participants to propagate to within that context\(^6\).

The application controls how all the pieces fit together (as it is the one that defines the context semantics) and rearranges the components when topology changes are detected.

\(^6\) Only those that are attached to a port in forwarding state.
To support protocol transitions, each participant holds a list of applicant and registrar machine states for each declared attribute, indexed by identifiers produced by the application.

A LeaveAll state machine is included in each participant while a Periodic Transmission state machine is defined at each port.

**MVRP OPERATION**

The mvrp daemon registers application configuration into the mrp module and starts the main loop. As part of the initialization process, a propagation context is created and the VLAN table is read from the RTU filtering database and cached; all the ports and participants required for protocol operation are created and attached to each other as part of the initial verification of port operational and forwarding state.

When a participant is initialised, it reads out the cached VLAN table and triggers Join.request service primitives for each registered VLAN. Once they are added to the propagation context, they announce this information also to the other participants.

The MRP protocol, which runs as part of the main loop, is in charge of handling timer events and received PDUs. Several timers run in parallel to control attribute aging and MRPDU transmission.

When an MVRP PDU is received in one port, it is parsed to extract its information about attribute events. This events modify the applicant and registrar state machines and eventually may result in Join or Leave indications being signaled to the MVRP layer and propagated through the other ports.

When a participant needs to transmit a message it requests a transmission opportunity. A transmission opportunity is scheduled by MRP depending on multiple factors (e.g. whether the participant is attached to point-to-point or shared media). Transmission involves first creating the MVRP PDU which is done by checking all applicant state machines for the given participant and inserting appropriate attribute events as indicated by the state transition tables and registrar state machines. Actual attribute values are obtained from the mvrp module based on MAD (MRP attribute declaration) indexes.

Periodically, an aging process takes place for all dynamically learned VLANs. A message is sent for each dynamic VLAN known to a participant, warning about its future removal. This forces other participants that still have interest in a particular VLAN to send back a response message, thus stopping the aging for that VLAN. This protection...
mechanism avoids storing information for VLANs that were removed, when removal could not be properly announced.

When a topology change is detected, MRPDUs are sent to connected switches to inform about VLANs moving from port to port\(^7\). The mvrp checks operational and forwarding state once for each MRP interaction.

Stopping the mvrp daemon makes it unregister the associated Ethernet address from the RTU filtering database so that MVRP PDUs are handled by the RTU as normal traffic once again.

**IPC COMMUNICATION**

Inter-process communication is based on the mini-ipc framework. This model is used to access to the RTU filtering database exported API from the MVRP daemon and to export the MVRP monitoring and control API to the management MIB modules.

The MVRP proxy and MVRP server contain method parameter marshalling and unmarshalling code. The proxy provides the API for methods exported by the MVRP while the server points to the callback functions that are declared as remotely accessible by the MVRP. The MVRP server runs as part of the main loop.

\(^7\) Connection with the STP instance to get the port forwarding state of the port is still pending. Since there is still no instance of STP, all operational ports are considered to be in forwarding state. This is a temporary solution until STP is available.
SYSTEM TESTS

TEST PROCEDURE

TESTBED CONFIGURATION

An MRP Ethernet frame generator was run on an industrial PC Xtreme 6200, running Linux Ubuntu 10.04.1 PC on an Intel(R) Celeron(R) M processor at 1.50GHz.

Generated MRP frames were sent from a Broadcom Netlink Gigabit Ethernet card to a WR switch node v2.0, using a Trendnet TFC-1000 MGB optical converter to connect to one of the available SFP uplink ports (wru1).

The Command Line Interface was run locally in a White Rabbit Switch v2 to monitor and control MVRP operation, using the CLI-SNMP gateway to get and set the configuration parameters through the IEEE 8021-Q-BRIDGE-MIB.
TEST RESULTS

<table>
<thead>
<tr>
<th>Test Case #1</th>
<th>MVRP Management and Configuration.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Check that the MVRP can be enabled and disabled by management both globally and per-port. Check that VLAN restricted registration configuration is taken into account by the MVRP daemon.</td>
</tr>
<tr>
<td>Scenario description</td>
<td>An instance of RTU, MVRP and SNMP daemons are running in the WRS.v2.</td>
</tr>
<tr>
<td>Test results</td>
<td>See following tables:</td>
</tr>
</tbody>
</table>

WR-Switch> show mvrp status
MVRP status: Enabled
WR-Switch> mvrp disable
WR-Switch> show mvrp status
MVRP status: Disabled
WR-Switch> mvrp enable
WR-Switch> show mvrp status
MVRP status: Enabled

WR-Switch> show interface information
### Test Case #2

**Name**

CLI MVRP PDU received

**Goal of the test case**

Check the behaviour of the MVRP daemon when a MVRP PDU is received in a given port. This test is useful to verify that: the MVRP is parsing correctly the PDU, the join indications are correctly generated and handled, the ‘LeaveAll’ timer is working, the leave indications are correctly generated and handled, and the ‘Last PDU Origin’ parameter...
**Scenario description**

An instance of RTU, MVRP and SNMP daemons are running in the WRS.v2.

A dummy PDU is sent from the PC.XTREM through the interface 'eth2'. The PDU carries the JOIN_IN event for the VLANs 2, 3, 4, 5 and 6. The test verifies the MVRP parameters and the VLAN Registration table before and after the transmission of the PDU. Once registered, the VLAN table is read after a given number of seconds, to check that the LeaveAll timer has actually expired and the VLANs have been de-registered.

**Test results**

See following tables:

**MVRP PDU to be sent:**

```
protocol_ver: 0
msg_count : 1
msg[0]
  .attrtype  :1
  .attrlen   :2
  .attr_count:5
  .attr[0]
    ..leaveall_event :0
    ..number_of_values:1
    ..value           :2
    ..event[0]       :JOIN_IN
  .attr[1]
    ..leaveall_event :0
    ..number_of_values:1
    ..value           :3
    ..event[0]       :JOIN_IN
  .attr[2]
    ..leaveall_event :0
    ..number_of_values:1
    ..value           :4
    ..event[0]       :JOIN_IN
  .attr[3]
    ..leaveall_event :0
    ..number_of_values:1
    ..value           :5
    ..event[0]       :JOIN_IN
  .attr[4]
    ..leaveall_event :0
    ..number_of_values:1
    ..value           :6
    ..event[0]       :JOIN_IN
```

**Before PDU transmission:**

```
WR-Switch> show interface information
MVRP
----------------------------------------
Port  PVID  Status   Registration Failed  Last PDU from
      ----  -------- ------------ ----------  ----------------
 0     0     Enabled      *        0           00:00:00:00:00:00
 1     0     Enabled      *        0           00:00:00:00:00:00
 2     0     Enabled      *        0           00:00:00:00:00:00
 3     0     Enabled      *        0           00:00:00:00:00:00
 4     0     Enabled      *        0           00:00:00:00:00:00
 5     0     Enabled      *        0           00:00:00:00:00:00
```
White Rabbit Multiple VLAN Registration Protocol – March 2012

6 0 Enabled * 0 00:00:00:00:00:00:00
7 0 Enabled * 0 00:00:00:00:00:00:00
8 0 Enabled * 0 00:00:00:00:00:00:00
9 0 Enabled * 0 00:00:00:00:00:00:00

WR-Switch> show vlan
VLAN FID Ports
---- --- -----------------------------------
0 0 0, 1, 2, 3, 4, 5, 6, 7, 8,
  9, 10, 11, 12, 13, 14, 15, 16,
  17, 18, 19, 20, 21, 22, 23, 24,
  25, 26, 27, 28, 29, 30, 31,

After PDU transmission:

WR-Switch> show interface information
MVRP
------------------------------------------------------
Port PVID Status Registration Failed Last PDU from
---- ---- -------- ------------ ---------- -----------
0 0 Enabled * 0 00:00:00:00:00:00:00
1 0 Enabled * 0 00:e0:50:00:2:24
2 0 Enabled * 0 00:00:00:00:00:00:00
3 0 Enabled * 0 00:00:00:00:00:00:00
4 0 Enabled * 0 00:00:00:00:00:00:00
5 0 Enabled * 0 00:00:00:00:00:00:00
6 0 Enabled * 0 00:00:00:00:00:00:00
7 0 Enabled * 0 00:00:00:00:00:00:00
8 0 Enabled * 0 00:00:00:00:00:00:00
9 0 Enabled * 0 00:00:00:00:00:00:00

Wait 10-15 seconds

WR-Switch> show vlan
VLAN FID Ports
---- --- -----------------------------------
0 0 0, 1, 2, 3, 4, 5, 6, 7, 8,
  9, 10, 11, 12, 13, 14, 15, 16,
  17, 18, 19, 20, 21, 22, 23, 24,
  25, 26, 27, 28, 29, 30, 31,

Test Case #3

<table>
<thead>
<tr>
<th>Name</th>
<th>CLI MVRP Restricted Registration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal of the test case</td>
<td>Check that the Restricted Registration parameter makes the Registrar state machines ignore the PDU messages when there is not a static VLAN entry classified as ‘Normal</td>
</tr>
</tbody>
</table>
Registration’ in the VLAN Registration table, and thus the ‘Failed Registration’ counter increases. Also check that this counter increases when we have an entry configured as ‘Fixed Registration’ for a given port and VLAN ID.

Scenario description
The scenario is the same that the one described in the test CLI MVRP PDU received. MVRP PDU processing should result in an attempt to register VLANs 2, 3, 4, 5 and 6.

Test results
See following table:

<table>
<thead>
<tr>
<th>Before PDU transmission:</th>
</tr>
</thead>
<tbody>
<tr>
<td>WR-Switch&gt; show vlan</td>
</tr>
<tr>
<td>VLAN   FID    Ports</td>
</tr>
<tr>
<td>----   ---    ----------------------------</td>
</tr>
<tr>
<td>0      0      0, 1, 2, 3, 4, 5, 6, 7, 8,</td>
</tr>
<tr>
<td>9, 10, 11, 12, 13, 14, 15, 16,</td>
</tr>
<tr>
<td>17, 18, 19, 20, 21, 22, 23, 24,</td>
</tr>
<tr>
<td>25, 26, 27, 28, 29, 30, 31,</td>
</tr>
<tr>
<td>WR-Switch&gt; show interface information MVRP</td>
</tr>
<tr>
<td>Port  PVID  Status   Registration Failed      Last PDU from</td>
</tr>
<tr>
<td>----  ----  -------- ------------ --------------  -------------</td>
</tr>
<tr>
<td>0     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>1     0     Enabled  Restricted   0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>2     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>3     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>4     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>5     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>6     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>7     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>8     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>9     0     Enabled      *        0           00:00:00:00:00:00</td>
</tr>
<tr>
<td>WR-Switch&gt; vlan 3 member 5</td>
</tr>
<tr>
<td>WR-Switch&gt; vlan 4 member 1</td>
</tr>
<tr>
<td>WR-Switch&gt; show vlan</td>
</tr>
<tr>
<td>VLAN   FID    Ports</td>
</tr>
<tr>
<td>----   ---    ----------------------------</td>
</tr>
<tr>
<td>0      0      0, 1, 2, 3, 4, 5, 6, 7, 8,</td>
</tr>
<tr>
<td>9, 10, 11, 12, 13, 14, 15, 16,</td>
</tr>
<tr>
<td>17, 18, 19, 20, 21, 22, 23, 24,</td>
</tr>
<tr>
<td>25, 26, 27, 28, 29, 30, 31,</td>
</tr>
<tr>
<td>After PDU transmission:</td>
</tr>
<tr>
<td>WR-Switch&gt; show vlan</td>
</tr>
<tr>
<td>VLAN   FID    Ports</td>
</tr>
<tr>
<td>----   ---    ----------------------------</td>
</tr>
<tr>
<td>0      0      0, 1, 2, 3, 4, 5, 6, 7, 8,</td>
</tr>
</tbody>
</table>

Note that we had VLAN 3 registered for port 5 and VLAN 4 registered for port 4 (i.e. ‘fixed registration’ for the port and VLAN ID pair, and ‘normal registration’ for the other ports and VLAN ID in the port set of each entry). The port 1, which receives the PDU, is configured with Restricted VLAN Registration, meaning that it will just register the VLAN 3 (for which it has ‘normal registration’). The other 4 attribute values in the PDU will increase the Failed Registration counter.
WR-Switch> show interface information

<table>
<thead>
<tr>
<th>Port</th>
<th>PVID</th>
<th>Status</th>
<th>Registration Failed</th>
<th>Last PDU from</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Enabled</td>
<td>Restricted</td>
<td>00:e0:50:00:02:24</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>Enabled</td>
<td>*</td>
<td>00:00:00:00:00:00:00</td>
</tr>
</tbody>
</table>