Application Note

RDPA

RDPA Traffic Management Operation Guide

RDPA-AN100-R

5300 California Avenue • Irvine, CA 92617 • Phone: 949-926-5000 • Fax: 949-926-5203 October 30, 2013

Revision History

Revision	*Date*	*Change Description*
RDPA-AN100-R	10/30/13	Initial release

Broadcom Corporation
5300 California Avenue
Irvine, CA 92617

Table of Contents

List of Figures

About This Document

Purpose and Audience

This document explains the Runner Data Path API (RDPA) Traffic Management (TM) architecture and QoS queue configuration via the CMS Web UI. It is aimed for software engineers using RDPA in their system.

Acronyms and Abbreviations

In most cases, acronyms and abbreviations are defined on first use.

For a comprehensive list of acronyms and other terms used in Broadcom documents, go to: http://www.broadcom.com/press/glossary.php.

Document Conventions

The following conventions may be used in this document:

Convention	***Description***
Bold	User input and actions: for example, type **exit**, click **OK,** press **Alt+C**
`Monospace`	Code: `#include <iostream>` HTML: `<td rowspan = 3>` Command line commands and parameters: `wl [-l] <command>`
< >	Placeholders for *required* elements: enter your <username> or `wl <command>`
[ ]	Indicates *optional* command-line parameters: `wl [-l]` Indicates bit and byte ranges (inclusive): [0:3] or [7:0]

Technical Support

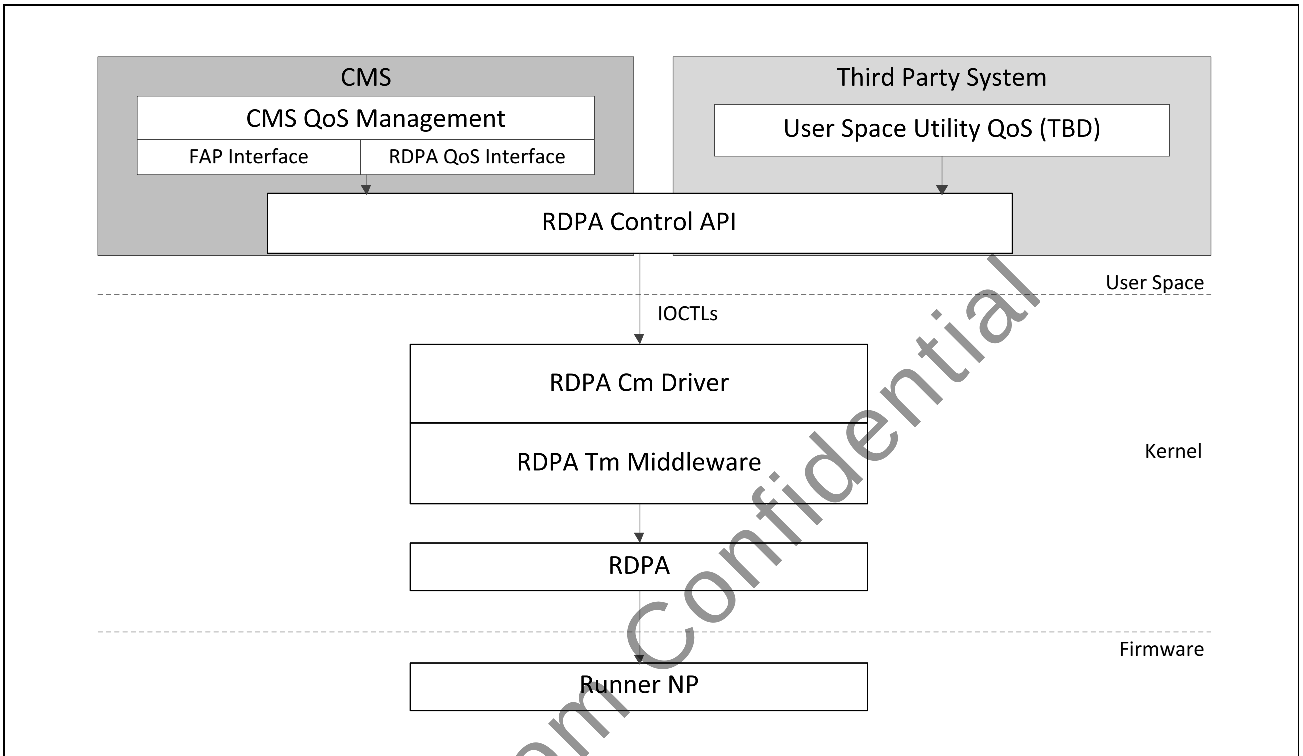
Broadcom provides customer access to a wide range of information, including technical documentation, schematic diagrams, product bill of materials, PCB layout information, and software updates through its customer support portal (https://support.broadcom.com). For a CSP account, contact your Sales or Engineering support representative.

In addition, Broadcom provides other product support through its Downloads and Support site (http://www.broadcom.com/support/).

RDPA TM Architecture

Figure 1 shows the RDPA TM architecture.

Figure 1: RDPA TM Architecture



CMS QoS Management represents an interface between TR-69 management (or Web UI) and the CMS Core APIs. This component is built around the configuration transactions, is stateless and is platform independent.

The RDPA QoS Interface represents a platform dependent glue layer between CMS and RDPA TM. This component translates the CMS transactions to the specific QoS model through the RDPA Control API.

RDPA Control API is the user space API which is responsible for the parameters encapsulation and the IOCTL calls. There is no decision making logic in this component. See the code examples below.

The RDPA Command Driver is a Linux Kernel character device which is responsible for the commands dispatching from/to the user space. The user space IOCTL call is received here and is redirected to the appropriate handler routine.

RDPA TM Middleware is the RDPA/BDMF abstraction layer. The actual RDPA/BDMF calls are encapsulated here. This component is responsible for the RDPA objects allocation/deletion, for the RDPA calls order and internal logic, etc.

The RDPA TM User Space Utility is the user space utility that encapsulates the RDPA TM IOCTL calls in case of a non CMS third-party system. We assume that the third-party residential gateway software will use the tc Linux utility to configure QoS models in Linux and Flow Cache and the rdpactl utility to configure the Runner accelerator QoS. There is also an option to avoid using rdpactl by directly calling the RDPA TM IOCTLs. This creates an open architecture which does not depend on specific customer implementation.

Code Examples

RDPA Control API

```
int rdpaCtl_TmConfig(
   int rootTmId,
   int portId,
   int dir,
   int level,
   int arbiterMode,
   int priority,
   int weight,
   int *pTmId)
{
   int rc = 0;The
   rdpa_drv_ioctl_tm_t tm;

   memset(&tm, 0, sizeof(rdpa_drv_ioctl_tm_t));

   tm.cmd = RDPA_IOCTL_TM_CMD_TM_CONFIG;
   tm.root_tm_id = rootTmId;
   tm.port_id = portId;
   tm.dir = dir;
   tm.level = level;
   tm.arbiter_mode = arbiterMode;
   tm.priority = priority;
   tm.weight = weight;

   rc = __sendTmCommand(&tm);

   if (!rc) {
       *pTmId = tm.tm_id;
       rdpaCtl_debug("NEW TM ID %d, dir %d, level %d, arbiterMode %d",
           *pTmId, dir, level, arbiterMode);
   }
   return rc;
}
```

RDPA Command Driver

```
static long rdpa_cmdIoctl(struct file *filep, unsigned int command, unsigned long arg)
{
   rdpaDrvIoctl_t cmd;
   int ret = RDPA_DRV_SUCCESS;
…
    switch( cmd )
    {
        case RDPA_IOC_TM:
        {
            ret = rdpa_cmd_tm_ioctl(arg);
            break;
        }
        default:
…
    }
    return ret;
} /* rdpa_cmdIoctl */
```

RDPA TM Middleware

```
case RDPA_IOCTL_TM_CMD_TM_REMOVE: {
   bdmf_error_t rc = BDMF_ERR_OK;
   bdmf_object_handle sched = NULL;
   rdpa_egress_tm_key_t tm_key = {};

   tm_key.dir = tm.dir;
   tm_key.index = tm.tmId;

   if ((rc = rdpa_egress_tm_get(&tm_key, &sched))) {
       ret = RDPA_DRV_ERROR;
       goto ioctl_exit;
   }
   if (sched) {
       bdmf_put(sched);
       bdmf_destroy(sched);
   }
}
```

RDPA Traffic Models

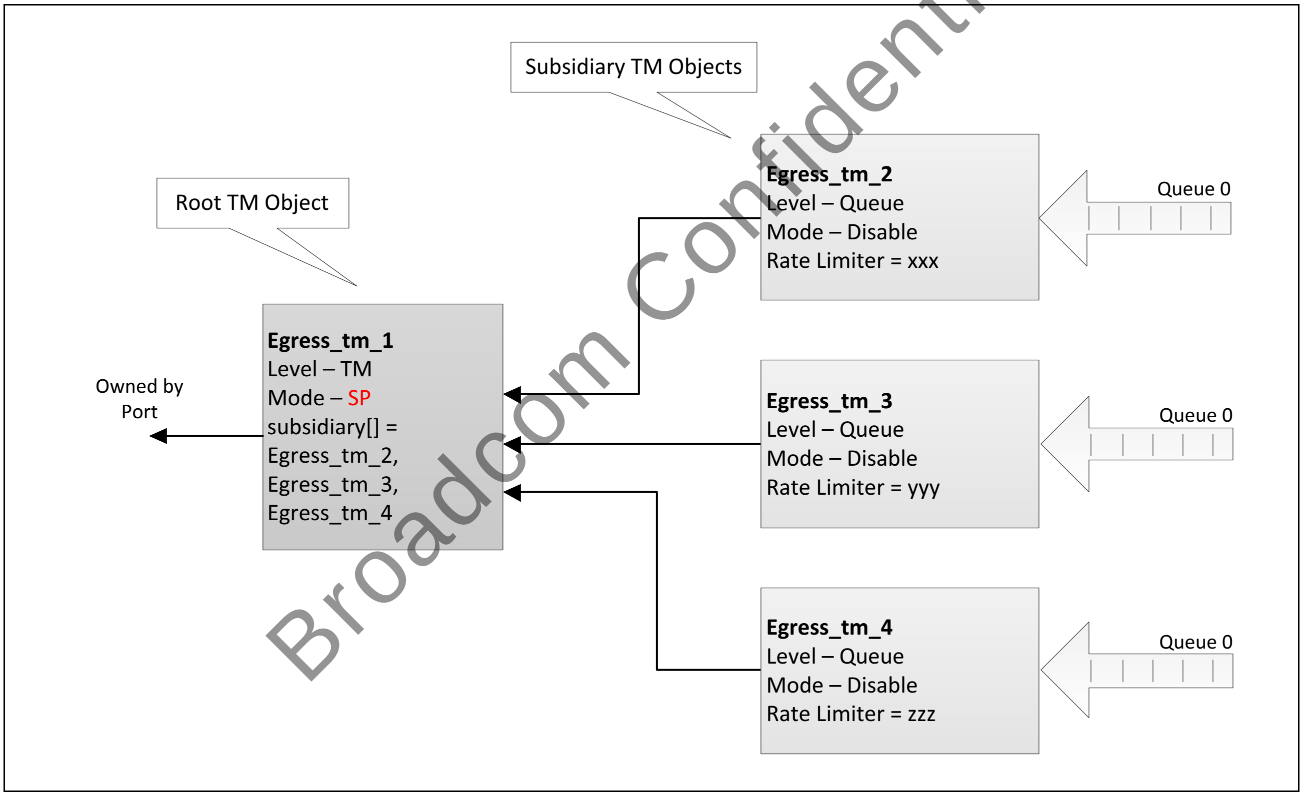
There are various traffic management models supported by RDPA. Currently the following models are implemented by CMS TM:

- Strict Priority (SP)
- Weighted Round Robin (WRR)

Strict Priority

The SP scheduling scheme is created using a single Root TM and a several secondary level TM objects, as shown in Figure 2. The Root TM defines the scheduler mode, in this case SP. The TM objects are configured with Rate Limiter value.

Figure 2: Strict Priority Scheduler



Example: The code below is a command line example for two queues.

Subsidiary TM Object

```
# bs /b/e egress_tm
Object: egress_tm/dir=us,index=39. Object type: egress_tm. Owned by: egress_tm/dir=us,index=37
==============================
                 dir : us
               index : 39
               level : queue
                mode : disable
          overall_rl : no
              enable : yes
                  rl : {af=0,be=0}
queue_cfg[0] : {queue_id=1,drop_threshold=127,weight=0,drop_alg=dt,red_threshold=0,
red_percent=0}
queue_stat[{channel={tcont/index=0},queue_id=1}] : 0
queue_stat[{channel={tcont/index=1},queue_id=1}] : 0
              weight : 0
```

Subsidiary TM Object

```
Object: egress_tm/dir=us,index=38. Object type: egress_tm. Owned by: egress_tm/dir=us,index=37
==============================
                 dir : us
               index : 38
               level : queue
                mode : disable
          overall_rl : no
              enable : yes
                  rl : {af=0,be=0}
queue_cfg[0] : {queue_id=0,drop_threshold=127,weight=0,drop_alg=dt,red_threshold=0,
red_percent=0}
queue_stat[{channel={tcont/index=0},queue_id=0}] : 0
queue_stat[{channel={tcont/index=1},queue_id=0}] : 0
              weight : 0
```

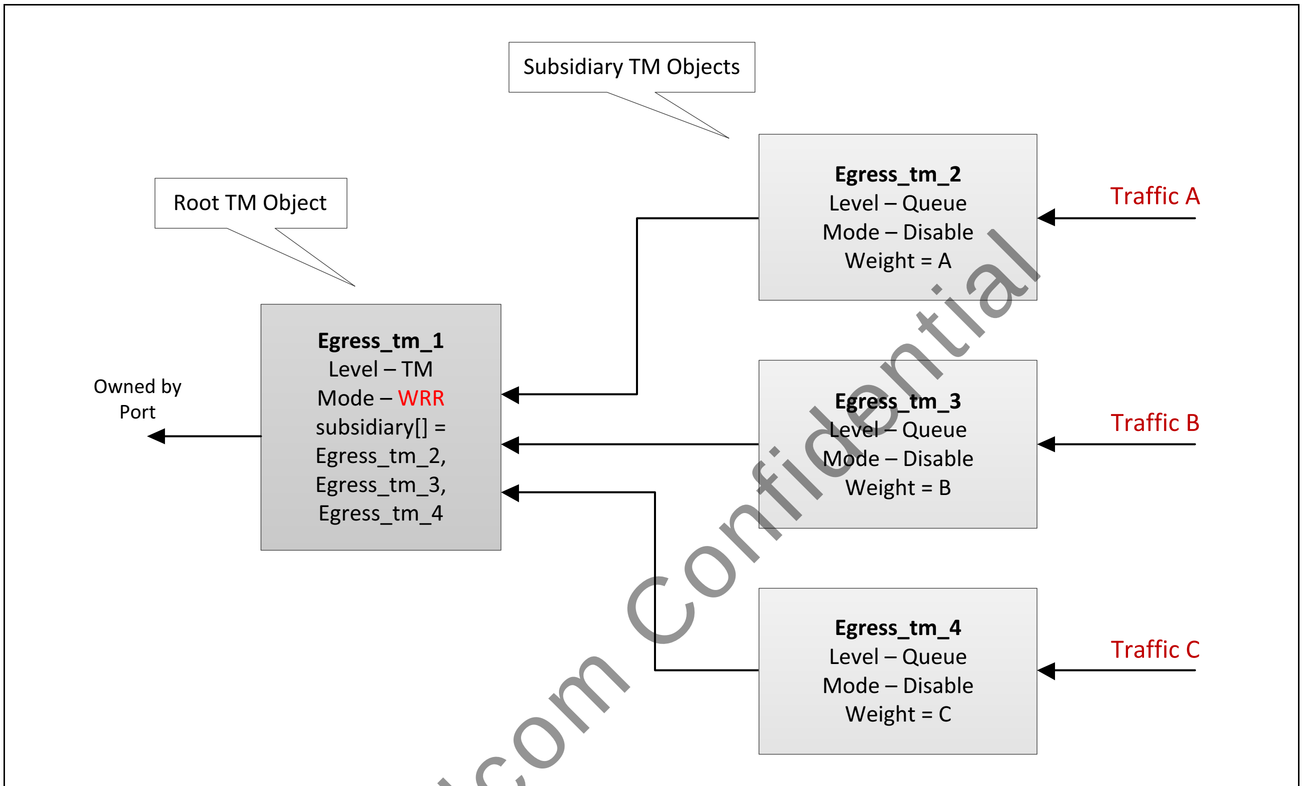
Root TM Object

```
Object: egress_tm/dir=us,index=37. Object type: egress_tm. Owned by: port/index=wan0
==============================
                 dir : us
               index : 37
               level : egress_tm
                mode : sp
          overall_rl : no
              enable : yes
                  rl : {af=0,be=0}
queue_stat[{channel={tcont/index=0},queue_id=1}] : 0
queue_stat[{channel={tcont/index=0},queue_id=0}] : 0
queue_stat[{channel={tcont/index=1},queue_id=1}] : 0
queue_stat[{channel={tcont/index=1},queue_id=0}] : 0
  *other elements not shown*
       subsidiary[6] : {egress_tm/dir=us,index=39}
       subsidiary[7] : {egress_tm/dir=us,index=38}
              weight : 0
```

Weighted Round Robin

The WRR scheduling scheme is built using a single Root TM and a several secondary level TM objects, similar to the SP scheme, see Figure 3. The Root TM defines a scheduler mode – WRR. The TM objects are configured with a queue weight. Weight is defined per TM object and queue weight is not used in this model.

Figure 3: Weighted Round Robin Scheduler



QoS Initialization and Configuration

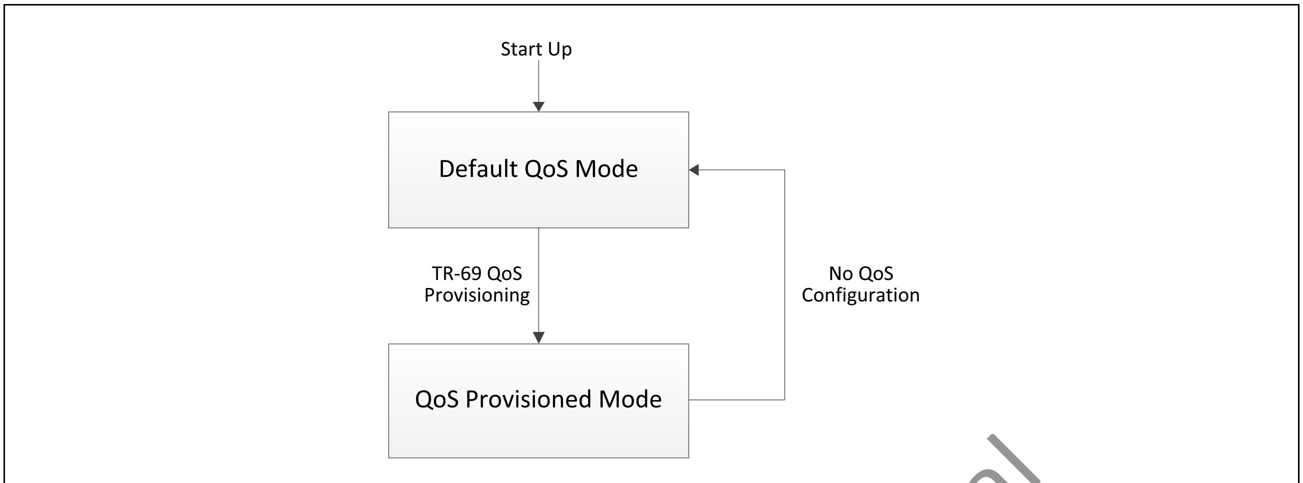
Configuration Modes

CMS TM and QoS configuration are encapsulated in the RDPA TM module. Default QoS configuration is performed during the module initialization at system startup. After startup the CMS System enters the Default QoS Mode.

During the CMS configuration session (TR-69 or Web UI) QoS parameters are provisioned. This is the QoS Provisioned Mode. QoS models for this mode are described in "RDPA Traffic Models".

If the QoS configuration is removed then the system returns to the default QoS mode. The default mode QoS model consists of a Strict Priority scheduler where a single TM object holds eight queues. Figure 4 shows the QoS initializing and provisioning flow.

Figure 4: QoS Initialization and Provisioning



Configuring A QoS Queue

This section gives an example of how to configure the QoS queues using the CMS Web UI.

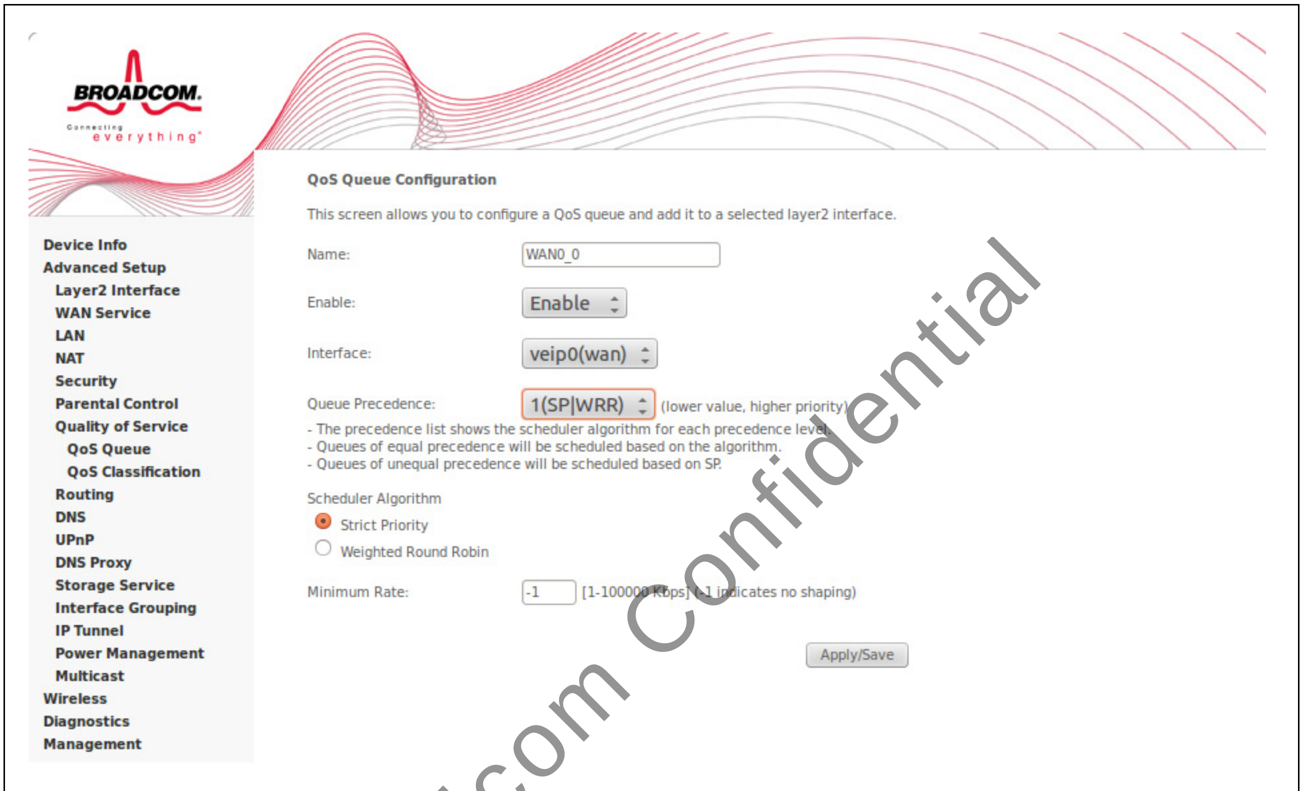
Follow the procedure to define the QoS.

1. Open the Web UI, select the Quality of Service tab, Figure 5.

Figure 5: QoS Queue Management Configuration

2. Check the Enable QoS checkbox.
3. Click **Apply/Save**.
4. Open the QoS Queue tab, click **Add Queue**.
   The QoS Queue Configuration screen opens Figure 6.

Figure 6: QoS Queue Configuration



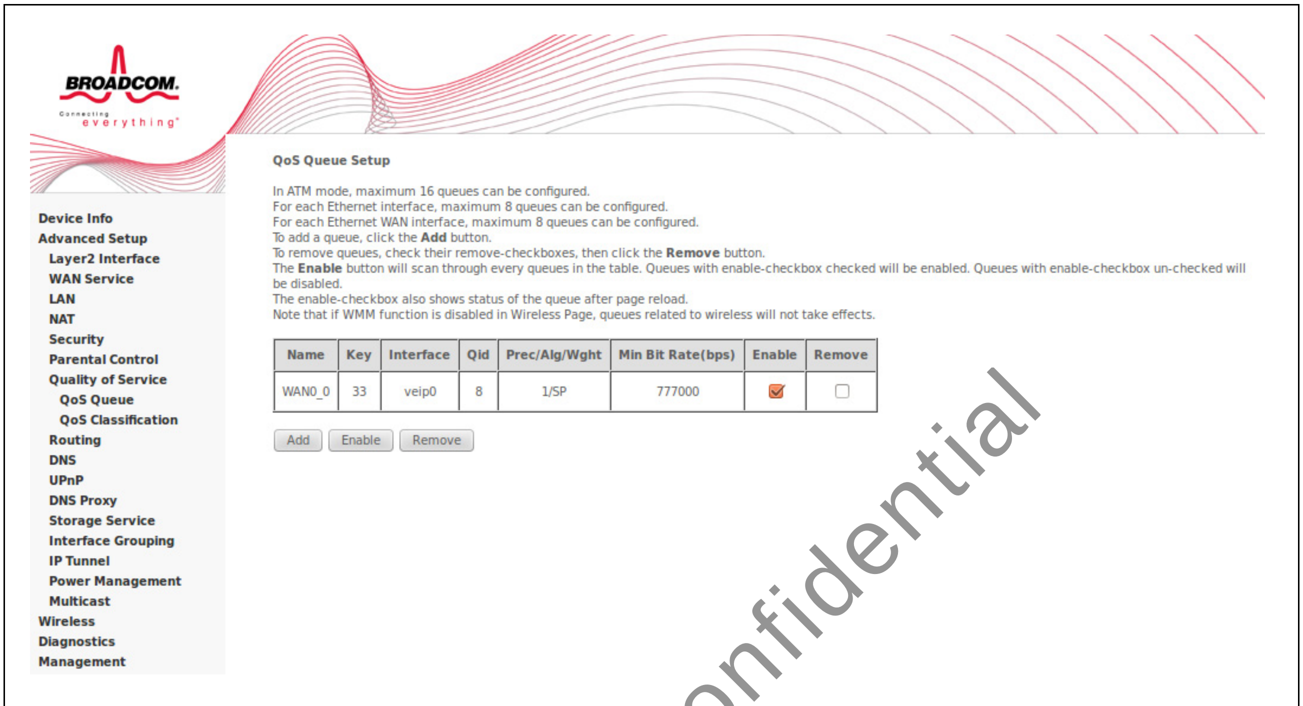
5. Type a queue name.
6. Choose an interface:
   - veip0(wan) - for GPON
   - EPON0 - for EPON
7. Select the queue priority. The lower value is a higher priority.
8. Select a scheduling algorithm.
   - Strict Priority
   - Weighted Round Robin

Note: The scheduling algorithm for subsequent queues is taken from this first configured queue. All other queues will either be SP or WRR.

9. For SP algorithm type in the Rate controller rate in kilobits per second or leave -1 to skip the rate.
   For WRR type in the queue weight.

10. Click **Apply/Save**. The QoS Setup screen is displayed Figure 7.

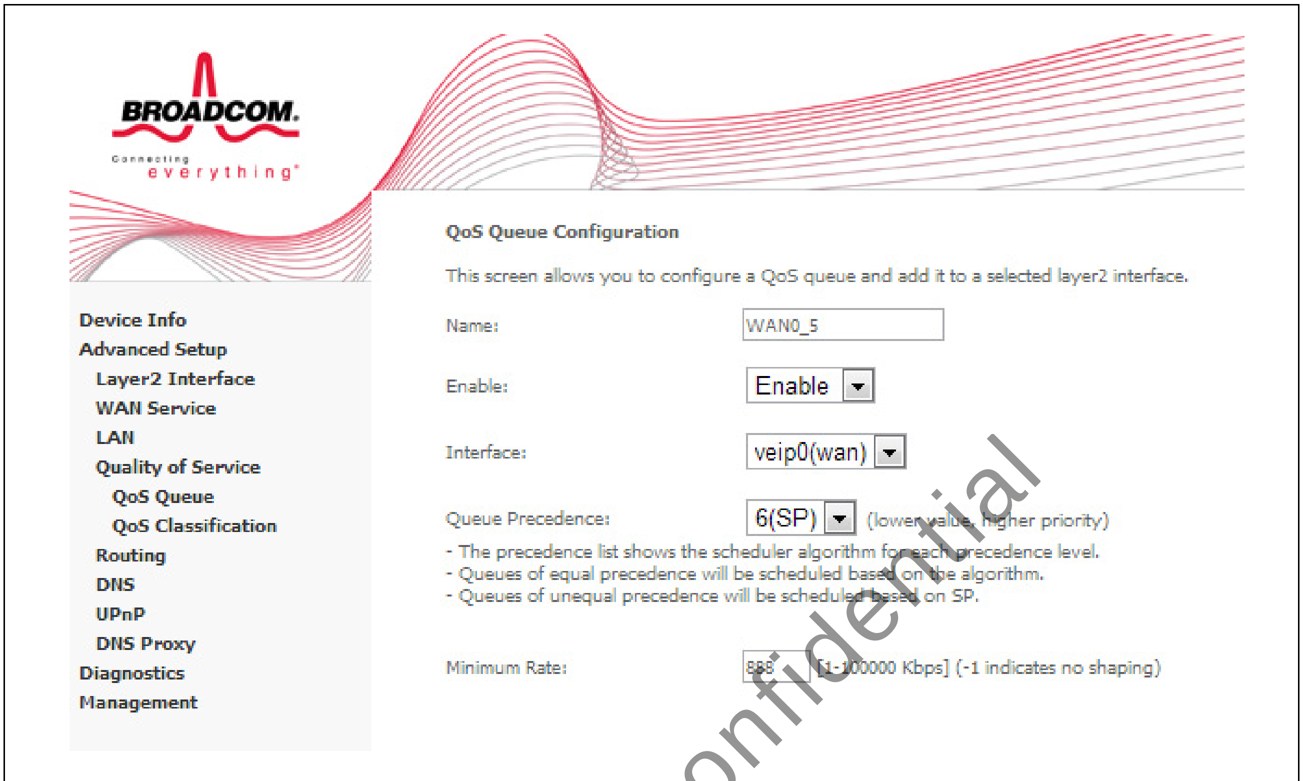
Figure 7: QoS Setup Screen



To add another queue follow the procedure from Step 11.

11. Click **Add**. The QoS Queue Configuration screen opens, but it displays the previously chosen configuration, as shown in Figure 8.

Figure 8: QoS Queue Configuration



12. Type a queue name.

13. Choose an interface:

- veip0(wan) - GPON
- EPON0 - EPON

14. Select the queue priority. Previous queue priorities do not appear in the list.

Note: The algorithm is taken from the first configured queue: SP or WRR. All subsequent queues take the algorithm of the first queue.

15. In this example, the SP algorithm was previously selected, so the rate controller rate is required. Enter kilobits per second or leave -1 to skip the rate.

16. Continue from Step 11 until all required QoS queues have been added and configured.

The downstream queue configuration is very similar, with the following exceptions:

- The SP only scheduling algorithm is available in the Downstream direction (no WRR).
- There is no rate per queue configuration.

RDPA Control APIs

```
/*******************************************************************************
* Function: rdpaCtl_ GetRootTmByIfname
*   portId,              // IN: RDPA Port Id
*   pRootTmId,           // IN: Root Traffic Management ID
*   pbFound              // OUT: TM Found/Not Found
*******************************************************************************/
int rdpaCtl_GetRootTmByPortId(
   int portId,
   int *pRootTmId,
   BOOL *pbFound);

/*******************************************************************************
* Function: rdpaCtl_GetTmByQid
*   portId,              // IN: RDPA Port Id
*   qId,                 // IN: Management Queue Id
*   pTmId,               // OUT: Traffic Management Object  ID
*   pbFound,             // OUT: Tm ID Found or NOT
*******************************************************************************/
int rdpaCtl_GetTmByQid(
   int portId,
   int qId,
   int *pTmId ,
   BOOL *pbFound);

/*******************************************************************************
* Function: rdpaCtl_RootTmConfig
* portId                 // IN: RDPA Port Id
* dir,                   // IN: Tm direction set
* level,                 // IN: Tm type
* arbiterMode            // IN: Tm arbiter mode (SP, WRR, etc.)
* pRootTmId,             // OUT: Root Traffic Management ID
*******************************************************************************/
int rdpaCtl_RootTmConfig(
   int     portId,
   int     dir,
   int     level,
   int     arbiterMode,
   int    * pRootTmId);
```

```
/******************************************************************************
* Function: rdpaCtl_TmConfig
* rootTmId,              // IN: Root Traffic Management (Tm) ID
* portId                 // IN: RDPA interface (port)
* dir,                   // IN: Tm direction set
* level,                 // IN: Tm type ()
* arbiterMode            // IN: Tm arbiter mode (SP, WRR, etc.)
* priority,              // IN: Tm Priority (for SP)
* weight,                // IN: TM Weight (for WRR)
* pTmId,                // OUT: Traffic Management Object  ID
******************************************************************************/
int rdpaCtl_Config(
   int     rootTmId,
   int     portId,
   int     dir,
   int     level,
   int     arbiterMode,
   int     priority,
   int     weight,
   int    *pTmId);

/******************************************************************************
* Function: rdpaCtl_RootTmRemove
*    tmId,                   // IN     :  Traffic Management (Tm) ID
*    dir,                    // IN     :  Tm direction set
* *****************************************************************************/
int rdpaCtl_RootTmRemove(
   int tmId,
   int dir);

/******************************************************************************
* Function: rdpaCtl_TmRemove
*   ifName,                  // IN: Interface name
*   tmId,                    // IN: Traffic Management (Tm) ID
*   dir,                     // IN: Tm direction set
* *****************************************************************************/
int rdpaCtl_TmRemove(
   int tmId,
   int dir);
```

```
/******************************************************************************
 * Function: rdpaCtl_QueueConfig
 * portId                // IN: RDPA interface (port)
 * tmId,                 // IN: Traffic Management (Tm) ID
 * q_id                  // IN: Queue ID
 * qsize,                // IN: Queue size
 * weight,               // IN: TM Weight (for WRR)
 * dir,                  // IN: Tm direction set
 * shapingRate,          // IN: af_rate
 * shapingBurstSize      // IN: be_rate
 * weight                // IN: Queue weight
 ******************************************************************************/
int rdpaCtl_QueueConfig(
   int portId,
   int tmId,
   int q_id,
   int qsize,
   int weight,
   int dir,
   int shaping_rate,
   int shaping_burst_size);

/******************************************************************************
* Function: rdpaCtl_QueueRemove
*    portId               // IN     :  RDPA interface (port)
*    tmId,                // IN     :  Traffic Management (Tm) ID
*    dir,                 // IN     :  Tm direction set
*    qid                  // IN     :  Queue ID
* *****************************************************************************/
int rdpaCtl_QueueRemove(
   int portId,
   int tmId,
   int dir,
   int qid);
```

Connecting everything®

BROADCOM CORPORATION
5300 California Avenue
Irvine, CA 92617

Phone: 949-926-5000
Fax: 949-926-5203
E-mail: info@broadcom.com
Web: www.broadcom.com

RDPA-AN100-R October 30, 2013