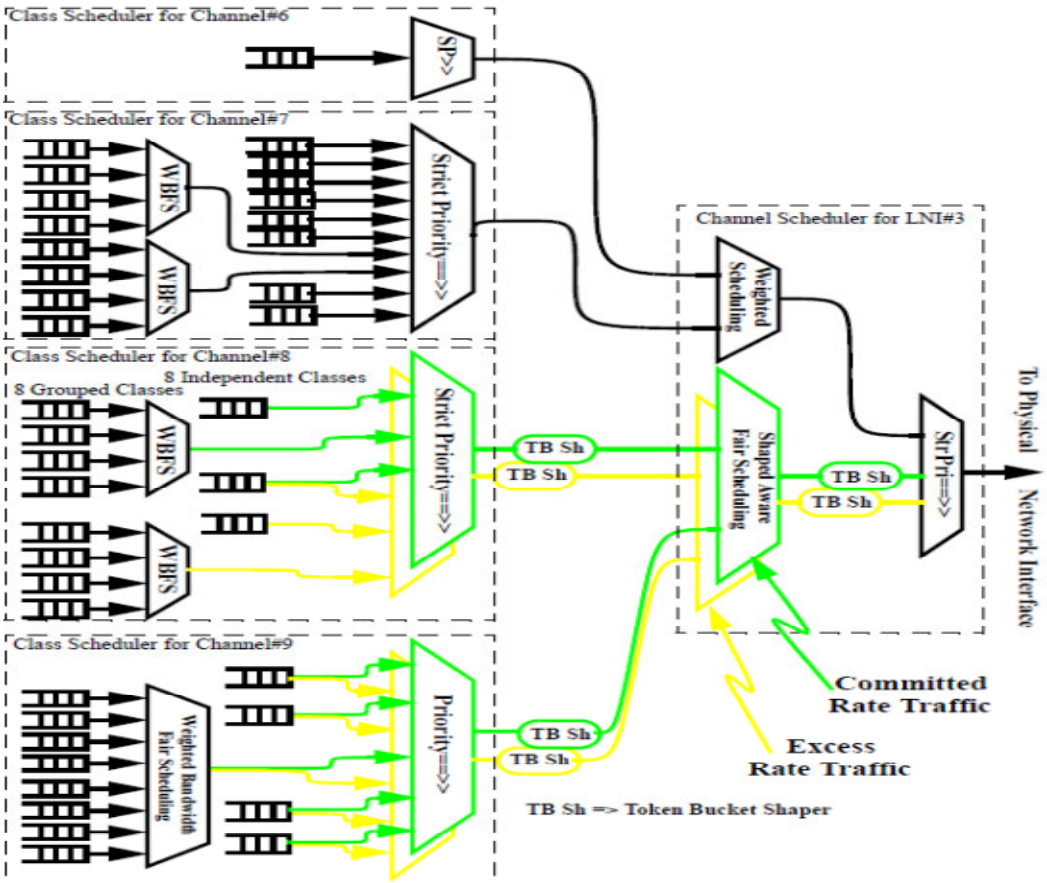
QorIQ QMan CEETM Implementation in USDPAA

1. CEETM Overview

For achieving higher throughput using Network Stack QoS, one of the solution is to push the QoS logic from software to hardware. CEETM(Customer-edge egress-traffic management) is DPAA enhancement first appearing in QorIQ T series processors in QMan, it provides hierarchical class based scheduling and traffic shaping. CEETM provides the features dual-rate shaping, weighted CQ scheduling, congestion management, this document introduces these features implementing in USDPAA.

Each instance of CEETM per FMAN supports dual-rate shaping(paired committed rate(CR) shaper and excess rate(ER) shaper), supports eight logical network interfaces(LNI), 32 channels available for allocation across the eight LNIs. Each channel can be configure to be unshaped or shaped, has eight independent class and eight grouped classes(one class group of eight or two class groups of four), supports weighted bandwidth fairness within grouped class groups. Each class has a dedicated class queue(CQ) with equivalent congestion management, is identified via a “logical frame queue identifier” to maintain semantic compatibility with non-CEETM queues.

A CEETM topology structure can be depicted as the following.



2. CEETM Shaping function

The QMan CEETM provides CR and ER shapers to shape the traffic with the given rate. Both shapers use the token credit rate and credit update reference period to determine the shaper's output rate. The token rate is specified in bytes with an 11 bit integer and a 13 bit fractional part, and can be configured via CEETM shaper configuration commands.

The following describes CEETM shaping related driver APIs design.

These APIs are used to convert shaper output rate in bps to token rate and vice versa, which can be applied to both LNI and channel shaping.

*int qman_ceetm_bps2tokenrate(u32 bps, struct qm_ceetm_rate *token_rate, int rounding);*

*int qman_ceetm_tokenrate2bps(const struct qm_ceetm_rate *token_rate, u32 *kbps, int rounding);*

LNI shapers are used to shaper or rate the aggregate of the class queue channels which have each been shaped.

Enables and disables shaping on the LNI.

*int qman_ceetm_lni_enable_shaper(struct qm_ceetm_lni *, int coupled, int oal);*

*int qman_ceetm_lni_disable_shaper(struct qm_ceetm_lni *);*

Set/get the shaper CR/ER token rate and token limit of the given LNI.

*int qman_ceetm_lni_set_commit_rate(struct qm_ceetm_lni *,const struct qm_ceetm_rate *token_rate,u16 token_limit);*

*int qman_ceetm_lni_get_commit_rate(struct qm_ceetm_lni *,struct qm_ceetm_rate *token_rate,u16 *token_limit);*

*int qman_ceetm_lni_set_excess_rate(struct qm_ceetm_lni *,const struct qm_ceetm_rate *token_rate, u16 token_limit);*

*int qman_ceetm_lni_get_excess_rate(struct qm_ceetm_lni *,struct qm_ceetm_rate,u16 *token_limit);*

Configure shapers of class queue channel, enable/disable shaping on the given channel.

*int qman_ceetm_channel_enable_shaper(struct qm_ceetm_channel *channel, int coupled);*

*int qman_ceetm_channel_disable_shaper(struct qm_ceetm_channel *channel);*

Set/get the channel shaper CR/ER token rate and token limit.

*int qman_ceetm_channel_set_commit_rate(struct qm_ceetm_channel *channel,*

*const struct qm_ceetm_rate *token_rate,u16 token_limit);*

*int qman_ceetm_channel_get_commit_rate(struct qm_ceetm_channel *channel,*

*struct qm_ceetm_rate *token_rate,u16 *token_limit);*

*int qman_ceetm_channel_set_excess_rate(struct qm_ceetm_channel *channel,*

*const struct qm_ceetm_rate *token_rate,u16 token_limit);*

*int qman_ceetm_channel_get_excess_rate(struct qm_ceetm_channel *channel,*

*struct qm_ceetm_rate *token_rate,u16 *token_limit);*

The following APIs allow the user to set the 8 independent CQs and 2 CQ groups within a shaped CEETM channel to be CR and/or ER eligible.

int qman_ceetm_channel_set_group_cr_eligiblility(struct qm_ceetm_channel

**channel, int group_b, int cre);*

int qman_ceetm_channel_set_group_er_eligiblility(struct qm_ceetm_channel

**channel, int group_b, int ere);*

*int qman_ceetm_channel_set_cq_cr_eligiblility(struct qm_ceetm_channel *channel,*

unsigned int idx, int cre);

*int qman_ceetm_channel_set_cq_er_eligiblility(struct qm_ceetm_channel *channel,*

unsigned int idx, int ere);

3. CEETM Class Congestion Management

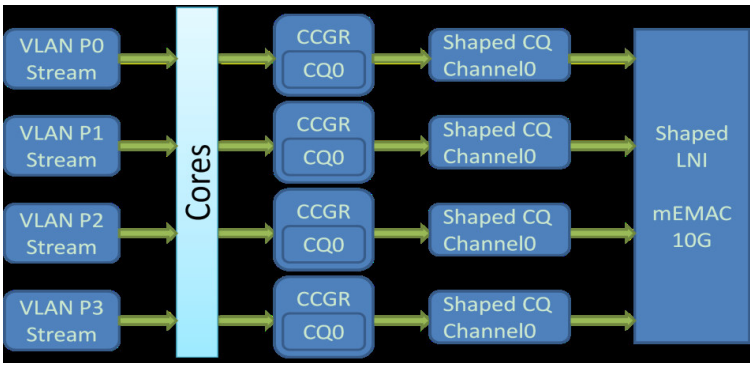
CEETM supports both Weighted Random Early Discard (WRED) and tail drop congestion management with its 512 Class Congestion Groups (CCGs). The CCG are divided into channels. Each channel contains 16 CCGs, and each CCG channel is tied one-to-one with a CQ channel. The CCG to be used for a particular CQ can be assigned to any of the 16 CCG within the channel.

Shaped CQ channel makes packets accumulate in CQ.

Once the quantity of packets in CQ reaches threshold of entering congestion set in CCGR of that CQ, the CCGR goes into congestion status and CSCN is triggered

Once the quantity of packets in CQ is less than threshold of exiting congestion, the CCGR exits congestion status and CSCN is triggered.

The following figure describes CCGR feature.



The CCG driver APIs design is described as the follow.

Claim a unused CCG, the congestion group is local to the given class queue channel, so only

class queues within the channel can be associated with that congestion group. The association of class queues to congestion groups occurs when the class queues are claimed.

*int qman_ceetm_ccg_claim(struct qm_ceetm_ccg **ccg,*

*struct qm_ceetm_channel *channel, unsigned int idx, void (*cscn)(struct qm_ceetm_ccg *,*

*void *cb_ctx, int congested), void *cb_ctx)*

qman_ceetm_ccg_release - Releases a previously claimed CCG.

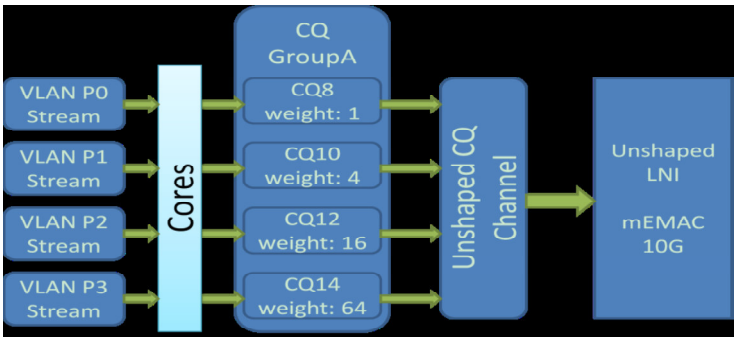
Configuring CCG APIs are used to specify attributes for a CCG. The 'we_mask' field controls which CCG attributes are to be updated, and the remainder specify the values for those attributes. A CCG counts either frames or the bytes within those frames, but not both ('mode'). A CCG can optionally cause enqueues to be rejected, due to tail-drop or WRED, or both (they are independent options, 'td_en' and 'wr_en_g,wr_en_y,wr_en_r'). Tail-drop can be level-triggered due to a single threshold ('td_thres') or edge-triggered due to a "congestion state", but not both ('td_mode').

/ Bits used in 'we_mask' to qman_ceetm_ccg_set(), controls which attributes of*
** the CCGR are to be updated. */*
#define QM_CCGR_WE_CDV 0x0000 / cdv */*
#define QM_CCGR_WE_MODE 0x0001 / mode (bytes/frames) */*
#define QM_CCGR_WE_TD_EN 0x0004 / congestion state tail-drop enable */*
#define QM_CCGR_WE_TD_MODE 0x4000 / tail-drop mode (state/threshold) */*
#define QM_CCGR_WE_TD_THRES 0x2000 / tail-drop threshold */*
#define QM_CCGR_WE_CS_THRES_IN 0x0002 / congestion state entry threshold */*
#define QM_CCGR_WE_CS_THRES_OUT 0x1000 / congestion state exit threshold */*
#define QM_CCGR_WE_CSCN_EN 0x0010 / congestion notification enable */*
#define QM_CCGR_WE_CSCN_TUPD 0x0008 / CSCN target update */*
#define QM_CCGR_WE_OAL 0x0800 / overhead accounting length */*
#define QM_CCGR_WE_WR_PARM_G 0x0400 / WRED parameters - green */*
#define QM_CCGR_WE_WR_PARM_Y 0x0200 / WRED parameters - yellow */*
#define QM_CCGR_WE_WR_PARM_R 0x0100 / WRED parameters - red */*
#define QM_CCGR_WE_WR_EN_G 0x0080 / WRED enable - green */*
#define QM_CCGR_WE_WR_EN_Y 0x0040 / WRED enable - yellow */*

#define QM_CCGR_WE_WR_EN_R 0x0020 /* WRED enable - red */
*int qman_ceetm_ccg_set(struct qm_ceetm_ccg *ccg, u16 we_mask,*
*const struct qm_ceetm_ccg_params *params)*
*int qman_ceetm_ccg_get(struct qm_ceetm_ccg *ccg,*
*struct qm_ceetm_ccg_params *params)*
Get the statistics provided by CEETM CCG counters
*int qman_ceetm_ccg_get_reject_statistics(struct qm_ceetm_ccg *ccg, u32 flags,*
*u64 *frame_count, u64 *byte_count)*
Set/get Congestion State Change Notification Target
*int qman_ceetm_cscn_dcp_set(struct qm_ceetm_ccg *ccg, u16 dcp_idx,*
u8 vcgid, unsigned int cscn_enabled,
*u16 we_mask, const struct qm_ceetm_ccg_params *params)*
*int qman_ceetm_cscn_dcp_get(struct qm_ceetm_ccg *ccg,*
*u16 dcp_idx, u8 *vcgid, unsigned int *cscn_enabled)*

4. CEETM Weighted CQ Scheduling

Each instance of CEETM supports 32 class queue channels for allocation across the 8 LNIs. Each channel contains a total of 16 class queues (CQ), with 8 independent classes and 8 grouped classes which can be configured as 1 group of 8 classes or 2 groups of 4 classes. The weighted bandwidth fairness scheduling applies within the grouped classes, and strict priority scheduling applies to 8 independent classes and 2 class groups.



In the above figure, the CQ GroupA in CQ channel contains 8 weighted CQ, four of which are enabled in this case Weighted CQs are assigned different weight.
The bigger weight for CQ, the less Latency for traffic through weighted CQ.

The followings APIs are designed to be invoked by the application program.
Claims a class queue within the channel group A.
*int qman_ceetm_cq_claim_A(struct qm_ceetm_cq **cq,*
*struct qm_ceetm_channel *channel, unsigned int idx,*
*struct qm_ceetm_ccg *ccg)*
Claims a class queue within the channel group B.
*int qman_ceetm_cq_claim_B(struct qm_ceetm_cq **cq,*
*struct qm_ceetm_channel *channel, unsigned int idx,*
*struct qm_ceetm_ccg *ccg)*
Grouped class queues have a default weight code of zero, which corresponds to a scheduler weighting of 1. This function can be used to modify a grouped class queue to another weight, valid values are from 0 to 255. (Use the helpers qman_ceetm_wbfs2ratio() and qman_ceetm_ratio2wbfs() to convert between these 'weight_code' values and the corresponding sharing weight.) As the weight code ranges from 0 to 255, the corresponding scheduling weight ranges from 1.00 to 248 in pseudo-exponential steps.

Configure/query the weight ofa grouped class queue.
*int qman_ceetm_set_queue_weight(struct qm_ceetm_cq *cq,*
*struct qm_ceetm_weight_code *weight_code)*
*int qman_ceetm_get_queue_weight(struct qm_ceetm_cq *cq,*
*struct qm_ceetm_weight_code *weight_code)*
Given a weight code ('wbfs'), an accurate fractional representation of the corresponding weight is given (in order to not lose any precision).
*int qman_ceetm_wbfs2ratio(struct qm_ceetm_weight_code *weight_code,*
*u32 *numerator, u32 *denominator)*
int qman_ceetm_ratio2wbfs(u32 numerator, u32 denominator,
*struct qm_ceetm_weight_code *weight_code, int rounding)*
Get the statistics provided by CEETM CQ counters.
*int qman_ceetm_cq_get_dequeue_statistics(struct qm_ceetm_cq *cq, u32 flags,*
*u64 *frame_count, u64 *byte_count)*

5. CEETM USDPAA demo Application and Test Result

CEETM USDPAA application is designed based on USDPAA PPAC/PPAM program architecture, the PPAC((Packet-Processing Application Core) component represents the common infrastructure to support such PPAMs(Packet-Processing Application Module). The PPAM portion implements this application specific logic, which receives the network packet, exchanges its source and destination address and send it out.
The egress flow will be forwarded to the specific class queue according to “TOS” field in IPv4 header. TOS is a 8-bit value, 4 msb in this case means CHANNEL id within a LNI while 4 lsb means CQ id (0-0xf) within a channel. This application claims CCGRs with congestion management of tail drop.

The following is CEETM initialization and configuration flow.

The CEETM application use the configuration is as the following.
<ceetm>
… …
<channel control="shaped" group="1" cr="250m" er="100m">
<groupA idx="2" op="both" />
<cq idx='0' op="both" />
<cq idx='1' op="both" />
<cq idx='2' op="both" />
<cq idx='3' op="both" />
<cq idx="8" weight=”1” />
<cq idx="9" weight=”2” />
<cq idx="10" weight="4” />
<cq idx="11" weight=”8” />
<cq idx="12" weight=”16” />
<cq idx="13" weight=”32” />
<cq idx="14" weight=”64” />
<cq idx="15" weight=”128” />
</channel>
</lni>
</ceetm>

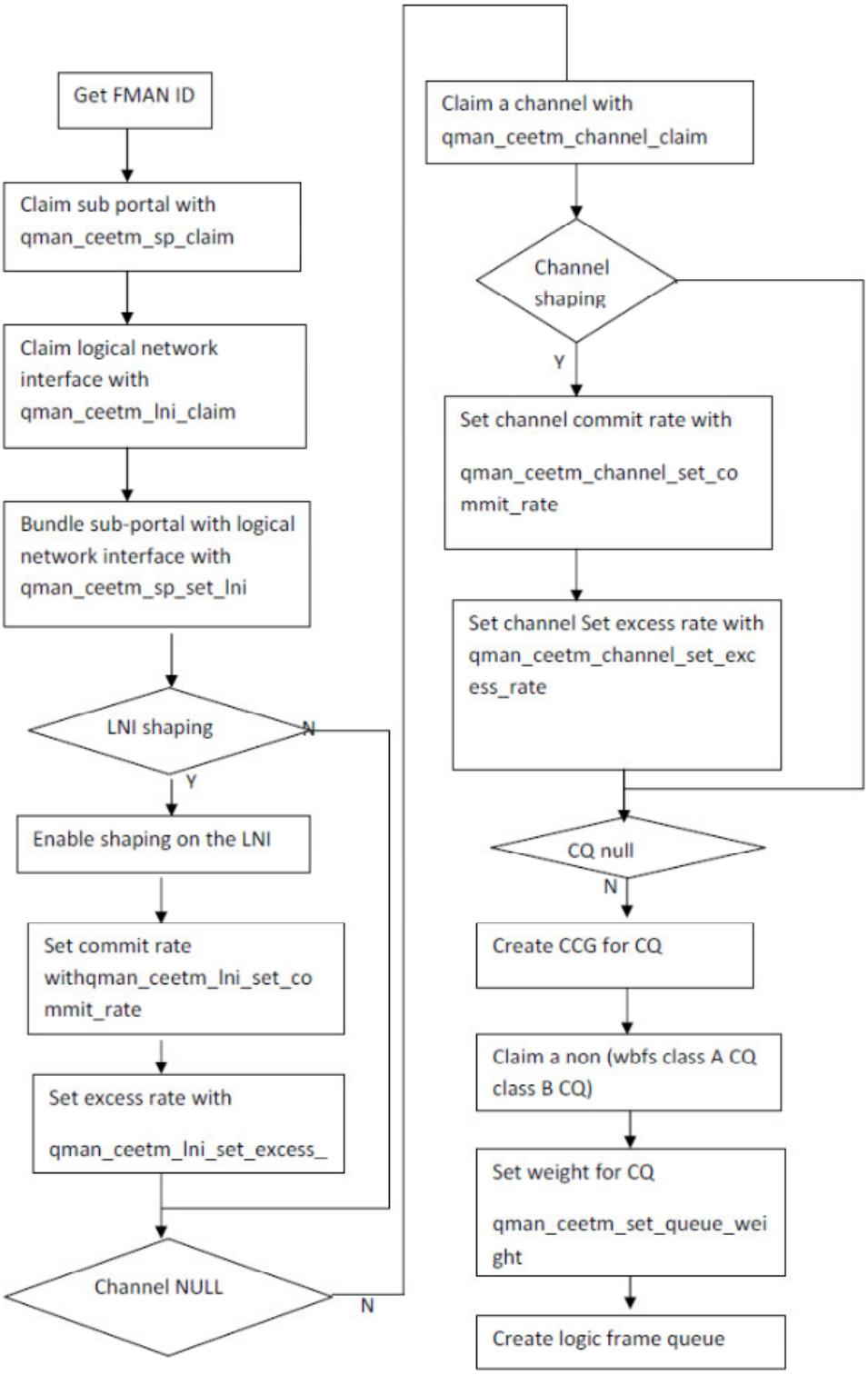
Generate two traffic flows with ToS as 0x3A and 0x3F separately with cq weight 4 and 128. Latency result is as the following.

Stream	Rx Port	Stream Index	Avg Latency(us)
StreamBlock 12	Port //3/7	262,145	5.42820519136187
StreamBlock 2	Port //3/7	262,144	1,766.78767242922

Two Stream blocks Tx and Rx detailed statistics information is as the following.

Tx Port	Rx Port	Stream Block	Stream Id	Stream Index	Tx Frame Count	Rx Frame Count	Tx Octet Count	Rx Octet Count	Rx Sig Frame Count	Avg Latency(us)
Port //3/7	Port //3/7	StreamBlock 2	262144	0	22544643	8706906	1442857152	0	8706905	1766.768
Port //3/7	Port //3/7	StreamBlock 12	262145	1	22544643	22544643	1442857152	0	22544643	5.428

CEETM USDPAA application demo is developed as the following.


Get FMAN ID
Claim sub portal with qman_ceetm_sp_claim
Claim logical network interface with qman_ceetm_lni_claim
Bundle sub-portal with logical network interface with qman_ceetm_sp_set_lni
LNI shaping
N
Y
Enable shaping on the LNI
Set commit rate withqman_ceetm_lni_set_commit_rate
Set excess rate with qman_ceetm_lni_set_excess_
Channel NULL
N
Claim a channel with qman_ceetm_channel_claim
Channel shaping
Y
Set channel commit rate with qman_ceetm_channel_set_commit_rate
Set channel Set excess rate with qman_ceetm_channel_set_excess_rate
CQ null
N
Create CCG for CQ
Claim a non (wbfs class A CQ class B CQ)
Set weight for CQ qman_ceetm_set_queue_weight
Create logic frame queue