QCN Pseudo Code

Version 2.3

Rong Pan
Definition - Variables

1. **IncomingFrame**: a packet frame which arrives at a congestion node or at its destination.
2. **IncomingFrame.flowid**: an incoming frame can be tagged with the field of its flow id.
3. **RL[*]**: a set of rate limiters.
4. **RL[i].state**: state of the rate limiter $i$: active or inactive.
5. **RL[i].flowid**: the flow id that is associated with the rate limiter $i$.
6. **RL[i].crate**: the current rate of the rate limiter $i$.
7. **RL[i].trate**: the target rate of the rate limiter $i$.
8. **RL[i].tx_bcoun**t: number of bytes left before increasing the stage of the byte counter.
9. **RL[i].si_count**: the stage of the byte counter that the rate limiter, $i$, is in.
10. **RL[i].timer**: the timer of the rate limiter.
11. **RL[i].timer_scount**: the stage of the timer that the rate limiter, $i$, is in.
12. **RL[i].qlen**: the queue length of the rate limiter queue.
13. **rlidx**: index of a rate limiter.
14. **FBFrame**: a feedback control frame which sends the congestion information, Fb, back to the traffic source; this packet frame can be sent either from any intermediate reflection point.
15. **FBFrame.SA**: the source MAC address of the feedback control frame.
16. **FBFrame.DA**: the destination MAC address of the feedback control frame.
17. **FBFrame.flowid**: the flow id of the feedback control frame.
18. **FBFrame.fb**: the congestion control information, Fb, of the feedback control frame.
19. **FBFrame.qoff**: the queue offset information carried in the feedback control frame, it equals Q_EQ-qlen.
20. **FBFrame.qdelta**: the queue delta information carried in the feedback control frame, it equals qlen-qlen_old
22. **qlen_old**: queue length (in pages) at last sample.
23. **Fb**: feedback value which indicates the level of congestion.
24. **qntz_Fb**: quantized negative Fb (-Fb) value.
25. **time_to_mark**: number of bytes left before the next sample will be taken.
Definition – Parameters

26. **Q_EQ:** the reference point of a queue. QCN aims to keep the queue occupancy at this reference level under congestion.

27. **W:** the control parameter in calculating the congestion level variable Fb.

28. **GD:** the control gain parameter which determines the level of rate decrease given a Fb < 0 signals.

29. **BC_LIMIT:** the parameter which determines the byte-counter time-out threshold.

30. **TIMER_PERIOD:** the parameter which determines the timer time-out threshold.

31. **R_AI:** the parameter which determines the rate increase amount in AI stage.

32. **R_HAI:** the parameter which determines the rate increase amount in HAI stage.

33. **FAST_RECOVERY_TH:** the threshold which determines when a RL will exit fast recovery (FR) stage, set to 5.

34. **MIN_RATE:** the minimum rate of a rate limiter, set to 10Mbps.

35. **MIN_DEC_FACTOR:** the minimum rate decrease factor, set to 0.5.

36. **C:** the speed of a link where a rate limiter is installed

37. **SWITCH_MAC_ADDRESS:** the congestion point MAC address which is used as SA in the feedback frame
QCN Reaction Point:

1. initialize()
2. {
3.     // * indicates all rate limiters
4.     RL[*].state = INACTIVE;
5.     RL[*].flowid = -1;
6.     RL[*].crate = C;
7.     RL[*].trate = C;
8.     RL[*].tx_bcount = BC_LIMIT;
9.     RL[*].si_count = 0;
10.    RL[*].timer_scount = 0;
11. }
12. foreach (FBFrame)
13. {
14.     // obtain the rate limiter index that is associated with a flowid
15.     // if no match, return the index of the next available rate limiter
16.     rldx = get_rate_limiter_index(FBFrame.flowid);
17. 
18.     if (RL[rldx].state = = INACTIVE) then
19.         if (FBFrame.fb != 0 & & FBFrame.qoff < 0) then
20.             // initialize new rate limiter
21.             RL[rldx].state = ACTIVE;
22.             RL[rldx].flowid = FBFrame.flowid;
23.             RL[rldx].crate = C;
24.             RL[rldx].trate = C;
25.             RL[*].tx_bcount = BC_LIMIT;
26.             RL[rldx].si_count = 0;
27.         else
28.             // ignore FBFrame
29.             return;
30.         endif
31.     endif
32. }
if (FBFrame.fb != 0) then

// use the current rate as the next target rate.
// in the first cycle of fast recovery.
// the Fb < 0 signal would not reset the target rate.
if (RL[rlidx].si_count != 0) then
    RL[rlidx].crate = RL[rlidx].crate;
    RL[rlidx].tx_bcount = BC_LIMIT;
endif

// set the stage counter
RL[rlidx].si_count = 0;
RL[rlidx].timer_scount = 0;

// update the current rate, multiplicative decrease
def_factor = (1 - GD * FBFrame.fb);
if (def_factor < MIN_DEC_FACTOR) then
def_factor = MIN_DEC_FACTOR;
endif
RL[rlidx].crate = RL[rlidx].crate * def_factor;
if (RL[rlidx].crate < MIN_RATE) then
    RL[rlidx].crate = MIN_RATE;
endif

// reset the timer
set_timer(rlidx, TIMER_PERIOD);
endif

} } self_increase(rlidx)

to_count = minimum(RL[rlidx].si_count, RL[rlidx].timer_scount);

// if in the active probing stages, increase the target rate
if (RL[rlidx].si_count > FAST_RECOVERY_TH ||
    RL[rlidx].timer_scount > FAST_RECOVERY_TH) then
    if (RL[rlidx].si_count > FAST_RECOVERY_TH &&
        RL[rlidx].timer_scount > FAST_RECOVERY_TH) then
        // hyperactive increase
        Ri = R_HAI * (to_count - FAST_RECOVERY_TH);
    else
        // active increase
        Ri = R_AI;
    endif
else
    // active increase
    Ri = 0;
endif
//at the end of the first cycle of recovery
if ((RL[rlidx].si_count == 1 || RL[rlidx].timer_scount == 1) &&
    RL[rlidx].brate > 10* RL[rlidx].crate) then
    RL[rlidx].brate = RL[rlidx].brate/8;
else
    RL[rlidx].brate = RL[rlidx].brate + Ri;
endif

RL[rlidx].crate = (RL[rlidx].brate + RL[rlidx].crate)/2;

//saturate rate at C
if (RL[rlidx].crate > C) then
    RL[rlidx].crate = C;
endif

foreach (Transmit Frame)
{
    //release the rate limiter when its rate has reached C
    //and its associated queue is empty
    if (RL[rlidx].crate == C && RL[rlidx].qlen == 0) then
        RL[rlidx].state = INACTIVE;
        RL[rlidx].flowid = -1;
        RL[rlidx].crate = C;
        RL[rlidx].brate = C;
        RL[rlidx].tx_bcount = BC_LIMIT;
        RL[rlidx].si_count = 0;
        RL[rlidx].timer = INACTIVE;
    else
        RL[rlidx].tx_bcount -= length(Transmit Frame);
    endif

    if (RL[rlidx].tx_bcount < 0) then
        RL[rlidx].si_count++;
        //if a negative FBframe has not been received after transmitting
        //BC_LIMIT bytes, trigger self_increase; margin of randomness 30%
        if (RL[rlidx].si_count < FAST_RECOVERY_TH) then
            expire_thresh = random_number_between(0.85,1.15)*BC_LIMIT;
        else
            expire_thresh = random_number_between(0.85,1.15)*BC_LIMIT/2;
        endif
    endif

    RL[rlidx].tx_bcount = expire_thresh;
    self_increase(rlidx);
endif

dendif
/* Timers */
timer_expired(rlidx)
{
    if (RL[rlidx].state == ACTIVE) then
        RL[rlidx].timer_scount++;
        self_increase(rlidx);
    //reset the timer
    //margin of randomness 30%
    if (RL[rlidx].timer_scount < FAST_RECOVERY_TH) then
        expire_period = random_number_between(0.85, 1.15)*TIMER_PERIOD;
    else
        expire_period = random_number_between(0.85, 1.15)*TIMER_PERIOD/2;
    endif
    set_timer(rlidx, expire_period);
    endif
}
QCN Congestion Point:

```
146. initialize()
147. {
148.    qlen = 0;
149.    qlen_old = 0;
150.    time_to_mark = Mark_Table(0);
151. }
152.
153. foreach (IncomingFrame)
154. {
155.     //calculate Fb value
156.     Fb = (Q_EQ - qlen) - w * (qlen - qlen_old);
157.     if (Fb < -Q_EQ * (2 * w + 1)) then
158.         Fb = -Q_EQ * (2 * w + 1);
159.     elseif (Fb > 0) then
160.         Fb = 0;
161.     endif
162.
163.     //the maximum value of -Fb determines the number of bits that Fb uses.
164.     //uniform quantization of -Fb, qntz_Fb, uses most significant bits of -Fb.
165.     //note that now qntz_Fb has positive values.
166.     qntz_Fb = -Fb(most significant bits);
167.
168.     //sampling probability is a function of Fb
169.     generate_fb_frame = 0;
170.
171.     time_to_mark -= length(IncomingFrame);
172.     if (time_to_mark < 0) then
173.         //generate a feedback frame if Fb is negative
174.         if (qntz_Fb > 0) then
175.             generate_fb_frame = 1;
176.         endif
177.         qlen_old = qlen;
178.     //Mark Table is described below. Margin of randomness 30%
179.     next_period = Mark_Table(qntz_Fb);
180.     time_to_mark = random_number_between(0.85,1.15)*next_period;
181.     endif
182.
183.     if (generate_fb_frame) then
184.         FBFrame.DA = IncomingFrame.SA;
185.         FBFrame.SA = SWITCH_MAC_ADDRESS;
186.         FBFrame.flowid = IncomingFrame.flowid;
187.         FBFrame.fb = qntz_Fb;
188.         FBFrame.qoff = Q_EQ - qlen;
189.         FBFrame.qdelta = qlen - qlen_old;
```
forward(FBFrame);

endif
}

//assuming 6 bits of quantization
Mark_Table(qntz_Fb) {

    switch (qntz_Fb/8){
        case 0: return 150KB;
        case 1: return 75KB;
        case 2: return 50KB;
        case 3: return 37.5KB;
        case 4: return 30KB;
        case 5: return 25KB;
        case 6: return 21.5KB;
        case 7: return 18.5KB;
    }
}