1 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_bcount**: 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_scound**: 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. **qlen**: current queue length (in pages), incremented upon packet arrivals and decremented upon packet departures.
20. **qlen_old**: queue length (in pages) at last sample.
21. **Fb**: feedback value which indicates the level of congestion.
22. **qntz_Fb**: quantized negative Fb (-Fb) value.
23. **time_to_mark**: number of bytes left before the next sample will be taken

---

1 EDCS-608482
Definition – Parameters

24. **Q_EQ:** the reference point of a queue. QCN aims to keep the queue occupancy at this reference level under congestion.
25. **W:** the control parameter in calculating the congestion level variable Fb.
26. **GD:** the control gain parameter which determines the level of rate decrease given a Fb < 0 signals.
27. **BC_Limit:** the parameter which determines the byte-counter time-out threshold.
28. **TIMER_PERIOD:** the parameter which determines the timer time-out threshold.
29. **R_AI:** the parameter which determines the rate increase amount in AI stage.
30. **R_HAI:** the parameter which determines the rate increase amount in HAI stage.
31. **FAST_RECOVERY_TH:** the threshold which determines when a RL will exit fast recovery (FR) stage, set to 5.
32. **MIN_RATE:** the minimum rate of a rate limiter, set to 10Mbps.
33. **MIN_DEC_FACTOR:** the minimum rate decrease factor, set to 0.5.
34. **C:** the speed of a link where a rate limiter is installed
35. **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_bcoun = BC_LIMIT;
9.     RL[*].si_count = 0;
10.    RL[*].timer_scount = 0;
11. }
12.
13. foreach (FBFrame)
14. {
15.     //obtain the rate limiter index that is associated with a flowid
16.     //if no match, return the index of the next available rate limiter
17.     rlidx = get_rate_limiter_index(FBFrame.flowid);
18.     if (RL[rlidx].state = = INACTIVE) then
19.         if (FBFrame.fb != 0) then
20.             //initialize new rate limiter
21.             RL[rlidx].state = ACTIVE;
22.             RL[rlidx].flowid = FBFrame.flowid;
23.             RL[rlidx].crate = C;
24.             RL[rlidx].trate = C;
25.             RL[*].tx_bcoun = BC_LIMIT;
26.             RL[rlidx].si_count = 0;
27.         else
28.             //ignore FBFrame
29.             return;
30.         endif
31.     endif
32. }
33.
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
    dec_factor = (1 - GD * FBFrame.fb);
    if (dec_factor < MIN_DEC_FACTOR) then
        dec_factor = MIN_DEC_FACTOR;
    endif
    RL[rlidx].crate = RL[rlidx].crate * dec_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
        Ri = 0;
    endif
}
//at the end of the first cycle of recovery
if ((RL[rlidx].si_count == 1 || RL[rlidx].timer_count == 1) &&
    RL[rlidx].rate > 10* RL[rlidx].crate) then
    RL[rlidx].rate = RL[rlidx].rate/8;
else
    RL[rlidx].rate = RL[rlidx].rate + Ri;
endif

RL[rlidx].crate = (RL[rlidx].rate + 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].rate = 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);
}
}
/* Timers */
timer_expired(rldx)
{
    if (RL[rldx].state == ACTIVE) then
        RL[rldx].timer_scount++;
        self_increase(rldx);
    //reset the timer
    //margin of randomness 30%
    if (RL[rldx].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(rldx, expire_period);

}
QCN Congestion Point:

initialize()
{
    qlen = 0;
    qlen_old = 0;
    time_to_mark = Mark_Table(0);
}

foreach (IncomingFrame)
{
    //calculate Fb value
    Fb = (Q_EQ - qlen) - W * (qlen - qlen_old);
    if (Fb < -Q_EQ * (2 * W + 1))
        Fb = -Q_EQ * (2 * W + 1);
    elseif (Fb > 0)
        Fb = 0;
    endif

    //the maximum value of −Fb determines the number of bits that Fb uses.
    //uniform quantization of −Fb, qntz_Fb, uses most significant bits of −Fb.
    //note that now qntz_Fb has positive values.
    qntz_Fb = −Fb(most significant bits);

    //sampling probability is a function of Fb
    generate_fb_frame = 0;

    time_to_mark -= length(IncomingFrame);
    if (time_to_mark < 0)
        //generate a feedback frame if Fb is negative
        if (Fb < 0)
            generate_fb_frame = 1;
        endif
        qlen_old = qlen;
    //Mark Table is described below. Margin of randomness 30%
    next_period = Mark_Table(qntz_Fb);
    time_to_mark = random_number_between(0.85,1.15)*next_period;
    endif

    if (generate_fb_frame)
        FBFram.DA = IncomingFrame.SA;
        FBFram.SA = SWITCH_MACADDRESS;
        FBFram.flowid = IncomingFrame.flowid;
        FBFram.fb = qntz_Fb;
        forward(FFrame);
    endif
190. } }  
191. 
192. 
193. //assuming 6 bits of quantization 
194. Mark_Table(qntz_Fb) { 
195. 
196. switch (qntz_Fb/8) { 
197. case 0: return 150KB; 
198. case 1: return 75KB; 
199. case 2: return 50KB; 
200. case 3: return 37.5KB; 
201. case 4: return 30KB; 
202. case 5: return 25KB; 
203. case 6: return 21.5KB; 
204. case 7: return 18.5KB; 
205. } 
206. }