[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 sent since the last negative feedback frame (Fb < 0).
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. **min_dec_factor:** the minimum decrease factor, a single step of decrease should not exceed this value.
20. **qlen**: current queue length (in pages). incremented upon packet arrivals and decremented upon packet departures.
21. **qlen_old**: queue length (in pages) at last sample.
22. **Fb**: feedback value which indicates the level of congestion.
23. **qntz_Fb**: quantized negative Fb (-Fb) value.

[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.

QCN Reaction Point:

```
initialize()
{
    //* indicates all rate limiters
    RL[*].state = INACTIVE;
    RL[*].flowid = −1;
    RL[*].crate = C;
    RL[*].trate = C;
    RL[*].tx_bcount = 0;
    RL[*].si_count = 0;
    RL[*].timer_count = 0;
}

foreach (FBFrame)
{
    //obtain the rate limiter index that is associated with a flowid
    //if no match, return the index of the next available rate limiter
    rlidx = get_rate_limiter_index(FBFrame.flowid);

    if (RL[rlidx].state = = INACTIVE) then
        if (FBFrame.fb != 0) then
            //initialize new rate limiter
            RL[rlidx].state = ACTIVE;
            RL[rlidx].flowid = FBFrame.flowid;
            RL[rlidx].crate = C;
            RL[rlidx].trate = C;
            RL[rlidx].si_count = 0;
        else
            //ignore FBFrame
            return;
        endif
    endif

```

```
        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].trate = RL[rlidx].crate;
                RL[rlidx].tx_bcount = 0;
            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 = B * (to_count − FAST_RECOVERY_TH);
            else
                //active increase
                Ri = A;
            endif
        else
                Ri = 0;
        endif
```

```
        //at the end of the first cycle of recovery
        if (RL[rlidx].si_count == 1 &&
           RL[rlidx].trate > 10* RL[rlidx].crate) then
                RL[rlidx].trate = RL[rlidx].trate/8;
        else
                RL[rlidx].trate = RL[rlidx].trate + Ri;

        RL[rlidx].crate = (RL[rlidx].trate + 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].rate = = C && RL[rlidx].qlen = = 0) then
            RL[rlidx].state = INACTIVE;
            RL[rlidx].flowid = −1;
            RL[rlidx].crate = C;
            RL[rlidx].trate = C;
            RL[rlidx].tx_bcount = 0;
            RL[rlidx].si_count = 0;
            RL[rlidx].timer = INACTIVE;
        else
             RL[rlidx].tx_bcount += length(Transmit Frame);
             //if a negative FBframe has not been received after transmitting
             //BC_LIMIT bytes, trigger self_increase
            if (RL[rlidx].si_bcount < FAST_RECOVERY_TH) then
                 expire_thresh = BC_LIMIT;
           else
                 expire_thresh = BC_LIMIT/2;
            endif
            if (RL[rlidx].tx_bcount > expire_thresh) then
                 RL[rlidx].si_count++;
                 RL[rlidx].tx_bcount = 0;
                 self_increase(rlidx);
            endif
        endif
    }
```

```
/*  Timers */
timer_expired(rlidx)
{
        if (RL[rlidx].state = = ACTIVE ) then
              RL[rlidx].timer_scount++;;
              self_increase(rlidx);

              //reset the  timer

              if (RL[rlidx].timer_scount < FAST_RECOVERY_TH) then
                     expire_period = TIMER_PERIOD;
              else
                     expire_period = TIMER_PERIOD /2;
              endif
              set _timer(rlidx, expire_period);

        endif
}
```

QCN Congestion Point:

```
initialize()
{
        qlen = 0;
        qlen_old = 0;
}

foreach (IncomingFrame)
{
        //calculate Fb value
        Fb = (Q_EQ − qlen) − W * (qlen − qlen_old);
        if (Fb < −Q_EQ * (2 * W + 1)) then
                Fb = −Q_EQ * (2 * W + 1);
        elseif (Fb > 0) then
                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;
        period_to_mark = Mark_Table(qntz_Fb); //Mark Table is described below.
        if (time_to_mark > period_to_mark) then
                //generate a feedback frame if Fb is negative
                if (Fb < 0) then
                        generate_fb_frame = 1;
                endif
                qlen_old = qlen;
                time_to_mark = 0;
        else
                time_to_mark += length(IncomingFrame);
        endif


        if (generate_fb_frame) then
                FBFrame.DA = IncomingFrame.SA;
                FBFrame.SA = SWITCH_MAC_ADDRESS;
                FBFrame.flowid = IncomingFrame.flowid;
                FBFrame.fb = qntz_Fb;
                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;
      }
}
```