Notes on Class-Based Queueing: Setting Parameters

Sally Floyd

Lawrence Berkeley National Laboratory
floyd@ee.lbl.gov

February 23, 1996

Abstract

These are informal notes about how to set the class parameters for Class-Based Queueing (CBQ) [FJ95]. Some of the guidelines, such as those for setting *maxidle* and *offtime*, apply to any of the link-sharing algorithms discussed in [FJ95]. Others are only necessary for Ancestor-Only link-sharing.

The current distribution of the CBQ code implements Ancestor-Only link-sharing; this should be updated to Top-Level Link-Sharing shortly. As explained in [FJ95], Ancestor-Only link-sharing is somewhat sensitive to the exact setting of the class parameters. Section 2 underscores that fact.

1 An overview of the guidelines

In CBQ, each class has variables *idle* and *avgidle*, parameter *maxidle* used in computing the limit status for the class, and parameter *offtime* used in determining how long to restrict throughput for overlimit classes.

Definition: idle. The variable *idle* is the difference between the desired time and the measured actual time between the most recent packet transmissions for the last two packets sent from this class. When the connection is sending perfectly at its alloted rate p, then *idle* is zero. When the connection is sending more that its allocated bandwidth, then *idle* is negative.

Definition: avgidle. The variable *avgidle* is the average of *idle*, and is computed using a exponential weighted moving average (EWMA). When *avgidle* is zero or lower, then the class is overlimit (the class has been exceeding its allocated bandwidth in a recent short time interval).

Definition: maxidle. The parameter *maxidle* gives an upper bound for *avgidle*. Thus *maxidle* limits the `credit' given to a class that has recently been under its allocation.

Definition: offtime. The parameter *offtime* gives the time interval that a overlimit class must wait before sending another packet. This parameter determines the steady-state burst size for a class when the class is running over its limit. This parameter is called *extradelay* in the ns simulator [MF95].

Definition: minidle. The cbq code in the ns simulator also has a parameter *minidle* that gives a (negative) lower bound for *avgidle*. Thus, a negative *minidle* lets the gateway `remember' that a class has recently used more than its allocated bandwidth. In the distributed code, *avgidle* has a lower bound of zero.

There are three types of classes: *leaf* classes (such as a video class) that have directly-assigned connections; *nonleaf* classes used for link-sharing (such as a class for a particular agency); and the *root* class that represents the entire output link. These types of classes are treated somewhat differently.

Leaf classes should not be penalized for borrowing unused bandwidth. When leaf class is overlimit and unable to borrow, it is simply restricted to its assigned bandwidth, regardless of how much bandwidth it borrowed previously. Thus if *avgidle* becomes nonnegative, it is reset to a lower bound of zero.

Summary of General Guidelines:

• **Maxidle:** Set *maxidle* to control the burstiness allowed to a class.

As Section 3 shows, to permit a maximum burst of n back-to-back packets, set

$$maxidle \leftarrow t(1/p - 1)\frac{(1 - g^n)}{g^n},$$

for t, p, g, and n as defined in that section.

The effect of maxidle is illustrated in the simulator *ns* with the following simulations: “ns test-suite-cbq.tcl cbqMax1” and “ns test-suite-cbq.tcl cbqMax2”.

• **Offtime:** For leaf classes, set *offtime* to regulate the steady-state burst size.

As Section 5 shows, a simple guideline is to set

$$offtime \leftarrow t(1/p - 1),$$

for t and p as defined in Section 3. This is the target waiting time to maintain a steady-state burst size of only one packet.

2 Guidelines for Ancestor-Only Link-Sharing

Additional guidelines are needed for Ancestor-Only Link-Sharing to prevent undesired borrowing, or equivalently, to encourage borrowing from lower levels before borrowing from higher levels.

Additional Guidelines for Ancestor-Only Link-Sharing:

• In Ancestor-Only link-sharing, assign a throughput to non-leaf classes that is slightly smaller than the sum of the assignments to the child classes. In the distributed code, this is done with the *nsecPerByte* class parameter. In the simulator ns, this is done with the "allotment" class parameter. This helps to prevent higher-priority classes from being able to borrow when lower-priority classes can send without borrowing.

In particular, for the root class set the allocated link-sharing bandwidth to less than 100% of the actual link bandwidth (e.g., 98% might do). This matters in particular for Ancestor-Only link-sharing; this guideline is not needed for Formal link-sharing.

• *Offtime* could be set larger for higher-level classes than for lower-level classes in the link-sharing structure.

Ideally, in Ancestor-Only Link-Sharing nonleaf classes SHOULD be penalized for exceeding their assigned bandwidth. This could be done with a (negative) *minidle* parameter, or by using a larger *offtime* parameter for non-leaf classes. Recall that for classes that are overlimit, "*offtime*" is added to "undertime", to define the time for which that class is not allowed to send a packet without borrowing. Thus, when a nonleaf class goes overlimit, *offtime* defines the duration for which that class does not permit borrowing. A higher value of *offtime* for non-leaf classes gives room for lower-priority classes that might be able to send without borrowing.

Assume that non-leaf class A and its parent non-leaf class B go overlimit at the same time. Then with Ancestor-Only link-sharing, "*offtime*" should be set for the two classes so that class A allows borrowing again before class B does. Assume that for both classes, *avgidle* has just become zero. Class A must wait at least $offtime_A$ seconds before allowing borrowing again, and class B must wait at least $offtime_B$ seconds. In this case, $offtime_B$ should be larger than $offtime_A$.

In my simulations I set *offtime* to the same value for all classes, and instead use a parameter *minidle* for non-leaf classes. By allowing *avgidle* to occasionally stay nonnegative, a negative value for *minidle* lets the averaging "remember" when the class has used more than its allocated bandwidth.

For non-leaf classes, this limits excessive borrowing from the class. Note that this only matters for Ancestor-Only link-sharing, where it is important to restrict classes from borrowing bandwidth when there are other classes that could send packets without borrowing bandwidth.

For higher-priority leaf classes with packet-by-packet round-robin, using either *minidle* or *extradelay* is necessary to limit leaf classes to their allocated bandwidth. This is shown in the simulator with "ns test-suite-cbq.tcl cbqMin1", "ns test-suite-cbq.tcl cbqMin2", and "ns test-suite-cbq.tcl cbqMin3".

3 Maxidle

Maxidle sets the upper bound for *avgidle*. We assume that maxidle is set when a class is created. *Maxidle* determines the maximum size burst allowed for a class that has sent no packets in the recent time interval.

Definitions: t, g, p. Let t be the transmission time for the most recent packet sent from this class. Let p be the fraction of the link bandwidth allocated to a particular class. Then for back-to-back packets from a class, the `target' interpacket time (the time between transmitting the two packets) is t/p, the actual interpacket time is t, and *idle* is $t\,(1 - 1/p)$. The formula for computing *avgidle* is

$$avgidle \leftarrow g\ avgidle + (1 - g)\ idle,$$

for g given by $15/16$, or maybe by $31/32$. The weight g determines the time constant of the averager. □

Assume that $avgidle$ initially has the value $maxidle$. Then after n back-to-back packets, *avgidle* is

$$g^n maxidle + \sum_{i=0}^{n-1} g^i\,(1 - g)\,idle$$

$$= g^n\ maxidle + idle\,(1 - g^n).$$

This derivation uses the fact that

$$\sum_{i=0}^{m} g^i = \frac{1 - g^{m+1}}{1 - g}.$$

If $avgidle$ reaches 0 after n consecutive packets, and $avgidle$ had the value $maxidle$ at the beginning of the burst, then the maximum size burst for that class is n packets. In order to allow a maximum size burst of n packets, $maxidle$ should be set to

$$maxidle = t_1(1/p - 1)\frac{(1 - g^n)}{g^n},$$

where t_1 is a "typical" packet transmission time, reflecting some "typical" packet size in bytes.

3.1 Maxidle with arbitrary packet sizes

Of course, because there might not be a predictable, "typical" packet size. Assume that the actual packet transmission times are $a\,t_1$, for some $a > 0$. Then what is the maximum number of back-to-back packets that could be sent, if *avgidle* is initially at the value for *maxidle* given by the equation above?

Idle will be $a\,t_1\,(1-1/p)$, and after $b\,n$ back-to-back packets,

$$\begin{aligned} avgidle &= g^{b\,n}\,maxidle + idle\,(1-g^{b\,n}) \\ &= g^{b\,n-n}t_1(1/p-1)(1-g^n) \\ &\quad +a\,t_1\,(1-1/p)\,(1-g^{b\,n}). \end{aligned}$$

We would like to know the value of b when *avgidle* first becomes zero.

Solving we get

$$g^{n(b-1)}t_1(1/p-1)(1-g^n)-a\,t_1\,(1/p-1)\,(1-g^{b\,n}) = 0,$$

$$\begin{aligned} g^{n(b-1)}(1-g^n) - a\,(1-g^{b\,n}) &= 0, \\ g^{n(b-1)}(1-g^n) + a\,g^{n(b-1)}\,g^n &= a, \\ g^{n(b-1)}(1+(a-1)\,g^n) &= a, \\ g^{n(b-1)} &= \frac{a}{1+(a-1)\,g^n}, \end{aligned}$$

and

$$b = \frac{\log \frac{a}{1+(a-1)\,g^n}}{n \log g} + 1.$$

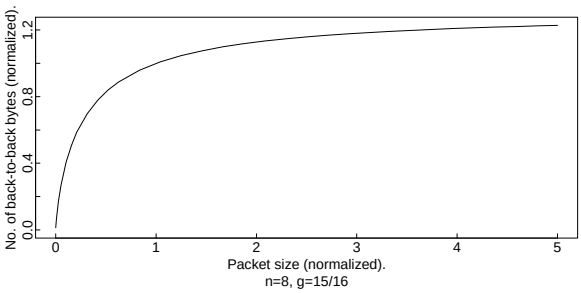


Figure 1: Number of back-to-back bytes allowed by *maxidle* given a range of packet sizes. Fraction of allocated throughput, for $g = 15/16$.

Assume that the initial t_1 transmission time was based on s-byte packets. Now, instead of sending $n\,s$ back-to-back bytes, with packets of $a\,s$ bytes we get to send $b\,n\,a\,s$ back-to-back bytes. Figure 1 shows $a\,b$ plotted as a function of a, for $n = 8$ and $g = 15/16$. Thus, *maxidle* is fairly effective in controlling the maximum number of back-to-back *bytes* even for a range of packet sizes.

4 Calculating undertime

Definition: undertime, now, overlimit. The scheduler checks the class variable *undertime* to see if a class can send packets without borrowing. If $avgidle$ is positive after a packet has been sent, then $undertime$ should be set to zero (or to something else less than the current time now). A class is not allowed to send a packet when $undertime > now$ and the class is unable to borrow.

For a regulated leaf class, which is not penalized for exceeding its allocation as a result of borrowing, then when $avgidle$ is at most zero, $undertime$ is set by adding *offtime* to the current time; this is regardless of the exact value of *avgidle*. The minimum value for *offtime* is the target waiting time *ptime*, for

$$ptime \leftarrow t(1/p-1).$$

5 Setting offtime

5.1 Setting offtime for leaf classes

Offtime can be used to regulate the steady-state burst size, and this steady-state burst size might be less than the maximum burst size allowed by *maxidle*. One possibility is to have a steady-state burst size of one packet; in this case *offtime* is simply set to the target waiting time *ptime*.

However, for efficiency or other reasons, it might be desirable to have a larger steady-state burst size than one packet. Assume that in steady state, with a class going on and off of the delay queue (that is, a class with plenty of demand that is being restricted to its link-sharing bandwidth), we want a steady-state burst of n packets. (This refers to a steady-state where the class sends a burst of n packets, goes on the delay queue, sends another burst of n packets, and so on.) This steady-state burst size n could be less than the size of the maximum burst discussed in Section 3. Let $avgidle_n$ be the value for $avgidle$ that allows a burst of size n before $avgidle$ reaches 0. Then

$$avgidle_n = t(1/p-1)\frac{(1-g^n)}{g^n}.$$

Assume that a class is put on the delay queue when *avgidle* becomes at most zero. Then we want to set *offtime* so that after *offtime* seconds, then if a packet is sent, the new value for *avgidle* will be $avgidle_{n-1}$, so exactly $n-1$ more consecutive packets can be sent until *avgidle* reaches zero again. This is true if

$$(1-g)\,idle = avgidle_{n-1},$$

for

$$idle = offtime + t - t/p.$$

This gives

$$offtime = \frac{1}{1-g} avgidle_{n-1} + t\,(1/p - 1)$$

$$= \frac{1}{1-g}\, t\,(1/p-1)\frac{(1-g^{n-1})}{g^{n-1}} + t\,(1/p-1).$$

For $g = 15/16$, $t = 0.01$, and $n = 8$, this is

$$offtime = 0.1014(1/p - 1).$$

For $g = 31/32$,

$$offtime = 0.0896(1/p - 1).$$

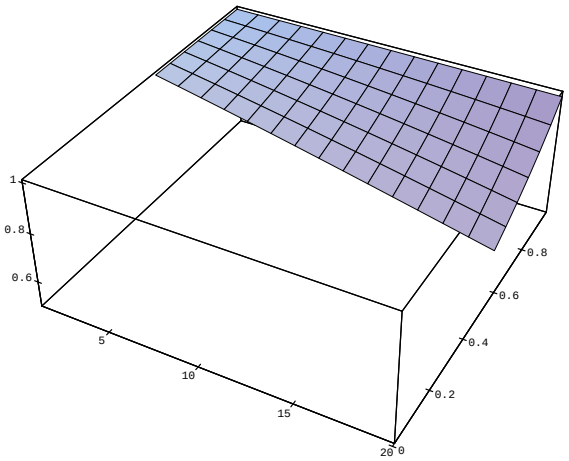
(This concurs with the findings later in this section that for a class with a steady-state burst size of 8, the throughput is higher with $g = 31/32$ than with $g = 15/16$.) As g increases, then *offtime* approaches closer to $8t(1/p - 1)$, the value needed for the class to achieve 100% of its throughput allocation.

What is a class's actual throughput, in this case? The class transmits n packets in $n\,t$ seconds, and then waits for *offtime* seconds. Thus the actual throughput, as a fraction of the maximum bandwidth of the link, is

$$\frac{n\,t}{n\,t + offtime}$$

$$= \frac{n}{n + \frac{1}{1-g}(1/p-1)\frac{(1-g^{n-1})}{g^{n-1}} + 1/p - 1}.$$

A connection that sends bursts of n packets in this manner will get slightly less that the specified fraction p of the bandwidth, for $n > 1$. Figure 2 shows the fraction f of its allocated throughput achieved by a delayed class, for $g = 15/16$.. The x-axis shows the steady-state burst size n and the y-axis shows the allocated throughput for the class. The z-axis shows the fraction of allocated throughput achieved by the delayed class. For this figure, we assume that $t = 0.01$ seconds, but the results are essentially the same for t as small as 0.01 ms. Figure 3 shows the same results for $g = 31/32$. This data argues for a small steady-state burst size, particularly for classes with small allocations. In our simulations, we use a steady-state burst size of $n = 8$ packets.



(What is the intuition behind this behavior? With a steady-state consisting of a burst of n packets followed by a delay, the computed $avgidle$ oscillates above and below the true steady-state average for the variable *idle*. If $avgidle$ oscillates, and the class is delayed when $avgidle$ reaches zero, then the class is delayed when the true average for idle is greater than zero, and the true throughput is less than the allocated throughput. The greater the amplitute of the oscillations in $avgidle$, the greater the loss of throughput. The "worst-case"