Decoupling Bandwidth and Delay Properties in Class Based Queuing

Fulvio Risso
Dipartimento di Automatica e Informatica – Politecnico di Torino
Corso Duca degli Abruzzi, 24 – 10129 Torino, Italy
risso@polito.it

Abstract— *This paper presents the Decoupled Class Based Scheduling, a CBQ-derived scheduling algorithm. Main advantages of D-CBQ are a new set of rules for distributing excess bandwidth and the ability to guarantee bandwidth and delay in a separate way, whence the name "decoupled"; moreover D-CBQ guarantees better delay bounds and more precise bandwidth assignment. This paper presents D-CBQ main points and discusses the choices for the implementation of the algorithm.*

1 Introduction

Historically, networks carried only predetermined types of traffic. For instance, the telephone network was able to transport voice, the Internet was able to transport data, and so on. Nowadays, networks want to integrate different kinds of traffic on the same physical infrastructure. For doing that, a modern network must be able to guarantee at least a certain amount of bandwidth and a maximum delay bound to each session. These parameters are usually independent each one from the others; for instance a session that requires assured bandwidth (for example a video stream) might not require strict delays. Networks usually guarantee differentiation among traffic by means of properly configured schedulers that are able to forward traffic in different ways according to the service required.

Priority Queuing (PQ) is the most effective scheduler from the "low delay service" standpoint, because it is able to forward prioritized traffic as soon as possible. Several other schedulers, particularly non-work conserving ones, are also excellent choices for delay requirements because they are able to forward traffic at predetermined intervals. Traffic is "paced"; therefore they are able to guarantee a maximum end-to-end delay bound to each session. However non-work conserving schemas are not widely accepted because they cannot exploit network resources at best (the output link can be idle even if there are packets in queue); moreover the guaranteed delay is usually higher than PQ schemas.

Fair schedulers [4] such as Weighted Fair Queuing [5] (WFQ) or Weighted Round Robin (WRR) are excellent choices for the "guaranteed bandwidth service" because of their ability to guarantee a predetermined amount of bandwidth to each session. Since WFQ is able to guarantee also a maximum delay bound ([6],[7]) to leaky bucket constrained sessions, it seems to be an excellent choice to satisfy bandwidth and delay requirements. However the delay experimented by a session in a WFQ scheduler could be larger than the PQ one; most important, delay is controlled by changing the bandwidth assigned to each session [11]. Since delay and bandwidth cannot be modified independently, the network may not be able to prevent a low-delay session from sending a large amount of data as well. This is a non-negligible issue in a modern network because a malicious user can exploit this point to overload the network; hence the network has to control (policing) explicitly the amount of traffic of each user. For this reason, the ability to *decouple* bandwidth and delay (i.e. granting bandwidth and delay bounds independently) is becoming a major point.

Besides guaranteeing bandwidth and delay, *hierarchical link-sharing* is getting increased importance. This requirement can be thought as a generalization of the "assured bandwidth" service because it guarantees that leaf classes receive their assigned bandwidth (for example class Data1 in Figure 1 must be able to get at least 40% of the total bandwidth), in the same way as fair schedulers do. Moreover, hierarchical link-sharing is used to impose specific rules for the distribution of bandwidth among different agencies sharing the same physical link. For instance, it guarantees the Data1 class to be able to use up to 60% of the link bandwidth (i.e. all the bandwidth granted to Agency A) when the Real-Time1 class is idle. Hierarchical schemas allow link-sharing to be potentially repeated inside each class of the hierarchy, moreover granting each class a minimum amount of bandwidth.

Previous schedulers are not able to meet delay, bandwidth and hierarchical-link sharing requirements at the same time. For instance, PQ is not able to guarantee a minimum share to low priority traffic. On the other hand, WFQ guarantees bandwidth and delay, but they are not decoupled. Neither PQ nor WFQ have hierarchical link-sharing capabilities.

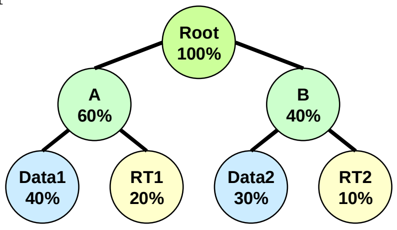


Figure 1. Hierarchical Link-Sharing.

Class Based Queuing [1] (CBQ), Hierarchical Packet Fair Queuing [2] (H-PFQ) and Hierarchical Fair Service Curve [9] (H-FSC) are able to satisfy previous requirements. All of them have to solve the theoretical problem of the incompatibility between link-sharing and delay goals, since there will be some time intervals in which satisfying one leads the other to be broken. This problem is addressed in different ways: H-PFQ privileges link-sharing goals, while CBQ and H-FSC privileges delay objectives. Poor delay capabilities are enough for H-PFQ to survive only in the academic literature.

CBQ is based on several mechanisms that basically merge PQ and fair capabilities to provide different kinds of service to data traffic. While the internals are quite complex, CBQ is easy to understand from the end-user point of view. Network managers need to define the link-sharing hierarchy and assign the proper amount of bandwidth and the desired priority to each class. Classes are served in decreasing order of priority, therefore high priority classes experiment a smaller delay.

H-FSC relies on a strong mathematical foundation (the service curve approach) but its behavior is difficult to predict (*which class will get the bandwidth next?*) because classes are selected using service curves[1] instead of static priorities. Configuration is less intuitive as well: network managers have to specify the link-sharing hierarchy and the service curve of each class instead of the couple bandwidth-priority of CBQ.

Thanks to its intuitiveness, CBQ is considered the most appealing advanced scheduler available today. However, an in-depth analysis of CBQ showed several problems, most noticeably that link-sharing and delay are not well decoupled (high priority sessions may also get more bandwidth) and that the rules used to distribute excess bandwidth are questionable.

Previous schedulers do not address a fourth requirement. Bandwidth guarantees can be divided in two objectives, the minimum bandwidth (a voice session can be willing to pay a large amount of money for that) and the preferred bandwidth (a voice session could be willing to pay less money to have CD-audio quality). This bandwidth differentiation is addressed only by D-GPS [13]; however the author believes this feature will add complexity into the algorithm without a valuable interest from end-users.

This paper presents an enhanced version of CBQ, called Decoupled-CBQ (D-CBQ), whose main points are the improvement of the rules used to distribute bandwidth according to the link-sharing structure and the decoupling of bandwidth and delay. D-CBQ is also able to guarantee tighter delay bounds and more precise bandwidth guarantees. This paper is structured as follows: Sections II and III present an in-depth analysis of CBQ and its leaky points. Section IV shows the main topics of D-CBQ; Section V shows the comparison between CBQ and D-CBQ. Finally, Section VI gives some conclusive remarks and presents the future directions we are interested in.

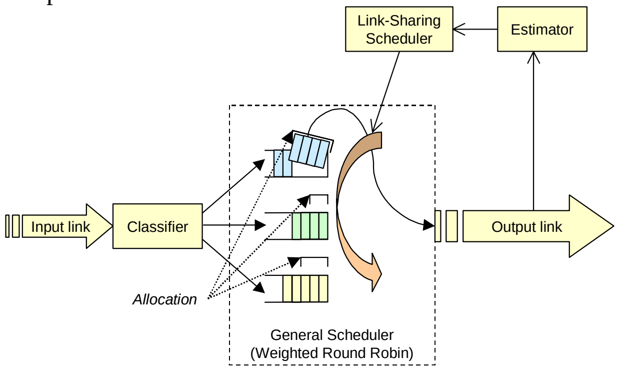


Figure 2. CBQ building blocks.

2 Class Based Queuing: an overview

CBQ architecture (Figure 2) is based on a generic "fair" scheduler controlled by a generic link-sharing scheduler. Incoming traffic is inserted (**classifier**) into the appropriate queue according to a set of filtering rules. **General scheduler** (usually WRR) extracts packets from queues and it guarantees each class to receive at least its nominal bandwidth. The **estimator** measures the inter-packet departure time for each class and checks whether the class is exceeding its allocated rate (*overlimit* class). The **link-sharing scheduler** cooperates with this "feedback block" and distributes the excess bandwidth according to the link-sharing structure. Basically, the link-sharing scheduler keeps control and suspends (for a specific amount of time) classes that exceed their allocated rate. Suspension time is calculated in such a way to force the class being consistent with its allocated bandwidth. Generally speaking, it appears like the suspended class is no longer active so that the WRR does not give any service to it until the suspension ends.

Link-sharing scheduler reconciles delay with link-

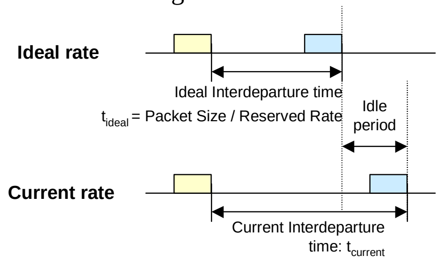
[1] An H-FSC service curve consists in two segments: the first one sets the delay properties of the session and the second one sets the long-term throughput. Each curve is made up of three parameters: the slope of the first segment (bps), the x-projection of the intersection point of the two segments (ms), the slope of the second segment (bps).

sharing capabilities by allowing a "Priority Queuing"-like service without starvation for lower priority classes.

2.1 General scheduler

General scheduler is usually made up of a set of cascading WRRs with different priorities. High priority classes are served using the first WRR until they are backlogged. A cascading WRR (strictly respecting priority) can start servicing its classes either when higher priority classes do not have any more packets in queue or none of these classes are allowed to transmit (because of the link-sharing scheduler).

In WRR each class can send a certain number of bytes (*allocation*) of data in each round. This number depends on the bandwidth allocated to the class and on how many bytes that class sent previously. The allocation is increased by a certain value (*allotment*, specific to each class) at the end of each round. Allotment is based on the bandwidth guaranteed to the class; therefore classes with large bandwidth can send more data each round. Allocation is decreased by the amount of bytes transmitted each time a class sends a packet; a class is allowed to transmit when its allocation is positive. Allocation can assume negative values (packets are non-divisible entities) but it cannot assume values larger than the class allotment.



Ideal rate
Ideal Interdeparture time
t_ideal = Packet Size / Reserved Rate
Idle period
Current rate
Current Interdeparture time: t_current

Figure 3. `idle` variable computation.

2.2 Link-sharing scheduler

CBQ uses several parameters in order to control the output pattern. `Idle` (Figure 3) is computed by difference from the current inter-departure time ($IDT_{current}$) and the ideal one (IDT_{ideal}):

(1) $$idle = IDT_{current} - IDT_{ideal} = IDT_{current} - \frac{p}{r \cdot f}$$

where `p` is the size of the packet being transmitted, `r` is the output link bandwidth and `f` is the faction of the link bandwidth allocated to the selected class.

`Idle` takes into account whether the class is sending more (`idle<0`) or less (`idle>0`) than its allocated rate. Nevertheless, the most important variable is `avgidle` that keeps track of the actual rate of the traffic. It is calculated by means of an EWMA function (of weight `w`):

(2) $$avgidle_{n+1} = (1-w) \cdot avgidle_n + w \cdot idle$$

Each class is suspended for a certain amount of time (`extradelay`) as soon as `avgidle` becomes negative; this forces the class rate to be compliant with the allocated bandwidth. `Extradelay` is calculated in order to allow the class sustaining a steady state burst on `m` packets after the suspension; therefore the suspension time must be:

$$avgidle_1 = (1-w) \cdot avgidle_0 + w \cdot idle = 0 + w \cdot \left[extradelay + \frac{p}{r} \cdot \left(1 - \frac{1}{f}\right)\right]$$

$$avgidle_2 = (1-w) \cdot avgidle_1 + w \cdot \frac{p}{r} \cdot \left(1 - \frac{1}{f}\right)$$

...

$$avgidle_m = (1-w) \cdot avgidle_{m-1} + w \cdot \frac{p}{r} \cdot \left(1 - \frac{1}{f}\right) = 0$$

(3) $$\Rightarrow extradelay = \frac{p}{r} \cdot \left(\frac{1}{f} - 1\right) \cdot \left(1 + \frac{1-(1-w)^{m-1}}{w \cdot (1-w)^{m-1}}\right)$$

A bigger `extradelay` makes the run-time allowed burst larger because that class can send more back-to-back packets before being suspended. This burst should be kept small at run time to permit a good level of multiplexing among the sessions that are using the same link. A suspended class will finally send a packet at time:

(4) $$undertime = last\_time + extradelay$$

where `last_time` represents the exit time of previous packet (belonging to that class) from the scheduler.

Finally, variables `minidle` and `maxidle` define a minimum and maximum bound to `avgidle`. The former avoids punishments for the excess service a class got in the past when other sessions were idle. `Minidle` was presented for the first time in [8] and it was set to zero. `Maxidle` sets a maximum value for `avgidle` to avoid that a previously idle class could send unpunished for too much time. `Maxidle` can be derived in a similar way to `extradelay` by taking into account that `maxidle` is reached *after* the first packet of a burst is being scheduled:

$$avgidle_1 = (1-w) \cdot avgidle_0 + w \cdot offtime \geq maxidle$$

$$avgidle_2 = (1-w) \cdot maxidle + w \cdot \frac{p}{r} \cdot \left(1 - \frac{1}{f}\right)$$

...

$$avgidle_M = (1-w) \cdot avgidle_{M-1} + w \cdot \frac{p}{r} \cdot \left(1 - \frac{1}{f}\right) = 0$$

(5) $$\Rightarrow maxidle = \frac{p}{r} \cdot \left(\frac{1}{f} - 1\right) \cdot \left(\frac{1}{(1-w)^{M-1}} - 1\right)$$

This Equation is different from the one derived in [8] because that Equation assumed `avgidle = maxidle` when first packet is scheduled.

Network managers usually do not set `maxidle` and `extradelay` because of their small degree of intuitiveness. These parameters are substituted by `maxburst` and `minburst`, which are respectively the maximum number of back-to-back packets that the class is allowed to send after a long idle period and the maximum number of back-to-back packets that a class is allowed to send during steady state. `Maxidle` and `extradelay` can be easily derived from `maxburst` and `minburst`: `minburst` is variable `m` in Equation (3), while `maxburst` is variable `M` in (5).

2.3 Distributing the excess bandwidth

In CBQ, a class is allowed to transmit if (1) the class is not overlimit *or* (2) the class has a non-overlimit ancestor at level *i*, and there are no unsatisfied classes in the link-sharing structure at lower levels than *i* (leaf classes are at level 0). These guidelines, called Formal Link-Sharing (more details in [1]), prevent a class from sending more than its allocation whenever another leaf exists that is backlogged and underlimit. Such a class is defined as *unsatisfied*.

Classes that are allowed to exploit excess bandwidth are called *unbounded*[2] classes [1]. They could make use of the excess bandwidth by sending at higher rates than its allocation and by consuming part of the excess bandwidth available on the parent class. Vice versa, a *bounded* class does not exploit second guideline because it cannot send more than its allocated bandwidth even if the output link is idle and the class is backlogged. Bounded classes can be useful to keep high priority traffic under control, aside of some economic issue (a customer who wants to have an unbounded class should pay more).

3 CBQ: An in-depth analysis

The most common CBQ implementations[3] can be found in the *ns-2* simulator and *ALTQ* tool, although both of them present some discrepancies from the original specification (some have been pointed out in [3]). However an in-depth analysis of the CBQ behavior shows some weakness that will be briefly pointed out. This analysis is focused particularly on the *ns-2* implementation, although differences between *ns-2* and *ALTQ* are negligible.

3.1 Formal link-sharing guidelines

Sometimes formal link-sharing guidelines have a too conservative approach in allowing underlimit classes transmitting. Figure 4 shows an example in which C2 (a CBR session whose rate is 40% of the total link bandwidth) is every now and then suspended because link-sharing guidelines detect that there is an unsatisfied class (A1) in the hierarchy. From formal link-sharing guidelines, C2 (that is an overlimit class) is allowed to transmit only if there are no unsatisfied classes at a level below of the class it is borrowing from (Agency C). However this is not correct from the link-sharing perspective: Agency C is underlimit because it owns 50% of the total bandwidth while C2 is using only 40% and there is no other traffic in there, therefore C2 should be allowed to send without limits.

Imprecise results obtained by formal link-sharing guidelines during small-scale intervals can be problematic when customers are willing to have their guaranteed bandwidth according to the amount of money they paid for.

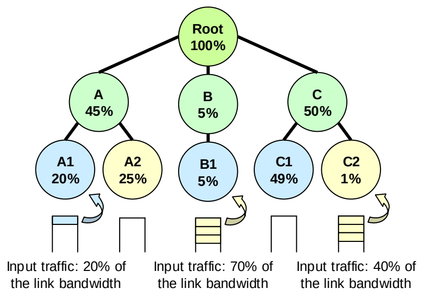

Root
100%
A
45%
B
5%
C
50%
A1
20%
A2
25%
B1
5%
C1
49%
C2
1%
Input traffic: 20% of the link bandwidth
Input traffic: 70% of the link bandwidth
Input traffic: 40% of the link bandwidth

Figure 4. Formal link-sharing guidelines problems.

3.2 Decoupling bandwidth and delay

Due to the several prioritized and cascading WRR schedulers, excess bandwidth is assigned to high priority unbounded classes first. In this way bandwidth and delay are not decoupled because high priority classes get a better service from both the viewpoints of delay (high priority classes are served first) and bandwidth (excess bandwidth is shared among high priority classes first).

We argue that bandwidth and delay must be set (and assigned) independently each one from the other in order to provide a more predictive (and fair) service. Furthermore this approach simplifies network management because it is easier to control the amount of high priority traffic flowing through the network.

Surprisingly (Table 2), high priority classes tend to exceed their allocation even when there is no excess bandwidth to distribute. This is due to some problems in computing internal parameters and will be pointed out in the next Section.

3.3 CBQ precision in the steady-state

Making CBQ respectful of given parameters has been proved being a non-trivial task [3]. Table 1 shows an example of a typical output trace of *ns-2* in which the imposed steady state burst (two packets) is not respected. Of course, the suspension time is not respected either

[2] Bounded classes are often called “without borrowing” classes, whereas unbounded classes are often called “borrowable” classes.

[3] Linux also has CBQ support, written by Alexey Kuznetsov.

(*15.2* ms instead of *9.8* ms). Often this does not affect too much the class throughput; however it remains the fact that imposed parameters are not respected.

Simulations show that CBQ is largely imprecise and each session has several problems in getting the exact share assigned to it. Some results showing the low degree of precision obtained by this scheduler will be shown in Table 2. Particularly, bounded classes always tend to get less than their share; in the meanwhile small classes tend to exceed their bandwidth (and large classes tend to get less) when borrow is enabled. Results get even worse when the small class has highest priority.

Class Structure:
- root class 2Mbps, two leaf classes (1% and 99% shares)
- Traffic: 100Kbps inbound traffic, Class 1 only
- Traffic pattern: Constant Bit Rate, packet size 120 bytes

CBQ parameters:
- `maxburst, minburst (M, m)`: 8, 2 packets
- Typical suspension time (`extradelay`): 9.821 ms

Packet #	Exit time (ms)	Inter-packet time (ms)	Bursts
1	10.560	0.048	First
2	10.608	0.048	
3	10.656	**15.264**	
4	25.920	0.048	Second
5	25.968	0.048	
6	26.016	**15.254**	

Table 1. Suspension time and steady state burst duration.

3.4 CBQ minor issues

The link-sharing scheduler suspends a class only when it is not allowed to transmit any further. This could happen when a class becomes overlimit or, more likely (in case of unbounded classes) when its parent is no longer allowed to transmit. What is important, however, is that parent class is never suspended; only leaf classes are. This introduces a certain degree of unfairness.

3.4.1 Delay vs. Link-Sharing guarantees

We can imagine that A2 in Figure 5 is idle and that A1 is using all its parent bandwidth (80%). After a while A1 becomes overlimit; however it is still allowed to transmit because of the excess bandwidth granted by Agency A. In any case, soon or later, Agency A will become overlimit and class A1 will be suspended.

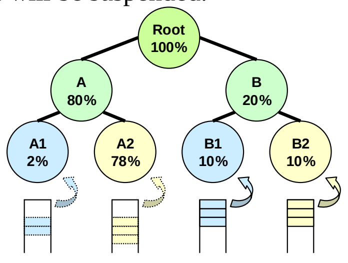


Figure 5. Delay vs. Link-Sharing guarantees.

Let's suppose that A2 starts having packets in queue: A2 is underlimit; therefore it is allowed to transmit even if its parent is not allowed to do the same. Since A2 has a large amount of allocated bandwidth, Agency A is going to use more bandwidth than the value allocated to it; therefore punishment for Agency B could be significant. This behavior is due to the formal link-sharing guidelines that privileges delay (leaf-level) guarantees at the expense of the link-sharing structure during small-scale intervals.

3.4.2 Suspending overlimit classes

This case is quite similar to the previous one: A2 is using all the parent bandwidth, thus Agency A (as well as A2) becomes overlimit. At that moment A1 starts transmitting packets and it continues until it becomes overlimit. As soon as this happens, A1 is suspended because Agency A is overlimit as well. The result is that class A1 is suspended even if excess bandwidth was almost entirely consumed by A2.

This is unfair but still reasonable: the suspension process for a leaf class is a matter of how much its parent and its siblings transmitted in the past as well as its own behavior. However there is another side effect. Since the suspension time depends on the leaf class and it can be quite large (in case of a class with small guaranteed bandwidth such as A1), leaf classes can be punished for the excess bandwidth that has been used by other sibling classes. This is a common problem in hierarchical schedulers (H-PFQ, for example) and it can be quantified by a specific index (Worst-case Fair Index [2]).

Leaf bounded classes do not have this problem because they are not allowed to transmit more than the allocated bandwidth (they are not allowed to borrow); therefore these classes are not influenced by other entities.

4 Decoupled Class Based Queuing

D-CBQ aims at the resolution of the problems pointed out in previous Sections. The most important modifications include new link-sharing guidelines, the decoupling between bandwidth and delay, better delay bounds and improved bandwidth precision. Minor issues include fixing some inaccuracy and some optimizations.

4.1 New Link-Sharing Guidelines

New link-sharing guidelines require the definition of the Bounded Branch Subtree (BBS). All the bounded classes plus all the classes that are child of the root class are called BBS-root. Each BBS-root generates a BBS that includes the set of classes that *share* a BBS-root as common ancestor plus the BBS-root itself; BBS can be embedded (Figure 6).

Each BBS acts as a new link-sharing hierarchy that is almost independent from the others. A class can belong to several BBSs, therefore it can have several BBS-root classes. Among these BBS-roots, the one with the lowest level in the link-sharing hierarchy is called L-BBS-root. The BBS generated by the L-BBS-root is called L-BBS.

The distribution of the bandwidth is done by means of a two-step process that gives precedence to unsatisfied leaf classes. According to the first rule, a leaf class is allowed to transmit immediately if it is underlimit and its L-BBS-root is underlimit as well. This prevents the L-BBS from consuming bandwidth reserved to other subtrees. When no classes are allowed to send according to the previous rule, a second rule specifies that excess bandwidth has to be distributed to the all the *unbounded* classes according to their L-BBS. A class (unbounded) is allowed to get more bandwidth when it has a non-overlimit ancestor A^4 at level *i* and there are no unsatisfied classes in its L-BBS at levels lower than *i*. This guarantees that the excess bandwidth is distributed inside the L-BBS; therefore an overlimit class is allowed to transmit if there is still bandwidth available in its L-BBS.

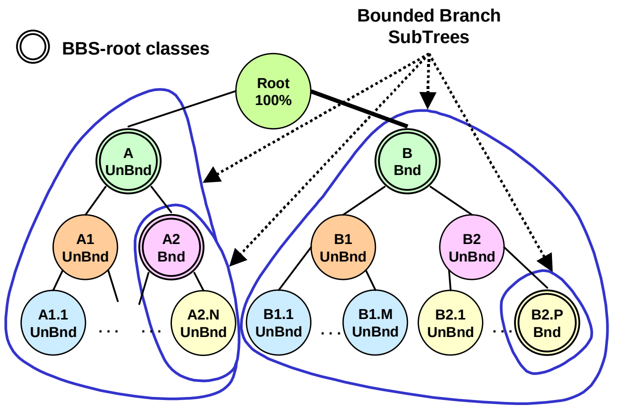


Figure 6. Bounded Branch Subtrees.

First rule allows a leaf class that is underlimit and bounded to send without constraints, while an underlimit and unbounded leaf class can be delayed when its L-BBS-root is overlimit. Bounded leaf classes are not influenced by the behavior of other classes, while unbounded classes are. This does not represent a problem because an unsatisfied class that is delayed will be served as soon as its L-BBS-root becomes underlimit. No starvation risks exist and underlimit leaf classes are always able to get their allocated rate, provided that the time-interval used to determine their throughput is appropriate (larger than the one used for bounded classes).

Unfortunately, these rules do not guarantee that any BBS-root never becomes overlimit. For instance, Agency B could become overlimit when class *B2.P* in Figure 6 sends data, therefore Agency B could get more bandwidth at the expense of Agency A. This is the common problem of hierarchical schedulers: link-sharing and leaf-class guarantees cannot always be met at the same time. D-CBQ chooses to privilege bounded classes that are always able to send (if they are respectful of their service rate). On the other side, these classes are not allowed to exceed their assigned rate.

[4] The class must be able to borrow from ancestor A. Basically, A can be either a generic ancestor belonging to the L-BBS or the root class (in case the L-BBS-root is unbounded).

4.2 Decoupling Bandwidth and Delay

D-CBQ distributes bandwidth by using two distinct systems of cascading WRR schedulers; the first one is activated when bandwidth is distributed according to the first link-sharing rule; the second one distributes excess bandwidth and it is enabled when bandwidth is allocated according to the second rule. First WRR uses priorities and guarantees each class to be able to get its allocated rate; second WRR does not take care of priorities and it serves classes according to their share.

This mechanism is rather simple but effective: network managers can assign each user a specific value of bandwidth and delay and they will be certain that, whatever the priority is, excess bandwidth will be distributed evenly to all the currently active classes.

4.2.1 Suspending overlimit classes

Suspending an overlimit class is a critical issue and it requires deciding when and how long a class has to be suspended.

D-CBQ suspends a class when, according to the second rule, the class is not allowed to transmit. D-CBQ can suspend any class (CBQ suspends leaf only); particularly D-CBQ suspends the highest-level ancestor whom the class is allowed to borrow from and that is overlimit. The problem pointed out in Section 3.4.1 does no longer exist: ancestor class is suspended; therefore all unbounded leaf classes that share this ancestor are no longer allowed to transmit. Bounded classes, of course, are still allowed to transmit (first link-sharing guideline).

A suspended class forces all its children (except bounded classes) to be "suspended". Suspension time (`extradelay`) will be the one that makes the subtree conformant to its allocated rate, i.e. it will be the one of the class that is being suspended. Since the suspension time depends on the bandwidth allocated to the intermediate class ("upper overlimit ancestor") instead of the one allocated to leaf classes, this modification is able to solve the problem pointed out in Section 3.4.2.

4.3 D-CBQ minor issues

A first modification is the definition of a criterion for setting `minidle` (Section 2.2) that was not completely specified in [1]. We assume that `avgidle` cannot become smaller than the one obtained by sending two back-to-back packets with size equal to the network MTU (Maximum Transfer Unit). Therefore the computation is based on the same steps that lead to Equation (3):

$$minidle = w \cdot (2-w) \cdot \frac{MTU}{r} \cdot \left(\frac{1}{f} - 1\right) \quad (6)$$

This value was chosen to differentiate the behavior of a class that is using excess bandwidth from the one that is just exceeding its allocation (when the last packet makes WRR allocation negative). Since a class cannot send more than one packet (of size MTU) beyond its WRR allocation, differentiation among these cases can be obtained by using two packets for setting `minidle`.

A second modification concerns the suspension time that takes into account that a class may send more than its allocation (packets are non divisible entities), therefore getting a negative `avgidle`. The modification (already included in ALTQ) adds a new variable `tidle` that represents the time a class has to be suspended in order to get the right share (i.e. the time needed to make `avgidle=0`). This variable takes into account how much the class exceeded its allocation during previous period, therefore:

$$\begin{aligned} &avgidle_{n+1} = 0 = (1-w)\cdot avgidle_n + w\cdot tidle \\ &\Rightarrow tidle = \frac{w-1}{w} avgidle_n \end{aligned} \quad (7)$$

and the resume time (for a suspended class) becomes the previous one (`last_time + extradelay`) plus this new term (`tidle`). This modification (coupled with the new event handler for waking up a sleeping class) is able to solve the precision issues, pointed out in Section 3.3.

4.4 D-CBQ optimizations

Some optimizations have been added to the original CBQ code. The first one allows D-CBQ to resume each suspended class at its correct time[5] by means of a new item into the `ns-2` event queue. A second optimization is the implementation of WRR scheduler in a more precise form[6]. A third one consists in the initialization of some variable to their proper value, for example `avgidle` and `undertime`[7]. These modifications, although marginal, concur to guarantee the improved behavior of D-CBQ compared to CBQ.

5 Results

Results (obtained through *ns-2* simulations and ALTQ tests) confirm that link-sharing has been improved, the decoupling between delay and bandwidth works well, bandwidth assignments are respected and delay bounds are tighter.

Table 2 reports the results obtained by CBQ and D-CBQ with a simple 1-level hierarchy (all classes are child of the root class). Even if these results are very limited, they show that priority does not influence bandwidth. In case of all classes competing for the bandwidth (first two tests), the classes get the assigned share. Moreover, last test shows that D-CBQ assigns the excess bandwidth to all the classes proportionally to their share (CBQ assigns that to the high priority class). The result is that D-CBQ looks more like a Weighted Fair Queuing than a Priority Queuing schema from this point of view and it is able to force malicious users not to send more than their allocated rate. Setting higher priorities (than means lower delays) in D-CBQ is no longer a way to obtain more bandwidth.

Class	Share	Priority	Traffic (Kbps)			
			In	CBQ out	D-CBQ out	Theor.
A	1%	LOW	100	55.25	21.46	20
B	99%	LOW	2000	1944.77	1978.56	1980
A	1%	HIGH	200	181.82	21.41	20
B	99%	LOW	2000	1818.19	1978.61	1980
A	10%	HIGH	2000	604.08	250.37	250
B	20%	LOW	---	---	---	---
C	70%	LOW	2000	1395.94	1749.65	1750

Table 2. Decoupling bandwidth and delay; all classes are allowed to borrow.

An extended characterization of D-CBQ can be found in [12]. Due to the high number of simulations, results have been summarized by means of appropriate Quality Index. The quadratic quality index (Q^2) limits the influence of small deviations from the expected value; therefore it tends to highlight tests in which the behavior is significantly different from the expected value and it is used to identify any idiosyncrasies between theoretical and real behavior. Vice versa, linear index ($|Q|$) is the relative difference of the simulation results compared to the theoretical ones and it can be used to show the precision of CBQ and D-CBQ against the expected result. Best results are obtained when these indexes tend to zero.

The comparison between CBQ and D-CBQ is summarized in Table 3, which shows the results of the link-sharing and delay bounds tests respectively. Results are extremely interesting and prove that D-CBQ performs far better than CBQ in all the cases.

	Quadratic Quality Index		Linear Quality Index	
Test	CBQ	D-CBQ	CBQ	D-CBQ
Link-Sharing	73.1666	0.0815	14.0525	1.7387
Delay	67.756	0.047	3.749	0.064

Table 3. Link-sharing and Delay test results (values * 100).

ALTQ tests confirm that D-CBQ has approximately the same complexity of CBQ, even in presence of non-optimized code. Moreover real D-CBQ implementation forced to keep in mind the scalability issues of our

[5] CBQ resumes classes when another event occurs in the simulator, for example when a new packet arrives.

[6] CBQ increases the WRR allocation of each class only when it is non-positive, making almost impossible to reach a full allocation. Moreover, the number of rounds needed to reintegrate the allocation is wrong because the allocation is sometimes set to zero arbitrarily.

[7] `avgidle` is initialized to its maximum allowed value (`maxidle`) instead of zero; see [8] for more details. `undertime` is initialized to a negative value in order to be able to schedule the first packet at once.

approach. For instance, classes cannot be waked up as soon as their `undertime` becomes positive because the system could be overloaded by several interrupts. D-CBQ adopts the same solution already proposed in ALTQ, i.e. it exploits the granularity of the kernel timer (usually 1 KHz) of a FreeBSD system in order to approximate the wake-up time in the best way.

6 Conclusions and future work

This paper presents D-CBQ, a CBQ-derived algorithm that decouples bandwidth and delay, sets new rules for the distribution of excess bandwidth, and improves both bandwidth precision and delay characteristics. D-CBQ has three main points. First, a new set of link-sharing guidelines that change the way service is allocated to each class and the way link-sharing is respected. Second, it includes a decoupling mechanism that allows high priority classes to consume only their allocated bandwidth and that allocates excess bandwidth to all classes no matter of their priority. Third, it defines a new algorithm that keeps rate under control by suspending classes that are directly responsible for exceeding their rate. Finally, it fixes several minor problems of CBQ.

All the modifications together guarantee a sensible improvement of link-sharing, decoupled capabilities, delay bounds. However, as stated in [9], link-sharing and leaf-classes properties cannot be guaranteed at the same time. D-CBQ makes no exception to this rule. D-CBQ tends to privilege leaf-classes properties for bounded classes, while it privileges link-sharing properties for the others. Bandwidth guarantees for unbounded classes are respected over larger scale intervals; these classes might not be allowed to send at their allocated rate in small intervals because the behavior depends on their ancestors as well. On the other side, link sharing has to be "relaxed" in case of bounded classes, because they do not depend on their ancestor for being able to transmit. However, this guarantees that bounded classes have better delay bounds because they are not influenced by the behavior of other sibling classes.

Summarizing, D-CBQ has different objectives and different management structures between bounded, devoted to traffic that requires strong service guarantees (for example real-time services), and unbounded classes, devoted to elastic traffic that can tolerate occasional delays and that could be willing to transmit data as much as possible. Bounded classes are able to exploit the allocated bandwidth quite precisely and they have better delay characteristics compared to CBQ; moreover they are not influenced by the behavior of other sibling classes. Vice versa, unbounded classes can go up to their share but other sibling classes can influence their behavior. Delay objectives can be reached by changing the priority of the class. Results confirm that both absolute and 99-percentile delay bounds are greatly improved, particularly the former. Delay distribution has been improved as well but we are not able, at this stage, to quantify the improvement. Excess bandwidth is now distributed independently of the priority; therefore setting highest priority for unbounded classes does no longer make sense.

Future work deserves several interesting points because an in-depth analysis of D-CBQ is still far from being completed. D-CBQ should be proper characterized in a multi-path environment, particularly for aspects concerning delay bounds. In order to make D-CBQ suitable for deployment in real networks, further studies about the statistical distribution of the delay experimented by high priority packets are desirable.

Acknowledgements

The author thanks Salvatore Iacono, Giordana Lisa and Kenjiro Cho for many discussions about CBQ internals, Ivan Ponzanelli and Lucio Mina for their help in testing and validating the prototypes, Panos Gevros, Mario Baldi and Jon Crowcroft for their comments.

This work has been partially sponsored by Telecom Italia Lab, S.p.A., Torino (Italy).

Bibliography

[1] Sally Floyd and Van Jacobson, *Link Sharing and Resource Management Models for Packet Networks*, IEEE/ACM Transaction on Networking, Vol. 3 No. 4, August 1995.

[2] Jon C.R. Bennett and H. Zhang, *Hierarchical Packet Fair Queuing Algorithms*, IEEE/ACM Transactions on Networking, 5(5):675-689, Oct 1997.

[3] Fulvio Risso and Panos Gevros, *Operational and Performance Issues of a CBQ router*, ACM Computer Communication Review, Vol. 29 No 5, October 1999.

[4] J. Nagle, *On packet switches with infinite storage*, IEEE Transactions on Communications, 35(4):435-438, April 1987.

[5] A. Demers, S. Keshav, and S. Shenker, *Analysis and Simulation of A Fair Queuing Algorithm*, ACM Computer Communication Review (SIGCOMM '89), pp. 3–12, 1989.

[6] A. K. Parekh and R. G. Gallager, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Single-node Case*, IEEE/ACM Transactions on Networking, vol.1, no. 3, pp. 344–357, June 1993.

[7] A. K. Parekh and R. G. Gallager, *A Generalized Processor Sharing Approach to Flow Control in Integrated Services Networks: The Multiple Node Case*, IEEE/ACM Transactions on Networking, vol. 2, no. 2, pp. 137–150, Apr. 1994.

[8] S. Floyd, *Notes on Class Based Queuing: Setting Parameters*, Informal notes, September 1995.

[9] Ion Stoica, Hui Zhang, T. S. Eugene Ng, *A Hierarchical Fair Service Curve Algorithm for Link-Sharing, Real-Time and Priority Service*, in Proceedings of SIGCOMM '97 September 1997.

[10] Fulvio Risso, *Delay Guarantees in D-CBQ*, Draft Paper, Politecnico di Torino, April 2001.

[11] Mario Baldi and Fulvio Risso, *Efficiency of Packet Voice with Deterministic Delay*, in IEEE Communications Magazine, vol. 28 n° 5, pg. 170-177, May 2000.

[12] Fulvio Risso, *Implementation and Characterization of an Advanced Scheduler*, in Proceedings of 1st International Conference on Networking (ICN '01), Colmar, France, July 2001.

[13] Francois Toutain, *Decoupled Generalized Processor Sharing: a Fair Queueing Principle for Adaptive Multimedia Applications*, Proceedings of INFOCOM '98, Volume 1, pg. 291-298.