POLITECNICO DI TORINO
Repository ISTITUZIONALE

Implementation and Characterization of an Advanced Scheduler

Original
Implementation and Characterization of an Advanced Scheduler / Risso, F.G.O.. - STAMPA. - 2093:(2001), pp. 85-97. (Networking - ICN 2001, 1st International Conference on Networking, Part I Colmar (France) July 9–13, 2001) [10.1007/3-540-47728-4_9].

Availability:
This version is available at: 11583/1417039 since: 2021-10-10T19:14:24Z

Publisher:
Springer

Published
DOI:10.1007/3-540-47728-4_9

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

07 November 2026

Implementation and Characterization of an Advanced Scheduler

Fulvio Risso

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

Abstract. Decoupled-CBQ, a CBQ derived scheduler, has been proved being a substantial improvement over CBQ. D-CBQ main advantages 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". This paper aims at the characterization of D-CBQ by means of an extended set of simulations and a real implementation into the ALTQ framework.

1. Introduction

Class Based Queuing [1], (CBQ) and Hierarchical Fair Service Curve [2] (H-FSC) represent an interesting solution to integrated networks that aim to provide hierarchical link-sharing, tighter delay bounds and bandwidth guarantees. However the configuration of HFSC is less intuitive from an Internet Service Provider point of view, therefore CBQ is the most appealing advanced scheduler available today.

An in-depth analysis of CBQ, on the other hand, shows several problems; most of them have been pointed out in [3] and [7]. This paper aims at the completion of [7] by summarizing the D-CBQ characteristics, presenting the implementation issues, and characterizing this new scheduler.

This paper is structured as follows. Section 2 summarizes D-CBQ characteristics; Section 3 presents the simulations used to validate the prototype, while Section 4 discusses the efforts in implementing D-CBQ in a real router. Finally, Section 5 presents some conclusive remarks.

2. Decoupled Class Based Queuing

The most important D-CBQ characteristics include new link-sharing guidelines, the decoupling between bandwidth and delay and the excellent precision in respecting bandwidth and delay.

1.1 New Link-Sharing Guidelines

New link-sharing guidelines require the definition of the new concept of the Bounded Branch Subtree (BBS). All the bounded[1] classes plus all the classes that are child of the root class are called BBS-root. Each BBS-root generates a Bounded Branch Subtree that includes the set of classes that *share* a BBS-root as common ancestor plus the BBS-root itself. BBSs can be embedded; an example can be seen in Fig. 1

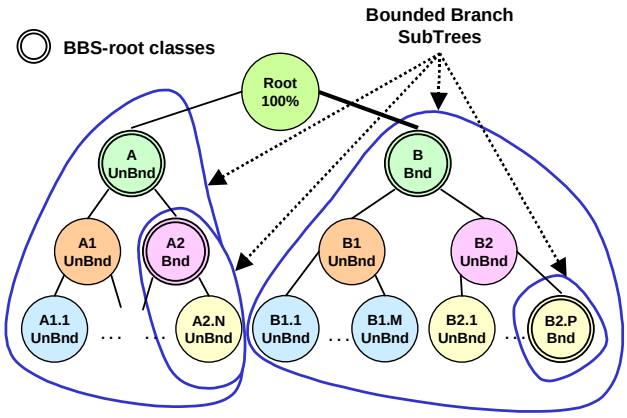


Fig. 1. Bounded Branch Subtrees. Classes can be either unbounded ("UnBnd") or bounded ("Bnd")

Each BBS acts as a new link-sharing hierarchy that is almost independent from the others. A class may 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[2] and its L-BBS-root is underlimit as well. This prevents the L-BBS from consuming bandwidth reserved to other subtrees.

A second rule distributes excess bandwidth to the all the *unbounded* classes (according to the L-BBS they belong) when no classes are allowed to send according to the first rule. A class is allowed to get more bandwidth when it has a non-overlimit[3] ancestor A[4] at level

[1] A bounded class is a class whose traffic that cannot exceed its allocated rate.

[2] A class is underlimit if its throughput (averaged over a certain period) does not exceed its allocated rate. See [1] for more details.

[3] A class is non-overlimit when it does not exceed its rate, averaged over a certain period.

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 provided that there is still bandwidth available in its L-BBS.

First rule allows a leaf class that is underlimit and bounded (that is the suggested configuration for real-time classes) to transmit without constraints, while an underlimit and unbounded leaf class can be delayed when its L-BBS-root is overlimit. Bounded leaf classes are never 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. Starvation does not occur and underlimit leaf classes are always able to reach their target rate; however the time-interval used to monitor their throughput must be larger than the time interval used to monitor bounded leaf classes.

Unfortunately, these rules do not guarantee that any BBS-root never becomes overlimit. An embedded L-BBS-root can be allowed to transmit (it is underlimit indeed), making the higher-level BBS-root overlimit. It follows that even D-CBQ does not respect perfectly the link-sharing structure; however this is the fee that has to be paid in order to have some classes that are always able to send (provided that they are respectful of their service rate). This fee is required to provide excellent support for guaranteed-bandwidth classes that, on the other hand, are not allowed to send more than their guaranteed rate.

1.2 Decoupling Bandwidth and Delay

This feature requires that the excess bandwidth will be distributed independently of the class priority. D-CBQ uses 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 (i.e. it gives precedence to high priority traffic) and it guarantees each class to be able to get its allocated rate; second WRR does not take care of priorities and it selects 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 sessions.

Suspending overlimit classes

It has been widely recognized that link-sharing rules and delay guarantees cannot be met at the same time; therefore there are small time intervals in which one of them cannot be guaranteed. The decoupling between bandwidth and delay has to be integrated with new rules that determine when a class has to be suspended (because either it or one of its parents is overlimit) and how long the suspension time will be.

---

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

First answer is based on the new link-sharing guidelines: a class is suspended when is not allowed to transmit according to the second rule. D-CBQ allows each class being suspended, whereas CBQ allows suspension of leaf classes only. In this case D-CBQ suspends the highest-level ancestor (i.e. the nearest to the root class) whom the class is allowed to borrow from and that is overlimit. Since ancestor class is suspended, 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).

Second question (*how long*) is based on the observation that a class (or a subtree, in the D-CBQ case) must be suspended for the time needed to be compliant to the allocated rate; hence the suspension time will be the one of the class that is being suspended, that is the ancestor class. It follows that the suspension time depends on the `extradelay` of the ancestor class; therefore it depends on the bandwidth allocated to the intermediate class instead of the one allocated to leaf classes. Generally speaking, suspension time depends on the "upper overlimit ancestor" instead of the leaf class.

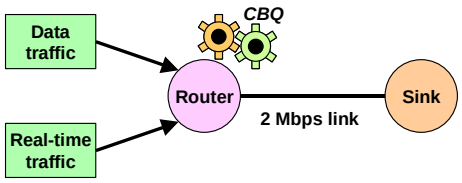


Fig. 2. Test topology.

3. Simulation results

This Section presents some comparative results between CBQ (in the implementation that comes with `ns-2`) and D-CBQ through the `ns-2` simulator. Results have been obtained by defining two different test suites; the first one devoted to bandwidth and link-sharing properties and the second one devoted to the delay objectives[5]. Each test suite consists in several tests; each one has a different class configuration (link-sharing structure, borrow, priority) and it is structured in several simulations that have different incoming traffic and different bandwidth allocated to each class.

D-CBQ (as well as CBQ) is, by nature, a non-work conserving algorithm although it can be easily modified in order not to leave the output link idle when there are unbounded

[5] Simulations measure the scheduling delay, i.e. the time between the arrival of the packet and the time the scheduler finishes its transmission on the output link.

backlogged classes. For instance, results compare CBQ with two different versions of D-CBQ, standard (described in Section 2) and *efficient*. Efficient D-CBQ (D-CBQe) looks like a work-conserving algorithm and it adds a new rule to D-CBQ. When no classes are allowed to transmit according to the link-sharing guidelines, D-CBQe sends a packet from the first unbounded and backlogged class it encounters. Therefore it is not a pure work-conserving algorithm because bounded classes are not able to exploit the output link idleness.

The efficient mechanism inserts a small degree of unfairness into D-CBQ. To keep the algorithm simple, the efficient process selects classes on a priority-based schedule; hence high priority classes can get more bandwidth. Moreover D-CBQ internal variables are not updated in case of a transmission due to this mechanism: the rational is that a class should not be punished for the bandwidth consumed when no classes are allowed to transmit.

Simulations use a simple topology (Fig. 2) composed by a single scheduler and several sources attached to it. Number of sources, their rate and their traffic pattern varies among the simulations. Sessions under test use CBR and Poisson sources because of their simplicity and their easiness to be controlled. Tests repeated using real sources (UDP/TCP sessions simulating real traffic) do not show any difference compared to previous ones.

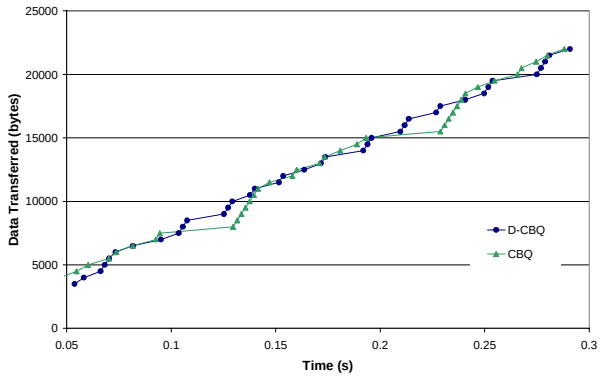


Fig. 3. New Link-Sharing guidelines.

New Link-Sharing guidelines

This test shows that D-CBQ is able to guarantee each session a more predictable service. Fig. 3 shows a typical trace in which both CBQ and D-CBQ are able to provide the correct share over large-scale intervals. However this is not true over small scale:

triangle trace shows that CBQ often suspends the class (large amount of time between two packets); this does not happen with D-CBQ.

Decoupling Bandwidth and Delay

D-CBQ has strong decoupling characteristics. For example Table 1 reports the results obtained with a simple 1-level hierarchy (all classes are children of the root class). Priority does not influence bandwidth in D-CBQ. In case of all classes competing for the bandwidth (first two tests), they are able to obtain the assigned share. Moreover, last test shows that CBQ assigns all the available bandwidth (not used by class B) to the high priority class; D-CBQ allocates the excess bandwidth to all classes proportionally to their share. 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 exceed their allocated rate. Setting higher priorities (than means lower delays) in D-CBQ is no longer a way to obtain more bandwidth.

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

Table 1. Decoupling bandwidth and delay; all classes are allowed to borrow.

Quality indexes

Previous tests are not able to validate for certain the goodness of D-CBQ from other points of view as well (delay, precision). Moreover, these tests have to be carried out using several different configurations; therefore data need to be summarized in order to display the results.

Comparison among different results is done using the following set of quality indexes:

$$Q^2 = \frac{1}{N} \cdot \sum_{i=1}^{N} \left(\frac{D_i - D_i^*}{D_i^*} \right)^2, \qquad |Q| = \frac{1}{N} \cdot \sum_{i=1}^{N} \left| \frac{D_i - D_i^*}{D_i^*} \right|$$

where D_i^* is the expected test result (the theoretical value for link-sharing tests; the best obtained delay in the current simulation for delay tests), D_i is the actual result of the

simulation and N is the number of simulations. The quadratic quality index (Q^2) highlights tests in which the behavior is significantly different from the expected value; therefore 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.

Link-sharing test suite details

Main objective is to verify the ability to provide each class with its target bandwidth; therefore sources (CBR) exceed the rate allocated to that class. The link-sharing test suite is made up of 10 different tests that use three different class configurations (details in Appendix I); each test aims at the evaluation of specific aspects of the algorithm.

A brief summary of the results (details can be found in [5]) is shown in Table 2: quality index for CBQ is by far the worst of all. Results confirm that D-CBQ performs far better than the original algorithm, particularly in tests with borrowing enabled and different priorities among classes. $|Q|$ shows that the difference between experimental results and theoretical one is greatly reduced from the 14.1% of CBQ to the 1.7% of D-CBQ.

An interesting point is that the efficient version of D-CBQ performs worse than the standard version. Efficient mode, in fact, inserts another trade-off between link utilization and precision parameters. For instance, a class could not be allowed to transmit at its target time when efficient mode is turned on because the scheduler could be busy servicing another packet.

	Quadratic Quality Index			Linear Quality Index		
Test	**CBQ**	**D-CBQ**	**D-CBQe**	**CBQ**	**D-CBQ**	**D-CBQe**
Test 1	0.3112	0.0111	0.0111	5.0110	0.8345	0.8345
Test 2	0.0000	0.0000	0.0000	0.0008	0.0008	0.0008
Test 3	0.2841	0.0646	0.0646	5.0140	2.0899	2.0899
Test 4	51.7934	0.0948	0.0948	30.5246	1.7590	1.7590
Test 5	0.2994	0.0721	0.0721	4.8157	2.2212	2.2212
Test 6	1091.2810	0.0886	0.0886	137.4755	1.6486	1.6486
Test 7	0.0008	0.0001	0.0001	0.2325	0.0671	0.0671
Test 8	0.0099	0.0001	0.0001	0.8036	0.0780	0.0780
Test 9	0.1672	0.0482	0.2211	3.5506	1.7568	2.7292
Test 10	0.3756	0.1706	0.3479	5.0937	2.5652	3.8744
Global	**73.1666**	**0.0815**	**0.1784**	**14.0525**	**1.7387**	**2.3698**

Table 2. Link-sharing test results (values * 100).

A small problem still remains: even D-CBQ is not able to transmit all the traffic when input sources are transmitting at their maximum rate. This is due to internals

approximations. A good practice consists in a slight over-provisioning of the bandwidth allocated to that class; an in-depth analysis of this phenomenon is left to future studies.

Delay test suite details

Delay test suite is made up of five tests that differ in the characteristics of the real-time sources. Simulations use both CBR and Poisson traffic for real-time sources, and a set of several VBR sessions for data traffic. Real-time sources transmit slightly less than the bandwidth allocated to their class; vice versa data traffic exceeds its allocation in order to make the output link congested. Three tests use a single CBR source for each session (each test has a different packet size); the fourth uses three CBR sources with different packet sizes (120, 240, 480 bytes) for each session, while the last uses Poisson sources. Last two tests use a token-bucket limiter to regulate real-time sources and to control input pattern (and source's peak rate) with excellent accuracy. Twelve different simulations with different link-sharing structure, priority and borrowing characteristics compose each test.

This paper summarizes the results related to the maximum delay experienced by packets and the delay experienced by the 99% of them (99-percentile). Results are given only for the real-time traffic because best effort one exceeds its allocation; therefore delay has no significance. Detailed analyses for delay bounds are left for future work [6]; here only a brief summary is given.

	Quadratic Quality Index			Linear Quality Index		
Test	**CBQ**	**D-CBQ**	**D-CBQe**	**CBQ**	**D-CBQ**	**D-CBQe**
CBR1	4.940	0.001	0.008	1.566	0.009	0.033
CBR2	158.514	0.000	0.010	8.236	0.000	0.034
CBR3	175.056	0.000	0.004	8.765	0.000	0.025
CBR-M	0.268	0.097	0.107	0.176	0.111	0.099
PS	0.000	0.138	0.127	0.003	0.201	0.168
Global	**67.756**	**0.047**	**0.051**	**3.749**	**0.064**	**0.072**

Table 3. Maximum delay tests: results (values * 100).

Starting from the maximum experimented delay, results (shown in Table 3) confirm that D-CBQ outperforms CBQ in all tests. Packets flowing through CBQ have a maximum delay that is far larger than the one experimented by D-CBQ. CBQ performs better only in a few simulations and this is due to its different (and wrong) implementation of the WRR mechanism[6].

[6] A detailed analysis of the CBQ code shows that it does not implement correctly the WRR mechanism and it often sets a class allocation to zero arbitrarily instead of leaving it negative.

D-CBQ improvement concerning the 99-percentile delay bound (Table 4) is not so evident such as in the maximum delay bound. D-CBQ, however, still guarantees smaller delays: delays of the 99-percentile of the CBQ packets are 4 times larger than the D-CBQ ones. Results in which CBQ outperforms D-CBQ (like PS) are due (again) to the different WRR implementation.

	Quadratic Quality Index			Linear Quality Index		
Test	**CBQ**	**D-CBQ**	**D-CBQe**	**CBQ**	**D-CBQ**	**D-CBQe**
CBR1	0.361	0.000	0.034	0.315	0.001	0.055
CBR2	0.036	0.002	0.016	0.080	0.013	0.049
CBR3	0.017	0.001	0.011	0.052	0.009	0.041
CBR-M	0.111	0.007	0.033	0.113	0.041	0.079
PS	0.000	0.071	0.083	0.005	0.118	0.125
Global	**0.105**	**0.016**	**0.035**	**0.113**	**0.036**	**0.070**

Table 4. 99-percentile delay tests: results (values * 100).

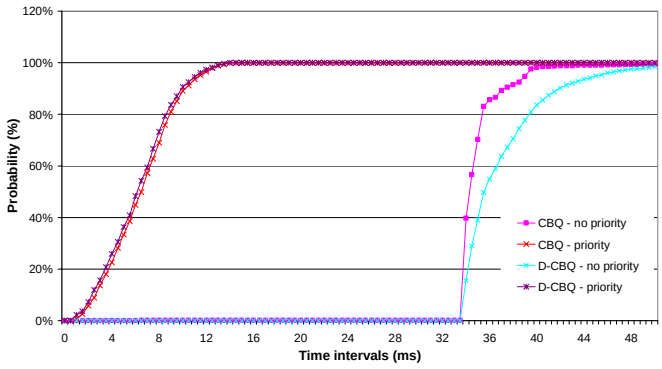


Fig. 4. A typical cumulative distribution of the delay in CBQ and D-CBQ.

Fig. 4 shows a typical distribution of the delay in CBQ and D-CBQ and it points out that delay distribution for high priority classes is almost the same, even if CBQ has a small percentage of packets that have significantly larger delays than D-CBQ.

D-CBQe performances are never better than D-CBQ. Even if the efficient part of the algorithm affects unbounded classes only, this feature influences real-time classes (that

---

This operation has a non-negligible impact on classes with large packets compared to their allocation: these classes do no longer need to wait several rounds before being able to transmit. Some configurations take particular advantage of that, hence CBQ may show better delay bounds.

are usually bounded) as well because it forces sometimes real-time classes to wait longer before transmitting a packet.

4. ALTQ implementation

ALTQ implementation is almost the same as the ns-2 one. The most important difference is the inability to wake up a class exactly at its target time, since this would require setting a new timer event each time a class is suspended. This might overload the processing power of the machine; therefore the suspension time is approximated within discrete intervals based on the kernel timer (set to 1KHz on our machines) available in the BSD kernel. Other differences are related to real-word issues, for example the presence of output interface buffers (virtually a FIFO queue after the CBQ scheduler) and the possible mismatch between the theoretical wire speed and the real one (for instance, header compression, link-layer headers and more may alter the real speed). First problem, pointed out in [3], is of great importance and is largely reduced in ALTQ 3.0 (several interface drivers have been modified). The second point does not have solutions at this moment and it results on some mismatch between some theoretical values (for example the packet departure time) and real ones.

ALTQ implementation has been validated using a subset of the test already used in ns-2 and results confirm the goodness of D-CBQ as well as in the simulations. The most important result, however, is that D-CBQ complexity is slightly more than CBQ one, although in presence of un-optimized code. This is evident primarily in the borrow tests in which D-CBQ has to check the status of the class' ancestors.

Test	Packet size (bytes)	CBQ	D-CBQ	Difference
No Borrow	40	7080	6599	-6.8%
	58	6992	6463	-7.6%
Borrow	40	6847	6186	-9.7%
	58	6691	6140	-8.2%

Table 5. ALTQ tests snapshot: maximum throughput (packets per second) on a Pentium 133 machine.

5. Conclusions

D-CBQ has been proved being a substantial improvement over CBQ. Its characteristics allow the deployment of this scheduler on networks with advanced requirements (hierarchical link-sharing, bandwidth guarantees, delay bounds).

Efficient D-CBQ has been shown being not worthy from the link-sharing and delay point of view. However further analyses are needed in order to evaluate the advantages of this algorithm on best effort traffic: we should expect some improvements in term of link utilization and throughput for these classes.

Next step will be a better characterization of D-CBQ from the viewpoint of the delay in order to give a mathematical indication of the maximum delay bound experimented by a D-CBQ session.

Source code for `ns-2` and ALTQ, together with test script, is online at the Author's website.

Acknowledgements

The author thanks Salvatore Iacono, Giordana Lisa and Kenjiro Cho for many discussions about CBQ internals. Best thanks also to Ivan Ponzanelli and Lucio Mina for their insightful 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] 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.

[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] The VINT Project, UCB/LBNL/VINT Network Simulator - ns (version 2). Available at http://www-mash.cs.berkeley.edu/ns/.

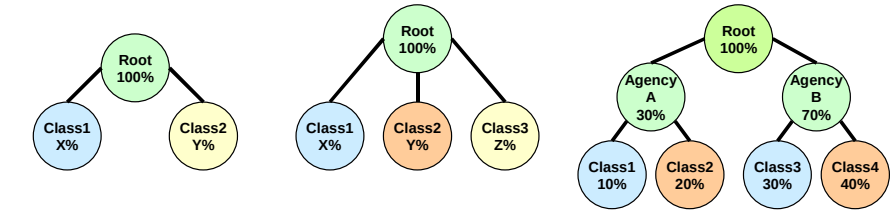
[5] Ivan Ponzanelli, *Garanzie di servizio con schedulers di tipo Class Based Queuing*, Laurea Thesis, Politecnico di Torino, July 2000. In Italian.

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

[7] Fulvio Risso, *Decoupling Bandwidth and Delay Properties in Class Based Queuing*, in Proceedins of the Sixth IEEE Symposium on Computers and Communications (ISCC 2001), July 2001

Appendix I

This appendix presents the details of the link-sharing and delay test suites.



First link-sharing structure

Test 1: no borrow, single flow
Test 2: borrow, single flow
Test 3: no borrow, both flows, same priority
Test 4: borrow, both flows, same priority
Test 5: no borrow, both flows, different priorities
Test 6: borrow, both flows, different priorities

Second link-sharing structure

Test 7: borrow, two flows, same priority
Test 8: borrow, two flows, different priorities

Third link-sharing structure

Test 9: different borrow, flows and priorities configurations
Test 10: same as test 9; classes with different packet sizes

Fig. 5. Link-sharing test suite: link-sharing structure.

The link-sharing test suite is made up of 10 different tests that use the three different class configurations shown in Fig. 5. Each test aims at the evaluation of specific aspects of the algorithm. In detail:

- Test 1: precision of the traffic carried by a single class, taken in isolation
- Test 2: ability to exploit all the link bandwidth by a single class, taken in isolation
- Tests 3, 4: ability to share correctly the bandwidth among peer classes; tests have been performed with and without borrowing
- Tests 5, 6: same as tests 3 and 4; classes have different priorities
- Tests 7, 8: ability to share the excess bandwidth correctly; classes can have either equal or different priorities
- Tests 9,10: ability to respect the imposed bandwidth among classes with different priorities, borrow configuration, incoming traffic; Test 10 repeats the same simulations using different packet sizes among classes. Configuration details are shown in Table 6, as well as the expected throughput of each class.

Delay test-suite is similar to the previous one: it consists in five link-sharing hierarchies, with different class configuration and traffic. A summary of the test characteristics is reported in Fig. 6.

Simulations		Classes					
		A	1	2	B	3	4
1	Priority Borrow/Traffic Expected throughput	LOW Y/--- 	LOW Y/Y 200	LOW Y/Y 400	LOW Y/--- 	LOW Y/N ---	LOW Y/Y 1400
2	Priority Borrow/Traffic Expected throughput	LOW N/--- 	LOW N/Y 200	LOW N/Y 400	LOW Y/--- 	LOW Y/N ---	LOW Y/Y 1400
3	Priority Borrow/Traffic Expected throughput	LOW N/--- 	LOW N/Y 200	LOW N/Y 400	LOW Y/--- 	LO Y/N 	LOW Y/N
4	Priority Borrow/Traffic Expected throughput	LOW Y/--- 	LOW Y/Y 200	LOW Y/Y 400	LOW Y/--- 	LOW Y/Y [7] 200	LOW Y/Y 1200
5	Priority Borrow/Traffic Expected throughput	LOW N/--- 	LOW Y/Y 200	LOW Y/Y 400	LOW Y/--- 	LOW Y/N 	LOW Y/Y 1400
6	Priority Borrow/Traffic Expected throughput	HIGH Y/--- 	HIGH Y/Y 200	HIGH Y/Y 400	LOW Y/--- 	LOW Y/N 	LOW Y/Y 1400
7	Priority Borrow/Traffic Expected throughput	HIGH Y/--- 	HIGH Y/Y 200	LOW Y/Y 400	HIGH Y/--- 	LOW Y/N 	HIGH Y/Y 1400
8	Priority Borrow/Traffic Expected throughput	HIGH N/--- 	HIGH Y/Y 200	HIGH Y/Y 400	LOW Y/--- 	LOW Y/N 	LOW Y/Y 1400
9	Priority Borrow/Traffic Expected throughput	HIGH N/--- 	HIGH Y/Y 200	LOW Y/Y 400	HIGH Y/--- 	LOW Y/N 	HIGH Y/Y 1400

Table 6. Details of tests 9 and 10 (throughput in Kbps).

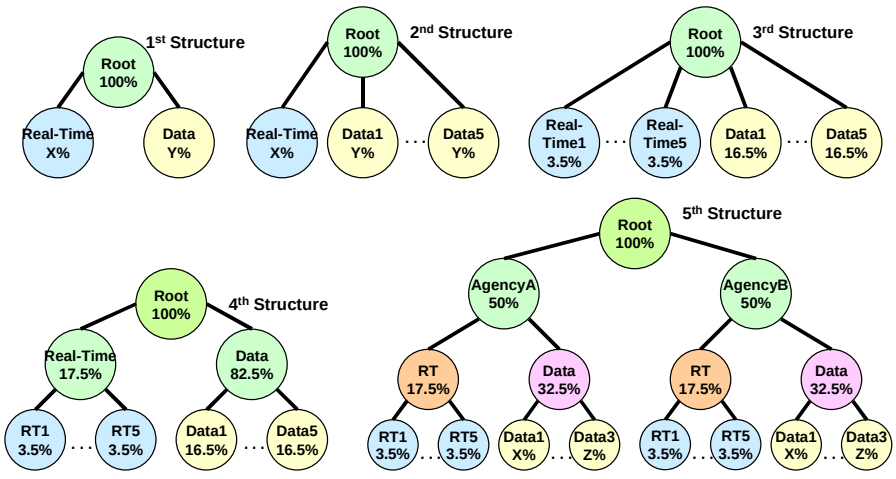


Fig. 6. Delay tests: the link-sharing structure.

[7] This test uses an on-off source, with 33% activity period.