Operational and Performance Issues of a CBQ router

Fulvio Risso* and Panos Gevros
Department of Computer Science,
University College London,
Gower Street WC1E 6BT,
London, U.K.
{f.risso, p.gevros}@cs.ucl.ac.uk

Abstract

The use of scheduling mechanisms like Class Based Queueing (CBQ) is expected to play a key role in next generation multiservice IP networks. In this paper we attempt an experimental evaluation of ALTQ/CBQ demonstrating its sensitivity to a wide range of parameters and link layer driver design issues. We pay attention to several CBQ internal parameters that affect performance drastically and particularly to "borrowing", a key feature for flexible and efficient link sharing. We are also investigating cases where the link sharing rules are violated, explaining and correcting these effects whenever possible. Finally we evaluate CBQ performance and make suggestions for effective deployment in real networks.

1 Introduction

Internet resource management requires mechanisms that control the allocation of resources on a per hop as well as on an end-to-end (per flow) basis. In this paper we focus on Class Based Queuing (CBQ), a "per hop" mechanism, investigating the implications of its deployment in real networks and its effects on the performance of end-to-end transport.

CBQ is a strong candidate as a building block for introducing new Internet service models (from standardised Integrated Services [1] to newly proposed Differentiated Services [2, 3]) because it can provide:

- per "entity" traffic isolation, with flexibility in defining "the entity" and therefore the degree of aggregation (per flow, per user, etc.)
- degrees of freedom for introducing a wide range of policies (based on services, protocol and network address information)
- bottleneck link sharing

However there is a substantial level of complexity involved in the deployment of resource management mechanisms. Their deployment is still at an early stage, their effects on end-to-end performance are not always straightforward and usually investigated only by simulations. This work is an experimental approach that focuses on the analysis of a CBQ implementation on a real network.

The paper is organised as follows: Section 2 presents an overview of the CBQ mechanism and the ALTQ implementation. Section 3 discusses the experiments and the methodology. Section 4 analyses the effects of the network driver architecture on the CBQ performance. Section 5 shows how well CBQ satisfies the link sharing goals in presence of various traffic mixes and CBQ configurations. Section 6 attempts to characterise the CBQ performance in terms of link utilisation, forwarding ability, link sharing precision. Section 7 analyses some practical networking issues (fragmentation and interaction between CBQ and RSVP) related to the deployment of CBQ for providing integrated services. Section 8 summarises the results and presents directions for future work.

2 An Overview of CBQ

Class Based Queueing [4] is a scheduling mechanism that provides link sharing between agencies that are using the same physical link. This is an improvement over the use of dedicated pipes for each agency because link sharing guarantees that any *excess* bandwidth resulting from an agency that is not fully utilising its share is redistributed to the other agencies (according to their relative allocations), improving link utilisation. With the CBQ's hierarchical link sharing capabilities, each agency can assign its own bandwidth to different kinds of traffic allocating the right share to each one. In this case the advantages of the hierarchical link sharing become evident: the unused bandwidth of an agency's class is distributed first to its leaf classes instead of being shared with other agencies.

CBQ operation is based on the interaction between a *general scheduler* and a *link sharing scheduler*. The *general scheduler* guarantees the appropriate service to each leaf class, distributing the bandwidth according to their allocations. The *link sharing scheduler* distributes the excess bandwidth according to the link sharing structure.

The general scheduler can be anything ranging from simple Packet Round Robin (PRR) to the more sophisticated Weighted Round Robin (WRR). The link sharing scheduler is more complex, because it has to take into account the throughput of each class. The link sharing scheduler estimates the class' use of the output link bandwidth and marks a class as *underlimit* (if it is transmitting at a lower rate than its allocation), *at limit* (if its rate is equal to its allocation), or *overlimit* otherwise. The mechanism for deciding which leaf class is allowed to send is complex and involves checking the status of every class (leaf or not) in the hierarchy. The resulting overhead can be prohibitively high, therefore approximations of the link sharing scheduler have been proposed. These proposals do not guarantee to respect exactly the link sharing guidelines, but they provide definitely lower complexity in managing the CBQ parameters.

* F.Risso is with Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy (risso@polito.it). At the time of this work he was visiting University College London.

2.1 ALTQ/CBQ Implementation

Alternate Queueing ALTQ [5, 6] framework provides a range of queueing schemes for realizing resource sharing and quality of service in the BSD networking code. It is available for FreeBSD [7] and includes Weighted Fair Queueing [8] and CBQ schedulers, buffer management algorithms like RED [9] and RIO [10] for DiffServ [2] networks.

The ALTQ/CBQ implementation provides by default fixed allocation to each class so that when a class is not fully utilising its bandwidth the excess bandwidth cannot be redistributed to other classes and is simply wasted. To allow the redistribution among the link sharing hierarchy, the administrator explicitly specifies the `borrow` option for each class, indicating whether the class is allowed to "borrow" bandwidth from its parent.

The ALTQ/CBQ implementation uses WRR or PRR for general scheduler and a modified Top-level Link Sharing (instead of the Formal Link Sharing Guidelines) algorithm for the link sharing scheduler. The WRR scheduler computes its allocation so that a number of bytes equal to the number of classes times the maximum packet length (determined by the link layer MTU) can be transmitted in each round. This value is calculated assuming that all classes have the same share, therefore classes with higher allocations can send more than one packet each round. A class stops sending packets when it finishes its slot or when it becomes overlimit.

The Top-level Link Sharing scheduler allows one class to borrow only up to level N, where N is set by a heuristic; the higher N is, the greater the chance of the leaf class to borrow. When the parent's leaf class is also overlimit, a large value of N allows the class to borrow from higher level ancestors up to a level N. However the parameter N is unique in the CBQ scheduler that means that it cannot be customly defined for different branches but this keeps complexity low.

ALTQ modifies the original heuristic with these new rules:

1. if a packet arrives for a not-overlimit class, set N to the depth of the class
2. if N is i and a packet arrives for an overlimit class with an underlimit ancestor at a lower level than i (say j), then set N to j
3. at scheduling a packet, if there are no underlimit classes due to the current N level, increase N by 1 and then try to schedule again
4. if no packet can be sent, set N to the maximum level allowed in the system (32), so that next round the chances to send a packet are maximised.

In general a class can borrow only if its parent is underlimit or if it has an underlimit ancestor and this can lead to a non-work-conserving behaviour under certain conditions. Non-work-conserving service can be avoided with the `efficient` option, so that the first overlimit class encountered will be able to send a packet even if all its ancestors in the link sharing structure are overlimit.

3 Environment and Methodology

The experiments were carried out with FreeBSD [7] based PC routers[1] with the ALTQ kernel[2] and a kernel clock of 500 Hz. The machines ranged from an Intel Pentium 166MHz to AMD K6-350

[1] Running FreeBSD 2.2.8-RELEASE.

[2] ALTQ-1.1.3.

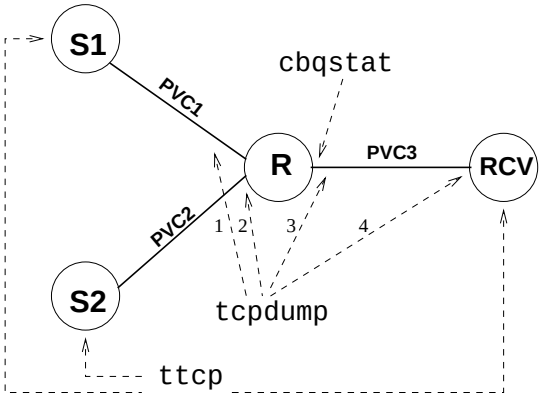

Figure 1: Network topology used in the experiments.

and were equipped with Adaptec ATM cards[3]. We used ATM PVCs (AAL5, LLC/SNAP encapsulation) with rate configured by software. The topology of our experiments is shown in Figure 1 and involved test with machines in the UCL local testbed, at the Essex University and at the NASA Goddard Space Flight Center (NASA-GSFC). We have tested all combinations of link MTU, bandwidth and delay for the network configurations shown in Tables 1, 2 and 3. The round trip delays on all PVCs were tested with `ping` (1.5 KBytes packet size).

We used `ttcp` [11] and `netperf` [12] for generating TCP and UDP traffic. TCP had the default maximum window size of 16 Kbytes. The traces of the flows were collected at the router input and output interfaces using `tcpdump` [13] and the CBQ accounting information given by the `cbqstat` utility [6].

The experiments were repeated several times in order to ensure the statistical validity of the results and average values are presented where appropriate. The experiments were performed in a completely controlled environment with no other traffic present on the links and the results were almost identical, therefore the figures show typical traces. Data were analysed off line after the end of each experiment to obtain "bytes transferred-over-time" and "throughput-over-time" graphs.

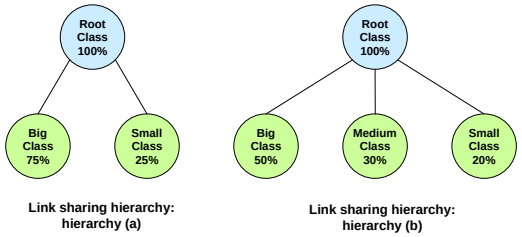

Figure 2: Link sharing hierarchies used in the experiments.

4 CBQ and network driver issues

This section presents some ALTQ implementation details and their effect on CBQ operations. Figure 3 shows how ALTQ/CBQ fits into the networking part of the BSD code.

[3] Adaptec ANA-59x0. The router had two ATM cards because of a known limitation of this version of ALTQ that requires a single PVC on the card where ALTQ/CBQ is running.

Table 1: Link MTU (Kbytes)

	PVC1	PVC2	PVC3
test-a	9.18	9.18	9.18
test-b	1.5	1.5	9.18
test-c	1.5	1.5	1.5

Table 2: Link Bandwidth (Mbit/sec)

PVC1	PVC2	PVC3
3	3	2
45	45	10

Table 3: Round Trip Delays (msec)

PVC1	PVC2	PVC3	Location
9	9	9	UCL local testbed
9	9	25	UCL - ESSEX
9	9	94	UCL - NASA

The packet processing is done by the classic BSD networking routines (`if_input()`, `ip_input()`, `ip_forward()`, `ip_output()`, `if_output()`) [14] up to the point where the packet has to be treated by the interface specific output routine (`atm_output()` for our experiments).

When CBQ is applied to an output interface, the packet does not follow the standard processing path (from the `if_output()` directly to the interface card), but it is examined by the classifier and is enqueued in the appropriate CBQ queue (`cbq_enqueue()`). From this point the processing is driver-specific: for example Figure 4 shows the sequence of the function calls for ATM output processing.

The following operations are specific to the ATM driver used in our experiments. The function that actually sends a frame out (`en_start()`) calls `cbq_dequeue()` which selects the first non-empty queue that must receive service (according to the current scheduling discipline) and dequeues the packet at its head. The ATM driver first "places" the packet in one of its software buffers[4], then the packet is transferred (in DMA mode, if possible) to the ATM card memory (`en_txdma()`) and then transmitted on the physical link (as shown in Figure 3) on a FIFO basis according to the contents of the hardware buffer. The ALTQ ATM driver sets the software buffer to 20 KBytes and the hardware one to 32 KBytes.

4.1 The ATM driver output function

The ATM driver used in these experiments is a clone of the original BSD ATM driver [15] appropriately modified to support ALTQ.

In the original ALTQ ATM driver code the `en_start()` routine (that is used for dequeuing packets from the CBQ buffers) *loops as long as there are packets waiting in the CBQ queues and there is enough space in the ATM software buffer*. This routine is also called when an ATM receiver interrupt has been serviced (i.e. when a new packet arrives at the router). This behaviour has undesirable side effects on CBQ operation. In fact, the `en_start()` routine runs at higher priority level (`splimp`) than other kernel code preventing

[4]We use the term *software buffer*, although this is not a "proper" buffer: when a packet needs to be transferred (DMAd) to the ATM card, there must be no more than N `mbuf` bytes already waiting to be transferred. This value N is called "buffer size".

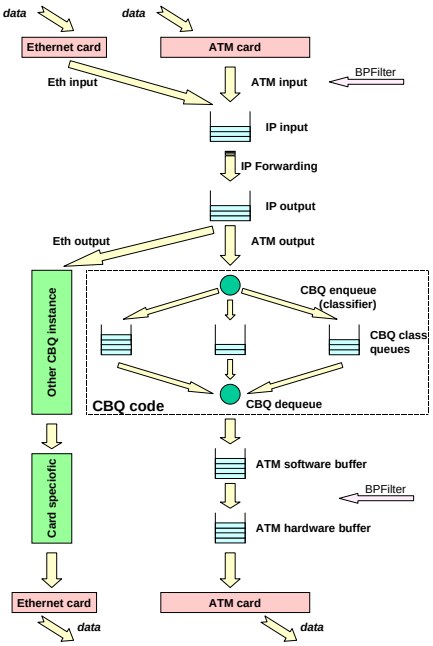

Figure 3: Packet processing in ALTQ.

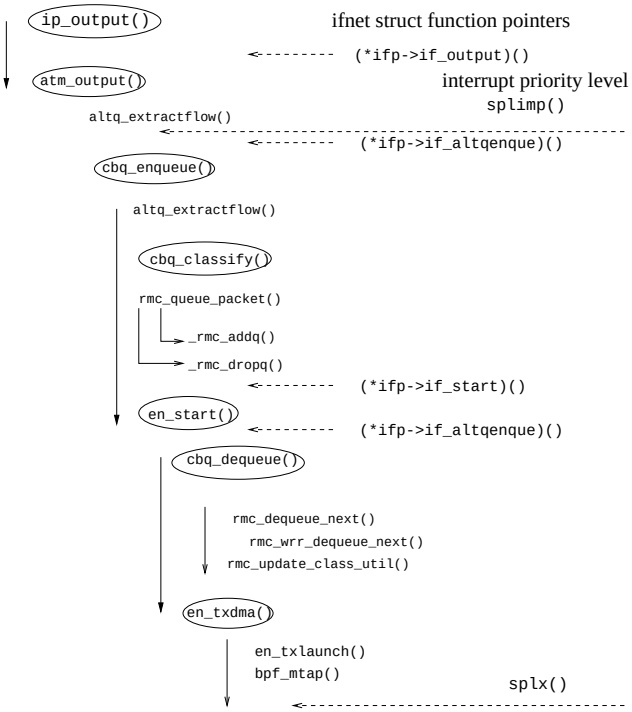

Figure 4: Functions calls for packet output.

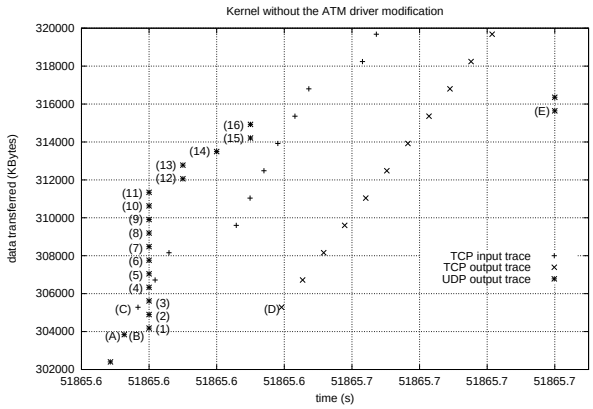

Figure 5: CBQ Link output pattern with the unmodified ATM driver.

the kernel from being able to refill the CBQ queues with new packets when the ATM driver is executed.

We modified the ATM driver by removing both the loop in the `en_start()` and the call to this function upon new packet arrival.

The comparison between Figure 5 (the original driver) and Figure 6 (the modified one) shows that the modification improves CBQ behaviour significantly. The output pattern in the first graph is burstier and more important the link-sharing guidelines are not respected. In fact, Figure 5 shows that TCP packet (C) enters the router when the TCP queue is empty (previous packet, (A), has already been forwarded, point (B)). Instead of serving packet (C), the original driver shows a burst of 16 UDP packets (sent from the UDP queue) followed by a long period where only TCP packets are being served, until point (E) where a UDP packet is sent again.

With the modified version of the ATM driver the service patterns on the output link are clearly improved as is evident from Figure 6, achieving correct link sharing at finer granularity (smaller time scale) and better approximation of fluid flow behaviour. There are no more long bursts of UDP packets: between the TCP arrival (point (C)) and its retransmission (point (L)) there are only 6 UDP packets to be served (points (D) to (I)).

Unless otherwise specified, all the experiments presented in this paper were done with this slightly modified version of the BSD ATM driver.

4.2 The effect of the ATM output buffer

The two ATM output buffers affect the operation of the CBQ mechanism and have the main responsibility for undesired delays in the forwarding process since the delays incurred in other kernel routines (i.e. `ip_input()`, `ip_output()`) are negligible.

Figures 6 and 7 show the link output pattern when the ATM software buffer (the number of `mbuf` bytes waiting to be transmitted on the PVC) has its default value (20 Kbytes), and when it is reduced to 2 KBytes. The CBQ class configuration is the one in Figure 2a, with one TCP flow in the "Big Class" and one UDP in the "Small Class" and all the PVCs configured with an MTU of 1500 bytes. The packet traces were obtained using `tcpdump` [13], and therefore do not account for the time spent in the ATM hardware buffer because `bpf` [16] marks a packet as transmitted when it leaves the ATM software buffer (Figures 3,4).

In Figures 6 and 7 the TCP class queue (at the time corresponding to point (C)) is indeed empty, because the last TCP packet received at the input interface (point (A)) has already been transmitted

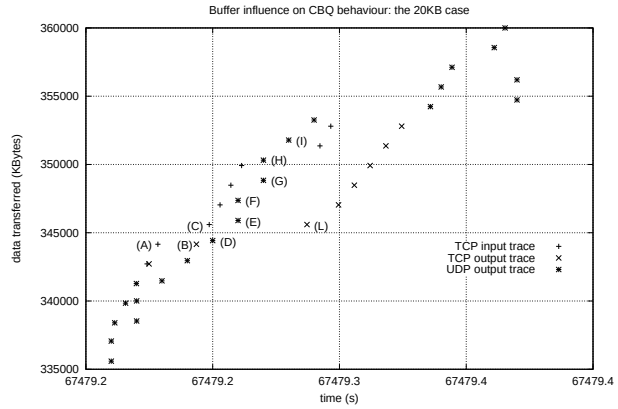

Figure 6: CBQ Link output pattern, modified ATM driver, ATM output buffer 20Kbytes.

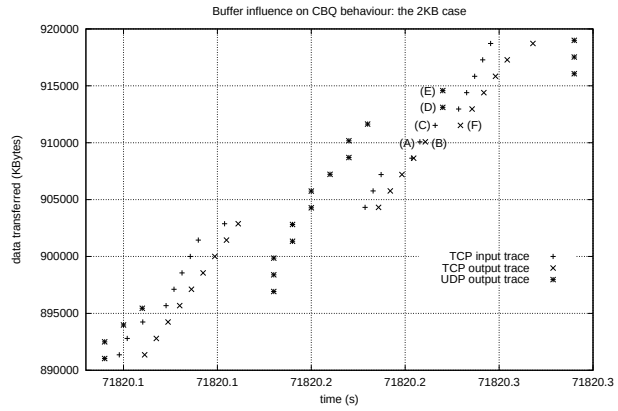

Figure 7: CBQ Link output pattern, modified ATM driver, ATM output buffer 2Kbytes.

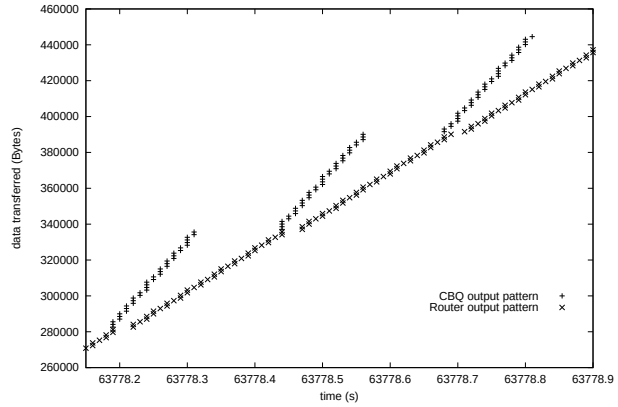

Figure 8: Scheduler and Router output pattern.

(at point (B)). The TCP class is also eligible for service because it has not been serviced recently; before point (A) mainly UDP packets were being serviced so that the TCP class cannot be overlimit. In these conditions, the CBQ scheduler serves immediately the TCP packet, moving it to the software buffer. If this has a non-negligible size (for example when it is 20 KBytes) there can still be a significant number of UDP packets waiting in there. The output pattern (captured by `tcpdump`) shows that even if the TCP class queue is empty when packet (C) arrives, a fairly large number of UDP packets (points (D) to (I)) is transmitted on the output link before point (L), when the TCP packet is eventually transmitted.

From the trace of the 2 KBytes ATM output buffer it can be seen that no more than two UDP packets (points (D) and (E)) are transmitted between the arrival of a TCP packet in an empty queue and its transmission. In fact, the additional level of buffering inserted by the ATM output buffers can be seen equivalent to that of a 2000 Km T3 (45 Mbps) link, that can have a non-negligible impact on performance.

Another effect of the per PVC output buffer is that *the CBQ scheduler appears to be dequeuing packets at a higher speed than the output link bandwidth assigned to the root class*. The trace in Figure 8 shows the same flow captured at the router after the CBQ scheduler and on the machine at the other end of the CBQ link (PVC3): the CBQ scheduler tends to serve packets faster than the output link capacity in short time intervals. Since the link sharing scheduler marks a class as overlimit by comparing the expected packet completion time and the actual packet completion time (where the latter is calculated according to the time a packet leaves the CBQ scheduler), the class is wrongly marked as overlimit and is being regulated. This will be discussed in detail in Section 6.1.

5 Link Sharing Goals

The key issue for CBQ is to allocate to each class its nominal bandwidth. In this section we examine to what extent ALTQ/CBQ is able to achieve its link sharing goals, especially when it allows borrowing between the classes. We have performed tests with TCP and UDP classes, identified cases where the results of bandwidth sharing were in fact different from those expected and showed how these undesirable effects can be avoided.

5.1 UDP Classes

In the first experimental scenario the CBQ router is configured with the link sharing hierarchy shown in Figure 2b and the source machines are generating three UDP flows (one for each class). In this case all classes have permanent backlog.

The leftmost part of Figure 9 shows CBQ behaviour when the leaf classes are *not allowed to borrow* from the root class. A leaf class can send until it has consumed its round-robin allocation or it has become overlimit, then the scheduler starts serving the next leaf class. The link sharing goal is thereby achieved by the cooperation of the link sharing scheduler and the general scheduler.

When *borrowing is allowed* (in the rightmost part of Figure 9) a class can become overlimit and still continue to send packets. When it finishes its WRR allocation the scheduler *checks if other classes have packets in queue and are allowed to send* (i.e. they are not suspended). Since all classes have *enough* backlog, the link sharing rules are guaranteed by the Weighted Round Robin mechanism. In both cases (borrow activated or not) the link sharing hierarchy rules are perfectly respected.

However the graph in Figure 9 shows that there is not *precise rate control*: the bandwidth allocations for the three classes are not

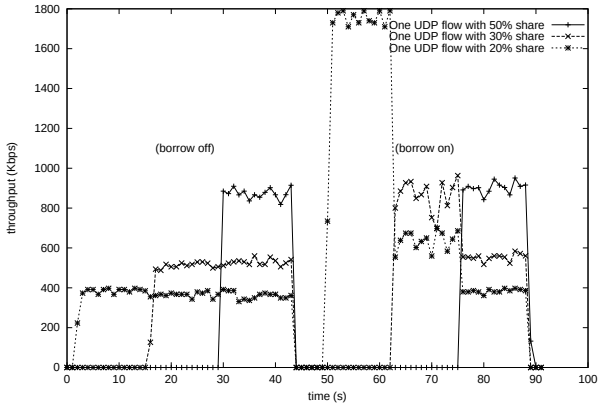

Figure 9: Link sharing among UDP flows with sufficient backlog.

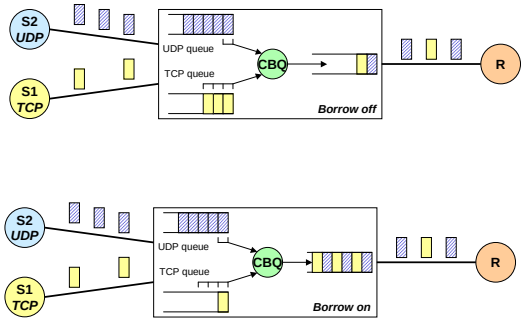

Figure 10: Different distribution of TCP packets when enabling borrow.

exactly respected according to the link sharing structure and precise bandwidth control cannot be achieved. Moreover, especially when borrowing is enabled, a class cannot use the entire bandwidth of its parent even if there is no competition by other classes; this will be better explained in Section 6.1.

5.2 TCP and UDP Classes

The second experimental scenario uses the link sharing hierarchy in Figure 2a with one TCP flow and one UDP flow per class, allocating them 75% and 25% of the output link bandwidth. The results in Figure 11 show that the link sharing rules are respected when there is no borrowing but they are less straightforward when borrowing is enabled (Figures 12 and 13).

When the leaf classes are not allowed to borrow, they cannot use more than their bandwidth allocation and whenever they exceed this value they quickly become suspended. This prevents a large number of packets from accumulating in the ATM output buffers.

When borrowing is allowed CBQ tends to serve packets at a higher speed than the output link bandwidth when examined in sufficiently small time intervals (Figure 8) because of the presence of the ATM output buffer.

The first effect of these buffers is that their size can cause non-negligible queueing delay and have serious impact on the perfor-

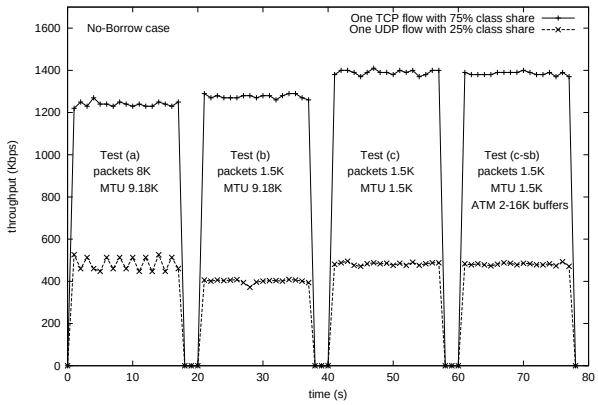

Figure 11: TCP and UDP classes, one flow per class: no borrow.

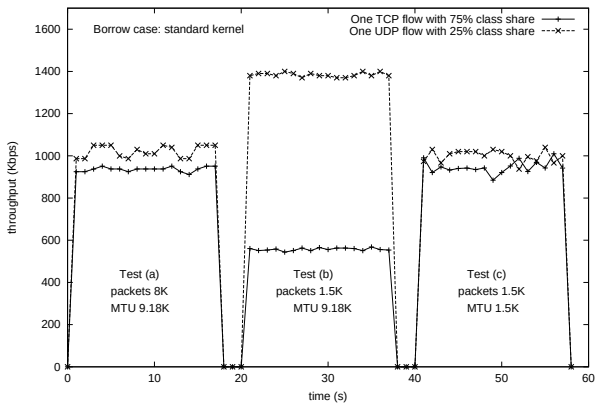

Figure 12: TCP and UDP classes, one flow per class: borrow.

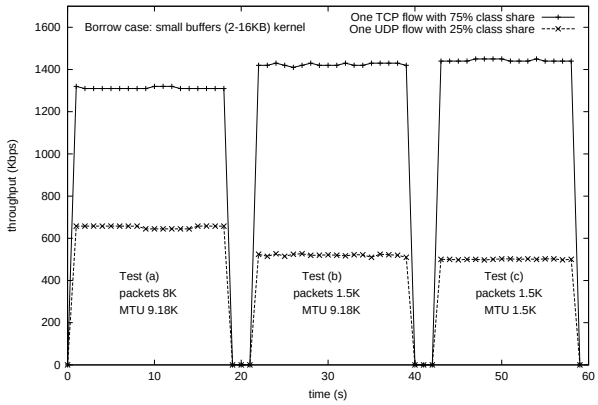

Figure 13: TCP and UDP classes, one flow per class: borrow with small ATM output buffer.

mance of transport protocols like TCP especially in the case when they account for a large fraction of the end-to-end bandwidth delay product. TCP's throughput[5] is affected by the Round Trip Time (RTT) which in turn depends on the queueing, transmission and propagation delay of the links in the end-to-end path.

The second effect is the different distribution of packets in the CBQ and ATM queues with and without borrow. This is shown in Figure 10 assuming that the TCP connection can have a maximum of seven packets "in flight". When borrowing is not allowed the TCP connection is able to create enough backlog in its CBQ class queue. On the other hand when borrow is allowed the packets of the TCP connection are mainly queued at the ATM output buffer. *The WRR mechanism is not able to fulfill the class allocation because its queue has not enough backlog*, therefore the class cannot fully exploit its allocated bandwidth.

Figure 10 shows that the WRR mechanism is not able to fulfill its allocation for the TCP class (three TCP and one UDP packet each round) and the TCP connection is penalised. This is confirmed in Figure 12-test (b): the class WRR allocation is large because it is computed taking into account the large MTU (9.18 KBytes) of the CBQ link (PVC3), and the TCP throughput is the worst of all three tests.

Reducing the ATM output buffers clearly improves the performance of the TCP flow. Figure 13 shows the same tests as Figure 12 when the ATM driver has been built with smaller size output buffers: the "software" buffer reduced to 2 KBytes and the hardware one to 16 KBytes. In all tests the competing flows adhere to the link sharing rules; the lower TCP throughput observed in test (a) is a known problem of TCP with 16 Kbytes limited maximum window size and large MSS due to the large (9.18 Kbytes) ATM MTU which reduces TCP essentially to a "stop-and-wait" protocol [17].

5.3 TCP Classes

The third experimental scenario uses the class hierarchy in Figure 2b with one TCP flow per class and the results are shown in Figure 14. The class bandwidth allocations are observed only when borrowing is not allowed; otherwise the TCP flows share equally the bandwidth between them. The main buffering point in this case is the ATM output buffer and not the CBQ class queues; the situation is exactly the same as the one described in the previous section. The ATM output buffers first increase the connection RTT, then provide insufficient backlog (Figure 10) in the input queues so that the WRR mechanism is unable to differentiate the service among the classes (i.e. the bigger classes cannot fully exploit their WRR allocation). In fact, the link share was respected when the test was repeated with the small buffer kernel configuration (2+16 KBytes).

5.4 TCP and Link Sharing with Borrow

The lesson learnt from the above experiments was that CBQ with borrow is able to provide the correct share among different classes when all the classes have *adequate* backlog. Obviously, having a smaller ATM output buffer helps significantly since a large ATM output buffer can become the main queueing point in the system defeating the link sharing rules. The presence of the driver output buffers affects especially those flows with small bandwidth-delay product (like the experiments in the local testbed because of the small delays), because it inserts a non-negligible delay on the end to end path.

Figure 15 shows results from the experiments in the wide area (UCL-NASA). The standard buffer case shows that the TCP/UDP

[5]The maximum throughput for a TCP connection is bounded by $\frac{Maximum_window}{RTT}$.

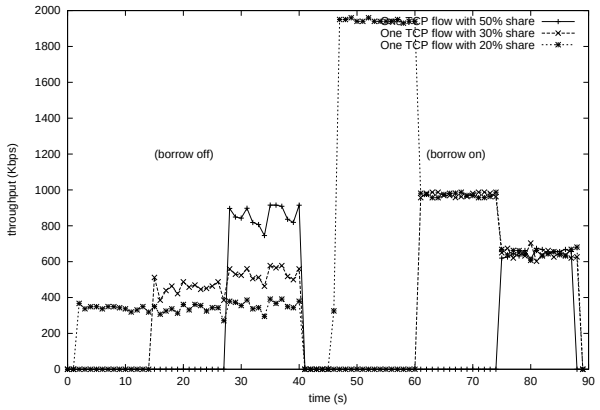

Figure 14: One TCP flow per class: behaviour with and without borrow.

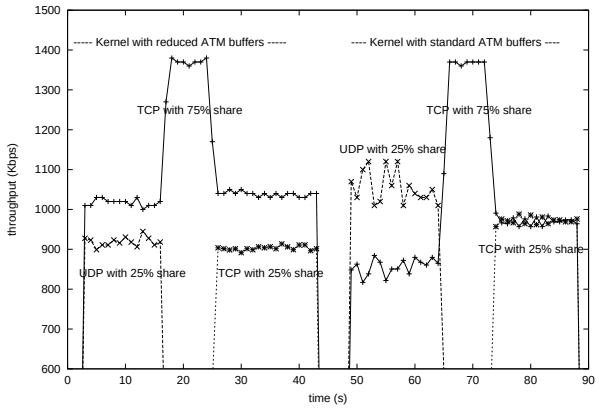

Figure 15: Long distance path: performance with reduced and standard ATM buffers.

ratio is worse than the local tests, while in the TCP/TCP test both flows share the bandwidth equally between them. The small buffer case shows that the situation has improved; the TCP in the "big" class gets more service than the flow in the "small" class but the result is still not as expected. The TCP flow, that in the local area obtains the correct service (in presence of small buffers) is no longer able to do this because of its inability to provide enough backlog due to the high bandwidth-delay product of this path.

The best way to achieve correct link sharing with TCP flows is to avoid the limitations of the maximum TCP window size (using the appropriate socket option `SO_SNDBUF, SO_RECVBUF`) or to ensure sufficient degree of flow aggregation in each class. Both methods aim to increase the maximum number of packets "in flight", by creating enough backlog in the CBQ class queues (as opposed to the link layer buffers) and thus making WRR differentiation possible.

Figure 16a shows the CBQ behaviour with 5 TCP connections in each class when borrowing is permitted and in this case all the classes get the right share. Figure 16b shows the corresponding queue length variation over time: there is now enough backlog that permits the WRR mechanism to fully exploit the allotment of each class.

When using RED at the CBQ class queues it is important that its `min_thresh` parameter is chosen in such a way that the queue length (backlog) in bytes is not smaller than the WRR allocation for that class. Otherwise the WRR may not be able to send the appropriate amount of data in each round to sustain the class' link share. ALTQ/CBQ allows RED drop policy in its queues but it does not allow configuration of the RED parameters simultaneously with CBQ. However when the experiments involving TCP were repeated with RED, they did not show any change in the class share.

Another cause of the CBQ delay is the WRR scheduler itself, because the service time depends mainly on the number of classes configured into the system: the more classes there are, the larger the delay "guaranteed" to each class. Therefore an arriving packet can be forced to wait longer before being serviced despite the allocation, increasing the RTT and affecting performance.

6 Performance

This section evaluates ALTQ/CBQ performance in terms of link utilisation, maximum obtainable throughput and sensitivity to the average packet size within a class.

Tests in this section are done with another modification of the ALTQ standard kernel. In fact, ALTQ/CBQ implementation maintains two global variables (Figure 17):

- `ifd->now[qo_]` is the *actual finish time* of the last packet in the head of queue `qo_` (i.e when the CBQ scheduler dequeues the packet and passes it to the ATM driver).
- `ifd->ifnow_` is the *expected finish time* for a packet in the head of queue `qo_` according to the output link speed.

ALTQ uses the second one to compute the *wake up time* of a class that has been suspended, while other CBQ implementations (ns-2 simulator, for example) and the CBQ original paper itself suggest to use the first one. We used `ifd->now[qo_]` instead of `ifd->ifnow_` to compute the suspension time for a class that is overlimit. All tests in this section (unless otherwise specified) were done with this modification; moreover we show the improvement of the modified kernel over the original one.

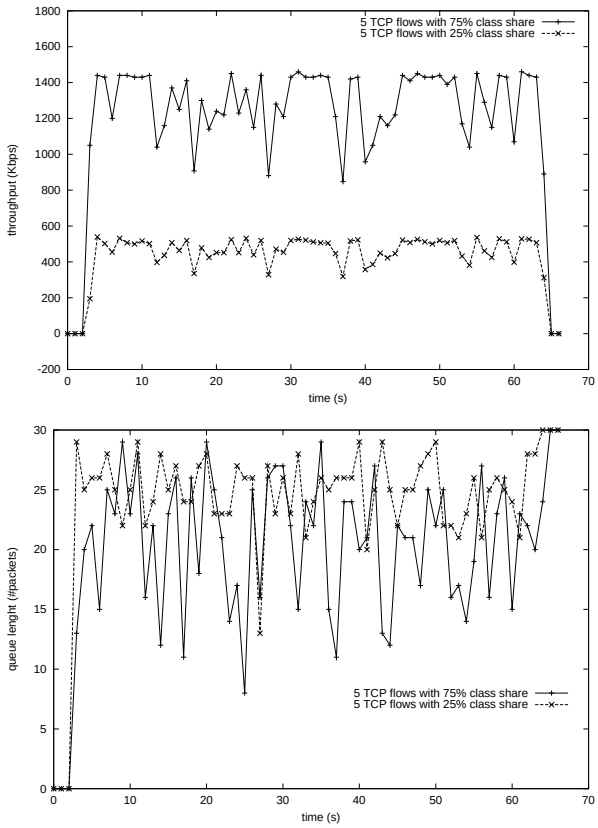

Figure 16: Five TCP flows per class: class shares are respected even with borrow.

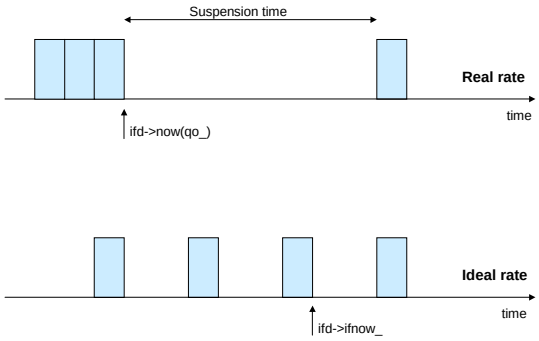

Figure 17: CBQ: actual and expected finishing time.

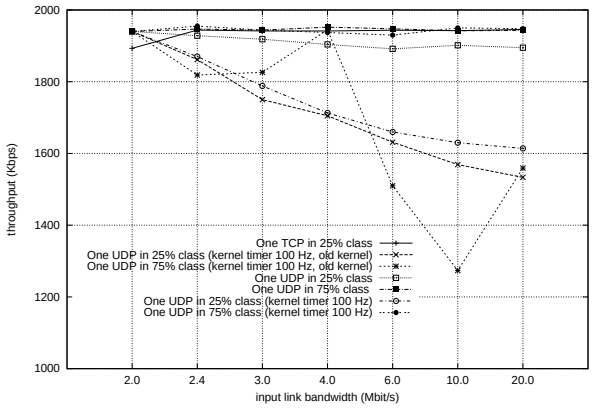

Figure 18: Maximum throughput obtainable by UDP and TCP within different classes.

6.1 Link Utilisation

As was pointed out in Section 5.1, CBQ is unable to guarantee the full utilisation of the output link even when the borrowing is allowed.

Lets assume to have a single UDP flow and that the output rate is smaller than the input rate. If there is enough space available in the output buffers CBQ dequeues a packet moving it into the software buffer. For a transitive period CBQ is able to dequeue packets at a higher rate than its output link bandwidth.

In such a situation, the gap between the variables `ifd->now[qo_]` and `ifd->ifnow_` (Section 6) increases progressively: the former increases according to the input link speed (a packet is served as soon as it arrives at the router), the latter increases according to the "true" output link speed. When the gap between them exceeds a certain threshold, ALTQ/CBQ resets `ifd->ifnow_` to the `ifd->now[qo_]` value (i.e the "real" system time), in order to prevent the gap from becoming too big. This does not come for free: becoming `ifd->ifnow_` smaller and being this a global variable, *it seems that the last packet left the CBQ scheduler very quickly, even faster than the rate allocated to the parent class*. This makes the parent class to appear to be overlimit and the leaf class gets suspended. It is important to notice that this does not mean that the router stops sending packets, because the buffers are able to sustain the output link until they are emptied.

The *class suspension time* depends on the class bandwidth share: the smallest the class allocation, the longest the suspension time. If the bandwidth share is small enough, the buffers are emptied before the suspension is resumed: the output link becomes idle and the throughput drops. As the bandwidth mismatch between input and output link increases (i.e the faster the input link compared with the output one), the more frequent becomes the need to *adjust* the gap between `ifd->now[qo_]` and `ifd->ifnow_`. As a result, under some conditions it is not possible the full utilisation of the output link bandwidth.

Figure 18 shows that TCP flows are not affected by this problem. In fact the number of back-to-back packets sent by TCP is not large (limited by its window) so it is unusual that a TCP flow gets suspended. UDP flows are affected by this problem especially if they belong to a class with relatively small share and the kernel timer is set to a small value (Section 6.1.1).

ALTQ implements the `efficient` option (settable in the configuration file) to overcome this problem. This option makes the scheduler *work conserving* so that it is able to send a packet from the first overlimit (and allowed to borrow) class it encounters even if all classes of the link sharing structure are overlimit. While the `efficient` option can be used to increase link utilisation, it magnifies the effect of the ATM output buffers because it keeps them full most of the time. In fact, the same tests of Section 5.2 repeated with the `efficient` option show that TCP throughput deteriorates even further.

The problem of poor link utilisation becomes less evident when the ATM output buffers are smaller (for example $2 + 16$ KBytes). In this case even UDP is usually able to use the entire class bandwidth independent from the class allocation. Obviously, the "efficient" flag seems to be useless in this case.

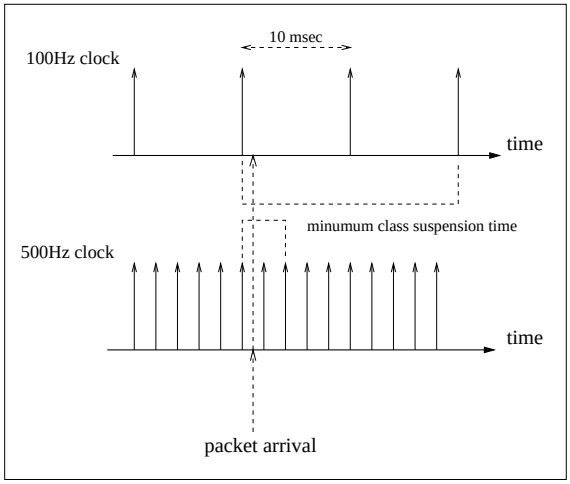

Figure 19: The effect of kernel timer on class suspension time (minimum 2 clock ticks).

6.1.1 The effect of the kernel timer

The kernel is driven by an internal clock that interrupts at regular intervals and each interrupt is called a *tick* (Figure 19).

The clock rate is programmable and is set at system startup time in the *hz* global variable. The default value for the clock rate is 100 Hertz; a smaller value may cause degradation in system response time but too high a value may cause too much system overhead, therefore it should be as high as possible avoiding excessive overhead.

When a class becomes suspended, its "wake up" time is kept in the `undertime` variable and an appropriate timeout, calculated in clock *ticks*, is set. As shown in Figure 19, the wake up time is set to a minimum of two clock ticks that for a standard 100 Hz clock means a minimum suspension time between 10 and 20 msec. A class is usually resumed when a new packet has arrived at the router (provided that its suspension time has finished) so that the timeout is the last resort for resuming the class when other mechanisms are not effective.

The precision of the kernel timer is important especially when the input packet rate is low (for example because of the big packet size), because it affects the ability to resume the suspended class at the right time. From this point of view, the highest the clock frequency, the best the precision obtained.

Figure 18 shows that the UDP flow into big class does not suffer from this problem because a class is suspended only a few times. Vice versa the UDP into small class has almost the ideal behaviour with the 500 Hz kernel timer, but its throughput decrease with the 100 Hz one.

Figure 18 shows also the comparison between the results obtained with the old kernel and the new one (modified suspension time computation), both with a 100 Hz timer: the improvement of the latter over the former is evident.

6.2 Forwarding ability

We attempt to stress the CBQ router to discover the overhead of CBQ specific per-packet operations (classify, enqueue, schedule, dequeue). For a given output link capacity (PVC3) we vary the offered load, number of configured classes and the packet size of a single UDP flow that we drive through the router.

Table 4: Maximum throughput for CBQ and FIFO schedulers.

IP packet size (bytes)	CBQ (pps)	FIFO (pps)
44	20831	24012
60	20662	23926
92	18691	21435
156	18333	20951
284	17615	19619

Table 5: Class Throughput for different number of configured classes

Number of Classes	Throughput (pkts/sec)
1	20513
5	20353
10	20245
20	19871
50	18718
100	15918

Table 4 shows that a router applying CBQ to an interface decreases its throughput approximately 10 to 14% compared to the same router with standard FIFO scheduling. This result was obtained when the router was forwarding one UDP flow and the CBQ was configured with a single class.

For a given packet size the router throughput is maximised for a certain input load. Figure 20 shows performance when changing the *offered load*: when the input load is smaller than its forwarding ability the router is able to manage all the traffic. When the offered load (packets/sec) increases beyond a certain level the throughput drops because the router is busy servicing interrupts from incoming packets and can only do limited packet forwarding.

Table 5 shows the impact of the number of classes (ranging from one up to 100) on CBQ router throughput. The test is performed when router is forwarding one UDP flow (IP packet size 60 bytes) and the offered load (input link capacity) was adjusted in order to maximise throughput. When increasing the number of classes, the CBQ router has to do an extensive search to determine which class each packet belongs to and, due to a non-optimised classifier, the CBQ router throughput drops significantly.

The effects are limited when CBQ has statically configured classes because they are usually less than a hundred[6]. However this can be a problem when CBQ is used as a traffic control module for RSVP [18] because in this case it allows arbitrarily small allocations (the minimum bandwidth per class does no longer exist) and a large number of classes can be created.

6.3 Packet size sensitivity

CBQ operation is highly sensitive to the packet size and link layer MTU. Since WRR uses the output link layer MTU to calculate each class allocation, when the average packet size is significantly smaller than the link layer MTU each class is allowed to send more than one packet each round. This can cause increased *burstiness* that has undesirable effects on the following routers along the path.

[6]Present ALTQ implementation permits to have more than 100 static classes, but the share must be an integer value. More than 100 classes means that someone has less than 1% share, and the CBQ does not give them service unless the borrow flag is activated.

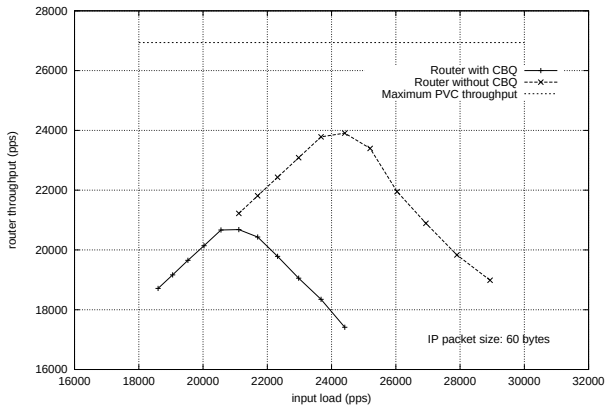

Figure 20: Throughput with different input capacity.

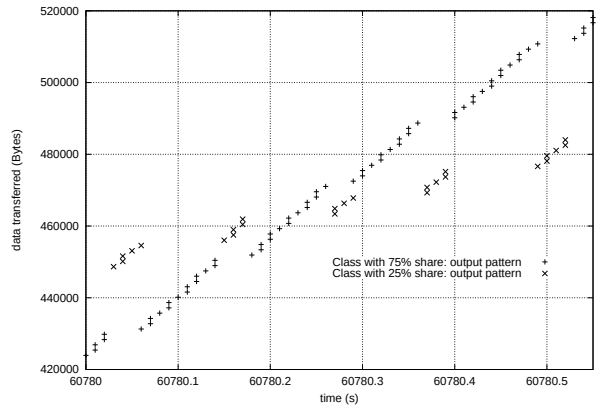

Figure 21: Bursts on the output link.

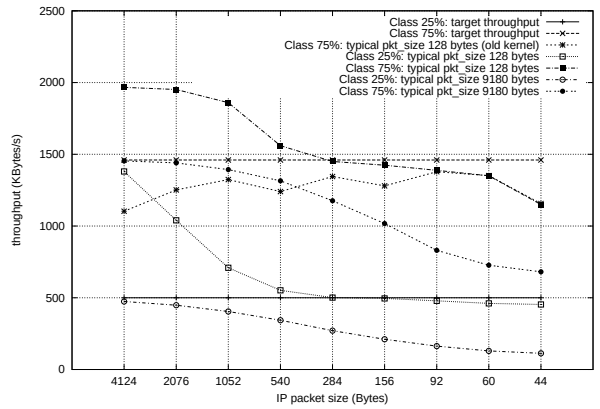

Figure 22: Throughput with different packet sizes.

Figure 21 shows the traffic pattern on the output link (class hierarchy of Figure 2a) when the MTU is 9.18 KBytes and the packet size is 1.5 KBytes (configuration (b) in Table 1): even the class with the smallest allocation can send more than one packet each round. This problem is hardly avoidable because the WRR cannot use other parameters to compute its allocation.

Another problem is that the CBQ scheduler calculates several internal parameters according to an *average packet size* specific for each class, that, if not specified into the configuration file, is considered equal to the output link MTU. In presence of large *configured - real* average packet size mismatch, the class throughput is significantly different from the expected value.

Figure 22 shows the throughput of a class (hierarchy of Figure 2a, borrow disabled) when sending only one UDP flow at a time, repeated with different packet sizes. The throughput is less than the expected value when the average packet size is smaller than the configured one (tests with 9180 Bytes); vice versa throughput increases when the average packet size is bigger than the imposed one (128 Bytes). Figure 23 shows a packet trace from the output link when the class sends packets much smaller than the average size; after it has transmitted a certain number of packets it becomes suspended by the link sharing scheduler and the output link becomes idle. Figure 22 also shows that the throughput obtained by the old kernel (original `undertime` computation) is clearly less predictable compared to the new one.

When the average packet size is far bigger than the expected value (in Figure 22 we consider a packet of 540 Bytes when the average is set to 128) the throughput increases unexpectedly. In fact, the parameter `minidle` sets a lower bound to `avgidle` preventing it from becoming too small, that usually happens when the class has big packets compared to the average value. In this case, when the class is resumed after a suspension CBQ "forgets" how much this class has sent in the past and the class can now send a lot more than its limit.

A partial improvement could be setting the `minburst` parameter (i.e. the number of back to back packets allowed by the link sharing scheduler before forcing a suspension) to a higher value. This will increase the `offtime` parameter so that a class is allowed to send more packets, followed by a longer suspension time. In this way the throughput obtained by a class with larger packet sizes compared to the average value is closer to the expected one. The drawback is an increased burstiness of the output pattern.

Average packet size can be hard to tune because of the diversity in packet sizes for flows aggregated in the same class. For instance, even TCP traffic that uses most frequently large packets (e.g. HTTP downloads) does not have a typical average packet size because of the acknowledgement packets (usually 40 bytes) or interactive applications like telnet. This is particularly true for bidirectional "virtual links", for example those used in a Virtual Private Networks (the most likely environment for deploying CBQ), that have data and acknowledgement packets within the same class.

Average packet size sensitivity is a well-known problem in the CBQ scheduler [19] and depends on the computation of the `offtime` parameter. This was calculated in such a way of being able to send a maximum number of *packets* (of average packet size) back to back before class suspension. Since the main parameter that triggers a class suspension is the "number of packets" instead of the "number of bytes", classes with small packets are not able to use their link share while classes with big packets get more bandwidth than the expected value.

6.4 Bursty flows

CBQ has two parameters (`maxburst` and `minburst`), specific for each class, that are used to adapt the CBQ behaviour to bursty flows. They are used to calculate the maximum number of back to back packets a class is allowed to send before having a suspension.

The first one is used when the flow starts transmitting after a long idle period; the second is the one used when the class is continuously backlogged (steady state).

It can be easily proved that maximum throughput that a flow can obtain decreases when its `maxburst` and `minburst` param-

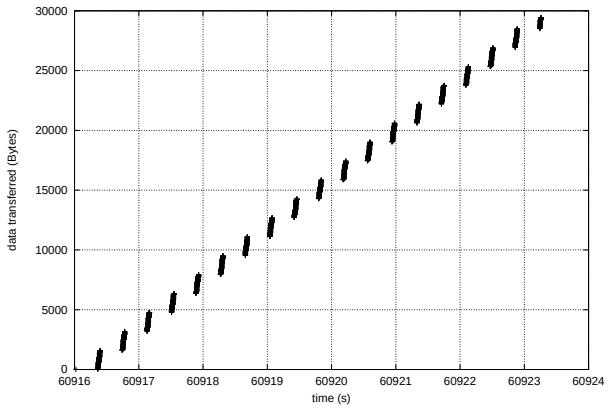

Figure 23: Output pattern with small packets: breaks in the pipe.

Table 6: Throughput for bursty flows.

Minburst (pkts)	Class 75% (Kb/s)	Class 25% (Kb/s)
2	1458	494
5	1436	484
10	1410	484
20	1400	483

eter increases. This is a problem related to the original CBQ proposal [4].

The result can be seen in Table 6; when the class is configured to support bursty flows, it is allowed to send more back to back packets but it is not able to reach its assigned share.

7 Practical issues in CBQ deployment

This section discusses some practical issues that concern CBQ deployment in real networks. There will be considered the effects of fragmentation on end-to-end performance and the interaction of the scheduler with the Resource reSerVation Protocol (RSVP) [18].

7.1 The effects of fragmentation

When a packet enters in the CBQ router, the classifier module checks its network and/or transport protocol headers and places it to the appropriate queue.

When a packet is fragmented (for instance due to a mismatch in link layer MTUs or to a UDP packet larger than the link layer MTU), the transport layer headers are missing from all fragments but the first one. The classifier may not be able to classify the packet appropriately and in this case the packet is put into the default class.

Fragmentation can prevent a flow from getting its allocated share because packets are inserted into the wrong CBQ queue. Fragmentation can also cause reordering problems (the first fragment of a packet can be served after the following ones because they belong to different queues) into the end system.

Fragmentation should generally be avoided, but when deploying advanced schedulers there are yet more reasons to do so.

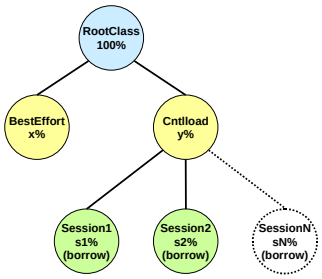

Figure 24: Link sharing configuration with RSVP.

7.2 RSVP integration

CBQ can be used as a traffic control module for RSVP in an Integrated Services Network. When the RSVP daemon accepts a new connection, CBQ creates dynamically a new class in its class hierarchy. The CBQ daemon starts automatically when the *rsvpd* is activated, loading the standard configuration file. Figure 24 shows a typical RSVP configuration.

The manual configuration requires the creation of the *Best Effort* and *Controlled Load* classes. When the RSVP daemon accepts a new reservation, the CBQ mechanism creates a new leaf class (under *Controlled Load* class) and assigns it the bandwidth indicated in the reservation message by the *token rate* parameter[7]. These leaf classes are by default allowed to borrow from their parent class.

RSVP is well integrated inside the CBQ kernel and it uses the classes in the same way as manually configured ones. Due to this high integration, it is not possible to perform a fine tuning of the CBQ parameters, particularly the packet size and priority. All the CBQ scheduler problems already presented can affect RSVP sessions as well. Moreover some of these problems that could be avoided with the manual class configuration (for example by specifying `packetsize`) cannot be avoided for the dynamically created RSVP classes and a class can get the wrong share due to a poor choice of the average packet size.

Deploying CBQ with RSVP in a network can lead to unpredictable results for end-to-end performance. For instance, if a reservation is carried out for a single TCP flow, the throughput might not be the one expected because of the limitations of the internal CBQ mechanism.

ALTQ/CBQ with RSVP can currently support only the Controlled-Load Service model since the Guaranteed Service is not supported by the current RSVP implementation [20].

8 Conclusions

This paper presents an evaluation of the ALTQ/CBQ implementation in terms of capabilities and performance. The CBQ operation is evaluated in-depth; some pathological behaviours are identified and corrected. For the cases that are ALTQ-specific we suggested and tested appropriate fixes demonstrating the improvement over the original implementation.

This is the case of the ATM driver architecture, the wrong computation of the suspension time and the link sharing that is not always respected (especially when the output buffers are not negligible compared to the size of the end-to-end pipe). For instance, the limitations on the link sharing goals are more likely to occur when borrowing is enabled, that is when the CBQ has major advantages over the use of dedicated links.

[7] Peak rate is not specified in the Controlled Load service.

Other aspects are pointed out without implementing fixes (for example the problem of the average packet size or the maximum throughput for bursty flows) because they are inherent to the CBQ algorithm and not a problem of the specific implementation.

Throughout the paper we make suggestions about CBQ deployment issues in a real network environment. These range from the possibly high delay in servicing a class (despite its share) due to the WRR scheduler, the need for enough backlog to insure the correct link share, the warning concerning use of RED in the CBQ queues, the precision of the rate control due to the kernel timer, the problems with throughput and burstiness due to the average packet size used in internal CBQ calculations. Finally, our experiments confirm that CBQ cannot be used as a mechanism for fine grained rate control.

In our future work we plan to investigate how effective CBQ is in a real network in terms of the Quality of Service parameters (delay, jitter, loss, throughput) and the way these parameters are affected by different configuration choices.

Acknowledgements

This work has been partially supported by CSELT (Centro Studi E Laboratori di Telecomunicazioni), Turin (Italy). We would like to thank Profs. Jon Crowcroft and Peter Kirstein that made this work possible with their help and support. Our thanks also to Kenjiro Cho, for answering our questions about the ALTQ internals.

References

[1] IETF Integrated Services Working Group (intserv). http://www.ietf.org/html.charters/intserv-charter.html.

[2] IETF Differentiated Services Working Group (diffserv). http://www.ietf.org/html.charters/diffserv-charter.html.

[3] Z. Wang. User-Share Differentiation (USD) Scalable bandwidth allocation for differentiated services. Internet Draft, Internet Engineering Task Force, December 1997. Work in progress available at http://www.ietf.org/internet-drafts/draft-wang-diff-serv-usd-00.txt.

[4] S. Floyd and V. Jacobson. Link-sharing and Resource Management Models for Packet Networks. *IEEE/ACM Transactions on Networking*, 3(4):365–386, August 1995.

[5] Kenjiro Cho. A Framework for Alternate Queueing: Towards Traffic Management by PC-UNIX based Routers. In *USENIX Annual Technical Conference*, New Orleans, Louisiana, 1998. available at ftp://ftp.csl.sony.co.jp/pub/kjc/papers/altq98.ps.gz.

[6] ALTQ: Alternate Queueing for FreeBSD. http://www.csl.sony.co.jp/person/kjc/software.html.

[7] The FreeBSD Project. http://www.freebsd.org/.

[8] A. Demers, S. Keshav, and S. Shenker. Analysis and Simulation of a Fair Queueing Algorithm. In *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 1–12, Austin, Texas, September 1989. ACM.

[9] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[10] David D. Clark and Wenjia Fang. Explicit Allocation of Best Effort Packet Delivery Service. *IEEE/ACM Transactions on Networking*, 6(4):362–373, August 1998.

[11] ttcp, Chesapeake Computer Consultants, Inc. available from http://www.ccci.com/tools/ttcp/.

[12] Netperf: Network Performance Measurement Tool. available from http://www.netperf.org/.

[13] V. Jacobson, C. Leres, and S. McCanne. tcpdump. Available from ftp://ftp.ee.lbl.gov/tcpdump.tar.Z.

[14] Gary R. Wright and W. Richard Stevens. *TCP/IP Illustrated Volume 2: The Implementation*. Addison-Wesley, Reading, Massachusetts, 1994.

[15] Charles D. Cranor. Integrating ATM networking into BSD. available from http://www.ccrc.wustl.edu/pub/chuck/psgz/bsdatm.ps.gz, August 1998.

[16] S. McCanne and V. Jacobson. The BSD packet filter: A New Architecture for User-level Packet Capture. In *Winter USENIX Technical Conference*, pages 259–269, San Diego, California, January 1993. Usenix.

[17] Kjersti Moldeklev and Per Gunningberg. How a Large ATM MTU causes Deadlocks in TCP Data Transfers. *IEEE/ACM Transactions on Networking*, 3(4), August 1995.

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. RSVP: a new resource ReSerVation protocol. *IEEE Network*, 7(5):8–18, September 1993.

[19] S. Floyd. Notes on CBQ and Guaranteed Service. available from http://www.aciri.org/floyd/papers/guaranteed.ps, July 1995.

[20] USC/ISI RSVP Implementation. Available from http://www.isi.edu/div7/rsvp/.