A CBQ-Based Dynamic Resource Allocation Mechanism for Diffserv Routers

Rares Serban[1], Chadi Barakat[1], Walid Dabbous[1]

[1]*Planète Project, INRIA Sophia Antipolis*
2004, Route des Lucioles, BP.93, 06902, Sophia Antipolis, France
`{Rares.Serban, Chadi.Barakat, Walid.Dabbous}@sophia.inria.fr`

Résumé :

Class Based Queueing (CBQ) is a link-sharing and resource management mechanism for packet networks. The weights of CBQ control the way with which the available bandwidth at the output interface of the router is distributed among the different classes of the input traffic. CBQ also disposes of rules for bandwidth borrowing based on a hierarchy of classes. In [1], we proposed a dynamic, self-tuning mechanism for allocating resources in Diffserv routers called DRAM (Dynamic Resource Allocation Mechanism). The routers in [1] are supposed to implement a simple version of CBQ equivalent to Weighted Fair Queuing (WFQ), where the bandwidth borrowing feature of CBQ is disabled. DRAM tunes the weights of WFQ so that each class of traffic realizes its needs, and the resources of the network are efficiently utilized. The tuning is automatic, self-configurable, easy to deploy and to manage, without any additional signaling. In this paper, we study the impact of CBQ mechanisms related to bandwidth borrowing on the performance of DRAM. We show that the simultaneous use of DRAM and CBQ leads to better overall performance due to the bandwidth borrowing capabilities of CBQ; Service Level Agreement (SLA) profiles are more respected and network resources are better utilized than when DRAM is used with WFQ. The performance improvements are validated by a campaign of experiments on a real network testbed.

MOTS-CLÉS : CBQ, Diffserv, WFQ, Service Level Agreement.

1. Introduction

Service Level Agreements (SLA) between providers and customers define Service Level Specifications (SLS) with traffic conditioning specification, monitoring service capabilities, service availability and the fees corresponding to each level. The Traffic Conditioning Specification (TCS) defines service level parameters (bandwidth, packet loss, peak rate, etc), offered traffic profiles and policies for excess traffic.

Various sets of QoS (Quality of Service) mechanisms and policies are tuned using the traffic conditioning specification. The heart of the QoS technology is a packet scheduling mechanism, also known as queuing mechanism. Sophisticated queuing mechanisms can provide performance bounds of bandwidth, delay, jitter and loss and thus, can meet the requirements of real-time services. Queuing mechanisms are also vital to IP services to avoid congestion and to provide fairness and protection, which leads to more stable and predictable network behavior. These mechanisms are responsible of providing bandwidth guarantee and traffic protection, and of distributing correctly the available bandwidth among the classes of services supported by the QoS framework.

Internet Service Providers (ISP) guarantee some level of service for their users. Given the SLA of a stream of packets, the network has to allocate enough resources so that the service required by that stream is guaranteed. The allocation can be done in different ways, depending on the total amount of resources available in the network and the number of customers. When the number of the customers approaches a maximum limit, the resources start to be rare and the Quality of Service (QoS) required by customers cannot be realized. Thus, some kind of Call Admission Control (CAC) [2] has to be implemented by the network, to protect already accepted customers from newly arriving ones.

The allocation can be done using many queuing disciplines. Two factors determine whether a packet scheduling (a queuing discipline) is able to support QoS in the Internet: its ability to provide bandwidth guarantee and to fulfil the delay requirement of the data flows. Providing guarantee for minimum bandwidth when there is a large demand is important to ensure that competing data flows do not degrade each other's throughput. Without proper bandwidth guarantee, non-responsive transport protocols as UDP finish up by consuming most of the bandwidth, and by shutting down responsive transport protocols as TCP.

Queuing disciplines are used in the two well known QoS architectures: Integrated Services and Differentiated Services. Intserv [3] is an IETF framework for guaranteed services and for controlling the load of traffic along a transmission path. This framework is based on the Resource reservation protocol (RSVP). RSVP itself is a signaling protocol to set up the traffic control modules of the routers based on SLA parameters. QoS is guaranteed since each router along the transmission path holds a resource reservation table able to handle flows. In IntServ, two classes of service are defined: guaranteed service and controlled-load service [4]. A flow is a stream of packets belonging to the same user/application (the 5-tuple criterion). The tuning of routers in IntServ is done by RSVP. Parameters of the packet scheduler in each traffic module are changed in a dynamic manner based on signaling properties of RSVP.

Differentiated Services (Diffserv) is an IETF framework for classifying traffic into classes, with different service levels for each class [5]. The edge routers in a Diffserv network mark/shape/police flows based on their SLAs and the core routers offer packets belonging to these flows different treatments using the marks they carry. Core routers handle aggregates of flows instead of individual flows, which is known to considerably reduce the complexity of Diffserv, compared to its counterpart IntServ [6], where core routers allocate resources on a per-flow basis. The treatment a core router gives to packets from one service class is called PHB (Per Hop Behavior). The PHB classes (or service classes) defined in Diffserv are: Best Effort (BE), Assured Forwarding (AF) and Expedited Forwarding (EF). EF packets are queued in separate buffer and are served before packets of the other classes. Packets of the different AF classes are queued in separate buffers and are served using a packet scheduler mechanism (e.g., CBQ, WFQ, WRR, CBWRR, etc.).

We focus in [1] on the tuning of core routers in a DiffServ network. Routers are supposed to implement a simple version of CBQ equivalent to Weighted Fair Queuing (WFQ), where the bandwidth borrowing feature of CBQ is disabled. The purpose of [1] is to dynamically tune the weights of WFQ, which are actually set in a static way by manual work based on a trial-and-error process. To this end, a dynamic, self-tuning mechanism is proposed. We call this mechanism DRAM (Dynamic Resource Allocation Mechanism). The weights of WFQ control the way with which the available bandwidth at the output interface of the core router is distributed among DiffServ classes. For simplicity reasons, only two AF classes are considered, in addition to the classical BE class. DRAM ensures that AF and BE classes are protected from each other, and that the excess of bandwidth, which is not allocated to high priority traffic is fairly shared among the different classes.

In this paper, we study the impact of CBQ mechanisms related to bandwidth borrowing on the performance of DRAM. We show that the simultaneous use of DRAM and CBQ leads to better overall performance due to the bandwidth borrowing capabilities of CBQ; SLA profiles are more respected and network resources are better utilized than when DRAM is used with WFQ. The improvement in performance when DRAM and CBQ are combined together is validated by a campaign of experiments on a real network testbed.

The remainder of this paper is organized as follows. A short presentation of CBQ is given in Section 2. Section 3 reminds the behavior of the Dynamic Resource Allocation Mechanism described in [1]. Section 4 details on the aim of this paper. Section 5 and Section 6 study the interaction between CBQ and DRAM based on the scenarios proposed in [1] and illustrate the gain in overall performance. Section 7 concludes the paper.

2. Class Based Queuing - CBQ

CBQ was proposed and studied in [7]. It is a packet-scheduling algorithm that supports hierarchical link sharing and preserves bandwidth guaranteed. CBQ presents a solution for unified scheduling for link-sharing and real-time purposes. It isolates real-time and best-effort traffic by allowing the separation of traffic in different traffic classes that are then treated according to their requirements, e.g. by giving priority to real-time traffic.

Link-sharing allows multiple organizations or multiple protocols to share the link bandwidth and to distribute available bandwidth according to the class tree structure. Each class has its own queue and is assigned its share of bandwidth. A child class can borrow bandwidth from its parent class as long as the available bandwidth is disposable. Although CBQ has the concept of using a general scheduler, which is used in the absence of congestion, and a link-sharing scheduler for rate-limiting classes, most implementations implement both mechanisms in a single scheduler.

CBQ works as follows: The classifier assigns arriving packets to the appropriate class. The estimator estimates the bandwidth recently used by a class. This is done by calculating an exponentially weighted moving average over the discrepancy between the actual inter-departure time of two packets of a class to the inter-departure time corresponding to the specified rate of the class. If a class has exceeded its predefined limit, the estimator marks the class as *overlimit*. The scheduler determines the next packet to be sent from the various classes based on priorities and states of the classes. CBQ has been implemented in Linux RedHat 7.2 using the algorithm skeleton from NS simulator. The original CBQ implementation is the classical WRR (Weighted Round Robin) but in Linux is DRR (Deficit Round Robing). The Top-Level link-sharing algorithm employs heuristics to control how far the scheduler needs to traverse the class tree. Implementation of this algorithm is complicated so CBQ Linux simplified implementation uses DRR independently of the borrow level form. This leads to sharing policies: when class A has higher priority than B, both are borrowing from C then A gets as much as it can and the remainder is served to B (but B still gets its own rate). If B has the same priority as A then borrowed bandwidth would be divided between A and B in the same proportion as their own base rates are.

3. Dynamic Resource Allocation Mechanism - DRAM

The tuning of core routers in a Diffserv network is actually done in a static way by manual work based on a trial-and-error process. A static tuning is time-consuming and costly for network manager. A static tuning may lead to an inefficient utilization of network resources and unfairness among the Diffserv classes. A dynamic tuning of core routers is needed to satisfy as much as possible all reservations of customers, and to fairly distribute the excess of bandwidth. To realize such objective, we proposed DRAM in [1] and we validate its performance with real experiments.

DRAM makes the following two assumptions. First, the network is over-provisioned, i.e. there are enough resources in the core of the network to support the high priority traffic marked at the edge. Second, the resources of the network may not be enough to support the total amount of traffic coming from the edge, i.e. the high priority plus the low priority. The decision on whether to accept a new SLA is done at the edge of the network, and core routers dynamically tune their parameters so as to absorb the marked traffic and to use efficiently the network resources.

We consider in [1] the allocation of bandwidth based on the average rate of high priority traffic of each DiffServ class. The interval over which the data rate is averaged is an important parameter of our mechanism. It must not be too small, since the system may become unstable, especially when we have a transport protocol like TCP, which adapts its rate as a function of the reaction of the network. A small averaging interval also results in high computational overhead. For simplicity of the analysis, the model we studied in [1] omits the EF service and focuses on AF and BE classes of services. We consider two-drop precedence per AF class. At the edge of the network, compliant packets of AF are marked with high priority (called IN packets), and non-compliant packets are injected into the network with low priority (called OUT packets). IN and OUT packets are buffered in the same queue in core routers, but are dropped differently at the onset congestion. The idea is to start dropping OUT packets while protecting IN packets. When all OUT packets are dropped and the congestion persists, IN packets start to be discarded. The mechanism proposed in [8] to support such a preferential dropping is called RIO (RED IN/OUT).

In summary, DRAM measures the rate of IN packets, and sets the weights of the CBQ buffer so as to absorb all IN packets and to distribute fairly the rest of the bandwidth (called excess) among OUT and BE packets. The excess bandwidth distribution rule forms important part of the mechanism. Here are two examples that have been studied: *fair division* - the excess bandwidth is distributed among classes using equal weights, *weighted division* - the excess bandwidth is distributed among classes using a priori weights. Note that the CBQ version used in [1] disables bandwidth borrowing. When one class transmits at less than its allocated rate, the remaining rate is distributed among the other classes proportionally to their allocated rates, without any control from the network manager.

4. CBQ as a basic component of DRAM

One of the components of a Dynamic Resource Allocation Mechanism is the packet-scheduling algorithm. With packet scheduling algorithm we control the interactions among traffic streams, different performance-oriented traffic classes and different administrative traffic classes. The interactions among traffic streams define rules to share the bandwidth of one output link. The traffic classes can share the total bandwidth using sharing rules combined with priority. The administrative rules are provided using a hierarchy of traffic classes. The first set of interactions can be controlled with weight parameter. The second one depends on the traffic design rules. Different types of traffic can share each one another the bandwidth in function of their requirements. The optimum solution is each class of streams to receive the minimum requirements in order to assure a good behavior of the total bandwidth traffic.

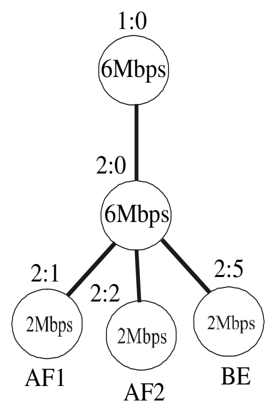
Class Based Queuing (CBQ) is a packet scheduler that is able to preserve the bandwidth allocated to the classes and to support hierarchical link sharing. However, it does not scale well with the number of data flows as it incurs additional delay with each new data flow admitted. Due to this additional delay, the admission control cannot predict the delay experienced by the existing flows after admitting a new flow. Although CBQ is able to provide lower delay to higher priority flows, it also introduces jitter to the traffic owning to its packet-scheduling artifact. As such, CBQ is not suitable for fine-grained scheduling of real-time traffic. It is more suited for providing bandwidth guarantee to aggregated data flows.

We consider that using DRAM with CBQ enhances link sharing and resource allocation management. Our Dynamic Resource Allocation Mechanism does not replace CBQ. It ensures that SLAs are respected and allows at the same time an efficient utilization of network resources. As we said in Section 1, our assumption is that a customer is satisfied if its IN packets get through the network without being dropped. We also assume that it is in the interest of customers and operator that the excess bandwidth in the network is fairly distributed among the classes: AF1, AF2 and BE. There are two levels of bandwidth distribution. The first one is defined by the link-sharing algorithm of CBQ and the second one is set by the network administrator. Given the variability of the traffic and the change of SLAs, it is very likely that a static tuning of core routers implementing CBQ does not realize the above objectives. The problem that can be caused by a static tuning of weights at the output interface of a CBQ core router, can be summarized as: bias against the IN packets of one or more AF traffic classes and unfairness in the distribution of the excess bandwidth. DRAM solves the problems caused by a static tuning of CBQ weights, and CBQ enhances DRAM with its mechanisms (class hierarchy and priority) for bandwidth sharing and borrowing.

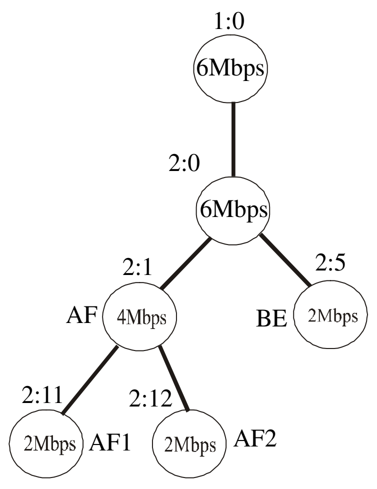
In Section 5, we study the problems caused by a static tuning of the CBQ core router using two hierarchies of classes: a flat level hierarchy and a multilevel hierarchy of classes. In Section 6, we introduce DRAM and we check if the link-sharing hierarchy of the classes mechanism of CBQ affects the performance of DRAM or not.

5. Studied CBQ class hierarchies

This section shows the behavior of CBQ using the two static bandwidth link-sharing configurations shown in Figure 1. We study the two configurations and the influence of the linux "bounded" parameter. The network configuration used is described in [1]. The 6Mbits/s link bandwidth is allocated to DiffServ classes like in Figure 1. The number of classes, the hierarchy of classes, and the efficiency of the packet filtering process affect the latency and the performance achieved by each class of traffic. In our case, the number of classes (of services) used is three: AF1, AF2 and BE.



a) simple configuration



b) hierarchical configuration

Figure 1. Two configurations of CBQ classes

Static tests help to discover the behavior of the CBQ implementation. The following tests verify the link sharing of the Linux CBQ implementation for a congested link by means of latency and throughput. The data source for class AF1 is a

UDP flow that sends 1470 bytes packets. The data source for class AF2 is a long-lived TCP flow. The data source for class BE is a UDP flow that sends 1470 bytes packets.

The first sharing test uses the simple configuration of CBQ from Figure 1a. The link bandwidth (C = 6Mbit/s) is shared by the three classes following the Top-Level link sharing rule described in [7]. Each class of traffic is allocated 1/3 of the link bandwidth. The root class 1:0 is used by dsmark classifier and is linked to the class 2:0. Class 2:0 represents the root class for CBQ configuration. Class 2:0 has three children classes AF1, AF2 and BE.

The test scenario is the following. First, all three data sources send at a rate 4Mbit/s. Each class rate of CBQ scheduler is provisioned to 2Mbit/s. Then, the source BE successively diminishes its rate to 2Mbit/s, 1Mbit/s, 600Kbit/s, 500Kbit/s and 100Kbit/s.

R_{AF1}[Mbit/s]	R_{AF2}[Mbit/s]	R_{BE}[Mbit/s]
2.1	2.0	1.9
2.3	2.0	1.0
2.4	2.2	0.6
2.5	2.0	0.5
3.4	2.1	0.1

Table 1. Test scenario for simple configuration of CBQ

R_{AF1}[Mbit/s]	R_{AF2}[Mbit/s]	R_{BE}[Mbit/s]
2.1	2.0	1.9
2.6	2.1	1.0
2.7	2.3	0.6
2.6	2.1	0.5
2.3	2.1	0.1

Table 2. Test scenario for simple configuration of CBQ using "bounded" parameter

Table 1 shows the behavior of link sharing between AF1, AF2 and BE classes for the class hierarchy of Figure 1a. Class AF1 being allocated the highest priority borrows the unused bandwidth of class BE. Class AF2 gets its allocated rate 2.0Mbit/s. We measure the delay through the AF1 class. The value of delay decreases when AF1 gets more bandwidth (i.e. from 0.856ms to 0.653ms).

The same test is resumed but this time using the "bounded" parameter of CBQ. The "bounded" parameter blocks the borrowing of bandwidth by a class from its parent class (i.e. it bounds the rate obtained by a class to its allocated rate). Table 2 shows the bandwidth allocation error where we see that classes AF1 and AF2 are still trying to borrow bandwidth from the BE class. Class AF1 has increased its rate from 2Mbit/s to the values indicated in the table. It is very hard to implement precisely the bandwidth borrowing mechanism of CBQ. We measure in this case the delay of AF1 packets. The value of the delay is approximately constant (i.e. 0.858ms).

The second sharing test uses the hierarchical configuration of CBQ from Figure 1b. The rates of sources and the types of traffic are the same as above. The total link bandwidth is shared by two classes with the same priority: AF and BE. As such, the root class 2:0 has two children AF and BE. AF class has two children AF1 and AF2. The root class 1:0 is used by dsmark classifier and is linked to the class 2:0. AF class is configured to share 2/3 of the total bandwidth (C=6Mbit/s) and class BE shares 1/3 of the link bandwidth. Each child of AF class shares 1/2 of class AF bandwidth. AF1 class has higher priority than AF2 class.

R_{AF1}[Mbit/s]	R_{AF2}[Mbit/s]	R_{BE}[Mbit/s]
2.2	1.9	1.9
2.6	1.9	1.0
2.9	2.0	0.6
3.0	2.4	0.5
3.4	2.5	0.1

Table 3. Test scenario for hierarchical configuration of CBQ

R_{AF1}[Mbit/s]	R_{AF2}[Mbit/s]	R_{BE}[Mbit/s]
2.1	2.0	1.9
2.3	2.1	1.0
2.4	2.3	0.6
2.6	2.1	0.5
2.7	2.2	0.1

Table 4. Test scenario for hierarchical configuration of CBQ using "bounded" parameter

Table 3 shows the behavior of the sharing bandwidth between classes and the case when the parameter "bounded" is not used. The data source BE decreases its rate and sets it to the values shown in the table. Clearly, class AF1 borrows most of the unused bandwidth of BE class since it has higher priority compared to AF2.

Table 4 shows the bandwidth allocation error when the parameter "bounded" is set to all classes of the hierarchical configuration of Figure 1b. Again, the AF classes are still borrowing bandwidth from the BE class even though there rates are bounded to their allocated rates.

The delay measured for AF1 class in the test scenario for hierarchical configuration without "bounded" parameter decreases when the AF1 class gets more bandwidth. This test has shown that CBQ is able to increase the quality of services for the classes with priority and it is able to control the transmission rate at the network interface.

The test scenario using hierarchical configuration and "bounded" parameter has shown that the delay on class AF1 has no constant values like in the test scenario with simple configuration of CBQ using "bounded" parameter.

6. Examples to compare link-sharing schemes

The goal of this section is to compare the link-sharing configurations from Figure 1 using the Dynamic Resource Allocation Mechanism presented in Section 3. This section proves that the behavior of our mechanism does not affect the bandwidth link sharing of CBQ. Moreover, the bandwidth link sharing of CBQ is improved when using DRAM since DRAM tunes the weights of CBQ as a function of the incoming traffic.

6.1. Link-sharing test if one reservation is not satisfied and one is satisfied

The network test configuration is illustrated and described in Figure 2. The hierarchy of classes defined at the output interface of the core router is illustrated in Figure 1. Consider the scenario where AF1 class generates a total rate of 4.5Mbit/s and AF2 class generates a total rate of 0.5Mbit/s. The rate of data carried by IN packets of class AF1 is equal to $R_{AF1.1}$=2.5Mbit/s. All packets of class AF2 are marked as IN, $R_{AF2.1}$=0.5Mbit/s. In the static tuning case, the buffer is supposed to distribute the bandwidth C = 6Mbit/s equally among the three classes. The BE traffic rate is set to 3Mbit/s (Figure 2).

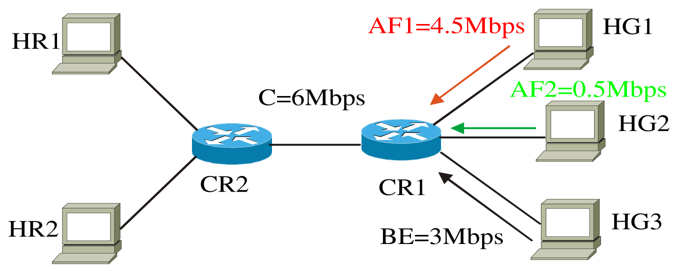


Figure 2. Test-bed network for link-sharing abilities if one reservation is not satisfied and one is satisfied. CR - core router; HR - host receiver; HG - host generator.

In this scenario, class AF2 is satisfied since all its IN packets get through the network. However, class AF1 is not satisfied, since the CBQ buffer limits its rate to 2Mbit/s, whereas it is sending IN packets at a higher rate of 2.5Mbit/s.. The total rate of IN packets generated by the two classes is less than C, therefore it is possible to find another tuning that allows all IN packets to get through, and corrects the bias against class AF1. When running our mechanism, the rate allocated to AF1 increases to more than $R_{AF1.1}$ and the rate allocated to AF2 is kept larger than $R_{AF2.1}$.

Figure 5 shows two cases. One is when the parameter "bounded" is not included in the simple configuration hierarchy. Because the traffic of AF2 is low and marked IN profile, the rest of the bandwidth is borrowed by the AF1 and BE classes. The second case is when we set the parameter "bounded" to block the sharing algorithm. In both cases, AF1 has the highest priority. We observe that the two AF classes always realize their desired rate. The BE service gets the smallest amount of bandwidth, since it has zero guarantee. But, the throughput of AF1 is different for the two presented cases. When the “bounded” parameter is off, the link-sharing algorithm of CBQ gives the bandwidth unused by AF2 to the AF1 class since the AF1 class has the highest priority. The "fair division" rule of the excess bandwidth is not strictly enforced when using the link-sharing algorithm with the “bounded” parameter set to off. If the "bounded" parameter is on, the excess bandwidth is shared using the "fair division" rule and is more controlled.

The same cases are presented in Figure 6 using the hierarchical configuration of CBQ. AF class has the 2/3 from the link bandwidth and the BE class 1/3. Inside the class AF, the weight of each sub-classes is 1/2. AF1 class has the highest priority and the AF2 class the lowest. AF and BE classes have the same priority.

Using this hierarchy we observe from Figure 6 (in the case without bounding) that the traffic of BE is more "nervous" because the Linux kernel is not a real-time kernel, hence it does not schedule its processes with different levels of priorities. It does not make sense for a process that transmits 2 Mbit/s to be given the same CPU time as the one that transmits only 200Kbit/s. We can observe this problem when we are using a complex hierarchy of traffic streams. In our case we use two levels of hierarchy, the first one with no priorities and the second one with priorities. This problem represents a lack of CBQ Linux implementation.

Figure 7 shows another case where we test the link-sharing behavior using CBQ for dynamic guaranteed bandwidth and for static guaranteed bandwidth. AF1 has UDP flows and AF2 and BE have long-lived TCP flows. The AF2 class of service has low traffic, less than guaranteed. In the static case, all classes are guaranteed 2Mbit/s. Link-sharing algorithm reacts giving the excess bandwidth from class AF2 to classes AF1 and BE. Having the same priority to all classes, the excess bandwidth is divided in same proportion. The BE service gets approximately the same amount of bandwidth like AF1 class.

The case when we use DRAM, the bandwidth guaranteed by the network to AF1 is set to 3Mbit/s, instead of 2Mbit/s as before (this follows the change in the contract according to a new agreement between customer and ISP). We use the "fairly division" rule. In this case, BE service gets the smallest amount of bandwidth, since it has zero guarantee.

6.2. Link-sharing test if all reservations are not satisfied

This case we consider when all reservations are not satisfied, due to an appropriate static tuning of CBQ weights. The total rate of IN packets generated by the AF1 and AF2 classes is less than C and the total traffic is more than C (capacity of link). If CBQ limits the maximum rate of each class to 2Mbit/s, both classes AF1 and AF2 will not be satisfied since some of their IN packets will be dropped. On contrary, clients of BE class are favored, since they obtain more than their fair share of the excess bandwidth.

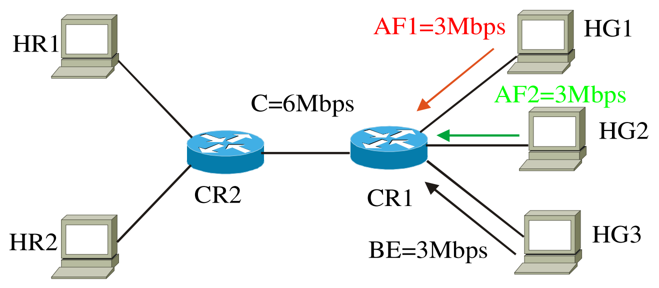


Figure 4. Test-bed network for link-sharing ability if all reservation are not satisfied. CR - core router; HR - host receiver; HG - host generator;

Figure 8a shows the static case of CBQ using link-sharing algorithm. We measure each class at the output interface of the core router. In this test we use simple configuration hierarchy. All the classes have the same priority. The network configuration used to conduct the test is presented in Figure 6. The implementation schema is described in [1].

As we observe from Figure 8a, CBQ can maintain the bandwidth of several concurrent data flows. Figure 8b shows the case when we use DRAM with flat hierarchy configuration. This proves that our algorithm changes the weights of the CBQ in order to satisfy the guaranteed bandwidth (represented by the IN packets) and to distribute the excess bandwidth using "fair division" rule. We can observe that in this case, the link-sharing algorithm has the same behavior like the rule used in our algorithm. Figure 8c shows the case when we use DRAM with the hierarchy configuration from Figure 1. We observe low values of AF1 and AF2 classes when are using the hierarchy configuration compared with flat configuration. We measure the delay in both cases for AF1 class. For the flat hierarchy, the delay is 0.832 ms and for the hierarchy configuration the delay is 0.893 ms (average values).

7. Conclusions

CBQ Linux implementation can influence the behavior of our algorithm. Flat hierarchy configuration has less delay of packets for AF1, AF2 classes than hierarchy configuration. More nodes in hierarchy link-sharing structure introduce delay and jitter. Another observation is that our algorithm using hierarchic configuration has a non-predictable behavior. The excess sharing bandwidth rule is not exactly respected in the case of hierarchic configuration. The tuning process is heavy with hierarchic configuration than the flat hierarchic configuration. The "bounded" parameter introduces delay but gives more control of the DRAM behavior. It blocks the excess link-sharing algorithm.

8. References

1] Rares Serban, Chadi Barakat and Walid Dabbous, "Dynamic Resource Allocation in Core Routers of a Diffserv Network", ASIAN02, Hanoi, December 2002;

[2] H. G. Perros, K. M. Elsayed, "Call Admission Control Schemes: A Review", *IEEE Communications Mag.*, Vol.34, No.11, November 1996, p.82-91;

[3] Lixia Zhang, Steve Deering, Deborah Estrin, Scott Shenker and Daniel Zappala, "RSVP: A New Resource Reservation Protocol", *IEEE Network*, No.7, September 1993, p.8-18;

[4] J. Wroclawski, "Specification of the Controlled-Load network element service", RFC 2211, September 1997;

[5] S. Blake, D. Black, M. Carlson, "An Architecture for Differentiated Services", RFC 2475, December 1998;

[6] J. Wroclawski, "The use of RSVP with IETF Integrated Services", RFC 2210, September 1997;

[7] Sally Floyd and Van Jacobson, "Link-sharing and resource management models for packet networks", *IEEE/ACM Transactions on Networking*, No4, August 1995;

[8] David D. Clark, Wenjia Fang, "Explicit Allocation of Best-Effort Packet Delivery Service", *IEEE/ACM Transactions on Networking*, Vol.3, No.4 August 1995;

[9] Sally Floyd and Michael Francis Speer, "Experimental Results for Class-Based Queueing", November 11, 1998;

[10] Kenjiro Cho, "A framework for Alternate Queueing: Towards Traffic Management by PC-UNIX Based Routers", Sony Computer Science Laboratory, Inc., Tokyo, Japan 1410022;

[11] KJ Loh, Irwin Gui, KC Chua, "Performance of a Linux Implementation of Class Based Queueing", 1999;

[12] D.Hoffman, M. Speer, An early access experimental realease of Solaris RSVP/CBQ;

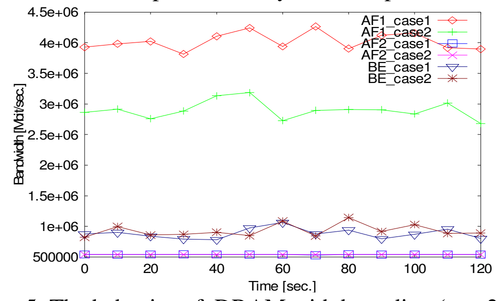


Figure 5. The behavior of DRAM with bounding (case2) and without bounding AF1 class (case1) using simple configuration

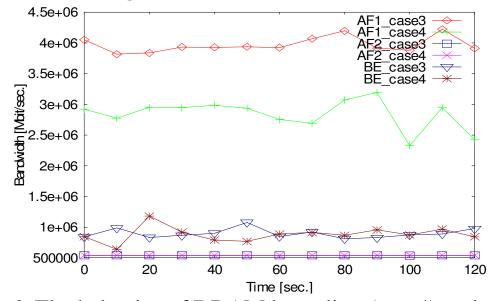


Figure 6. The behavior of DRAM bounding (case4) and without bounding AF1 class (case3) using hierarchical configuration

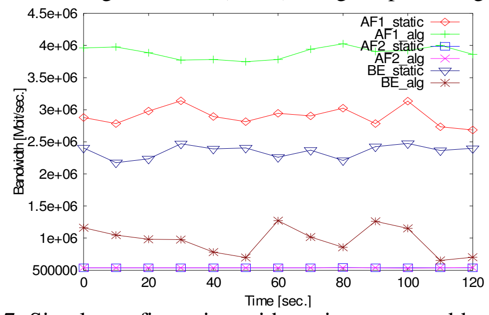


Figure 7. Simple configuration with static guaranteed bandwidth and with dynamic guaranteed bandwidth

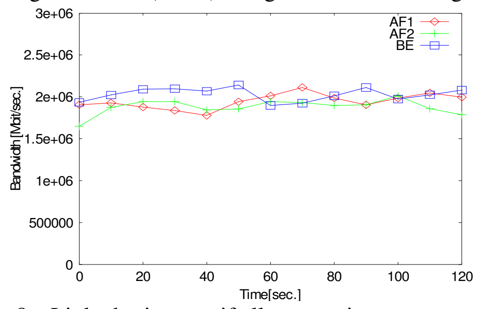


Figure 8a. Link-sharing test if all reservation are not satisfied using configuration hierarchies

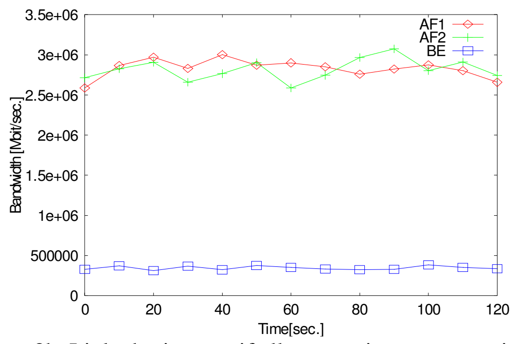


Figure 8b. Link-sharing test if all reservation are not satisfied using configuration hierarchies

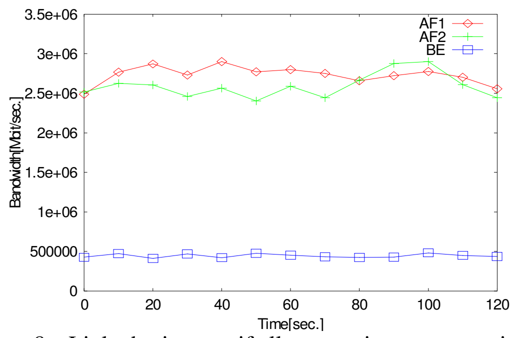


Figure 8c. Link-sharing test if all reservation are not satisfied using configuration hierarchies