Bandwidth Management for IP QoS

Nikolaos Vardalachos

MRes Telecommunications Student,

Dept. of Electrical and Electronic Engineering,
UCL, Torrington Place, London.

mt99005@ee.ucl.ac.uk

Abstract: The paper presents a simple bandwidth management algorithm for achieving QoS in IP networks. The algorithm is tested on a simulated network topology created using the simulation tool NS. Results from the use of the algorithm show that the use of the bandwidth management algorithm improved the performance of the class, which was being supported.

1. Introduction.

The Internet was initially set out to be a network used in transferring documents and files, in general, between users. As it become more and more popular, people tried to use it in different ways than the original one. Now, apart from files, voice, video, multimedia traffic is also loaded on the network. IP networks, and hence Internet, provide a best effort connectionless datagram delivery service, i.e. the packets transmitted can arrive out of order at the receiver, or corrupted or even may be lost. This is because each packet is treated independently. Since it is connectionless and no fixed circuit is allocated to each user, all users compete for the same resources. So, there is no guarantee for providing a specific level of Quality of Service (QoS) [4] to each user, with respect to delay, jitter, packet loss and throughput.

Different types of traffic (e.g. packet audio/video, bulk data, interactive data) often have different requirements in terms of throughput and delay, and these different types of packets are often sent at widely different data rates. Certain points in the network act like 'bottlenecks', and it is at these points that the different types of traffic compete for resources. Since over-provisioning the network is not an efficient method of avoiding these bottlenecks, a way of sharing these resources in way so that the packets which require better QoS are preferentially treated at these 'bottlenecks', compared to those which require less tight bounds on QoS. This can be done by using class based queuing.

When setting up the class based queues, it is essential to know the level of traffic to be carried by each class, in order to be able to dimension the classes appropriately. If the network operator/designer does not know what level of traffic per class to expect, then the bandwidth allocations per class can be randomly set, and then the operator will need to monitor the classes in order to make any corrections. This can be done by building a management plane on top of the network, which will overlook some network statistics, and respond adequately to maintain a certain level of QoS.

This paper describes a management algorithm, which monitors the queue size of the most important leaf class of a Class Based Queue (CBQ) and readjusts the hard bandwidth allocations per class. The paper is structured as follows, section 2 gives an overview of the CBQ, section 3 describes the management algorithm, section 4 describes the experimental set up, section 5 contains the results and conclusions are drawn in section 6.

2. Class Based Queuing

Class Based Queuing [1] is a variation of priority queuing with several output queues (classes) defined. The preference by which each of the queues is being serviced and the amount of queued traffic, measured in bytes that can be drained in each pass of the servicing rotation, can be also defined. This queuing discipline intends to provide some semblance of fairness by prioritising queuing services for certain types of traffic, while not allowing any class to monopolise the system (link) resources and bandwidth.

The classes can be overlimit, underlimit, at limit. A class is called overlimit if it has recently used more than its allocated link sharing bandwidth (in bytes/second, as averaged over a specified time interval), underlimit if it has used less that a specified fraction of its link sharing bandwidth, and at limit otherwise. The limit status of each class is determined by the estimator, and is used to determine when explicit action should be taken to correct the link sharing behaviour of the traffic.

Different parameters [3] can be modified in the CBQ and it's classes in order to achieve the required CBQ performance. If two classes are set up, one of the parameters that can be set is that of whether sharing of bandwidth is allowed between the classes. If one class is not using it's allocated bandwidth and the other class requires more bandwidth, then it can borrow from the other class. Otherwise, if all the CBQ classes are overlimit, then each class will receive only the bandwidth, which was initially allocated to the class, i.e. the 'hard allocated' bandwidth.

3. The Bandwidth Management Algorithm

The Bandwidth Management algorithm presented in this paper can be used to achieve this. It's aim is to reallocate bandwidth when all the CBQ classes are congested, in such a way such as that the packet losses for the most important CBQ class/ classes are minimised. The algorithm developed is quite simple conceptually. The manager will continuously check the occupancy of the highest priority class (size of the queue). If both of the classes are under limit, then the algorithm will take no action, since sharing between the two classes is allowed. Another possible state is when one class is over limit, and the other has not reached its allocated bandwidth, then the first one can borrow bandwidth from the other and still not trigger the algorithm. Another possible state for the CBQ is when both classes have reached the bandwidth they were allocated, and the offered traffic for the highest priority class cannot be accommodated by it's bandwidth allocation. This will have as a result the fast built up of the highest priority's queue.

The bandwidth management algorithm will be checking the queue size of that class, and when a threshold is being crossed, then the manager will increase the hard bandwidth allocations of the top class by a factor x. So, the other class will be left with 100-(1+x)*k % of the total link bandwidth, where k is the proportion of bandwidth allocated to the class and x is the factor by which this is raised. This increase of bandwidth allocations is not allowed to happen forever, since this will result in the starvation of the other class. So, a lower limit is defined for the minimum bandwidth proportion that the lower class can be allocated, after which the bandwidth management algorithm will not be able to increase the first class' bandwidth allocation.

In order for the algorithm to be able to achieve the targets set above, it is required that it knows which nodes are using CBQs. So, the algorithm when it is initialised, it reads a topology file, containing information on the connectivity between the nodes, which have CBQs installed and the bandwidth allocations per class. The algorithm then, stores those values in memory and uses them to do its bandwidth calculations whenever it is invoked.

When the top class becomes under limit, the algorithm will restore the original hard bandwidth allocations to each of the affected classes by reducing the bandwidth in the classes in a similar way as it increased it.

4. Experimental Set Up

In order to be able to study the algorithm developed, simulation had to be used. The simulator used was the Network Simulator [2]. The topology (figure 1) used in the experiments was simple in order to examine the effect the algorithm had. A CBQ was set up at node 3, having two classes with respective bandwidth allocations of 0.75 and 0.25 of the link bandwidth and with queues of 20 packets each. Sharing between the two classes was allowed. Node 1 had a UDP agent with a CBR application running on top of it, with λ=1000kb/s, which was binded on class 2 (0.25). Then on node 2, another UDP agent was set up, with λ varying with time from 200kb/s to 750kb/s to 1000kb/s and finally to 750kb/s. Then the communication between the management algorithm and the simulated network had to be established (figure 2).

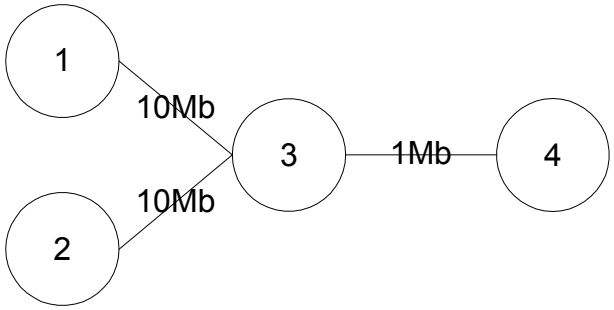

Figure 1: The experimental topology

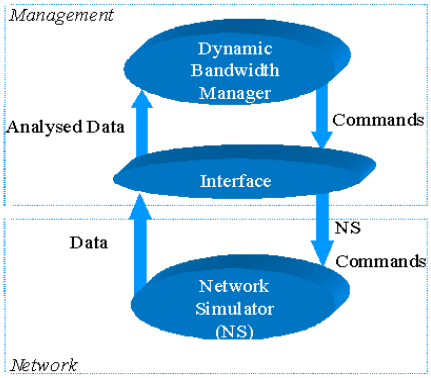

Figure 2: The experimental set up

During the simulation, the simulator sends it's trace file through a named pipe to the management plane, which performs some statistical calculations (calculates the queue lengths, etc) on the data received. These calculations are carried out by the interface between the bandwidth management algorithm and the network (simulator). If the queue size of the highest priority class exceeds the threshold value set by the algorithm, then the bandwidth management algorithm is invoked, which then decides which command will be passed back to the simulator. The commands of the bandwidth manager are passed to the simulator using a named pipe as well. If no threshold crossing has occurred, then no command will be passed to the simulator and the simulator will continue with the simulation, otherwise the command will be executed and then the simulator would resume.

5. Results

Two experiments were carried out, one with the management algorithm and the other with it. The offered traffic (figure 3) was the same for both experiments. In the first experiment, the bandwidth management algorithm wasn't used, i.e. the simulation was run with no management plane. In the first section of figure 4 , from 0 to 34 sec, the class 1 (0.75) UDP source transmits at 200kb/s and the class 2 (0.25) UDP source at 1Mb/s. It can be seen that the sharing mechanism of CBQ is used since class 1 does not fully use it's allocated bandwidth. So, the throughput for class 1 is 200kb/s (equal to the offered traffic) and class 2 borrows some bandwidth from class 1, and has a throughput of 800kb/s. In the second section of figure 4, from 34 to 55 sec, the class 1 UDP source changes it's rate to 750kb/s. So, class one requires all it's allocated bandwidth and sharing is no longer feasible. The throughput for class 1 and class 2 become 750kb/s and 250kb/s respectively,

which are the hard bandwidth allocations for the two CBQ classes. In the third section of figure 4, from 55 to 75 sec, the class 1 source increases it's rate even more to 1Mb/s. In this case, no change can be made, because both classes have used their allocated bandwidth, and hence continue to have the same throughput as before but class 1 has

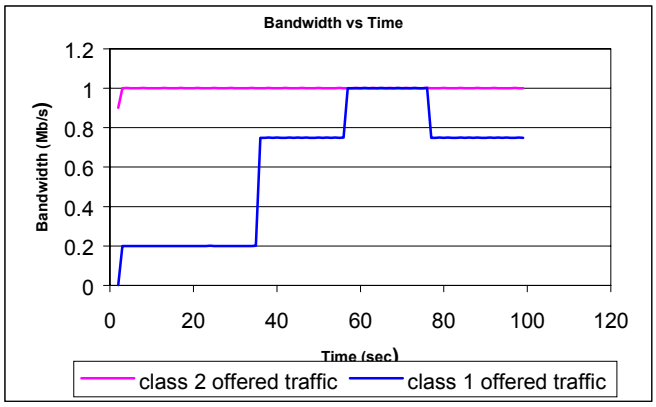

Figure 3: Offered Traffic

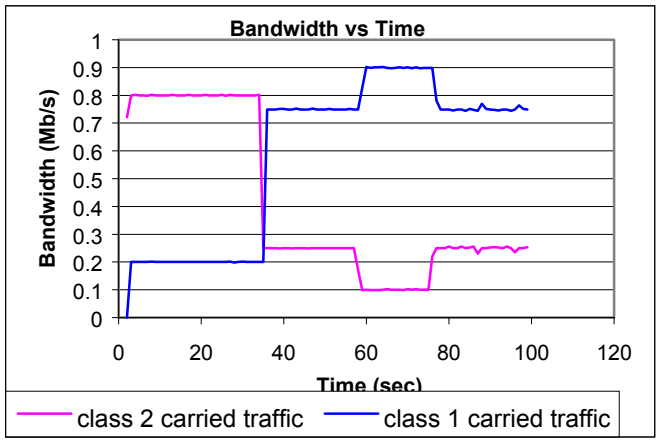

Figure 4: No algorithm running

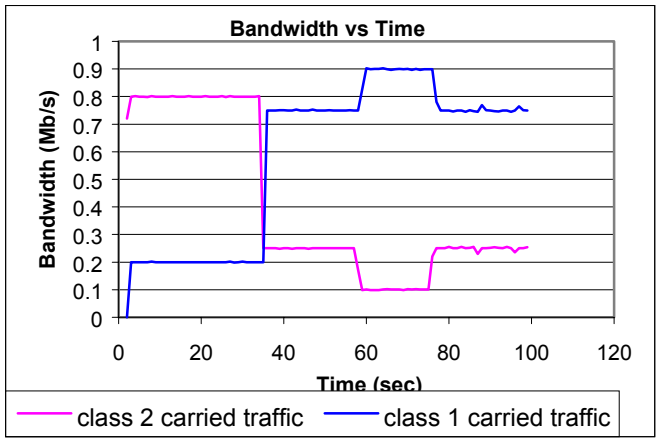

Figure 5: Algorithm in operation

drops. In the last section of figure 4, from 75 to 99 sec, the class 1 UDP source decreases it's rate to 750kb/s. In this case, as expected, the throughput for each of the sources remains the same, equal to the bandwidth allocated to each of the classes and class stops having dropped packets.

In the second experiment (figure 5), the bandwidth management algorithm was used. The difference in the two experiments can be seen in the 3rd section of the graph where class 1 asked for more bandwidth than it was allocated. This resulted in an increase of its queue occupancy. When the queue size becomes greater than 10 packets (which is

the algorithm's threshold), the bandwidth management algorithm is invoked and gradually increases the bandwidth allocated to class 1. Eventually, the bandwidth allocated to class 1 reaches the upper bandwidth allocation limit, which is equal to 900kb/s. This limit is dictated by a lower bandwidth limit set by the algorithm for class 2, in order to protect class2 from resources starvation. Also, as the rate of the class 1 source, between 75 and 99 sec, decreased, the offered traffic can easily fit class 1 (0.9 now). This results in the gradual decongestion of class's 1 queue, which in effect triggers the bandwidth algorithm. The algorithm then reduces the class 1 allocation gradually to 750kb/s, giving class 2 the remaining of the bandwidth.

6. Conclusions

The architecture in figure 2 provides an experimental set up, for testing management algorithms in general. The only thing that would be required would be a new interface between the simulator and the new management system, in order the required information to be extracted from the trace files of the simulator. The algorithm described, provides a way of managing the bandwidth allocations of the different classes inside a CBQ when the offered traffic per class is not known, and one class is considered to be more important than the others. It also reserves some bandwidth for the classes, which are not considered to be important, in order to prevent the top class to occupy all the bandwidth on the link. The service provided with this algorithm is not qualitative but quantitative, since it only discriminates between the different classes of a CBQ. It was also, realised that the algorithm had certain limitations. Since, the algorithm protects the remaining classes from starvation, there will be an upper limit to the available bandwidth to be allocated to the highest priority class (the class supported by the algorithm). So, if the offered traffic is more than that maximum value, then the algorithm will not be able to have any effect, and the overall class performance will degrade.

Acknowledgements

I would like to thank David Griffin for offering his continuous support and constructive remarks on this paper and my MRes work in general.

References

[1] Floyd, S., and Jacobson, V., "Link-sharing and Resource Management Models for Packet Networks" IEEE/ACM Transactions on Networking, Vol. 3 No. 4, pp. 365-386, August 1995

[2] UCB/LBNL/VINT "Network Simulator – ns (version 2)", **http://www-mash.cs.berkeley.edu/ns/ns.html**

[3] Floyd, S., "Notes of Class-Based Queueing: Setting Parameters", Informal notes, September 1995

[4] Ferguson, P and Huston, G., "Quality of Service, Delivering QoS on the Internet and in Corporate Networks, Wiley Computer Publishing, 1998