Implementation and performance measurements of a delay-bounded HPD algorithm in an ALTQ-based router

Johanna Antila  
Helsinki University of Technology  
PO. Box 3000  
02015 HUT  
FINLAND  
Email: johanna.antila@hut.fi

Marko Luoma  
Helsinki University of Technology  
PO. Box 3000  
02015 HUT  
FINLAND  
Email: marko.luoma@hut.fi

Antti Paju  
Creanord Oy  
Hämeentie 153 C  
00560 Helsinki  
Finland  
Email: antti.paju@creanord.com

Abstract — In this paper we show the first working implementation of an adaptive, measurement-based scheduling algorithm called delay-bounded HPD (DBHPD) in a FreeBSD-based ALTQ prototype router. We describe how we have implemented DBHPD and discuss what kind of difficulties were encountered in the implementation. We present measurement results of the DBHPD implementation with FTP, HTTP, Video Streaming and VoIP traffic in underload, overload and heavy overload conditions. We show that the algorithm operates well according to the theoretical model and preserves the desired delay-bound as well as the delay ratios between the classes. We also compare DBHPD to an existing Class-Based-Queueing (CBQ) algorithm that is widely used both in research and in the industry. We show that DBHPD is able to achieve at least as good link utilization as CBQ and in addition results in much better and predictable differentiation in terms of delays and more controlled packet losses.

I. INTRODUCTION

During the recent years adaptivity has become the hot topic in networking. Adaptive mechanisms are desirable since they enable a self-configurable network that is able to adjust itself to varying traffic conditions. The operation of adaptive mechanisms is heavily based on measurements: the state of the network is monitored to produce an estimate of a desired quantity, which may then be used in different time scales of network control.

We focus on the packet-level time scale and investigate how measurements can be utilized to make a packet scheduling algorithm adaptive. The basic idea of adaptive scheduling algorithms is to dynamically adjust the class resources either periodically or on a packet per packet basis, so that the policy chosen by the operator will be fulfilled regardless of the traffic conditions. Our approach is to base the resource adaptation on packet delay: In [10], we proposed a delay-bounded HPD (DBHPD) scheduling algorithm for combined absolute and proportional delay differentiation in the DiffServ context. In this algorithm, the most delay sensitive class is assigned an absolute delay bound. If this bound is about to be violated, a packet is directly dispatched from this class, otherwise the operation is based on the delay ratios between classes according to [7].

So far we have evaluated the DBHPD algorithm only with simulations by using traditional performance metrics such as throughput, delay and packet loss as criteria. In [10], [11] and [12] we studied the performance of the DBHPD algorithm as well as static and other adaptive scheduling algorithms both in a single bottleneck case with relatively high abstraction level traffic models and in a larger network topology with realistic traffic models. We also investigated what kind of traffic mapping principles are suitable for these algorithms. In [13] we developed new delay estimators for calculating the long term delay component for the DBHPD algorithm and evaluated the filtering properties of these estimators. From this previous research we concluded that adaptive scheduling algorithms and especially the DBHPD algorithm lead to more consistent quality differentiation than static algorithms. Efficient delay estimators can further improve the level of differentiation.

However, considering only traditional performance metrics as evaluation criteria may lead to false conclusions since an algorithm that behaves well in simulations may not lead to desired results in real world. In fact, the algorithm may be impossible to implement in real-time environment if it uses too complex computation and esti-
mation procedures. Thus, an algorithm can be considered valuable only after the implementation complexity of the algorithm as well as the resulting performance in a real implementation has been assessed.

To our understanding the DBHPD algorithm has not been implemented in a prototype or a commercial router before. In this paper we show the first working implementation of the DBHPD algorithm in a FreeBSD-based ALTQ prototype router. ALTQ [2] [3] is a general Quality of Service framework for a BSD system that currently supports the following scheduling algorithms: Weighted Fair Queueing (WFQ) [9], Class Based Queueing (CBQ) [1], Hierarchical Fair Service Curve (HFSC) [8] and Joint Buffer Management and Scheduling (JoBS) [5]. Of these algorithms CBQ is the most well-known and most widely used in measurement papers. CBQ can be considered as an adaptive scheduling algorithm in the sense that it provides heuristic rules for borrowing capacity in a case where some class is running out of resources. Thus it is reasonable to use CBQ as a baseline when evaluating the performance of the DBHPD algorithm.

The main goal of this paper is to describe how we have implemented DBHPD in the ALTQ prototype, what kind of challenges were encountered in the implementations and how these challenges were solved. We also present measurement results of the DBHPD implementation and show that DBHPD is able to achieve the same link utilization as the existing CBQ algorithm and in addition results in much better differentiation in terms of delays.

II. THEORETICAL BACKGROUND OF THE MEASURED SCHEDULING ALGORITHMS

This section presents the theoretical model for DBHPD and CBQ algorithms. It should be noted that the theoretical models may deviate from the real implementations since the hardware and software set certain limits for what can be implemented in practice.

A. Class Based Queueing (CBQ)

Class Based Queueing (CBQ) [1] is the most well known hierarchical bandwidth sharing algorithm. In CBQ the roles of the different hierarchy levels are following:

- **Root class** contains the link resource that is to be divided among the traffic classes.
- **Leaf classes** represent the actual traffic classes that are served by the link.
- **Intermediate classes** are responsible for resource sharing among the leaf classes. The intermediate classes act as parents for the leaf classes, allowing the leaf classes to borrow resources.

The resource allocation in CBQ is enforced with two different schedulers: a general scheduler and a link sharing scheduler.

- **General scheduler** is all that is required when there are enough resources available for all leaf classes.
- **Link sharing scheduler** is required when some of the leaf classes get congested. In this case, the link sharing scheduler enforces rules by which the congested classes can borrow resources from the parent classes.

The actual implementation for the general scheduler and the link sharing scheduler has not been defined. The idea is that any rate-based scheduler, such as WFQ, WRR or DRR can be used as the general scheduler and also the borrowing rules of the link sharing scheduler can vary among implementations.

B. delay-bounded HPD (DBHPD)

The delay-bounded HPD (DBHPD) algorithm first checks if the packet in the highest class (class 0) is about to violate its deadline. Denote by $d_{\text{max}}$ the delay bound in the highest class, by $t_{\text{safe}}$ a safety margin for the delay bound, by $t_{\text{in}}$ the arrival time of the packet in the highest class queue and by $t_{\text{curr}}$ the current time. The packet in the highest class queue is considered to be violating its deadline if

$$t_{\text{in}} + d_{\text{max}} < t_{\text{curr}} + t_{\text{safe}}.$$  \hspace{1cm} (1)

If delay violation is not occurring, the algorithm takes into account the delay ratios between the other classes. Denote by $\bar{d}_i$ the average queueing delay of class $i$ packets and by $\delta_i$ the Delay Differentiation Parameter (DDP) of class $i$. The ratio of average delays in two classes $i$ and $j$ should equal the ratio of DDPs in these classes

$$\frac{\bar{d}_i}{\bar{d}_j} = \frac{\delta_i}{\delta_j}, \quad 1 \leq i, j \leq N.$$ \hspace{1cm} (2)

In [7] this is interpreted so that the normalized average delays of traffic classes must be equal, i.e.,

$$\bar{d}_i = \frac{\bar{d}_i}{\delta_i} = \frac{\bar{d}_j}{\delta_j} = \bar{d}_j, \quad 1 \leq i, j \leq N.$$ \hspace{1cm} (3)

The DBHPD algorithm selects for transmission at time $t$, when the server becomes free, a packet from a backlogged class $j$ with the maximum normalized hybrid delay [7]:

$$j = \arg \max (g\bar{d}_i(m) + (1 - g)\bar{w}_i(m)),$$ \hspace{1cm} (4)
where \( \tilde{d}_i(m) \) and \( \tilde{w}_i(m) \) denote the normalized average queueing delay and the normalized head waiting time (i.e. the waiting time of the first packet) of class \( i \) when \( m \) packets have departed and \( 0 \leq g \leq 1 \) is a weighting coefficient. Thus, the algorithm utilizes measurements of both short and long term queueing delays (\( \tilde{d}_i(m) \) and \( \tilde{w}_i(m) \)) in the scheduling decisions. The operation of the algorithm depends largely on how the average delay \( \tilde{d}_i(m) \) is calculated, because it determines the amount of history that is incorporated into the scheduling decisions.

In the original form [7] the average delay of class \( i \) after \( m \) packet departures, \( \tilde{d}_i(m) \), is calculated by a simple sum estimator as follows: Denote by \( D_i(m) \) the sequence of class \( i \) packets that have been served and by \( d_i(m) \) the delay of the \( m \)‘th packet in \( D_i(m) \). Then, assuming that at least one packet has departed from class \( i \) before the \( m \)‘th packet

\[
\tilde{d}_i(m) = \frac{\sum_{j=1}^{\left| D_i(m) \right|} d_i(m)}{\left| D_i(m) \right|}.
\]

However, this kind of calculation to infinity is not feasible in practice, since the counter for the sum of delay values easily overflows as enough packets have been departed from a certain class. Also, we do not want to incorporate infinite history into the estimator and thus into the scheduling decisions.

III. IMPLEMENTATION OF THE MEASURED SCHEDULING ALGORITHMS

We have used the Alternated Queueing (ALTQ) package [2] for implementing the DBHPD algorithm. ALTQ is a general Quality of Service framework for BSD systems. It contains basic scheduling, queue management and rate control functionalities which can be assembled together in a various ways to realise different types of QoS networks. In our work we used CBQ scheduling block and modified the priority queueing to implement DBHPD scheduling.

A. Implementation of the CBQ algorithm

For CBQ we have utilized the existing implementation in the ALTQ-package. In this implementation the general scheduler is either Weighted Round Robin (WRR) or Packet Round Robin (PRR). When looking the implementation of these schedulers in a more detailed level WRR contains also deficit counter and therefore is actually close to Deficit Round Robin (DRR) [6]. We used WRR as it allows greater flexibility and accuracy in capacity allotment. Possible queue management algorithms in ALTQ are tail-drop, RED and RIO. We decided to use only tail-drop since we wanted to see the effect of pure scheduling. However, we realize that queue management increases the performance of TCP traffic and thus the effect of different algorithms should be evaluated in the future.

Actual implementations of CBQ borrowing mechanism contain a lot of approximations which degrade the accuracy in which CBQ operates and also cause performance penalties. ALTQ contains two modes for borrowing:

1) **Borrow**: A class is allowed to borrow if its ancestor class is underlimit. This makes the system only work-conserving among sub-trees that share common resources.

2) **Efficient**: A class is allowed to borrow even if its ancestor class is not underlimit. Such a situation may occur if borrowing has taken place in a manner that the system is in suspension (e.g. no packets are allowed to be sent if timer has not expired) even though the link has free resources. Efficient mode takes a packet from the first scanned queue and passes that on the link. This makes the whole system work-conserving but can cause unexpected behaviour for classes with small capacity fraction.

Details of the CBQ implementation can be found in [2]. We have chosen to use only simple borrowing in our measurements in order to achieve more predictable performance and more controllable top-level borrowing. Our CBQ link-sharing hierarchy is presented in Figure 1.

B. Implementation of the delay-bounded HPD algorithm

As far as we know, the delay-bounded HPD algorithm has not been implemented in a real or in a prototype router before. We have implemented the algorithm on
the basis of the priority queue structure in ALTQ (ALTQPRIQ). The DBHPD algorithm relies heavily on the measurement of time. Time is a difficult concept for operating system kernels as it deals with floating point numbers. Majority of the operating systems do not allow floating point operations in the kernel space as they lead to diminished performance. The work-around is the usage of two interconnected integers; in case of time one for the full seconds and the other for microseconds. Manipulation of two interconnected integer values consumes always some extra time compared to a single integer. In our implementation, the head-of-line packet delay is obtained by utilizing the kernel `gettime` routine twice: first when the packet is enqueued in the queue and secondly when the scheduling decision is made by polling the head-of-line packets. However, usually this happens more than twice as there are several packets competing on the link resource and queueing time calculation has to be done each time a scheduling decision is made.

In the theoretical model the long term average delay $\bar{d}_i(m)$ was calculated with the simple sum estimator. With simple sum estimation the system is counting the average value with an infinite memory. Infinite memory in a real router is impossible concept as the actual kernel is based on either 32 bit or 64 bit registers. This would eventually lead to overflow of registers. Therefore, in the implementation, the long term delay is obtained with an Exponential Weighted Moving Average (EWMA) filter as follows:

$$\bar{d}_i(m) = (\gamma d_i(m) + (1 - \gamma)\bar{d}_i(m - 1)),$$  \hspace{1cm} (6)

where $0 \leq \gamma \leq 1$. Now calculation to infinity is not required and the amount of history can be determined by the selection of the $\gamma$ parameter. Normally weighting coefficients are between zero and one, but in kernel space floating point arithmetics should be avoided. Therefore, all operations are scaled by the magnitude of number range in coefficients. DBHPD requires a lot of time calculations and therefore we streamlined the calculation to contain only binary shifting. This means that weighting coefficients and differentiation parameters are powers of two, allowing multiplication by shifting the actual delay value by the power of a coefficient. Operating with integers increases efficiency but also reduces the granularity of parameters. However, the deviation from ideal operation is still acceptable.

For the future experimentation, we have developed even more sophisticated estimators for the DBHPD algorithm [13] but these estimators have not been implemented in ALTQ yet.

### IV. MEASUREMENT SETUP

Measurement topology used in our test is presented in Figure 2. The topology consists of a router having connection into two individual networks:

1) **Server network:** QuickTime Video server, Web server, FTP server and VoIP clients
2) **Client network:** QuickTime Video clients, Web clients, FTP clients and VoIP clients

Interconnecting networks were 100Mbps ethernet VLANs implemented in a single ethernet switch. Router in between was a PC running FreeBSD-4.5 operating system on a 433MHz Intel Celeron architecture having 128MB of memory. The FreeBSD kernel was compiled to have 1kHz clock resolution to make internal interrupt handling more fine grained.

We used four different applications in the measurements: FTP, HTTP, QuickTime Video Streaming and VoIP. FTP, HTTP and QuickTime Video Streaming traffic was produced with Spirent’s Avalanche (client side) / Reflector (server side) test appliances. VoIP traffic was emulated with Spirent’s AX/4000 broadband test system by defining suitable traffic generation pattern.

Three user instances were defined in Avalanche: one for FTP, one for HTTP and one for Video Streaming. The FTP instance requests one file of size 1MB, the HTTP instance loads 40 www-pages having short break after 5 pages and the Video instance streams a 60 s VBR video-clip with the average speed of 200kbps. In www-transfers HTTP version 1.0 was used, meaning that a separate TCP-connection was established for each object within a page. However, pages used in the measurement contained only a single object. The page sizes were defined with a list having 40 entries from the geometric distribution of mean 16.5kB. The mean value and distribution represents actual size distribution of real web-
VoIP calls were created in AX/4000 by defining a burst of 13 packets arriving every 20ms. During the burst packets are not sent back to back rather having constant rate of 30Mbps. This pattern approximates 13 simultaneous VoIP calls between two ports of AX/4000.

The mapping of traffic into different classes was performed so that the VoIP uses the first, delay-bounded class, Video second class, HTTP third class and FTP fourth class. The relative traffic shares for different applications were following: (FTP: 30%, WWW: 40%, Video: 20%, VoIP: 10%). Desired traffic load for FTP, HTTP and Video was produced by defining how many users/hour avalanche created. In addition, the user generation pattern may be flat, stair-step, bursty or sinusoid in order to create more load fluctuations. We have used a flat pattern for HTTP and Video and a sinusoid pattern for FTP.

Denote by $\bar{\rho}$ the desired mean load in the output link, by $f_i$ the load fraction produced by class $i$, by $C$ the link capacity (bits/s) and by $W$ the workload (bits) created by a single user instance in class $i$. The required number of simusers/hour in class $i$ can be calculated as:

$$N = \frac{\bar{\rho} \times f_i \times C}{W} \times 3600s$$

Since the operation of the DBHPD algorithm is based on packet delays, we were particularly interested in recording the distribution of delays in each traffic class. In Avalanche/Reflector this type of information is not provided and thus a packet capturing tool such as tcpdump would be required to track the packets generated by FTP and HTTP. However, it is also possible to obtain the delay distributions indirectly by defining a measurement probe for each traffic class. Suitable probes are small packets that are sent in continuous intervals and have accurate time-stamps. We used this approach since AX/4000 provides efficient facilities to record delay distributions of traffic streams. Each probe sends a 46 byte packet every 10 ms, resulting in a 50kpbs CBR stream (raw IP-packets). The delay distributions of the probes do not correspond exactly to the distributions of the actual application traffic but the probes provide an approximation that is accurate enough for our purposes. Link utilizations, application throughputs and packet losses are derived from tcpdump files captured from both sides of the router.

![Fig. 3. Link utilizations with HPD and CBQ scheduling (Load 90%).](image)

V. PROVISIONING PARAMETERS AND MEASUREMENT SCENARIOS

Provisioning for CBQ was performed by simply allocating each class a bandwidth share that corresponds to the load fraction of that class. The bandwidth fractions of the classes are shown in Figure 1. In delay-bounded HPD the delay-bound was set to 5ms and the delay ratios to 4. Queue lengths in both algorithms are set to 15, 15, 80 and 200 packets. The settings for delay-bounded HPD correspond well to the parameters used in our previous simulation studies [10]–[12].

Both algorithms were tested in underload (90% mean theoretical load), overload (100% mean theoretical load) and heavy overload (110% mean theoretical load). Load level of the system was varied by changing the emulated link speed within the scheduling units of router interfaces. Theoretically there should be little over 9Mbps offered traffic in the system. Thus, in the measurements with 90% load level link speed was set to 10Mbps while in the measurements with 110% load level it was set to 8Mbps. CBQ was tested both with borrowing and no borrowing. When borrowing is on, CBQ uses heuristics to distribute excess resources among the classes up to the top-level of sharing which is the parent class of the leaf classes. This parent is the intermediate class: i) realtime for VoIP and Video, and ii) non-realtime for WWW and FTP.

VI. MEASUREMENT RESULTS

In this section we present the results from the CBQ and DBHPD measurements. We show link utilizations,
cumulative delay distribution functions for each class as well as application throughputs and packet losses for both algorithms. When comparing the results of different load levels one should remember that difference in load level was achieved by changing the link capacity. This causes lower throughput with the same offered traffic. The delays consist of queuing delay, processing delays and propagation delay, of which queuing delay is the dominating component.

A. Scenario 1: Theoretical link load 90%

Figure 3 shows the link utilizations for DBHPD and CBQ with a theoretical link load of 90%. In CBQ borrowing is used in order to heuristically divide extra capacity. From the Figure it can be observed that DBHPD is capable of providing full link capacity for the elastic TCP traffic while CBQ can not provide same level of adaptivity. This results from the fact that part of the capacity is dedicated to real-time traffic which is not using full share of its resources.

Tables I and II show application throughputs and packet losses in this scenario. The throughput for FTP and HTTP has been recorded by averaging the throughputs of individual objects. This results in huge variances for HTTP as there are very small pages (fit into a single packet) and considerably larger ones. DBHPD provides more throughput for FTP due to its capability to adjust to time varying load levels. On the other hand, CBQ allocates considerably more resources for HTTP even
## Table I
Statistics for HPD with delay bound (Load 90%)

<table>
<thead>
<tr>
<th>Traffic</th>
<th>Throughput kbps</th>
<th>Loss %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Dev</td>
</tr>
<tr>
<td>FTP</td>
<td>2451</td>
<td>1524</td>
</tr>
<tr>
<td>WWW</td>
<td>4642</td>
<td>32588</td>
</tr>
<tr>
<td>Video</td>
<td>209</td>
<td>0.4</td>
</tr>
<tr>
<td>VoIP</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

## Table II
Statistics for CBQ with borrow (Load 90%)

<table>
<thead>
<tr>
<th>Traffic</th>
<th>Throughput kbps</th>
<th>Loss %</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Dev</td>
</tr>
<tr>
<td>FTP</td>
<td>1041</td>
<td>687</td>
</tr>
<tr>
<td>WWW</td>
<td>5359</td>
<td>17005</td>
</tr>
<tr>
<td>Video</td>
<td>208</td>
<td>0.8</td>
</tr>
<tr>
<td>VoIP</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

though FTP should use a larger share of the resources. It should be noted that the theoretical provisioning goal was in this occasion 4Mbps for HTTP and 3Mps for FTP. We can observe that DBHPD is clearly closer to the goal of 25% difference in capacity allocation - even though DBHPD operates on delay, not on capacity. Deviation of CBQ from the target capacity is due to the nature of implementation. Borrowing of capacity in a packet per packet manner is computationally intensive task. Therefore, approximations are made in the calculations by allowing borrowing only over certain time-windows. This easily leads to starvation of low capacity classes.

Tables I and II show application throughputs and packet losses in this scenario. The throughput for FTP and HTTP has been recorded by averaging the throughputs of individual objects. This results in huge variances for HTTP this gives as there are very small pages (fit into a single packet) and considerably larger ones. DBHPD provides more throughput for FTP due to its capability to adjust to time varying load levels. On the other hand, CBQ allocates considerably more resources for HTTP even though FTP should use a larger share of the resources. It should be noted that the theoretical provisioning goal was in this occasion 4Mbps for HTTP and 3Mps for FTP. We can observe that DBHPD is clearly closer to the goal of 25% difference in capacity allocation - even though DBHPD operates on delay, not on capacity. Deviation of CBQ from the target capacity is due to the nature of implementation. Borrowing of capacity in a packet per packet manner is computationally intensive task. Therefore, approximations are made in the calculations by allowing borrowing only over certain time-windows. This easily leads to starvation of low capacity classes.

Figure 4 presents the delays for CBQ and DBHPD in this scenario. By comparing delay distributions and medians of these algorithms previous notions on borrowing become obvious. When CBQ is used, HTTP traffic in class 3 receives most of the network capacity which can also be seen in very small delays. The difference in real-time traffic classes is not that large between CBQ and DBHPD but in general DBHPD provides much smaller variance for the delay. This is important for the real-time communications which requires a playback compensation buffer that is dimensioned relative to the delay variance. From the delay distributions we can conclude that DBHPD provides much more reasonable service than CBQ because CBQ offers extra capacity to a class with the highest link-share, regardless of borrowing. We believe that the implementation of CBQ algorithm is so complex that accuracy of its operation is far from the theoretical operation.

### B. Scenario 2: Theoretical link load 100%

Figure 5 show the link utilizations for DBHPD and CBQ with a theoretical link load of 100%. In CBQ borrowing is used in order to heuristically divide extra capacity. It can be observed that both algorithms result in approximately the same link utilization, DBHPD providing slightly more steady utilization.
Fig. 6. Delay distributions for CBQ and DBHPD (Load 100%)

TABLE III
STATISTICS FOR HPD WITH DELAY BOUND (LOAD 100%)

<table>
<thead>
<tr>
<th>Traffic</th>
<th>Throughput kbps</th>
<th>Loss %</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>1736</td>
<td>1826</td>
</tr>
<tr>
<td>WWW</td>
<td>2723</td>
<td>16142</td>
</tr>
<tr>
<td>Video</td>
<td>208</td>
<td>1</td>
</tr>
<tr>
<td>VoIP</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Tables III and IV show application throughputs and packet losses in this scenario. DBHPD provides more throughput for FTP and Video while CBQ allocates again more resources for the HTTP. The packet losses are clearly smaller for real-time traffic in DBHPD. This is due to the fact that in 100% scenario actual load is little over 100% and mainly caused by video traffic which sends a little bit over its allocated capacity. In CBQ this
will lead to packet loss in real-time traffic subclass. On the other hand, DBHPD is capable of adapting to the packet loss by using capacity from elastic traffic.

Figure 6 present the delays for CBQ and DBHPD in this scenario. By comparing delay distributions and medians of these algorithms it is obvious that DBHPD is able to provide delay differentiation according to the specified provisioning parameters. The delay bound in the highest class is not violated, the class ordering between the rest of the classes is correct and the specified delay ratio between classes is preserved. However, in CBQ the delays are distributed in an irrational way between the classes and furthermore, class ordering is violated between class 2 and class 3 since class 3 experiences lower delays.

The shape of the distributions show that in DBHPD the delay distribution is quite linear while the distribution for CBQ is more concave. Another important observation is that the distributions for CBQ are considerably wider especially for class 1 and class 4. This implies that DBHPD is able to provide more predictable service in terms of delays.

C. Scenario 3: Theoretical link load 110%

Figure 7 show the link utilizations for DBHPD and CBQ with a theoretical link load of 110%. Since the system is pathologically congested both scheduling algorithms provide operation which is strictly bounded by their link-sharing rules: in CBQ capacity is allocated to each leaf class and in DBHPD the operation is based mainly on the proportional delay constraints.

Tables V and VI show application throughputs and packet losses in this scenario. As can be expected, the packet losses are considerably higher and throughput as smaller than in the 100% load case.

Figures 8 show the delay distributions for DBHPD and CBQ with borrow when the offered theoretical load is 110%. This corresponds to the heavy overload case. It can be seen that the shape of the distributions remains very similar but the mean and maximum delays increase considerably since the system is in a pathological congestion situation. Also the gap between the delay medians for DBHPD and CBQ is significantly larger compared to previous scenarios.

VII. CONCLUSIONS

In this paper we have shown the very first implementation of an adaptive scheduling algorithm delay-bounded HPD (DBHPD) in a FreeBSD based ALTQ router. DBHPD is based on proportional delay constrained scheduling which makes it suitable for application differentiated QoS networks: it provides low delay service for real-time traffic and allocates resources between the other classes based on delay ratios. We have also conducted measurements with this algorithm in different load conditions. From the results presented in Section VI-B we can conclude that DBHPD not only provides good network utilization but also distributes the delays between the service classes in a manner that makes both real-time and non-real-time communication

<table>
<thead>
<tr>
<th>Traffic</th>
<th>Throughput kbps</th>
<th>Mean</th>
<th>Dev</th>
<th>Loss %</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>953</td>
<td>728</td>
<td>0.03</td>
<td>0.1</td>
</tr>
<tr>
<td>WWW</td>
<td>602</td>
<td>1360</td>
<td>0.02</td>
<td>1</td>
</tr>
<tr>
<td>Video</td>
<td>205</td>
<td>3.4</td>
<td>2.4</td>
<td>1.8</td>
</tr>
<tr>
<td>VoIP</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traffic</th>
<th>Throughput kbps</th>
<th>Mean</th>
<th>Dev</th>
<th>Loss %</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP</td>
<td>652</td>
<td>426</td>
<td>0.07</td>
<td>0.1</td>
</tr>
<tr>
<td>WWW</td>
<td>3527</td>
<td>13567</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>Video</td>
<td>182</td>
<td>16</td>
<td>16.7</td>
<td>9.8</td>
</tr>
<tr>
<td>VoIP</td>
<td>-</td>
<td>-</td>
<td>18.4</td>
<td>-</td>
</tr>
</tbody>
</table>
viable.

We have also compared DBHPD with a well known and widely used algorithm Class Based Queueing (CBQ). Our results show that CBQ is only a semi-adaptive algorithm and is not able to provide as good tracking of offered traffic in contrast to available resources as DBHPD does. Besides the ability to adapt to experienced packet delays DBHPD also controls the packet losses which are due to buffer overflows (e.g. too long queueing times). In Section VI-C we presented a pathological congestion situation where no excess capacity is available. In this scenario queueing delays are extremely large, and with no adaptivity large packet loss is unavoidable for real-time traffic.

Our future work concerns implementation of more sophisticated estimation algorithms for the long term average delay and also implementation of this algorithm in an embedded router architecture with network processors. We also aim to instrument the measurement system so that processing and memory consumption of both algorithms can be tracked.

ACKNOWLEDGEMENTS

This work has been partly supported by the European Union under the E-Next Project FP6-506869 and by the Ironet project funded by TEKES.
REFERENCES


