Competitive Dynamic Bandwidth Allocation

Amotz Bar-Noy* Yishay Mansour† Baruch Schieber‡

Abstract

We propose a realistic theoretical model for dynamic bandwidth allocation. Our model takes into account the two classical quality of service parameters: latency and utilization, together with a newly introduced parameter: number of bandwidth allocation changes, which are costly operations in today's networks. Our model assumes that sessions join the network with a certain delay requirement rather than a bandwidth requirement as assumed in previous models. In addition, the network has a certain utilization requirement. Given bounds on latency and utilization, we design online algorithms that minimize the number of bandwidth allocation changes.

1 Introduction

The phenomenal proliferation of communication networks during the recent years is due to both growth in the number of users and inflation in their bandwidth demand. Although the available bandwidth is increasing dramatically, it is still one of the bottleneck resources in communication networks. Sharing this resource efficiently is the key to a successful communication network.

In traditional local area networks bandwidth sharing is done implicitly, for example, by using a shared media for transmission. In the next generation of networks there is much more emphasis on allowing requirements to be specified explicitly by the user. This manifests itself as *Quality of Service (QoS)* parameters, which vary over a large range of parameters. Among the most important parameters are *latency (delay)* and *bandwidth*. (Examples of this trend can be found in the ATM standards and in the IPv6 standard.)

A simplistic approach is to allow users to specify a constant bandwidth requirement per session. Having a constant bandwidth allocation has many advantages from both the user and network perspectives. Once established, it is completely predictable, and enables a simple pricing model that depends on the total bandwidth consumption, i.e., the product of the bandwidth allocation and the duration of the session. However, only for very few tasks (e.g., real-time voice) the required bandwidth is known in advance. Even video communication involves a variable requirement of bandwidth (due to compression), and in other applications with bursty nature of traffic the required bandwidth may change dramatically over time, usually in an unpredictable manner. (See Figure 1 for an example of a stream of bits requested by a session.) In order to accommodate such situations it is reasonable to require the network to allow dynamic modification of the bandwidth allocation. (See [GKT95, CRS95, ACHM96] for trends in applying dynamic bandwidth allocation.)

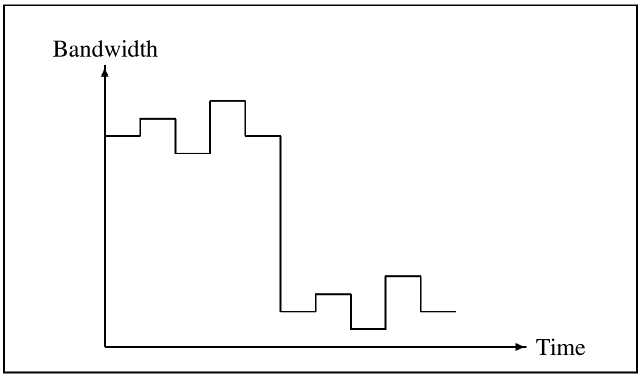

Figure 1: An example of bandwidth demand.

When considering static bandwidth allocation for tasks with varying and unpredictable bandwidth requirements, the latency and utilization parameters interplay. The latency of a session is defined as the maximum over all bits sent in the session of the time elapsed between the submission of a bit at the sending end and its arrival at the receiving end. Utilization is defined as the ratio between the total bandwidth allocated from the time connection was established to the number of bits received during this time. Clearly, latency is one of the most important parameters from the end user perspective. (As can be easily related to by any "web surfer".) On the other hand, utilization is one of the most important performance parameters from the network point of view. The two view points do not match, and there is an inherent tradeoff between utilization and latency. The tradeoff is best illustrated by its two extremes.

*Dept. of Electrical Engineering, Tel Aviv University, Tel Aviv, Israel 69978, E-mail: *amotz@eng.tau.ac.il*. This work was supported by the consortium for broadband communication administered by the Chief Scientist of the Israeli Ministry of Commerce and Industry.

†Dept. of Computer Science, Tel Aviv University, Tel Aviv, Israel 69978, E-mail: *mansour@math.tau.ac.il*. This work was supported by the consortium for broadband communication administered by the Chief Scientist of the Israeli Ministry of Commerce and Industry.

‡IBM – Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, E-mail: *sbar@watson.ibm.com*.

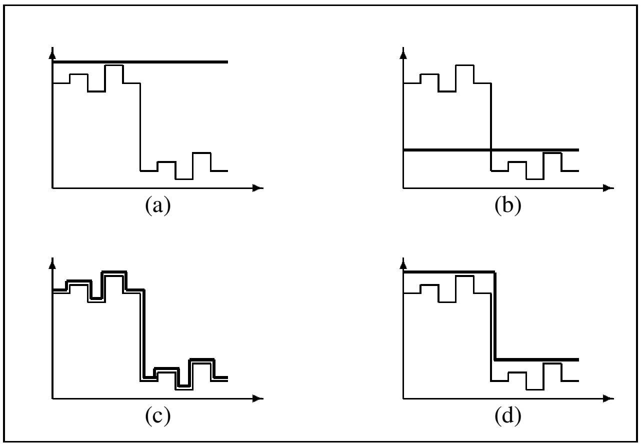

Figure 2: Two static bandwidth allocations: (a) A short delay and a low utilization with a high bandwidth allocation. (b) a high utilization and a long delay with a low bandwidth allocation. Two dynamic bandwidth allocations: (c) a short delay and a high utilization with many bandwidth allocation changes. (d) Few bandwidth allocation changes resulting in "good" utilization and delay.
Thin lines represent bandwidth demand whereas thick lines represent bandwidth allocation.

One extreme is to allocate a lot of bandwidth to each session which results in low utilization and could be costly for the user since pricing may depend on bandwidth consumption. Moreover, the network would be able to support only a few sessions simultaneously. The other extreme is to allocate bandwidth just sufficient to deliver the data. This would result in high latency, due to bursts. Naturally, the user wishes to get a decent latency at a fair price. Nevertheless, the exact tradeoff depends on the user preferences taking into consideration that improving one parameter would increase the cost of the other parameter. The two extremes allocations described above are depicted in Figure 2 (a) and (b).

In dynamic bandwidth allocation a third parameter that influences the cost of the session is the number of bandwidth allocation changes. First, it takes time to setup the modified bandwidth allocation. Second, in today's ATM switches it would normally require the invocation of software in every switch on the session path, which would lengthen the response time even more and consume resources at the switch. Clearly, this would translate also to the price of a bandwidth change. Thus, it is reasonable to assume that when using dynamic bandwidth an additional goal would be to minimize the number of changes. This brings us to another tradeoff in the case of tasks with varying and unpredictable bandwidth requirements. This tradeoff is again illustrated by its extremes. One extreme is not to change the bandwidth and remain with a fixed allocation, this results either in high latency if the fixed bandwidth allocation is minimal or in low utilization if the allocation is maximal (see Figure 2 (a) and (b)). Another extreme is to allocate bandwidth dynamically for each message or packet (see Figure 2 (c)). This might yield "good" utilization and latency, however, the high number of changes would be a burden on the network, and makes such a scheme completely unrealistic. Therefore, from the network perspective it makes sense to try to minimize the number of bandwidth changes while achieving "good" latency and utilization (see Figure 2 (d)). A major problem in achieving such a goal is the online nature of the problem: decisions have to be made without knowing the future requirements.

The tradeoff between the three parameters, utilization, latency and number of changes was studied experimentally in [GKT95, ACHM96]. They compared different heuristics for allocating bandwidth, and ran simulations based on real network traffic. The number of changes was limited by either requiring that the modification be done periodically, or by other means. In this work we provide a theoretical model for the above experimental works, and develop competitive algorithms within this framework.

We believe that our framework keeps the essence of the tradeoffs, while proposing an abstraction which is tractable to analyze. We do not compare the relative price of the three components, but rather fix two parameters (latency and utilization) and optimize the third (number of allocation changes). In our abstraction we consider only end stations and assume that the latency between them is caused only by the queues generated at the end station due to low existing bandwidth allocation. In this paper we focus on the above three parameters and ignore a fourth important parameter – data loss – which is a crucial quality of service parameter. Therefore, we assume that the size of the queues of the end stations are large enough to satisfy the given latency and utilization demand.

1.1 Results

We first study the case of a single session. In a single session we have a single sending end and a single receiving end, however, the arrival rate of bits to the sending end varies unpredictably over time. The single session setting can be viewed as an abstraction for the case where we have end users connected to a network with a limited bandwidth link (say 64Kb). The network can almost surely supply users with any bandwidth in the range, however the users would like to dynamically change the bandwidth in order to minimize its cost while maintaining a reasonable latency.

Our online algorithm has three parameters. The first is B_A the maximum bandwidth it can allocate to the session. The second is D_A an upper bound on the latency. The third is U_A a lower bound on the utilization. Given these three parameters our aim is to minimize the number of bandwidth changes. To ensure a feasible solution we assume that there is a way to serve the session using maximum bandwidth B_A, latency D_A and utilization U_A.[1] To measure the performance of our algorithm we compare it to a clairvoyant (offline) algorithm, i.e., an algorithm that knows the future arrival rate. We impose slightly more stringent requirements on this offline algorithm. We require it to serve the session with latency at most $D_O = D_A/2$ and utilization at least $U_O = 3U_A$ using maximum bandwidth $B_O = B_A$. The number of bandwidth changes made by our online algorithms is guaranteed not to exceed $O(\log B_A)$ or $O(\log(1/U_A))$ times the number of bandwidth changes made by any offline algorithm with the more stringent constraints.

Next we study the case where sessions interact with each other. Namely, there are several sessions all sharing the same bandwidth. This scenario can be viewed as modeling an IP provider that given a fixed amount of bandwidth needs to serve many sessions providing them with a bounded latency.

The parameters in this case are: k the number of sessions, B_A the maximum bandwidth shared by all the k sessions, and D_A the maximum latency. (Here we ignore utilization, our next algorithm takes care of this constraint as well.) Our aim is, as before, to minimize the number of bandwidth changes. Again, to measure the performance of our online algorithm we compare it to an offline algorithm with more stringent constraints. This offline algorithm has to serve the k sessions using total bandwidth $B_O = B_A/4$ (or $B_A/5$), and guaranteeing delay of at most $D_O = D_A/2$. We design online algorithms that make $O(k)$ times more bandwidth changes than any such offline algorithm. Our first algorithm is phased, it makes its

[1] From now on, whenever we consider an algorithm with given constraints we always assume that all the input streams are feasible; i.e., can be served within these constraints.

bandwidth modifications at the beginning of the phase and keeps it fixed within the phase. The second algorithm is continuous, and makes its bandwidth modifications whenever there is a need, and therefore is more natural to implement.

Our last algorithm is a hybrid algorithm combining the algorithms for the single session case and the multi-session case. It considers the case of k sessions, all sharing maximum bandwidth B_A. The delay of each session has to be at most D_A, while the total utilization has to be at least U_A. The goal is again to minimize the number of bandwidth changes while achieving these constraints. This case can be viewed as modeling an IP provider that carries traffic over a public network. The provider is billed according to the total bandwidth consumption and the number of bandwidth changes performed. Thus to minimize the provider's cost both have to be minimized. On the other hand, the users expect a certain performance level, which is manifested in the latency.

We compare our online algorithm to a more stringent offline algorithm with maximum delay $D_O = D_A/2$, minimum total utilization $U_O = 3U_A$, and total bandwidth $B_O = B_O/7$ (or $B_O = B_O/8$). The number of bandwidth changes made by our online algorithm is guaranteed not to exceed $O(k \log B_A)$ or $O(k \log(1/U_A))$ times the number of bandwidth changes made by any offline algorithm.

Remark: In all cases we compare our online algorithms to offline algorithms with more stringent constraints. In other words we allow the online algorithm some "slack" in the delay, utilization, and maximum bandwidth. The slack factors could be different, and actually there exists a tradeoff between these factors. In this paper we focused on solutions in which all of the slack factors are *constant* (for delay, utilization and total bandwidth). In the full version we show some impossibility results which indicate that online algorithms cannot achieve the same parameters as the offline algorithm. Specifically, we prove that if we restrict the online algorithms to have either the same delay and utilization in the single user case, or the same bandwidth and delay in the multi-user case, then the number of bandwidth allocation changes they must make is unbounded. It follows that any online algorithm needs some slack. We also prove that the competitive ratios for the single user case are tight.

1.2 Related work

There is a considerable amount of work on competitive algorithms for bandwidth allocation. We give a brief review here with a special emphasis on the differences between our model and the related models. The works of [AAF+93, AAP93] discuss the case where there are incoming sessions with a given bandwidth requirement. The system needs either to assign a path to a session with the desired bandwidth, or to reject the session. The goal is to maximize the total throughput of the network (i.e., the bandwidth of the accepted sessions). Once a session is accepted and gets its bandwidth allocation, it remains fixed throughout the lifetime of the session. The works of [GG92, GGK+93, BNCK+95, ABFR94, AGLR94] also discuss the case of incoming sessions, each with a certain bandwidth requirement. The difference is that in their model sessions that were accepted may be later preempted. However, preempted sessions (like rejected sessions) do not contribute to the throughput. In contrast to both models, the model in this work assumes that sessions join the network with a certain delay and utilization requirement (rather than a bandwidth requirement). The bandwidth required to maintain the delay and utilization requirements of the sessions is allocated dynamically.

Another related model appears in [SK94, LPR94]. The system they consider has a bound on the number of sessions that can be active simultaneously. Each time the system needs to make a decision which sessions to keep active. The goal is to minimize the number of status changes, i.e., making an active session inactive. In this setting there is no bandwidth requirement and the decision for each session is "binary" (active or not active). We can view our model as an extension of this model, since we adjust dynamically not only the number of active sessions, but also the bandwidth allocated to each session.

2 The single session case

In this section we present an algorithm for the case of a single session. Our goal is to design an algorithm that would deliver the stream of incoming bits within latency at most D_A, would utilize at least U_A fraction of the allocated bandwidth, and would minimize the number of bandwidth changes. Since future bit arrival rates are not known in advance, this problem is of an online nature. To ensure feasibility we assume that the maximum bandwidth B_A, suffices to deliver the arriving bits with latency at most D_A. We prove that the number of changes made by our algorithm is at most $\log(B_A)$ times the number of changes made by any offline algorithm with the same maximum bandwidth $B_O = B_A$, maximum delay $D_O = D_A/2$ and minimum utilization $U_O = 3U_A$.

To simplify the presentation, we assume that B_A is a power of two. Let $\ell_A = \log_2 B_A$. The algorithm works in stages, where each stage is preceded by a RESET operation. Below, we prove that in each stage any offline algorithm that obeys the latency and utilization constraints must make at least one bandwidth allocation change while the online algorithm does at most ℓ_A changes.

Let t_s be the start time of a stage, and let t be any time within the stage. The algorithm maintains two parameters $\mathtt{high}(t)$ and $\mathtt{low}(t)$. Under the assumption that the offline algorithm does not change its bandwidth assignment from time t_s to time t, $\mathtt{high}(t)$ and $\mathtt{low}(t)$ serve as upper and lower bounds on this bandwidth assignment. The value of $\mathtt{high}(t)$ is determined by the utilization constraint, while the value of $\mathtt{low}(t)$ is determined by the latency constraint. Let t_e be the first time in which $\mathtt{high}(t_e) < \mathtt{low}(t_e)$, at this time the current stage ends and a new stage is about to start. We can end the stage because we know that the offline cannot maintain the same bandwidth allocation in the time interval $[t_s, t_e]$, and therefore has performed at least one change in the bandwidth allocation. However, before starting the new stage, we have to make sure that bits belonging to the previous stage are delivered. We achieve this in the RESET operation, in which we allocate B_A bandwidth until the queue is empty, and then the new stage starts.

More formally, whenever a stage is started the queue is empty. At each time unit t within the stage, $\mathtt{low}(t)$ and $\mathtt{high}(t)$ are computed. The bandwidth allocated by the online algorithm, denoted by B_{on}, is set to the smallest power of two that is at least $\mathtt{low}(t)$. If at some time t we get $\mathtt{high}(t) < \mathtt{low}(t)$, the stage is ended, and a RESET operation starts. During the RESET, B_{on} is set to B_A until the queue becomes empty. At this time a new stage begins. Figure 3 is a formal presentation of the algorithm. In the description, we denote the incoming queue of bits by Q. We assume that bits that have to be delivered are put in Q. The algorithm is started by invoking RESET.

We now show how to compute $\mathtt{low}(t)$ and $\mathtt{high}(t)$. At time t_s we set $\mathtt{low}(t_s) = 0$, for any $t > t_s$:

$$\mathtt{low}(t) = \max_{t':\, t_s \le t' \le t} \; \max_{w:\, 0 \le w \le t' - t_s} \left\{ \left\lceil \frac{IN[t'-w, t')}{w + D_O} \right\rceil \right\},$$

where $IN[t'-w, t')$ denotes the number of incoming bits in the time interval $[t'-w, t')$. In words, for each time interval (window) of size w that ends at t' ($t_s \le t' \le t$), we compute the minimum bandwidth required to deliver the bits received at the window within the offline latency D_O. We set $\mathtt{low}(t)$ to be the maximum

```
RESET:
    B_on := B_A
    wait until the first time Q is empty then
        STAGE
STAGE:
    for each time unit t do
        compute low(t) and high(t)
        if high(t) < low(t) then
            RESET
        if B_on < low(t) then
            B_on := min{2^i | 2^i ≥ low(t), i integer}
    end
```

Figure 3: The single session algorithm

among all the resulting bounds. This implies that under the assumption that the offline bandwidth has not changed from time t_s to time t, it must be at least $\mathtt{low}(t)$. Note that to actually compute $\mathtt{low}(t)$ we can use the identity

$$\mathtt{low}(t) = \max\left\{\mathtt{low}(t-1), \max_{0\leq w\leq t-t_s}\left\{\left\lceil\frac{IN[t-w,t)}{(w+D_O)}\right\rceil\right\}\right\} .$$

Utilization: The computation of $\mathtt{high}(t)$ stems from our definition of utilization. Let us diverge now and define utilization. There are two approaches to define utilization: one is global and the other is local. In the global approach the total bandwidth allocated from the time connection was established is compared to the number of bits received during this time. The ratio of these two quantities reflects the overall "waste" in bandwidth, and thus plays a role in measuring the overall performance of the connection. One drawback of such an approach is that it tends to ignore periods of time in which the utilization is very low, as long as these are compensated by other periods of high utilization. Consequently, we prefer to use a local approach in which we fix a time window parameter, denoted W, and bound the ratio of the number of bits received during any time window of size W to the total bandwidth in this time. Formally, the utilization is defined as

$$\min_{t:\, W\leq t}\frac{IN(t-W,t]}{B(t-W,t]}$$

where $IN(t-W,t]$ and $B(t-W,t]$ denote the number of incoming bits and the total allocated bandwidth in the time interval $(t-W,t]$. We expect that the utilization according to the global approach should be higher than the one from the local approach. Generally it is true (and provides the right intuition) although there are some special cases where it does not hold. Also note that the utilization depends on the incoming stream of bits, rather than on the actual bits transmitted. We chose this definition because it limits the algorithm influence over the utilization, and ensures that the utilization is monotone in the bandwidth allocation.

When choosing the parameter W, we would not like it to be too large, or we will suffer the deficiencies of the global approach, on the other hand it should be large enough, or otherwise the flexibility in allocating the bandwidth would be hampered. In this manuscript we assume that $W \leq D_O$. This assumption is not essential, it is done to simplify the presentation. When we consider the online algorithm we again relax the constraints and allow a bigger window up to $W + 5D_O$.

Recall that $\mathtt{high}(t)$ is the largest bandwidth allocation that would still give the desired utilization of U_O with no bandwidth allocation change since the beginning of the stage. Initially, for $t_s \leq t < t_s + W$, we set $\mathtt{high}(t) = B_A$. For any $t \geq t_s + W$:

$$\mathtt{high}(t) = \left\lfloor\frac{1}{U_O\cdot W}\min_{t_s+W\leq t'\leq t}\left\{IN(t'-W,t']\right\}\right\rfloor .$$

Observe that in order for the offline algorithm to maintain utilization U_O, and under the assumption that the bandwidth has not been changed from t_s to t, the bandwidth allocated by the offline cannot exceed $\mathtt{high}(t)$. Note that to actually compute $\mathtt{high}(t)$ we can use the identity

$$\mathtt{high}(t) = \min\left\{\mathtt{high}(t-1), \left\lfloor\frac{IN(t-W,t]}{U_O\cdot W}\right\rfloor\right\} .$$

The discussion above implies that it is feasible for the offline algorithm to maintain the same bandwidth from time t_s to time t only if $\mathtt{high}(t) \geq \mathtt{low}(t)$. Since the stage ends when $\mathtt{low}(t) > \mathtt{high}(t)$, we are guaranteed that the offline algorithm has changed its bandwidth allocation at least once during the stage. Since the bandwidth allocations made by the online algorithm within a single stage (and the following RESET) are monotonically increasing, and are in powers of two, the number of changes made by the online algorithm is at most $\ell_A = \log_2 B_A$.

Lemma 1 *The number of bandwidth allocation changes performed by our algorithm is bounded by ℓ_A times the number of changes done by any offline algorithm with delay D_O and utilization U_O.*

We proceed to prove that the above algorithm has delay $D_A = 2D_O$ and utilization $U_A = U_O/3$.

Claim 2 *Consider any time t, and let Q_{on} and B_{on} be the queue and the online bandwidth allocation at this time. Then, $B_{on} \geq q/D_A$, where q is the size of Q_{on}.*

Proof: Consider the stage that contains t. In case t is within a RESET operation, consider the stage preceding it. Define t_l to be the first time in this stage (or its following RESET) when B_{on} was set to 2^j, for some $j \geq l$. Define t_{-1} to be the start time of the stage. Let i be the maximum index such that $t_i \leq t$.

We prove by induction on the maximum index i that $q \leq 2^i \cdot D_A$. This is vacuously true for $i = -1$. Consider $i > -1$. We claim that the maximum number of bits received in the time interval $[t_i, t)$ is $2^i(t - t_i + D_O)$. To see this recall that $\mathtt{low}(t) \leq 2^i$ which means that allocating bandwidth of 2^i suffices to deliver all the bits received from time t_i to time t (exclusive) within a delay of D_O. (Note that if t is within a RESET this holds since then $2^i = B_A = B_O$.) This implies that these bits have to be delivered up to time $t + D_O$. The maximum number of bits that can be delivered in $t - t_i + D_O$ time units using this bandwidth is $2^i(t - t_i + D_O)$.

Now since B_{on} is set to 2^i at time t_i, it follows that the online algorithm is able to deliver at least $2^i(t - t_i)$ bits in this time interval. Let q' be the queue just before time t_i. We get $q \leq 2^i D_O + q'$. By our induction hypothesis $q' \leq 2^{i-1} \cdot D_A$ and the claim follows since $D_A = 2D_O$. □

Lemma 3 *The delay of the online algorithm is at most D_A.*

Proof: The lemma follows from Claim 2. Consider a bit that enters Q. At that time, we have that $B_{on} \geq q/D_A$, where q is the size of Q. Since the bandwidth is not decreased as long as the queue is not empty, this bit must have been delivered in D_A time units. □

Corollary 4 *Consider any time t, and let q_A be the size of the queue of the online algorithm at this time. Then, $q_A \leq q_O + B_O \cdot D_O$, where q_O is the size of the queue of any offline algorithm with bandwidth B_O and delay D_O.*

Proof: To obtain a contradiction assume that at some time t $q_A > q_O + B_O \cdot D_O$ for some offline algorithm with bandwidth B_O and delay D_O. Note that the offline algorithm can serve additional $B_O \cdot D_O - q_O$ bits at time t with bandwidth B_O delay D_O and utilization U_O. However, these additional bits at time t would increase the size of the queue of the online algorithm to $q_A + B_O \cdot D_O - q_O > 2B_O \cdot D_O$. Implying that the online algorithm cannot serve these additional bits with bandwidth B_O and delay $2D_O$; contradiction to Lemma 3. □

Lemma 5 *The utilization of the online algorithm is at least* $U_A = U_O/3$.

Proof: To prove the lemma we have to show that for any time t there exists a time window $(t - W', t]$, where $W' \leq W + 5D_O$, that satisfies the utilization condition:

$$\frac{IN(t - W', t]}{B_{on}(t - W', t]} \geq \frac{U_O}{3}.$$

We distinguish between several cases depending on the location of the time t.

Case 1: The window of size W that ends at t is contained within a stage. That is, $t_s \leq t - W$ and $t \leq t_e$, where t_s and t_e are the start and end points of this stage. Recall that for all $t_s \leq t \leq t_e$, we have $\texttt{low}(t) \leq \texttt{high}(t)$ and $B_{on}(t) \leq 2\texttt{low}(t)$. Therefore,

$$\begin{aligned} B_{on}(t - W, t] &\leq 2W\texttt{low}(t) \leq 2W\texttt{high}(t) \\ &\leq \frac{2IN(t - W, t]}{U_O} \leq \frac{3IN(t - W, t]}{U_O}. \end{aligned}$$

Note that in this case $W' = W$.

Case 2: The time unit t is within a RESET operation, and it is at least $2D_O$ time units after the *start* of the RESET. Consider the window $WIN = (t - 4D_O, t]$. Divide the window into two sub windows of equal size: $WIN_1 = (t - 4D_O, t - 2D_O]$, and $WIN_2 = (t - 2D_O, t]$ which lies within the RESET operation. Recall that by Claim 3 the delay of the online algorithm is at most $2D_O$. Hence, all of the bits delivered in the window WIN_2 arrived in WIN. During a RESET operation the bandwidth is fixed to B_A and hence $B_{on}(t - 2D_O, t] = 2D_O \cdot B_A$, which implies $IN(t-4D_O, t] \geq 2D_O \cdot B_A$. Clearly, $B_{on}(t-4D_O, t-2D_O] \leq 2D_O \cdot B_A$. It follows that

$$\frac{IN(t - 4D_O, t]}{B_{on}(t - 4D_O, t]} \geq \frac{2D_O \cdot B_A}{4D_O \cdot B_A} = \frac{1}{2}.$$

This implies that the utilization of the online algorithm is at least $1/2$. Since the utilization of the offline algorithm, U_O, is at most one, it follows that this utilization is greater than $U_O/3$. Note that in this case $W' \leq 4D_O$.

Case 3: The time unit t is within a RESET operation, and it is less than $2D_O$ time units after the *start* of the RESET. Let t_r be the start time of the RESET operation. Let D' be the smallest multiple of W that is at least $2D_O$. By our assumption $W \leq D_O$, and thus $D' < 3D_O$. Consider the time interval $(t_r - D', t_r]$. If it is not contained within a stage, change $t_r - D'$ to be the start time of the stage ending at t_r (that is, $D' = t_r - t_s$, where t_s is the start time of the stage ending at t_r). Consider the window $WIN = [t_r - D', t]$, of size less than $5D_O$. Divide it into two sub windows: $WIN_1 = (t_r - D', t_r]$, and $WIN_2 = (t_r, t]$. Since the delay of the algorithm is bounded by $2D_O$ (by Lemma 3), and since a stage always starts with an empty queue, all the bits delivered in WIN_2 arrived in WIN. Recall that during a RESET operation the bandwidth is fully utilized and hence $B_{on}(t_r, t] \leq IN(t_r - D', t]$. For $1 \leq i \leq \lfloor D'/W \rfloor$, similar to Case 1,

$$B_{on}(t_r - iW, t_r - (i-1)W] \leq \frac{2IN(t_r - iW, t_r - (i-1)W]}{U_O}.$$

Summing these inequalities we get

$$B_{on}(t_r - \lfloor D'/W \rfloor \cdot W, t_r] \leq \frac{2IN(t_r - \lfloor D'/W \rfloor \cdot W, t_r]}{U_O}.$$

Now, in case $t_r - D' = t_s$ we still need to bound $B_{on}(t_s, t_1]$, where $t_1 = t_r - \lfloor D'/W \rfloor \cdot W$. Note that $t_1 - t_s = D' - \lfloor D'/W \rfloor \cdot W < W \leq D_O$. From the algorithm it follows that in the first W time units of a stage the bandwidth is at most $2IN(t_s, t_1]/D_O$. This implies that $B_{on}(t_s, t_1] \leq 2IN(t_s, t_1]$. Summing all the bounds for the case $t_r - D'$ is not a start time of a stage we get

$$B_{on}(t_r - D', t] \leq \frac{IN(t_r - D', t](U_O + 2)}{U_O},$$

and for the case $t_r - D'$ is a start time of a stage we get

$$\begin{aligned} B_{on}(t_s, t] &\leq 2IN(t_s, t_1] + \frac{2IN(t_1, t]}{U_O} + IN(t_s, t] \\ &\leq \frac{IN(t_s, t](U_O + 2)}{U_O}. \end{aligned}$$

Since $U_O \leq 1$, in both cases we that that the utilization is no less than $U_O/3$. Note that in this case $W' \leq 5D_O$.

Case 4: The time unit t is less than W time units after the start of a stage. Let t_s be the start time of the stage. Consider $B_{on}(t_s, t]$. From the algorithm it follows that in the first W time units of a stage the bandwidth is at most $2IN(t_s, t]/D_O$. This implies that $B_{on}(t_s, t] \leq 2IN(t_s, t]$. By the two previous cases we know that there exists a window of size $W' \leq 5D_O$, such that $B_{on}(t_s - W', t_s] \leq 3IN(t_s - W', t_s]/U_O$. We get

$$\begin{aligned} B_{on}(t_s - W', t] &\leq \frac{3IN(t_s - W', t_s]}{U_O} + 2IN(t_s, t] \\ &\leq \frac{IN(t_s - W', t]}{U_O/3}. \end{aligned}$$

Note that in this case $W' \leq W + 5D_O$. □

We conclude with the main result of this section.

Theorem 6 *Our online algorithm uses maximum bandwidth B_A, has maximum delay D_A and minimum utilization U_A. The number of bandwidth allocation changes it performs is $O(\log B_A)$ times the number of changes performed by any offline algorithm with maximum bandwidth B_A, delay $D_O = D_A/2$ and utilization $U_O = 3U_A$.*

So far we considered only *local* utilization. In the full version we show that our algorithm would have the same performance also under *global* utilization. To complement this positive result we prove in the full version that any online algorithm with global utilization $O(U_O)$ and finite delay must have competitive ratio $\Omega(\log B_A)$. Things are different for *local* utilization. First, observe that for a fixed W and D_O, within any stage, if $t \geq t_s + W$ (where t_s is the start time of the stage), then $\texttt{high}(t)/\texttt{low}(t) = O(U_O)$. Using this observation we modify our algorithm and prove the following.

Theorem 7 *Our modified algorithm has delay $O(D_O)$, utilization $\Omega(U_O)$, and the number of bandwidth changes it performs is $O(\log(1/U_O))$ times the number of changes any offline algorithm with delay D_O and (local) utilization U_O must perform.*

3 The multi-session case

In this section we present two algorithms for the case of many sessions that are sharing the same channel. Let $k \geq 2$ be the number of sessions. We first assume that there is no utilization constraint. In the next section we show how to combine the single session algorithm with the multi-session algorithm taking into consideration the utilization constraint as well.

The multi-session case is different from the single session case since all the sessions have to share the same bandwidth pool, and no session owns the whole available bandwidth. Consequently, it may be that the only way to increase the bandwidth of a certain session is by decreasing the bandwidth allocation of some other session. In this section we show how to share the bandwidth among the sessions without too many allocation changes while still bounding the delay of each session.

We measure the performance of our algorithm against an offline allocation that delivers all bits in delay at most D_O using bandwidth B_O. We call such an algorithm a (B_O, D_O)-algorithm. Out of all the (offline) (B_O, D_O)-algorithms, consider the algorithm $\mathcal{O}$ with the least number of bandwidth changes. Our goal is to minimize the ratio of the number of bandwidth changes made by our online algorithm to the number of bandwidth changes made by algorithm $\mathcal{O}$. Our online algorithms have more resources, they use more bandwidth than B_O and suffer larger delay than D_O.

Before describing the algorithms, it might be helpful to discuss the following trivial solutions. The first algorithm is an online (kB_O, D_O)-algorithm. This algorithm allocates each session bandwidth B_O and therefore has optimal delay and no changes. The clear drawback of this solution is the huge waste in the bandwidth; whereas our online algorithms use only a constant factor more in the bandwidth and suffer only a constant factor more in the delay. The second simple solution is working in phases. During each D_O time units the algorithm stores all the arriving bits. Then it allocates bandwidth B_O in the next D_O time units to deliver all those bits. This solution incurs delay $2D_O$ and uses bandwidth $2B_O$. (This is true since as shown later, in a window of size D_O, at most $2D_O \cdot B_O$ bits could arrive.) However, the number of bandwidth changes is unbounded since the relative size of the multiple sessions may vary from phase to phase. Our online algorithms exploit a variation of this idea while bounding the number of allocation changes.

Below, we describe two algorithms: a phased algorithm and a continuous algorithm. The first performs the allocation changes once in any D_O time slots and the second performs these changes upon demand. Although the bounds achieved for the continuous algorithm are slightly inferior, we believe that it is more "natural" and better suited for implementation.

3.1 The phased algorithm

The algorithm uses a total bandwidth of $B_A = 4B_O$. This bandwidth is divided into two conceptual channels, the first is the *regular channel*, which has bandwidth $2B_O$ and the second is the *overflow channel* that has bandwidth $2B_O$. The bandwidth allocated to session i is B_i and is composed from an allocation in the regular channel B_i^r, and in the overflow channel B_i^o, i.e., $B_i = B_i^r + B_i^o$.

We assume that initially all queues are empty and all allocations are set to zero. The algorithm works in stages, where each stage is preceded by a RESET operation. Each stage is divided into phases of D_O time units each. We denote the queue of session i by Q_i. This queue consists of the queue in the regular channel (the *regular queue*), Q_i^r, and the queue in the overflow channel (the *overflow queue*), Q_i^o, where $Q_i = Q_i^r \cup Q_i^o$. When session i sends bits to be delivered they are put in Q_i^r. From time to time, the queue Q_i^r is emptied and all its content is moved to the queue Q_i^o. Let $|Q_i|$ denote the size of Q_i.

Each stage starts by allocating equal amounts of bandwidth B_O/k to the regular channels of each session. The algorithm runs in phases of D_O time units each. At the end of each phase the algorithm tests whether there is a need to increase the bandwidth allocation of the regular channel of any session i. Such an allocation has to be increased if the bandwidth allocated to the regular channel of session i is not enough to deliver the content of its regular queue in D_O time units, i.e., $|Q_i^r| > B_i^r \cdot D_O$. If the allocation has to be increased the algorithm adds B_O/k to the regular allocation of session i, i.e., $B_i^r = B_i^r + B_O/k$. It then empties the queue Q_i^r and moves its content to the overflow queue Q_i^o and allocates enough bandwidth to the overflow channel that would suffice to empty the overflow queue in the next phase, i.e., $B_i^o = |Q_i^o|/D_O$. If at the end of a phase the regular bandwidth allocation need not be changed (i.e., $|Q_i^r| \le B_i^r \cdot D_O$), then the overflow bandwidth is set to zero. We prove below that at this point the overflow queue is empty.

After updating the bandwidth allocation for all sessions, the algorithm tests if the total regular bandwidth allocation exceeds $2B_O$. In this case, the algorithm empties all the regular queues into the overflow queues, allocates enough overflow bandwidth to deliver the overflow queues in D_O time units, and enters a RESET state, that starts a new stage. (Figure 4 depicts a formal description of the algorithm.)

```
RESET:
    for i = 1 to k do
        B_i^r := B_O/k
    end
PHASE:
(Done every D_O time units starting D_O time units after RESET)
    for i = 1 to k do
        if |Q_i^r| ≤ B_i^r · D_O then
            B_i^o := 0
            (We show later that at this point |Q_i^o| = 0.)
        else (|Q_i^r| > B_i^r · D_O)
            B_i^r := B_i^r + B_O/k
            Move the content of Q_i^r to Q_i^o
            B_i^o := |Q_i^o|/D_O
        endif
    end
    if Σ_{i=1}^k B_i^r > 2B_O then
        for i = 1 to k do
            Move the content of Q_i^r to Q_i^o
            B_i^o := |Q_i^o|/D_O
        end
        RESET
    endif
```

Figure 4: The phased algorithm

To prove the correctness of the algorithm, we first argue that in each phase the allocated bandwidth is enough to serve all the bits that were accumulated in all the queues at the beginning of the phase.

Claim 8 *Let Q_i be the queue of session i at the beginning of a phase, and let B_i be the bandwidth allocated to session i in the phase. Then $B_i^r \cdot D_O \ge |Q_i^r|$ and $B_i^o \cdot D_O \ge |Q_i^o|$.*

Proof: The proof is by induction on the phases. The claim clearly holds for the initial phase since the queue of session i is empty. We distinguish between two types of phases: the first phase after a RESET and any other phase. Consider a first phase after a RESET. At the beginning of such a phase Q_i^r is empty, and B_i^o is set to $|Q_i^o|/D_O$, so the claim holds. Consider any other phase.

Case 1: The regular bandwidth of session i was not changed. In this case, $B_i^r \cdot D_O \ge |Q_i^r|$ and no bits were added to the overflow queue. By the inductive hypothesis it follows that the overflow queue is empty. This is because the overflow allocation in the previous phase was enough to deliver all its content.

Case 2: The regular bandwidth of session i was changed. In this case, at the beginning of the phase Q_i^r is empty, and B_i^o is set to $|Q_i^o|/D_O$. □

To bound the bandwidth and the delay of the online algorithm, we next bound the number of bits arriving in each phase.

Claim 9 *During any time interval $[t, t+\Delta)$ at most $(\Delta + D_O)B_O$ bits could arrive for all the sessions together.*

Proof: We know that there exists an offline (B_O, D_O)-algorithm for the input instance. During the Δ time units in the interval $[t, t+\Delta)$ this algorithm can deliver at most $\Delta \cdot B_O$ bits. In order to maintain that the delay is bounded by D_O, the rest of the bits arriving in this interval must be delivered in the next D_O time units. This accounts for at most $D_O \cdot B_O$ additional bits. □

The fact that the bandwidth of the regular channel is at most $2B_O$ follows from the definition of the algorithm. The following lemma bounds the bandwidth of the overflow channel.

Lemma 10 *At any time t, $\sum_{i=1}^{k} B_i^o \leq 2B_O$.*

Proof: It is sufficient to show that the claim holds at the beginning of each phase because bandwidth allocation changes are done only then. Consider a phase. Since enough bandwidth was allocated to deliver all the bits in the overflow queue in D_O time units, it follows that at the end of the previous phase $Q_i^o = 0$ for any session i. Therefore, all the bits that are in the overflow queues at the beginning of the phase arrived during the previous phase. By Claim 9 the total number of bits arrived in the previous phase is bounded by $2D_O \cdot B_O$. The lemma follows from the fact that $\sum_{i=1}^{k} B_i^o = \sum_{i=1}^{k} |Q_i^o|/D_O \leq 2B_O$. □

Next, we bound the delay incurred by the phased algorithm.

Lemma 11 *Any bit arriving at time t is delivered by time $t + D_A$, where $D_A = 2D_O$.*

Proof: Consider a bit b which arrives in a certain phase. If it is delivered in the same phase, then the delay is at most D_O. Otherwise, at the end of the phase, the bit is left in the regular queue only if the regular channel is capable of delivering it in the next D_O time units. Otherwise, this bit is moved to an overflow queue. Since the overflow queue is emptied by the end of the next phase, b will be delivered in the next phase which terminates before time $t + 2D_O = t + D_A$. □

Finally, we compute the competitiveness in number of bandwidth allocation changes. The first lemma bounds the number of bandwidth changes made by the online algorithm in a stage, and the second lemma states that any offline (B_O, D_O)-algorithm must have made at least one change in a stage.

Lemma 12 *The number of bandwidth changes made by the online algorithm in any stage is at most $3k$.*

Proof: Consider a stage. The stage is started by making k bandwidth changes; allocating B_O/k as the regular bandwidth of each session. This totals to bandwidth B_O. The stage is ended when the total regular bandwidth allocated is more than $2B_O$. Since each time the algorithm changes a regular bandwidth it increments it by B_O/k, the number of additional regular bandwidth changes is k. (Note that the way the algorithm is described currently, it makes $k+1$ changes. However, the algorithm can be modified to make only k changes.) Each time the regular bandwidth is incremented the overflow bandwidth is changed simultaneously as well. The overflow bandwidth may be changed again to zero a phase later, which accounts to an additional k changes. We get a total of at most $3k$ changes. □

Now we show that between two times the algorithm performs a RESET, any offline algorithm must have made at least one change to its bandwidth allocation.

Lemma 13 *In any stage, any (B_O, D_O)-algorithm has to change the bandwidth allocation at least once.*

Proof: To obtain a contradiction, assume that there is a single allocation that can sustain for more than one stage. Let OFF_i be the bandwidth allocation of the offline algorithm to session i. It follows that $\sum_{i=1}^{k} OFF_i \leq B_O$.

We claim that at any time the regular bandwidth allocated by the online algorithm to session i is at most B_O/k more than OFF_i. We prove this claim inductively. This clearly holds for the initial online allocation of B_O/k. Consider the time t in which the regular bandwidth allocation of session i was incremented by B_O/k. This was done at the end of a phase in which $Q_i^r > B_i^r \cdot D_O$, where B_i^r was the bandwidth before the increment. Let t_0 be the time in the stage in which the online algorithm set the regular bandwidth of session i to B_i^r. Note that at time t_0, the regular queue Q_i^r was emptied. Since we have that at time t, $Q_i^r > B_i^r \cdot D_O$, we must have that there exists some $t_0 \leq t' < t$ such that more than $B_i^r(t - t' + D_O)$ bits arrived for session i in the time interval $[t', t)$. By Claim 9, this implies that $OFF_i > B_i^r$.

The lemma now follows since at the end of the stage the total regular bandwidth allocated by the online algorithm is more than $2B_O$, implying that the total bandwidth allocated by the offline algorithm is more than B_O; a contradiction. □

We can now derive the theorem regarding the phased algorithm.

Theorem 14 *The online phased algorithm is a (B_A, D_A)-algorithm. The number of bandwidth allocation changes it performs is at most $3k$ times the number of changes performed by any offline (B_O, D_O)-algorithm, where $B_A = 4B_O$ and $D_A = 2D_O$.*

Remark: In many cases it is important to maintain the order among the bits of the same session (FIFO). Although the algorithm maintains two queues, it can maintain a single queue that will preserve the order. We simply empty the overflow queue before the regular queue, using the entire bandwidth B_i. The worst case delay remains $D_A = 2D_O$, since FIFO policy always outperforms any other policy in this regard. The rest of the claims hold by "renaming" the bits in order to maintain the values of Q_i^r and Q_i^o. That is, for the algorithm it does not matter which bits are in which queues. Therefore, we let the FIFO dictates which bits to deliver and then we move bits between the two queues such that their size would be as the original algorithm dictates.

3.2 The continuous algorithm

The online continuous algorithm uses a total bandwidth of $B_A = 5B_O$. Again, this bandwidth is virtually divided into a *regular channel*, which has bandwidth $2B_O$ and an *overflow channel* that has bandwidth $3B_O$. We use the same notations as in the phased algorithm: B_i^r, B_i^o, Q_i^r, and Q_i^o. Initially all queues are empty and all allocations are set to zero. The algorithm works in stages, where each stage is preceded by a RESET operation.

In the continuous algorithm, the test whether there is a need to modify the bandwidth allocation of the regular channel of session i is done during the stage, whenever bits are added to the regular queue of session i. The allocation has to be changed if $|Q_i^r| > B_i^r \cdot D_O$. In this event, the algorithm sets $B_i^r = B_i^r + B_O/k$, empties the queue Q_i^r and moves its content to the overflow queue Q_i^o, Then it sets $B_i^o = q/D_O + B_i^o$, where q is the original size of Q_i^r. After D_O time units the algorithm reduces this allocation, i.e., $B_i^o =$

$B_i^o - q/D_O$. After increasing the regular bandwidth allocation, the algorithm tests if the total regular bandwidth allocation exceeds $2B_O$. In this case, the algorithm empties all the regular queues into the overflow queues, allocates enough overflow bandwidth to deliver the overflow queues in D_O time units, and enters a RESET state that starts a new stage.

Figure 5 depicts a formal description of the algorithm. Note that in reality there is no need to test the condition in TEST(i) every time bits are added to Q_i^r. The frequency of the tests depends on the values of D_O and B_O.

```
RESET:
    for i = 1 to k do
        B_i^r := B_O/k
    end
TEST(i): (Done every time bits are added to Q_i^r)
    if |Q_i^r| > B_i^r · D_O then
        B_i^r := B_i^r + B_O/k
        q := |Q_i^r|
        Move the content of Q_i^r to Q_i^o
        B_i^o := B_i^o + q/D_O
        REDUCE(i, D_O, q/D_O)
    endif
    if Σ_{i=1}^k B_i^r > 2B_O then
        for i = 1 to k do
            q := |Q_i^r|
            Move the content of Q_i^r to Q_i^o
            B_i^o := B_i^o + q/D_O
            REDUCE(i, D_O, q/D_O)
        end
        RESET
    endif
REDUCE(i, D, B):
    wait D time units then
        B_i^o := B_i^o - B
```

Figure 5: The continuous algorithm

The proof that the ratio of the number of changes made by the continuous algorithm to the number of changes made by the optimal (B_O, D_O)-algorithm is $3k$ is identical to the proof for the phased algorithm. Next we bound the delay of the algorithm.

Lemma 15 *Any bit sent in time t arrives by time $t + D_A$, where $D_A = 2D_O$.*

Proof: Consider a bit b that was added to Q_i^r at time t. We distinguish between two cases.
Case 1: The bandwidth B_i^r was not incremented in the time interval $[t, t + D_O)$, and there was no RESET during this interval. In this case, during this time interval the content of Q_i^r was not transferred to Q_i^o. Let $|Q|$ be the maximum size of Q_i^r during this time interval. We have also that $B_i^r \geq |Q|/D_O$ in the entire interval. This implies that b is delivered within this time interval.
Case 2: At time $t' \in [t, t + D_O)$ either the bandwidth B_i^r was incremented or there was a RESET. In this case, at time t' the content of Q_i^r was moved to Q_i^o. The algorithm guarantees that the total overflow bandwidth in the interval $[t', t' + D_O)$ is at least the original size of Q_i^o plus the original size of Q_i^r divided by D_O. Therefore, the bit b is delivered by time $t' + D_O \leq t + 2D_O$, and the lemma follows. □

The total bandwidth used is simply the sum of the bandwidth of the regular channel and the overflow channel. The fact that the bandwidth of the regular channel is at most $2B_O$ follows from the definition of the algorithm. The following lemma bounds the bandwidth of the overflow channel.

Lemma 16 *At any time t, $\sum_{i=1}^{k} B_i^o \leq 3B_O$.*

Proof: It is sufficient to show that the lemma holds whenever bits are added to any of the queues Q_i^o, since only then B_i^o is increased. Consider session i, and let t_j be a time when B_i^o was incremented by bandwidth B. Let t_{j-1} be the previous time when B_i^o was incremented (or $t_{j-1} = 0$ if there is no such prior increment). Note that in the time interval $(t_{j-1}, t_j]$ the regular bandwidth B_i^r remained the same. Since Q_i^r is emptied at t_{j-1}, and since during the time interval (t_{j-1}, t_j) the size of Q_i^r was no more than $B_i^r \cdot D_O$, the value of B is at most the number of bits received in the time interval $(\max\{t_{j-1}, t_j - D_O\}, t_j]$ divided by D_O. The value of B_i^o at any time t is composed of the increases that occurred in the time interval $(t - D_O, t]$. Similar to the way we bound B, we can bound the sum of the increases by the number of bits that arrived for session i in the time interval $(t - 2D_O, t]$ divided by D_O. When we sum over all sessions we get that the total bandwidth in the overflow channel is bounded by the total number of bits that arrived in the interval $(t - 2D_O, t]$ divided by D_O. By Claim 9 the total number of bits that arrived in the interval $(t - 2D_O, t]$ is bounded by $3D_O \cdot B_O$. This bounds the total bandwidth in the overflow channel at any time t by $3B_O$. □

Theorem 17 *The online continuous algorithm is a (B_A, D_A)-algorithm. The number of bandwidth allocation changes it performs is at most $3k$ times the number of changes performed by any offline (B_O, D_O)-algorithm, where $B_A = 5B_O$ and $D_A = 2D_O$.*

Finally, the remark regarding the FIFO property holds for the continuous algorithm as well.

4 The combined algorithm

In this section we combine the single and multi-session scenarios discussed in the previous two sections. We consider a system with $k > 1$ sessions who are sharing the same channel. However, the total bandwidth used by these sessions is not fixed and has to satisfy a utilization constraint. The utilization is measured as combined utilization of all sessions, namely we compare the total allocated bandwidth and the total number of bits sent, rather than doing it for each session separately. We measure the performance of our online algorithm against an offline algorithm that delivers the streams of incoming bits for each of the k sessions using bandwidth B_O with latency at most D_O, and utilization at least U_O. (From now on in this section, whenever we refer to offline algorithms we refer only to offline algorithms with the above characteristics.) We describe an online algorithm that delivers the streams of incoming bits for each of the k sessions using $B_A = 7B_O$ bandwidth (or $B_A = 8B_O$ if the continuous multi-session algorithm is used), with latency at most $D_A = 2D_O$, and utilization at least $U_A = U_O/3$. The goal is to minimize the number of bandwidth changes done by the algorithm.

There are two types of bandwidth changes: changes of the total bandwidth allocation (global changes), and changes of bandwidth allocation to specific sessions (local changes). We compare the performance of our algorithm (i.e., number of local and global bandwidth allocation changes) to the performance of an offline algorithm with the characteristics stated above.

By combining our algorithms for the single session and the multi-session, we achieve an online algorithm that changes the total bandwidth at most $\log(B_A)$ times the number of global changes made by any offline algorithm, and changes the bandwidth of the sessions $O(k \log(B_A))$ times the number of local changes made by any offline algorithm. Again, we assume that B_A is a power of two and let $\ell_A = \log_2 B_A$.

We now present an informal description of the algorithm. The algorithm works in global stages, where each stage is preceded by a GLOBAL RESET operation. Below, we show that in each global stage any offline algorithm must have made at least one global bandwidth allocation change while the online algorithm does at most ℓ_A changes. Each global stage is divided into local stages. In each local stage the total bandwidth is unchanged and the only changes are in the bandwidth allocation to the sessions. We prove that in each local stage any offline algorithm must have made at least one local change, while the online makes $O(k\ell_A)$ such changes.

Consider a global stage and let t_s be its start time. The algorithm maintains two values $\texttt{high}(t)$ and $\texttt{low}(t)$. As in the single session case, under the assumption that the offline algorithm does not change its total bandwidth assignment from time t_s to time t, the values $\texttt{high}(t)$ and $\texttt{low}(t)$ are upper and lower bounds on this bandwidth assignment. Note that for computing $\texttt{high}(t)$ and $\texttt{low}(t)$ the algorithm considers the total bandwidth allocated and the total bits arrived for all the sessions. As before, the first time t_e in which $\texttt{high}(t_e) < \texttt{low}(t_e)$, the stage ends, and a GLOBAL RESET operation is started in which all the queues are emptied. Unlike the single user case a new global stage begins simultaneously with the GLOBAL RESET operation.

Each global stage is composed of local stages. Each such stage is identical to the one described in the multi-session algorithm. The only difference is that the value B_O used in the stage is the current value of B_{on} computed as in the single session algorithm. A local stage ends if one of the following conditions holds: (1) A GLOBAL RESET is started. (2) The value of B_{on} has to be changed. (3) The condition of the multi-session algorithm holds; i.e., $\sum_{i=1}^{k} B_i^r > 2B_O$.

The GLOBAL RESET is different from the RESET operation in the single session case. When the GLOBAL RESET starts the content of the sessions' queues are moved to a global overflow queue that is served by a global overflow channel of size $2B_O$. At the same time we start a new global stage using the remaining bandwidth. When these queues are processed by the global overflow channel, the channel is allocated proportionally among the sessions' queues. We have to show that $2B_O$ bandwidth would suffice to deliver the content of the global overflow queue with delay D_O. (Note that many such GLOBAL RESET operations may occur within D_O time units.) Observe that whenever the global overflow channel is fully utilized (which is always the case when it is used to empty the sessions' queues) its queue cannot exceed $2B_O \cdot D_O$ since this would indicate that the size of the queue of the offline algorithm is more than $B_O \cdot D_O$ (by Corollary 2); a contradiction.

The proofs of the claims bounding the delay, utilization, and number of changes are similar to the respective proofs in the single- and multi-session cases.

References

[AAF+93] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proc. 23rd ACM Symp. on Theory of Computing*, pages 623 – 631, 1993.

[AAP93] B. Awerbuch, Y. Azar, and S. Plotkin. Throughput-competitive of on-line routing. In *Proc. 34th IEEE Symp. on Foundations of Computer Science*, pages 32 – 40, 1993.

[ABFR94] B. Awerbuch, Y. Bartal, A. Fiat, and A. Rosén. Competitive non-preemptive call control. In *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 312 – 320, 1994.

[ACHM96] Y. Afek, M. Cohen, E. Haalman, and Y. Mansour. Dynamic bandwidth allocation. In *Proc. IEEE INFOCOM*, pages 880 – 887, 1996.

[AGLR94] B. Awerbuch, R. Gawlick, T. Leighton, and Y. Rabani. On-line admission control and circuit routing for high performance computing and communication. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 412 – 423, 1994.

[BNCK+95] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber. Bandwidth allocation with preemption. In *Proc. 25th ACM Symp. on Theory of Computing*, pages 616 – 625, 1995.

[CRS95] I. Cidon, R. Rom, and Y. Shavit. Analysis of one way reservation algorithms. In *Proc. IEEE INFOCOM*, pages 1256 – 1263, 1995.

[GG92] J.A. Garay and I.S. Gopal. Call preemption in communication networks. In *Proc. IEEE INFOCOM*, pages 1043–1050, 1992.

[GGK+93] J.A. Garay, I. S. Gopal, S. Kutten, Y. Mansour, and M. Yung. Efficient on-line call control mechanism. In *Proc. 2nd Israeli Conf. on Theory of Computing and Systems*, pages 285–293, 1993.

[GKT95] M. Grossglauser, S. Keshav, and D. Tse. RCBR: A simple efficient service for multiple time-scale traffic. In *Proc. ACM SIGCOM*, pages 219 – 230, 1995.

[LPR94] C. Lund, S. Phillips, and N. Reingold. IP over connection-oriented networks and distributional paging. In *Proc. 35th IEEE Symp. on Foundations of Computer Science*, pages 424 – 434, 1994.

[SK94] H. Saran and S. Keshav. An empirical evaluation of virtual circuits holding times in IP-over-ATM networks. In *Proc. IEEE INFOCOM*, 1994.