Discrete Broadcasting Protocols for Video-on-Demand

Chao Peng[1] ⋆ Hong Shen[1] Naixue Xiong[1] Laurence T. Yang[2]

[1]Graduate School of Information Science
Japan Advanced Institute of Science and Technology
1-1 Tatsunokuchi, Ishikawa, 923-1292, Japan
[2] Department of Computer Science,
St. Francis Xavier University, Antigonish, B2G 2W5, Canada

Abstract. The Video-on-Demand (VOD) service allows users to view any video program from a server at the time of their choice. Broadcasting protocols can be used to improve the efficiency of a VOD system. The Harmonic Broadcasting Protocol has been proved to be bandwidth-optimal, but it is not efficient for the local storage. In this paper, we present the Discrete Broadcasting scheme, which can intelligently adjust its solution according to available bandwidth and local storage in order to achieve an ideal waiting time.

1 Introduction

In a Video-on-Demand (VOD) system, a subscriber is expected to be able to watch his favorite video program in the server at the time of his choice. Usually such a system is implemented by a client-server architecture supported by certain transport networks such as CATV, telecom, or satellite networks. Clients use web browsers or set-top-boxes (STB) on their television sets. In a pure VOD system, each user is assigned a dedicated video channel so that they can watch the video they choose without delay and many VCR-like functions may be provided. But in this case the cost is too expensive, because it will quickly use up all of the available bandwidth on the VOD server when too many concurrent users are to be accommodated at the same time.

To reduce the tremendous bandwidth and I/O requirements, many alternatives have been proposed by sacrificing some VCR functions. Broadcasting is one of such techniques and is mostly appropriate for popular videos that are likely to be simultaneously watched by many viewers [9]. In this approach, the server uses multiple dedicated channels to execute frequent retransmissions of a "hot" video. Each client follows some reception rules to grab and store data from appropriate channels so as to play the whole video continuously. What distinguishes broadcasting from other VOD distribution methods is that the server's

⋆ Supported by "Fostering Talent in Emergent Research Fields" program in Special Coordination Funds for promoting Science and Technology by Ministry of Education, Culture, Sports, Science and Technology. E-mail: p-chao@jaist.ac.jp.

broadcasting activity is independent of the number of viewers. The bandwidth savings can be considerable since a few (10 or 20) very popular videos are likely to account for nearly 80% of demands [2].

The simplest solution is to periodically broadcast the video on several channels, each differentiated by some time. By this method the server needs at least K channels in order to keep the waiting time below L/K (here L is the length of the whole video). To enhance the efficiency in channel usage, many schemes [2, 4–8, 10] have been proposed by imposing a large enough client receiving bandwidth and an extra buffering space at the client side.

In this paper, we present the *Discrete Broadcasting Protocol*(DB), which can intelligently adjust its solution according to available resources such as available channels and local storage. It can reduce the average waiting time of a 120 minutes video to 3 minutes if we allocate a bandwidth of 4 times the consumption rate. It can also be modified for VOD service even when the local storage is very small. Some of our results have already been realized in industrial applications and have got a good performance.

2 Model and Analysis of VOD Broadcasting Protocols

The broadcasting problem in the VoD service can be described as follows: In a VOD system, given a video of size S (Mb) and consumption rate c (Mb/s), if the available bandwidth on the VOD server is B, the endurable delay for the client is D and the available storage size of the client is M, we should find a broadcasting scheme which can satisfy these tree constraints.

Usually we will bound the storage m and the delay d but minimize the bandwidth b. We can also bound m and b but minimize d, but the method is the same. What is different is the storage issue, yet it is often assumed to be unlimited and been neglected. The following are the parameters we need to consider in a VOD system.

The parameters of a given video	
L	The length of a video program, in seconds.
S	The size of a video program, $S = L \cdot c$, in Mb.
c	The consumption rate, in Mb/s.

Performance parameters of the system	
d	The max delay for any client, in seconds.
b	The bandwidth needed for the server, in Mb/s.
m	The maximum storage used by any client, in Mb.

Constraint parameters of the system	
D	The endurable maximum delay , in seconds.
B	The available bandwidth of the server, in Mb/s.
M	The minimum local storage size, in Mb.

Fig. 1 illustrates the basic ideas of VOD. Here we use a single tape with length L and width c to denote a whole video which is Constant Bit Rate (CBR) encoded,

so its size is $S = L * c$. At the server side, we will use a channel of bandwidth b to broadcast this video. Suppose the client sends a request for this video at time t_{req}, and starts to consume this video at time t_0, then the period between this two points is the delay $t_0 - t_{req}$ of this user. The maximum delay experienced by any client of a video is the viewing delay d of this video.

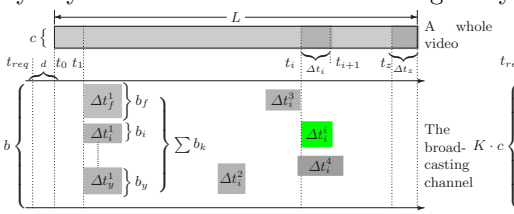

Fig. 1. A general framework for VOD

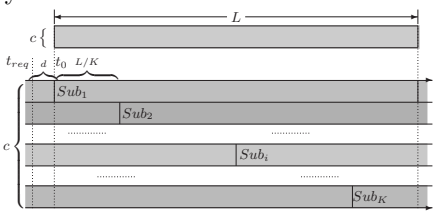

Fig. 2. The SBP Protocol

To improve the efficiency, we can divide a whole video into small segments (the segment starts at time t_i is denoted as Δt_i) and arrange these segments into the broadcasting channel according to a certain schedule. To guarantee that a client can watch the video smoothly, the required segment Δt_i must be already in the storage of the client's STB at time $t_{req} + d + t_i$.

Thus we need to make sure that the segment Δt_i can be downloaded during the period from t_{req} to $t_{req} + d + t_i$. Sometimes there may be more than one such segments during this period, for example, there are three Δt_i during the period from t_{req} to $t_{req} + d + t_i$ in Fig. 1. The client can choose to download the last appearance Δt_i^3 if he knows the schedule of all segments at the time he starts to download, for this may in some cases decrease the storage requirement. But in most cases he doesn't have such knowledge, then he has to download it at its first appearance Δt_i^1 and store it for future consumption. For a certain video, we can calculate the storage requirement. Refer to the shadow tape of the video in Fig. 1, at time t_i, all segments at the left side have been consumed and can be cleared from the storage, but some segments at the right side may have been downloaded or partially downloaded and they will stay in the storage until they are consumed. The total volume of these segments will reach a maximum value at some time, so it will be the storage requirement for users enter at t_{req}. Then the largest value among all users enter at different time is the minimum storage requirement m for this video.

As an example, the *Staggered Broadcasting Protocol* (SBP) in [1] rebroadcasts the whole video on $b/c = K$ distinct channels (each with bandwidth c) at equal time intervals L/K, and thus the maximum viewing delay will be L/K (Fig. 2). So for a $7200sec$ video, we need 12 such channels to guarantee a $600sec = 10min$ viewing delay, which is not so efficient for bandwidth.

3 The Discrete Broadcasting Protocol for VOD

In our *Discrete Broadcasting Protocol* model, we first assume that the bandwidth b allocated for the broadcasting channel will be a multiple of the consumption rate c. Our second assumption is that a CBR video of length L will be divided

into n segments with same size L/n. Let's arrange them by time order and use $S_i(1 \leq i \leq n)$ to denote the ith segment.

In this simplified model, the maximum delay depends on the maximum distance between the beginning time of any two neighboring S_1 segments. Since all segments are equally sized, we can assume that this distance is $k * (L/n)$, $k \in N$. Thus the maximum delay is near $k*(L/n)$ in the case that the client just misses the first frame of the video when he starts to download. If the minimum distance of any two neighboring S_1 segments is also $k * (L/n)$, and the arriving times of the clients are uniformly distributed, then the average delay will be $\int_{0^+}^{\frac{kL}{n}} (\frac{kL}{n} - x)dx / \frac{kL}{n} = \frac{kL}{2n}$.

To satisfy the smooth watching requirement, we need to make sure that the client's STB can find an S_i during any period of length $L*i/n$ which starts from the begin of an S_1. Then we can make sure that after a client start consume S_1, he can download a S_i before $t_{req} + d + L*(i-1)/n$ (or find S_i at then) when he should start to consume S_i. If we fix the distance between any two neighboring S_1 segments to be (L/n), then we need to put an S_i in any period of length $L*i/n$ and we can use a single channel of $1*c$ to broadcast S_1.

Now let's analyze the bandwidth requirement. Since we need to put an S_i in any period of length $L*i/n$, then S_1 will occupy $1*c$ bandwidth, S_2 will occupy $c*1/2$ and S_i will occupy $c*1/i$, we have that $b \geq \frac{c}{1} + \frac{c}{2} + ...\frac{c}{i} + ... + \frac{c}{n} = \sum_{i=1}^{n} \frac{c}{i} = c * H_n$.

Based on this analysis, we designed Algorithm1. Fig. 3 is the first 52 columns of one result scheduling table of Algorithm1 when the available bandwidth is $4 * c$. We can see in Fig. 3 that the whole video is divided into 26 equal-size segments. But according to the analysis, the number of segments should be $max\{n|\lceil H_n \rceil = 4\} = 30$. The gap lies in that every segments are equally-sized and are broadcasted using the same bandwidth $1 * c$, so the scheduling table is a discrete table. In such a discrete case, we cannot make sure that the distance between any two neighboring S_is is exactly i. The reason is because there will be a confliction when you try to put one segment at time t yet all 4 sub-channels at that time slot are already occupied.

Algorithm 1 The Discrete Broadcasting Algorithm
SERVER:

Divide the whole video into n equal-size segments;
Put all S_1 segments into the first sub-channel;
For $i = 2$ *to* n **do**
  $t_{cur} = t_{next} = 0$;
  While *the video is not finished* **do**
    Calculate the next time t_{next} to put S_i;
    If *find a vacancy in* $(t_{cur}, t_{next}]$ **then** *put S_i;*
      Else *report error and exit;*
    $t_{cur} = t_{next}$;
  End while loop
End for loop

CONSUMER:

```
1   Start downloading all segments;
2   If find segment S_1 then start viewing S_1;
3   For i = 2 to n do
4       If find segment S_i in the local storage then
5           start viewing S_i;
6       Else report error and exit;
7       For all segments S_k in the broadcast channel;
8           If k > i and S_k is not in the local storage then
9               Download S_k into the local storage;
10  End for loop
```

For example, see S_{14} in Fig. 3, its first appearance is at t_{10}, so its next appearance should be at t_{24} for the most efficient case. But we find that all 4 sub-channels at t_{24} slot are already occupied, thus we can only put it at t_{23}. Such kind of conflictions will happen more often as n increases. So $L/max\{n|\lceil H_n \rceil = b/c\}$ is a theoretical optimal lower bound for maximum waiting time and cannot be achieved in most cases. Notice that there are some blank positions at the head, but we cannot use them to accommodate more segments. Because if we put an S_{27} there, we cannot find a vacancy for the next S_{27} since the columns from t_5 to t_{52} are already full.

Suppose the length of the video program to be broadcasted is $120min = 7200sec$,

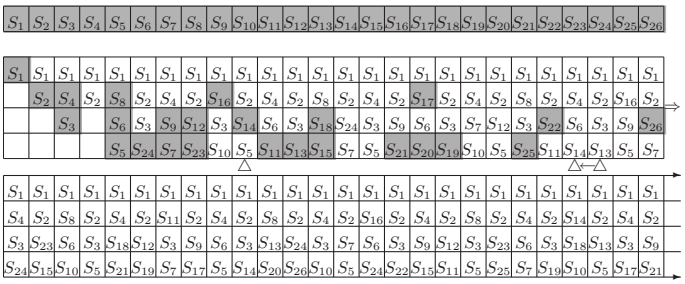

Fig. 3. The Full Discrete Broadcasting Protocol for VOD

then the length of each segment is $277sec$ if we adopt the table in Fig. 3. Thus the maximum delay is $277sec = 4.6min$ and the average delay is $2.3min$. If the format is high quality MPEG-II-compressed NTSC video at about $10Mbps$, then we have $r = 4 * c = 40Mbps$ while $m = 7200 * 10 * 8 * 9/26 \approx 3115Mbytes \approx 3.1Gbytes$, the ration $9/26$ is calculated from the table which means that a user needs to store at most 9 segments any time.

4 The Block Discrete Broadcasting Protocol

The scheduling table of DB will be very complex when n is large. So we present the *Block Discrete Broadcasting Protocol*(BDB), which arranges a short BLOCK table and then repeats broadcasting the segments according to this BLOCK table. Notice that there are some blank positions at the head of the table in Fig. 3, we can utilize them by using the BLOCK table method. And we need not to change the algorithm at the client side.

Algorithm 2 The Discrete Block Broadcasting Algorithm
SERVER:

```
1   Divide the whole video into n equal-size segments;
2   Put all S_1 into the first sub-channel of the BLOCK;
3   For i = 2 to n do
4       t_cur = t_next = 0;
5       While t_cur < l_BLOCK do
6         Calculate the next time t_next to put S_i;
7         If t_next < l_BLOCK then
8           If find a vacancy in (t_cur, t_next] then put S_i;
9               Else report error and exit;
10        Else
11          If find a vacancy in (t_cur, l_BLOCK] ∪ [0, t_next]
12                then put a segment S_i there;
13              Else report error and exit;
14        t_cur = t_next;
15      End while loop
16  End for loop
17  Repeat broadcasting BLOCK.
```

Suppose the length of a BLOCK table is l columns. To satisfy the smooth watching requirement, we need to make sure that the client's STB can find an S_i during any period of length $L * i/n$ which starts from an S_1. This means that we have to put an S_i in any i consecutive columns. Thus the space occupied by S_i will be no less than $\lceil \frac{l}{i} \rceil$ since this table will be repeatedly broadcasted. And the lower bound of the number of segments in a BLOCK table with l columns of Algorithm 2 will be: $b \geq l + \lceil \frac{l}{2} \rceil + \lceil \frac{l}{3} \rceil + ... \lceil \frac{l}{i} \rceil + ... + \lceil \frac{l}{n} \rceil = \sum_{i=1}^{n} \lceil \frac{l}{i} \rceil > l * H_n$.

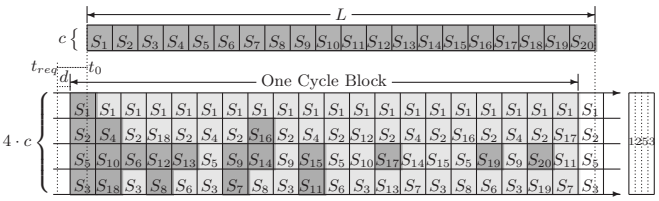

Fig. 4. The Scheduling Table of the Discrete Broadcasting Protocol

Fig. 4 is one output BLOCK table of Algorithm 2. We divide the whole video into $n = 20$ equal-size segments in this example and the length of the table is also $l = 20$ columns. The deep-grey shadowed segments in the BLOCK table show all those segments need to be downloaded for a client who sent a request at time t_{req}. Since $H_{20} \approx 3.60$ and $\sum_{i=1}^{20} \lceil \frac{20}{i} \rceil = 80$, the table has achieved the lower bound. For a $120min = 7200sec$ video with $c = 10Mbps$, the maximum delay is $7200/20sec = 6min$ and the average delay is $3min$. We can also calculate from the table that $r = 4 * c = 40Mbps$ while $m = 7200sec * 10Mbps * 9/(20 * 8) = 4050Mbytes = 4.05Gbytes$. Here we set $l = n = 20$, but this does not mean that we should always set $l = n$. In contrary, we can adjust the value of l

to find the most efficient schema. Consider an example when $b = 3 * c$, since $max\{n|\lceil H_n \rceil = 3\} = 10$, we cannot divide the video into more than 10 segments and load them into a scheduling table.

We start from $n = 8$, let $l = 8$ and we will have $\sum_{i=1}^{8} \lceil \frac{8}{i} \rceil = 24$. Fortunately a table with 3 lines and 8 columns can accommodate exactly 24 segments. Using Algorithm2, we construct the table in Fig. 5. Now let's try $n = 9$. If we let $l = 9$

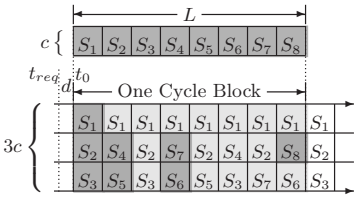

Fig. 5. A Block of BDB when n=8.

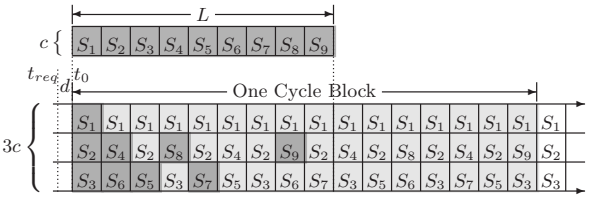

Fig. 6. A Block of BDB when n=9.

then $\sum_{i=1}^{9} \lceil \frac{9}{i} \rceil = 29$. Yet a table with 3 lines and 9 columns can accommodate only 27 segments. So we cannot construct such a BLOCK table. Then let us extend the length of the table. We find that when $l = 16$ we have $\sum_{i=1}^{9} \lceil \frac{16}{i} \rceil = 48$. While a table with 3 lines and 16 columns can accommodate exactly 48 segments. Thus we can construct the BLOCK table in Fig. 6.

5 Bound the Storage

We assume that there are enough local storage in all above schemes. And the true local storage requirements of them are all above 35% and around 40%. But sometimes in real applications we will encounter the problem of local storage shortage. Suppose there is a STB whose storage size is $2Gbytes$, which can satisfy the requirement for any video with $c \leq 5Mbps$ and $L \leq 7200sec$ since $m \leq 7200 * 5 * 40\%/8 = 1800Mbytes$. Now if there is a video with $c = 10Mbps$ and $L = 7200sec$, then $L * c = 9000Mbytes$ and $m = 3150Mbytes > 2Gbytes$.

If will still divide the video into n equal-size segments, then we can store $s = \lfloor n * 2/9 \rfloor$ segments by using $2Gbytes$. To guarantee the smooth watching of this video, we need to make sure that the client can find an S_i from the local storage at time t_i. We find that in most cases, the longer the distance between any two neighboring S_is, the longer the time S_i will be stored in the local storage. If too many segments need to be stored, then the limited local storage will be crammed sooner or later. The result is some segments must be discarded and the video cannot be smoothly consumed.

So one method to depress the local storage requirement is trying to reduce the distance between any two neighboring S_is when i is large. Let's reconsider about the BLOCK table, one characteristic of this table is that the maximum storage required will never exceed $S * l/n$. Since at a certain time when you are watching S_i, you need only to find the segments from S_{i+1} to S_{i+l-1}. You need not to consider about S_{i+l} because it can always be found at the period from when you start to watch S_{i+1} to when you finish S_{i+l}, which is a whole cycle BLOCK. If you find S_{i+l} at exactly the time when is should be consumed, you

can watch it directly because its broadcasting rate is the same as its consumption rate. Else you just load it into your local storage. Thus at anytime you need only to store the current segment and the next $l-1$ segments at most.

Of course this property is nonsense when $l \geq n$, but it does make sense when $l \ll n$. The following example in Fig. 7 show the point. In this example, we divide the video into $n = 19$ equal-size segments and arrange them into a $6 * 4$ BLOCK table. Since the length is only $l = 4$ and $\sum_{i=1}^{19} \lceil \frac{4}{i} \rceil = 24$, we have to use $6 * c$ bandwidth so that the table can accommodate more segments. Now according to the above analysis, the local storage requirement will be $9000 * 4/19 \approx 1895Mbytes < 2000Mbytes$. Since $l \ll n$ in this BLOCK table and the

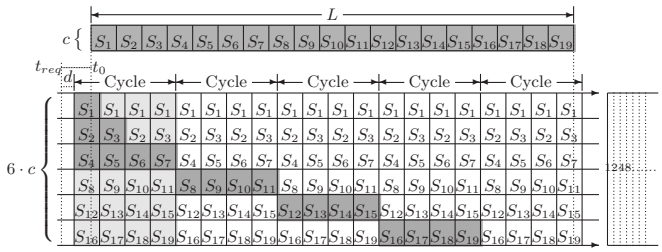

Fig. 7. The Storage Efficient Scheme for VOD

local storage is limited, we need only to slightly adapt the algorithms at the client side to satisfy the new requirement.

The deep-grey shadowed segments in the BLOCK table in Fig. 7 show all those segments need to be downloaded for a client who sent a request at time t_{req}, the maximum delay is $6.3min$ and the average delay is $3.2min$.

6 Performance Analysis and Comparison

The *Pyramid Broadcasting Protocol*(PB)[10] can provide shorter waiting time than previous schemes with the same available bandwidth. This protocol partitions an L-length video into n sequential segments of geometrical series increasing sizes and multiplexes M different videos into each logical channel.

A user should download the first segment at the first occurrence and start playing, then he will download the subsequent segment at the earliest possible time. To ensure smooth watching, each channel needs plenty of bandwidth and the I/O rate is very high, while local storage requirement can reach more than 70% of the whole video. To address these issues, the authors of [2] proposed the *Permutation-based Pyramid Broadcasting Protocol*(PPB). In PPB, each channel multiplexes its segments into p periodic bit streams and transmits them in $1/p$ times lower rate. But the local storage requirement of PPB is still high since the exponentially increasing speed may cause the last segment to be as large as 50% of the whole video, and the synchronization mechanism in it is very difficult to implement. So authors of [4] proposed the *Skyscraper Broadcasting Protocol* (SB). In SB, fixed bandwidth c is assigned to each logical channel. A video will be divided into n segments, the length of segments are [1, 2, 2, 5, 5, 12, 12, 25,

25, 52...]. Each of these segments will be repeatedly broadcasted on its dedicated channel at the consumption rate c. SB uses a value W as an upper bound to control the maximum size of each segment.

A significant progress was achieved by [5], in which the *Harmonic Broadcasting Protocol*(HB) was proposed. HB equally divides a video into n segments $[S = S_1 \uplus S_2 \uplus \cdots \uplus S_n]$, and each segment S_i will be divided into i equal-size subsegments $[S_i = S_i^1 \uplus S_i^2 \uplus \cdots \uplus S_i^i]$. Then HB allocates a single channel with bandwidth $C_i = c/i$ for each segment S_i. Thus the maximum delay is the length of the first segment $d = L_1 = L/n$ and the bandwidth is $b = H_n \cdot c$.

The *Stair Case Broadcasting Protocol* (SCB) in [6] and the *Fast Broadcasting Protocol* (FB) in [7] are based on HB. But in [8] it was observed that the user in HB may not get all data it needs to consume on time. The authors of [8] also proposed the *Cautious Harmonic Broadcasting Protocol*(CHB) and the *Quasi-Harmonic Broadcasting Protocol*(QHB). CHB and QHB are based on the same idea of HB, the new point is that they changed the arrangement of subsegments so that the average waiting time can be reduced by a factor near 2. But for the same Maximum Waiting Time, HB is still the best. Later they are proved to be Bandwidth-Optimal by Engebretsen and Sudan in [3].

The *Discrete Broadcasting Protocol* (DB) we propose in Section 3 is similar to HB. It can be deemed as the discrete version of HB. BDB is a simple version of DB, it's delay will be reduced when the BLOCK length is prolonged. But its performance is already quite good even when $l = n$. The Fig. 8 shows the number of segments we can reach in DB, BDB and HB by using the same Bandwidth. In Fig. 9 we compare the best maximum delay we can achieve in PB, SB, BDB, HB and the mean-delay of BDB by using the same Bandwidth. In both figures we set $l = n$ for BDB and we choose the best α value for PB. In

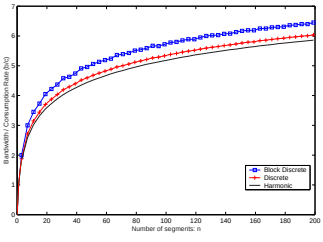

Fig. 8. Bandwidth Ratio v.s. Number of segments.

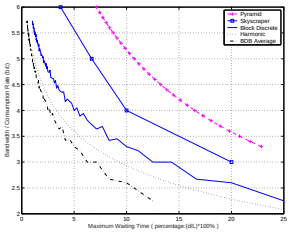

Fig. 9. The b/c Ratio versus the max-delay.

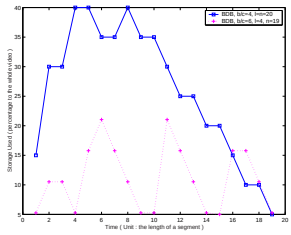

Fig. 10. Storage used in Fig. 4 and 7.

Section 4 we have discussed the solution of DB when the local storage is very small compare with the video size. For common BDB which uses a BLOCK of $l = n$, the maximum local storage requirement is around 40% of the whole video. But if we shorten the length of the broadcasting BLOCK, we can decrease the local storage requirement by the cost of increasing the bandwidth. Fig. 10 shows the local storage usage situation for two examples of the above two schemes. The data are calculated from the scheduling BLOCK Tables from Fig. 4 and Fig. 7 respectively. The following table compares the performance of these pro-

tocols. The sample video is a $120min$-long MPEG-II-compressed NTSC video at a consumption rate of about $c = 10Mbps$. The bandwidth is $b = 40Mbps$.

	HB	SCB	FB	DB	BDB	PB	PPB
Maximum Delay	4min	8min	8min	4.6min	6min	20min	30min
Average Delay	4min	4min	4min	2.3min	3min	10min	15min
Local Storage	3.4GB	2.1GB	4.2GB	3.8GB	3.6GB	6GB	6.75GB
Disk I/O rate	30Mbps	20Mbps	40Mbps	40Mbps	40Mbps	50Mbps	50Mbps

Table 1. Performance and resources requirements comparison

7 Conclusion

In this paper we present the efficient *Discrete Broadcasting Protocol* for VOD service. We also work out the *Block Discrete Broadcasting Protocol* as an extension of DB. Both DB and BDB can achieve lower average delay than the *Harmonic Protocol* with the same bandwidth. Furthermore, HB cannot work when the local storage is less than 37% of the whole video size. BDB can achieve a 6.3-minute max delay when the local storage is around 20% of the whole video size by using a bandwidth of 6 times the consumption rate. The discrete characterisc also makes our protocols more flexible and easy to implement.

References

1. K. C. Almeroth and M. H. Ammar, "The use of multicast delivery to provide a scalable and interactive Video-on-Demand service," *IEEE Journal on Selected Areas in Communications*, 14(5):1110-1122, Aug 1996.
2. C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation-based pyramid broadcasting scheme for Video-on-Demand systems," in *Proc. International Conference on Multimedia Computing and Systems*, pages 118-26, June 1996.
3. L. Engebretsen and M. Sudan, "Harmonic broadcasting is optimal," in *Proc. 13th annual ACM-SIAM SODA*, San Francisco, California, Pages: 431-432, Jan 2002.
4. K. A. Hua and S. Sheu, "Skyscraper broadcasting: a new broadcasting scheme for metropolitan Video-on-Demand systems," in *Proc. ACM SIGCOMM '97 Conference*, Cannes, France, pages 89-100, Sept 1997.
5. L. Juhn and L. Tseng, "Harmonic broadcasting for Video-on-Demand service," *IEEE Trans. on Broadcasting*, 43(3): 268-271, Sept 1997.
6. L. Juhn and L. Tseng, "Stair case data broadcasting and receiving scheme for hot video service," *IEEE Transactions on Consumer Electronics*, 43(4), 1110-1117, Nov 1997.
7. L. Juhn and L. Tseng, "Fast data broadcasting and receiving scheme for popular video service," *IEEE Transactions on Broadcasting*, 44(1):100-105, Mar 1998.
8. J.-F. Paris, S. Carter and D. D. E. Long, "Efficient broadcasting protocols for video on demand," in *Proc. MASCOTS '98*, Montral, Canada, pages 127-132, July 1998.
9. P.M. Smithson, J.T. Slader, D.F. Smith and M. Tomlinson, "The development of an operational satellite internet service provision," in *Proc. IEEE GlobalCom'97*, pp. 1147-1151, Nov 1997.
10. S. Viswanathan and T. Imielinski, "Metropolitan area Video-on-Demand service using pyramid broadcasting," *Multimedia Systems*, 4(4):197-208, 1996.