A Virtual Channel Technique for Supporting P2P Streaming[*]

Jau-Wu Huang, Nien-Chen Lin, Kai-Chao Yang,
Chen-Lung Chan, and Jia-Shung Wang

Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan
jwhuang@vc.cs.nthu.edu.tw

Abstract. Nowadays, some powerful client devices, e.g., smart phone, set top boxes and digital video recorders, are commonly used to enhance digital TV broadcasting services. This paper proposes a virtual channel platform by organizing these client devices to virtually support each user a dedicated channel according to her/his demand. In the proposed platform, each video is partitioned into many small segments before it is shared in a peer-to-peer network. A virtual channel is constructed by composing these video segments into a long video playout sequence. However, retrieving these small segments from a large scale peer-to-peer network could cause relatively large query overhead. To reduce the number of queries, we propose a virtual stream mechanism by aggregating popular adjacent video segments to logically form a long video object. The simulation results demonstrate that the proposed virtual channel platform can improve the service performance.

1 Introduction

The Digital TV (DTV), which is a new type of TV broadcasting technology, has become the most popular home entertainment nowadays. In addition, some powerful client devices, for example, smart phone, Set Top Box (STB) and Digital Video Recorder (DVR), are now widely integrated to the TV broadcasting service. Since these client devices usually have some computation powers, some storage spaces, and a network interface for Internet access, we then attempt to organize these client devices to provide a more flexible TV broadcasting service that delivers a sequence of demanded programs to each user via a "virtual channel", which is a personalized channel composed by her/his favorite programs.

In this paper, we propose a virtual channel platform to support an efficient virtual channel service. The idea of this platform is to organize the client devices to share their resource efficiently; therefore the availability of the videos in the system can be improved. We assume that all video programs are encoded into a common media format for compatibility, for exzample, MPEG-4 or H.264, before sharing them to all clients.

[*] The research is supported by the NSC99-2221-E-007-077-MY3, National Science Council, R.O.C.

We also propose a cache-and-share mechanism to improve the availability of the videos. Once a client obtains a video, this video will be stored and shared to others by means of a peer-to-peer file sharing technique, e.g., structured P2P (Pastry[2], Chord[3], CAN[4], CoopNet[5]), unstructured P2P (Gnutella[6], Napster[7], KaZaA[8]), or P2P streaming (CoolStreaming[9], PPLive[12], UUSee[13], SopCast[14]). These works are good for file sharing or video streaming; however, directly implementing them to support virtual channel service is not appropriate. The reasons will be stated later.

To support each user a dedicated virtual channel, we summarize the general viewing behaviors of users to four modes as follows:

A. Live mode

In this mode, users always watch the live program (for instance, live basketball game). In Fig. 1, we let the four users all watch a live video program A. When a user is coming, she/he always watches the latest frame of the live program.

B. Review mode

The review mode is based on the live mode. When a user is watching a live program, she/he could want to review some scenes, for example, splendid scenes or the scenes she/he just missed. Consider the example in Fig. 2. Let the live program A have a splendid scene at time 3. Therefore, some users, such as the user 1 and user 2 in this example, are probably to watch the scene A3 again.

C. Serial-play mode

In this mode, a user always watches an entire TV program from the head to the tail sequentially. Consider the example in Fig. 3. When a user requests the program A, she/he always stays in this channel until the entire program has end.

D. Interest-based mode

In this mode, a user can randomly access the TV programs according to her/his interest. Consider the example in Fig. 4. As the figure illustrates, the request sequences are irregular and are difficult to model.

Based on the above, we can summarize these general modes to two types: 1) sequential access; 2) random access. Undoubtedly, a perfect TV broadcasting platform should consider two goals as below:

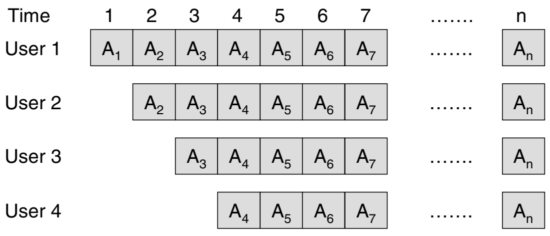

Time
1 2 3 4 5 6 7 .... n
User 1
A_1 A_2 A_3 A_4 A_5 A_6 A_7 .... A_n
User 2
A_2 A_3 A_4 A_5 A_6 A_7 .... A_n
User 3
A_3 A_4 A_5 A_6 A_7 .... A_n
User 4
A_4 A_5 A_6 A_7 .... A_n

Fig. 1. The behavior of live viewing

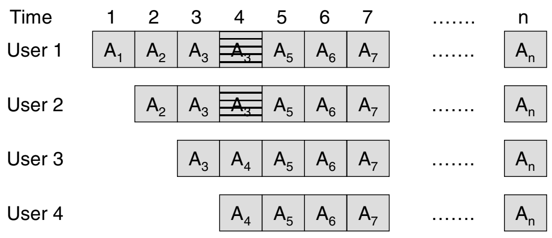


Fig. 2. The behavior of review viewing

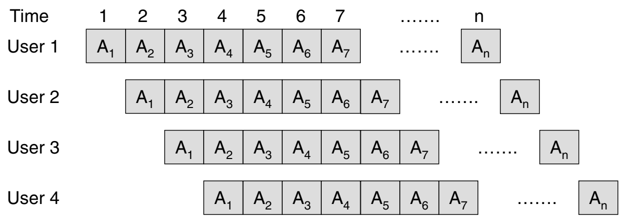


Fig. 3. The behavior of serial-play viewing

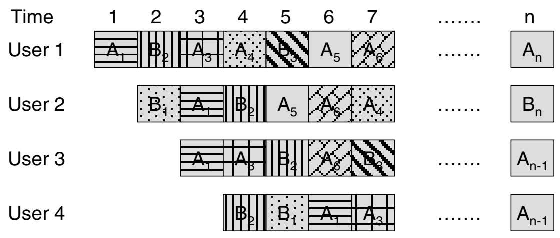


Fig. 4. The behavior of interest-based viewing

(1) supporting all the user viewing behaviors

Sequential access is a special case of random access. To support the above viewing behaviors, we assume that the platform must provide a random access interface to users.

(2) optimizing the system performance

A random access request pattern is not appropriate for performance optimization due to its irregularity. Therefore, we should try to aggregate some segments into a long video stream thus the system can deliver them sequentially.

Because 1) the system must provide a random access interface and 2) different parts of a video could have different popularities, we make an important assumption that the proposed platform partitions each video program into many small segments, such as [10][11][15][16] do. Then, a virtual channel can be constructed by retrieving these video segments and composing them into a long video playout sequence.

The rest of this paper is organized as follows. Section 2 introduces the concept of the proposed virtual channel platform and formulates its problem on query overhead. Section 3 then presents a virtual stream mechanism for reducing the query overhead. Section 4 presents some simulation results to evaluate the performance of the platform. Finally, the conclusion and future works are given in Section 5.

2 A Framework of Virtual Channel

2.1 Overview of the Proposed Platform

The primary components of the proposed platform can be summarized as follows:

A. The video source provider

With the rapid advances on network communication technologies, a client which can access the Internet and has some videos to share can also operate as a video source provider. Unfortunately, the storage and the reliability would be limited. To eliminate these restrictions in our platform, a backup service, which contains a large storage and has high reliability, to store the videos supported from the video source providers is essential. The details of the backup service will be introduced later.

B. The client

In the proposed virtual channel platform, each client could generate a request sequence including a series of video segments according to her/his demanded. To satisfy the demands of each client at anytime, we must increase the availability of each video program. Therefore, we not only introduce a backup service to store these videos but also adopt a cache-and-share mechanism to each client.

C. The backup service/the lookup service

In this paper, we introduce a backup service and a lookup service to increase the availabilities of the videos which are summarized as follows:

The backup service

In this paper, we adopt a backup service to store the video programs from the video source providers. The main functionalities of the backup service are 1) partitioning each video program into many small video segments; 2) storing these video segments in the buffers; 3) sharing these video segments to the clients. However, the storage of the backup service could still be exhausted. Therefore, we also adopt a LRU algorithm to replace the video segments which are not accessed for a long time.

The lookup service

In the current peer-to-peer file/video sharing platforms, each file/video is usually identified by a unique hash key. Thus, we introduce a lookup service to keep the mapping of the hash keys to the corresponding video segments.

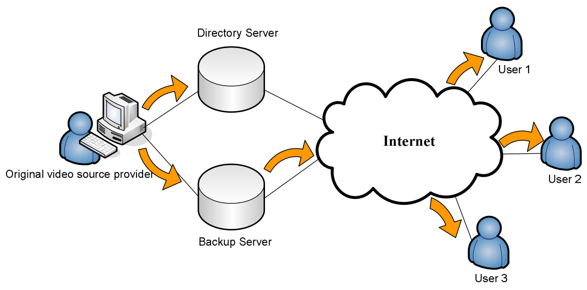


Fig. 5. An illustrative example of the proposed virtual channel platform

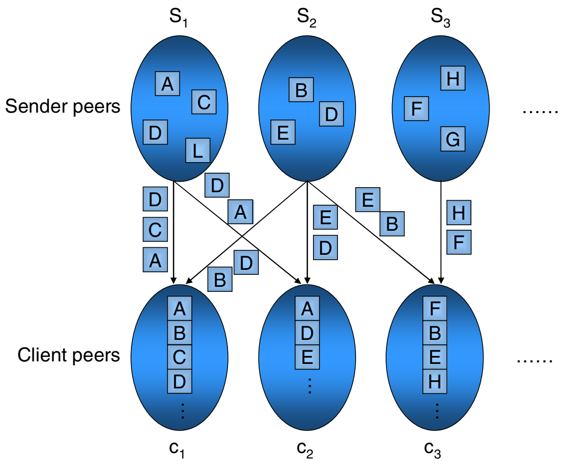


Fig. 6. An illustrative example of the delivery architecture

2.2 The Proposed Virtual Channel Platform

In this subsection, we introduce an illustrative example of the proposed virtual channel platform, as depicted in Fig. 5. The backup service and the lookup service can be implemented as a centralized or a distributed architecture. To simplify the following illustrations, we assume that the backup and lookup services are both implemented in a centralized way. Then, we will describe the details of the platform from three parts: 1) publishing a video program; 2) locating the sender peers; 3) retrieving the video segments.

A. Publishing a video program

When a video source provider publishes a video program, it will simultaneously forward the video stream to the backup server and the directory server. The backup server will partition the video into many small video segments and store them in its local buffer. On the other hand, the directory server will associate a unique hash key to each generated video segment and maintain the mapping in a hash table.

B. Locating the sender peers

When a client gets the hash key of a video segment, it still has to retrieve the video segment from one or more sender peers. The process to locate the sender peers can be implemented in either a centralized model or a peer-to-peer model.

(1) Centralized model

In the centralized model, the strategy is for each segment to publish the information of its contributed sender peers. However, this model would result in a performance bottleneck on the server of a large scale service.

(2) Peer-to-peer model

In a structured P2P system, a video source provider usually publishes the essential information to specific peers where requesters can easily discovered by means of a DHT (Distributed Hash Table) mechanism. However, maintaining such a DHT over a large scale P2P network will require considerable management overhead, and any update on peer status and system information will also cause an update on the content of the DHT. In the proposed virtual channel platform, a client can dynamically join or leave the system, and the cached video segments in its buffer could also update frequently. Thus, the structured P2P is not appropriate for our platform. Based on the above, we develop the delivery architecture of the proposed virtual channel platform based on the concept of unstructured P2P, just as CoolStreaming [9] does.

C. Retrieving the video segments

After locating the possible sender peers, the client then must retrieve the desired video segments from these sender peers. Fig. 6 shows an illustrative example of the delivery architecture in the proposed virtual channel platform. (It is important to remember that a sender peer could be the local buffer, another client, or the backup server.) As Fig. 6 illustrates, the client c_1 has gathered the hash keys {A, B, C, D} of its desired video segments. c_1 also understands that it can get the segments corresponding to {A, C, D} from S_1 and the segments corresponding to {B, D} from S_2.

2.3 The Details of the Proposed Delivery Architecture

In this subsection, we will discuss the details of a client to retrieve the desired video segments as follows:

(1) hash key conversion

After receiving the request sequence from a client, the directory server transforms the request sequence into a series of hash keys for P2P sharing.

(2) query and delivery

After receiving the responded hash keys, the user attempts to obtain the corresponding video segments according to these hash keys. First, the client checks if the video segments are already cached in its local buffer. If the segments are not cached in the local buffer, the client floods a query message to find at most w sender peers among all other peers and wait for responses. Let the number of available sender peers be x. Then, the delivery process would be either of the following cases:

Case I. $x \geqslant w$: The available number of sender peers is larger than the requirement. In this case, this client will choose w peers as the video sources.

Case II. $x < w$ and $x > 0$: The available number of sender peers is smaller than the requirement, so the client will simultaneously get the video segment from all of these x sender peers.

Case III. $x = 0$: The video segment is not cached in any peer in the P2P network. Therefore, the client must get the video segment from the backup server.

(3) local caching

After obtaining the desired video segments, the client will cache these video segments into its local buffer simultaneously with a LRU algorithm.

3 Overview of the Virtual Stream Mechanism

3.1 Definition of a Virtual Stream

A virtual stream is a set of video segments which are generated by the directory server and adopted to reduce the query overhead. In the proposed virtual channel platform, the virtual streams can be divided into two types: 1) beneficial virtual stream, and 2) elemental virtual stream. A beneficial virtual stream is a series of video segments which are frequently accessed by user requests, whereas an elemental virtual stream is formed with only one video segment. In other words, each individual video segment is called an elemental virtual stream. Then the directory server will associate an unique hash key with each virtual stream and record these hash keys into a hash table. To simplify the discussions, we assume that the size of each video segment is equal and is normalized to minutes of video data. Then, the directory server will 1) check all useable beneficial virtual streams and try to use them to replace the matched request subsequence of each user; 2) use the elemental virtual streams to replace the user request segments which are not matched by the beneficial virtual streams; 3) convert the request sequence to a series of hash keys associated to the virtual streams.

3.2 Definition of Popular Subsequences

Let the proposed virtual channel platform have n clients, and let each client C_i $(1 \leq i \leq n)$ send out a request sequence q_i $(1 \leq i \leq n)$, which is composed by d_i $(1 \leq d_i \leq y)$ video segments, to the directory server. (Note that y is the maximum number of video segments that a request sequence can contain.) Then, we have, $q_i = \{R_1, R_2, \ldots, R_{di}\}$. Consider a request subsequence v' with y' video segments $(2 \leq y' \leq L)$, i.e., $v' = \{U_1', U_2', \ldots U_{y'}'\}$. (Note that L is the maximum length of a virtual stream.) Then we define the term “match measure” MM as

$$MM(q_i, v') = \begin{cases} 1, & \text{if } v' \text{ is occurred in } q_i. \\ 0, & \text{if } v' \text{ is not occurred in } q_i. \end{cases} \tag{1}$$

We use this match measure *MM* to check if the request subsequence *v'* is matched in the request sequence q_i. If *v'* is found in q_i, then we have $MM(q_i, v') = 1$. Otherwise, we have $MM(q_i, v') = 0$. The match ratio of the request subsequence *v'* is then defined as follows:

$$MR(v') = \frac{\sum_{i=1}^{n} MM(q_i, v')}{n} \tag{2}$$

The above match ratio *MR* function for *v'* is equivalent to the probability that a request sequence contains *v'*. Then, if the match ratio of *v'* is larger than or equal to a threshold δ, i.e., $MR(v') \geq \delta$, the request subsequence *v'* is referred to as a popular subsequence and is treated as a beneficial virtual stream in this paper.

3.3 A Naïve Approach

A. Virtual stream generation

A naïve approach is to list all possible subsequences, check their match ratio *MR*, and select the highest ones as the beneficial virtual streams. However, this naïve approach requires too much memory space and computation overhead.

For example, we assume that the system have 10,000 video segments, whose lengths are all five minutes long. Let the system have 10,000 clients, and let each client request 60 video segments (i.e., 5 hours). Therefore, the number of possible request subsequences in worse case will be $10{,}000^{60} = 10^{240}$, because a video segment could be requested many times by a client. To find the most popular subsequences, we must derive all the possible subsequences in advance, and then check the match ratio *MR* of each one. Since a beneficial virtual stream will be at least two segments long, the amount of possible subsequences will be

$$\sum_{i=2}^{60} 10000^i,$$

which is an incredible large number. Based on the above, we can make two essential assumptions for virtual stream generation: 1) the number of subsequences to be checked must also be restricted, and 2) the length of each subsequence must be restricted.

B. Virtual stream scheduling

To reduce the query overhead, the directory server should reschedule the request sequences and try to replace with the beneficial virtual streams. The scheduling approach can be further divided into two steps: 1) sequentially scan all request sequences to check if any beneficial virtual stream is matched; 2) replace the matched subsequences with the corresponding beneficial virtual streams. However, this naïve scheduling approach has some problems: 1) too large memory space and computation overhead. 2) the match probability is low.

3.4 The Proposed Approach

In Section 1, we have summarized that the request sequences of the sequential access behaviors are more regular than those of the random access behaviors, so these request sequences are easily to find beneficial virtual streams with high match rates. Although this problem is caused by the request sequences themselves and looks impossible be solved by the directory server, it is not so serious because a user issues a random access based request sequence may not extremely care the playout order of demanded segments.

Based on the above, we let the directory server rearrange the "interest-based" user request sequences to increase the match probability of the beneficial virtual streams.

A. Virtual stream generation

To restrict the number of subsequences to be checked, we apply an interval batching mechanism to the directory server. Consider the example in subsection A again. The directory server will only deal with the request sequences which contain at most 12 segments. Therefore, the amount of possible subsequences will be

$$\sum_{i=2}^{12} 10000^{i},$$

which is much smaller than the original one. However the result is still too large to be stored in memory. To reduce the number of possible subsequences to be checked, define a cost function to accurately evaluate the impact of a virtual stream.

In this paper, we define a cost function, which is called "estimated performance gain", to estimate the performance gain of each subsequence. The subsequences with the largest performance gains will be selected as beneficial virtual streams. The estimated performance gain ($E(G)$) is defined as below:

After the virtual stream is applied, assume that the amount of query messages becomes n_Q', the total number of requests becomes r', and the number of requests served by the local buffers becomes r_L '. Then the amount of query messages n_Q' can be represented as

$$n_Q' = ((r' - r_L') \cdot n_P) \tag{3}$$

Let r_B' be the number of segments delivered by the backup server after the virtual stream is applied. Therefore, the estimated performance gain ($E(G)$) will be

$$E(G) = (n_Q - n_Q') \cdot o_Q - (r_B' - r_B) \cdot o_S \tag{4}$$

The algorithm of virtual stream generation, which is designed based on the equation (4), is illustrated as follow:

In this paper, we only focus on the popular video segments and the directory server applies a heuristic to generate the beneficial virtual streams in every interval time as follows:

First, the directory server finds the most M frequently accessed video segments in this interval. Then, it generates all possible subsequences by spanning the frequently accessed video segments. Since a request sequence can be rearranged, we only generate the subsequences by composing the segments as a nondecreasing order on the segment generating time. For example, let the popular segments be A, B, and C. So the possible subsequences will be {A, B}, {A, B, C}, {B, C}, and {A, C}. Finally, the directory server evaluated the estimated performance gains $E(G)$ based on the equation (4) and selects the subsequences with the largest F gains as the beneficial virtual streams, and then it will associate each of them an unique hash key.

B. Virtual stream scheduling

In this subsection, we will introduce the proposed approach for virtual stream scheduling with a rearranging strategy. Note that while rearranging a request sequence, each demanded video segment R_i can not be scheduled to a time slot before its generating time $t(R_i)$, i.e., can not cause a "time violation". However, this procedure could take a lot of computation overhead. To reduce the computation complexity, we let the directory server sort the interest-based request sequences to a nondecreasing order on the segment generating time $t(R_i)$.

4 Simulation Results

In this section, we propose two platforms based on the Gnutella P2P overlay network to compare their system performance over various physical network topologies, as illustrated in Table 1.

We let each client have 10 neighbors, and let the TTL of each query message be 5 hops for limiting the query range of a request. Besides, we assume that all video programs are originally from several TV broadcasting channels. Let the lengths of all videos be one hour, and let each video be divided into 12 five-minute video segments (M = 12). The execution time of each simulation is 24 hours. In each scheduling interval which is one hour long, the directory server will generate at most 10 beneficial virtual streams, and each virtual stream could have at most five segments. We use the term "normalized load" to evaluate the system performance of both platforms:

$$\text{normalized load} = \frac{\text{system cost in the enhanced platform}}{\text{system cost in the baseline platform}}$$

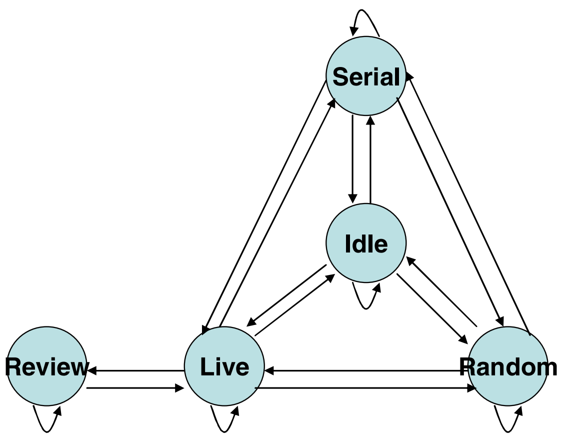


Fig. 7. The finite state machine to model the user viewing behavior

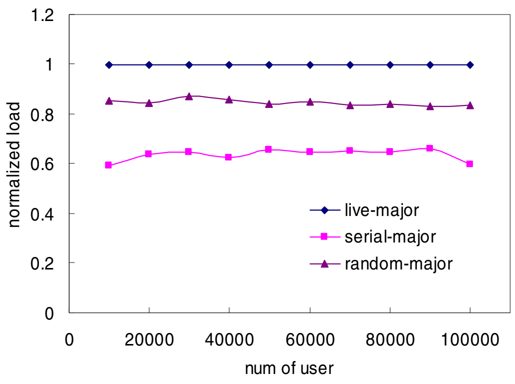


Fig. 8. The impacts of request patterns

Table 1. The two platforms for comparison

Baseline platform	Without beneficial virtual streams
Enhanced platform (our scheme)	Schedule the user request sequences with beneficial virtual streams.

4.1 The Model to Formulate the User Requests Determining the State of Viewing Behavior

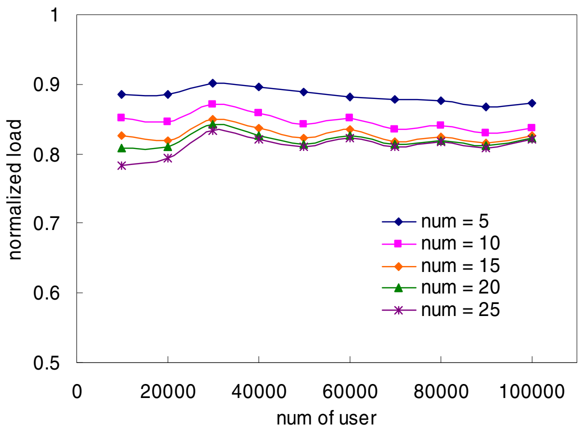
In this subsection, we use a finite state machine to model the behavior of each user, as illustrated in Fig. 7. Under this finite state machine, the user can start from one of these four states (i.e., live, serial, random, idle), and then she/he can select to stay at the same state or go to another state in the next time slot.

Choosing the Channel

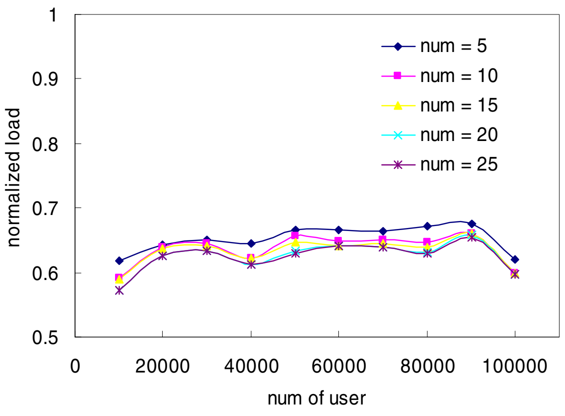
As mentioned above, we have assumed that the simulated system have three channels and define a channel table to specify the weight on each state for accessing each channel in the simulation. Table 2 shows an illustrative example of a channel table. In this example, if a user is in the Live state, she/he has a 50% to choose channel 1, a 30% to choose channel 2, and a 20% to choose channel 3.

Table 2. An example of channel table

	Channel 1	Channel 2	Channel 3
Live state	50%	30%	20%
VoD state	20%	70%	10%
Random state	10%	30%	60%



(a)



(b)

Fig. 9. The impact of the number of generated beneficial virtual streams F: (a) for random-major, (b) for serial-major

Requesting a video segment

After the viewing state and the demanded channel have been determined, we have to combine the finite state machine with the channel table to model the user viewing behaviors.

4.2 Performance Evaluation

A. Impact of the request patterns

Now we evaluate the system performance of the baseline platform and the enhanced platform under different request patterns: 1) live-major: with high ratio of live viewing; 2) serial-major: with high ratio of serial-play viewing; 3) random-major: with high ratio of interest-based viewing. In this simulation, each client averagely issues 0.97 requests every five minutes, and the request distribution is determined by the model as mentioned above. We then vary the number of clients from 10,000 to 100,000 to investigate the normalized load under these settings.

In Fig. 8, the result agrees that the serial-play mode is much easier to generate both long and frequently accessed beneficial virtual streams. Therefore, in the following simulations, we only focus on the viewing behaviors based on random-major and serial-major mode, and modify the number of the beneficial virtual streams in a scheduling interval, the maximum length of a virtual stream, to investigate the corresponding service performance.

B. Impact of the number of beneficial virtual streams

We adjust the maximum number F, of beneficial virtual streams generated in each scheduling interval time. Fig. 9(a) and Fig. 9(b) illustrate the simulation results of a random-major and a serial-major request pattern, demonstrating that a larger F implies a lower normalized system load, and the request pattern in a serial-major mode has a larger improvement than in a random-major mode.

5 Conclusions

We have proposed a virtual channel platform, which organizes DTV client devices to virtually support each user a dedicated channel according to her/his favor. We have introduced a peer-to-peer overlay network for sharing their resources, such as their buffers and network bandwidth, to each other, thus improving overall video availability. In addition, each video program is partitioned into many small segments before it is shared in this peer-to-peer network. However, such a peer-to-peer delivery architecture would cause considerable query overhead, so we also have proposed a virtual stream mechanism which aggregates the popular connected video segments into a large video stream to reduce the amount of query messages. The simulation results demonstrate that our virtual stream mechanism can significantly reduce the query overhead by increasing tiny backup server load, thus the system performance can be improved. Our future work is to develop a more efficient algorithm for generating beneficial virtual streams to further improve service performance.

References

1. Digital Video Broadcasting Project (DVB), `http://www.dvb.org`
2. Rowstron, A., Druschel, P.: Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Liu, H. (ed.) Middleware 2001. LNCS, vol. 2218, pp. 329–350. Springer, Heidelberg (2001)
3. Stoica, I., Morris, R., Kaashoek, M., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for Internet applications. In: Proc. ACM SIGCOMM 2001 (August 2001)
4. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. ACM SIGCOMM 2001 (August 2001)

5. Padmanabhan, V.N., Wang, H.J., Chou, P.A., Sripanidkulchai, K.: Distributing streaming media content using cooperative networking. In: Proc. ACM NOSSDAV (May 2002)
6. Gnutella, `http://www.gnutella.com/`
7. Napster, `http://www.napster.com`
8. KaZaA, `http://www.kazaa.com`
9. Zhang, X., Liu, J.C., Li, B., Peter Yum, T.-S.: CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. In: Proc. IEEE INFOCOM (March 2005)
10. Viswanathan, S., Imielinski, T.: Metropolitan area video-on-demand service using pyramid broadcasting. Multimedia Systems 3, 197–208 (1996)
11. Hua, K.A., Sheu, S.: Skyscraper broadcasting: a new broadcasting scheme for metropolitan video-on-demand systems. In: Proc. ACM SIGCOMM 1997 (1997)
12. PPLive, `http://www.pplive.com/`
13. UUSee, `http://www.uusee.com/`
14. SopCast, `http://www.sopcast.com/`
15. Chi, H., Zhang, Q., Jia, J., (Sherman) Shen, X.: Efficient Search and Scheduling in P2P-based Media-on-Demand Streaming Service. IEEE Journal on Selected Areas in Communications 25(1) (January 2007)
16. Cheng, X., Liu, J.: NetTube: Exploring Social Networks for Peer-to-Peer Short Video Sharing. In: Proc. IEEE INFOCOM 2009, Rio de Janeiro, Brazil (April 19-25, 2009)