A Virtual Channel Technique for Supporting Live/On-demand Streaming

NIEN-CHEN LIN[1,*], CHEN-LUNG CHAN[2], AND JIA-SHUNG WANG[2]
[1]*Chunghwa Telecom Laboratories, Taiwan*
[2]*Department of Computer Science, National Tsing Hua University, Taiwan*

ABSTRACT

Nowadays, some powerful client devices, e.g., 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 establish a peer-to-peer overlay to virtually support each user with a dedicated channel according to their demands. In the proposed platform, each video program is partitioned into many small segments before it is shared. A virtual channel is constructed by composing the necessary video segments, which are possibly from different videos, into a long video playout sequence for the user. However, retrieving these small segments from a large scale peer-to-peer network could cause a 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.

Key words: DTV, streaming video, peer-to-peer network.

1. INTRODUCTION

Digital TV (DTV) (Digital Video Broadcasting Project [DVB], 2007), which is a new type of TV broadcasting technology, has become the most promising means of home entertainment nowadays. In a traditional TV service, when users want to watch their favorite programs, they will have to switch between multiple channels.This viewing behavior has become a habit nowadays, however, it is not good enough because the channel switching is complex and will probably cause missed scenes. In addition, such channel switching could be time consuming in a DTV service because the number of available channels could become extensive.

Since the viewing habits of different users might be significantly different, a tradition channel-based browsing model is no longer flexible enough for modern users. Virtual channel (Chorianopoulos, Lekakos & Spinellis, 2003; Chorianopoulos & Spinellis, 2003, 2004) is a new service model that integrates live broadcasting and stored video content to support flexible organization and dynamic presentation of TV programs. In such a cross channel platform, a user can have their own video playout sequence which is composed of their favorite programs. Furthermore, the platform enables users to issue some VCR-like commands such as play, pause, next, and previous. The most attractive thing is that users can now not only be end viewers but also be video suppliers because they can publish their

* Corresponding author. E-mail: nclin@cht.com.tw

personal videos on the Internet. With these advanced applications, a virtual channel is considered as the next generation multimedia service model.

Nowadays, some powerful client devices, e.g., set-top boxes (STB) and digital video recorders (DVR), are widely integrated to enhance the TV broadcasting service. These client devices usually have some computational powers, some storage space, and a network interface for Internet access. In this paper, we propose a virtual channel platform by organizing these client devices to form a distributed video delivery platform to improve the availability of each video program. That is, the video source of a client now could be from a TV channel, the local disk, or other clients in the network. A virtual channel is established by applying streaming video techniques to ensure the user can smoothly switch among the video programs, i.e., they will not suffer from the drawbacks caused by channel switching. To do so, the platform should provide a video discovery service so that each user can efficiently find their favorite programs from all possible video sources. From this point of view, the proposed virtual channel platform looks similar to a general peer-to-peer (P2P) video delivery platform.

P2P overlay networks have been widely studied and implemented. Based on their overlay structures, the P2P overlay networks can be roughly divided into two categories: structured P2P (e.g. Pastry (Rowstron & Druschel, 2001), Chord (Stoica, Morris, Kaashoek & Balakrishnan, 2001), CAN (Ratnasamy et al., 2001), and CoopNet (Padmanabhan, Wang, Chou & Sripanidkulchai, 2002)) and unstructured P2P (e.g., BitTorrent (2003), LimeWire (2001), and KaZaA (2003)). Some of these techniques are further enhanced to support streaming video, e.g., CoolStreaming (CoolStreaming, 2005; Zhang, Liu, Li & Yum, 2005)/ PeerCast (2001)/ PPLive (PPlive, 2006; Hei, Liang, Liang, Liu & Ross, 2006) for live streaming and Bitos (Vlavianos, Iliofotou & Faloutsos, 2006)/ BASS (Dana, Li, Harrison & Chuah, 2005)/ LiveBT (Lv et al., 2007) for video-on-demand. In these previous works, a client caches the video clips it received and shares these clips with others, thus reducing the server load and network traffic. Such a P2P sharing model is efficient for file sharing and video streaming and, in fact, our virtual channel platform is also developed based on a P2P model. However, there is still a critical difference between the proposed virtual channel platform and these P2P video streaming platforms, i.e., the different request behaviors.

In the virtual channel platform, the dedicated channel of a user is entirely composed of their demanded video scenes. That is, the actual playout sequence of a user is determined by their viewing behavior. The general viewing behaviors of a user can be summarized as four modes: *live mode*, *review mode*, *serial-play mode*, and *interest-based mode*. (1) In the live mode, a user synchronously plays out the video frame which is being transmitted by the video source. Figure 1 shows an example in which four users are concurrently watching a live video program, A. In this case, all the users are playing the latest frame of the live program (e.g., playing the frame A_3 at the time slot 3), implying a sequential access pattern on the video A. (2) The review mode is an extension of the live mode. When a user is watching a live TV program, they may want to review some previous scenes, e.g., noteworthy scenes or the scenes they just missed. Consider the example in Figure 2. Let the

live program A have a noteworthy scene at the time slot 3. Therefore, some users, such as user 1 and user 2 in this example, may watch the scene A_3 again at the time slot 4. The request patterns in this case are still primarily composed of sequential access. (3) In the serial-play mode, a user always watches an entire TV program from the start to finish sequentially, as Figure 3 illustrates. This is the primary access pattern of video-on-demand applications. (4) Unlike the previous three viewing behaviors, the interest-based mode implies a random access pattern. Consider the example in Figure 4, which supports four videos A, B, C, and D. In this mode, a user can randomly access all the TV programs according to their current interest, e.g., user 1 could play the frame A_1 at the time slot 1 but play B_2 at the time slot 2. The resultant request sequences become irregular because they might be composed of random scenes from distinct video programs.

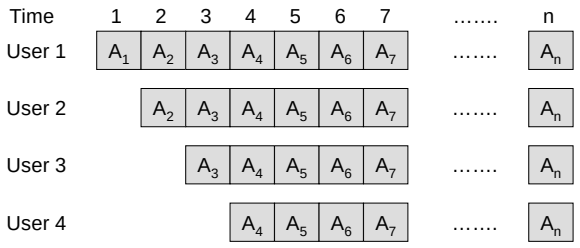

Figure 1. Live viewing behavior.

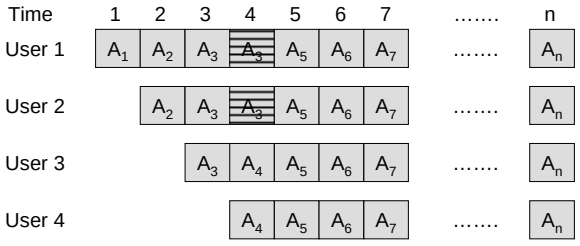

Figure 2. Review viewing behavior.

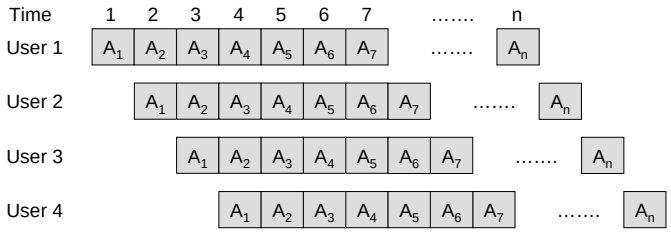

Figure 3. Serial-play viewing behavior.

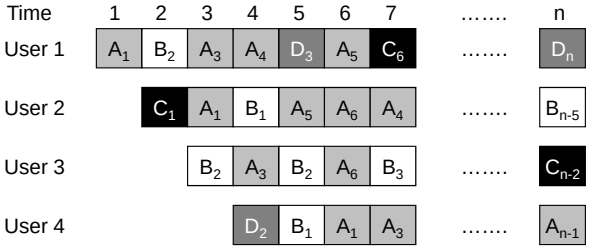

Figure 4. Interest-based viewing behavior.

To provide each user with a dedicated channel, all the above request modes should be efficiently supported by the virtual channel platform. Undoubtedly, the first three modes can be represented by the interest-based mode, because a sequential access pattern can be viewed as a special case of a random access pattern. This implies that the platform must provide a random access request interface to users.

Due to such random access issues, each video program should be divided into many small segments in advance, so that each virtual channel can be represented by the combination of these video segments. Once a user requests a virtual channel, the demanded video segments will be assembled into a long playout sequence, and the resultant video stream will be delivered. This procedure looks similar to P2P *swarming*, which also divides a video stream into many pieces and resembles the pieces on the client side. Thus, treating each video segment as a video program and sharing them via P2P swarming could be a straightforward solution for a virtual channel. However, the size of a video segment could be extremely small since it could only be a short video scene, implying that the number of available segments in the network could be extremely large. Also, these segments could have no

dependency and must be discovered individually. Distributing these extremely small video segments over a large scale P2P overlay network could induce a considerable query overhead, i.e., a user might send thousands of query messages to discover a segment of only few mega bytes.

To share the video segments efficiently, the proposed virtual channel platform is built atop a P2P video streaming architecture, so the extensive query overhead also becomes the critical problem on our platform. However, considering that most request modes are still based on sequential access, the requests should be served in a sequential access manner as much as possible. The idea we propose in this paper is a virtual stream mechanism that aggregates video segments which are frequently being played together to logically form a long virtual video stream. With these long virtual streams, a user can simultaneously discover multiple video segments via only a query for a virtual stream, thus the number of query messages can be significantly reduced.

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 suggestions for future work are given in Section 5.

2. THE FRAMEWORK OF VIRTUAL CHANNEL

As we mentioned above, the core architecture of the proposed virtual channel platform is a P2P overlay network which has an extensive number of small video segments sharing it. However, there are still some critical differences between our platform and a general P2P swarming system. Here we introduce the essential "logical" components used in our platform:

- *The video source provider*: The goal of a video source provider is to provide original video programs. A video source provider could be a TV broadcasting channel, a traditional video server, or a powerful client which can access the Internet and has some videos to share.
- *The client*: A client device in our platform is not only the sink of a virtual channel but also can be a video source provider. With the P2P overlay network established by these powerful clients, the availability of each video program can be further improved.
- *The backup service*: The storage and the reliability of a client would be limited. To extend the life time of each video program, we suggest that a "backup service", that can stably store the videos from the video source providers, is also essential. Such a backup service can be implemented on a centralized file server, a proxy, a content distribution network (CDN), or even on peer-to-peer storage. That is, the backup service can also be implemented by the P2P overlay network itself if a reliable peer-to-peer storage management

mechanism is applied. The main functions of the backup service include: (1) partitioning each video program into many small video segments; (2) storing these video segments in the buffers; (3) sharing these video segments with the clients. Once the storage of the backup service is exhausted, a LRU algorithm is applied for cache replacement.

- *The lookup service*: The lookup service is used to keep the mapping of the video segments and their hash keys. The same as the backup service, it can also be implemented by a centralized directory server, a tracker, a distributed database, or even by the P2P network itself.

The proposed virtual channel platform is constructed from four components, as depicted in Figure 5. Next we describe in detail the three parts of the platform: (1) publishing a video program; (2) locating the supplier peers; (3) retrieving the video segments.

2.1 Publishing a Video Program

When a video source provider publishes a video program, the video stream will be simultaneously forwarded to the backup service and the lookup service. The backup service will partition the video into many small video segments and store them in its local buffer. The lookup service will associate an unique hash key to each generated video segment and maintain such mapping in a hash table.

2.2 Locating the Supplier Peers

Once a client requests a virtual channel, it invokes the lookup service to translate the virtual channel to a list of hash keys of the essential video segments. Then the client retrieves the video segment from one or more supplier peers. The process to locate the candidate suppliers can be implemented in either a centralized model or a peer-to-peer model. Since a centralized model could easily result in a performance bottleneck, a peer-to-peer model is recommended. The peer-to-peer models can be further classified into two categories: distributed hash table (DHT) and flooding query model.

In a structure P2P system, a video source provider usually publishes the essential information to specific peers where it can easily be discovered by requesters by means of a DHT mechanism. However, any update of peer status and system information will also cause an update of the content of the DHT. In the virtual channel platform, a client can dynamically join or leave the system, and the cached video segments in its buffer can also be updated much more frequently than those in traditional P2P streaming platforms. Thus, the DHT model is not appropriate for our platform. Based on the above, supplier discovery of the proposed virtual channel platform is developed based on a flooding query model similar as the Gnutella protocol (The Gnutella Protocol Specification, 2001). The query process becomes: First, the client checks if the required 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 a number of its neighboring peers to find the

candidate suppliers and wait for responses. The neighbors will also forward the query messages to their neighbors unless the predefined query range is exceeded. After gathering the response messages, the client can choose a number of nodes as its suppliers from the candidate list.

Let the maximum number of suppliers that a client can have be w, and let the number of candidate suppliers returned by the query process be x. Then we have the following three cases:

Case *I*. $x \geq w$:

The available number of candidate suppliers 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 number of candidate suppliers is smaller than the requirement, so the client will simultaneously get the video segment from all of these x peers.

Case *III*. $x = 0$:

The video segment is not be cached in any peer in the P2P network. Therefore, the client must get the video segment from the backup service.

2.3 Retrieving the Video Segments

After locating the supplier peers, the client then has to retrieve the desired video segments from these supplier peers. If the buffer space of the client is exhausted, a LRU algorithm is applied for cache replacement. Figure 6 shows an illustrative example where the client c_1 has to download the segments {A, B, C, D}. Based on the query result, it will get {A, C, D} from S_1 and {B, D} from S_2. If it has no buffer space to cache {A, B, C, D}, it will remove the segment whose access time is the earliest from its buffer.

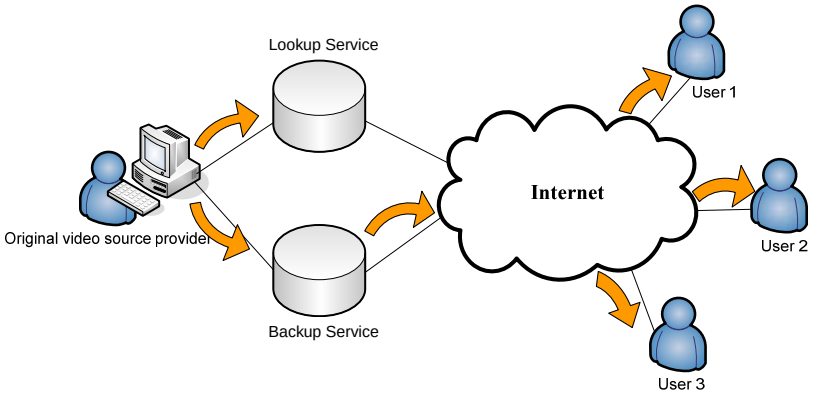

Figure 5. An illustrative example of the proposed virtual channel platform.

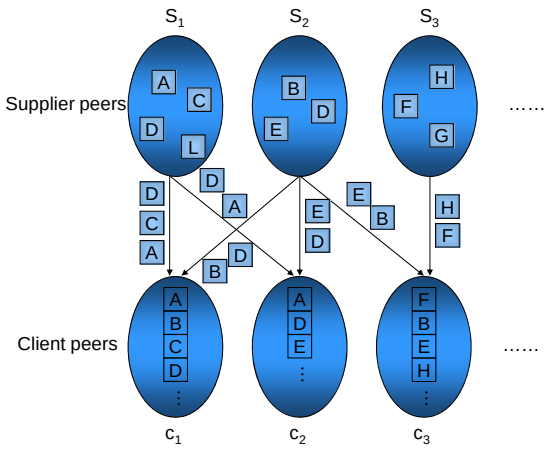

Figure 6. An illustrative example of the P2P delivery architecture.

Now we derive the theoretic performance of the proposed virtual channel platform, which is represented by the term "system cost." Let c be the total system cost, let c_B be the total cost of the backup service, let c_Q be the total cost of the query messages, and let c_D be the total cost for delivering the data over the P2P overlay network. Therefore, the system cost will be

$$c = c_B + c_Q + c_D. \tag{1}$$

Let r be the total number of requests, let r_L be the number of requests served by the local buffers, let r_P be the number of requests served by other peers, and let r_B be the number of requests served by the backup service. Obviously, the total number of requests will be

$$r = r_L + r_P + r_B. \tag{2}$$

Let the cost of downloading a video segment from the backup service be o_S. Then, the total cost of the backup service c_B will be

$$c_B = r_B \cdot o_S. \tag{3}$$

Let the total number of peers that a query process involves be n_P. Then, the total number of query messages n_Q will be

$$n_Q = ((r - r_L) \cdot n_P). \tag{4}$$

Let the cost of delivering a query message be o_Q. Therefore, the total cost of query messages c_Q will be as below:

$$\begin{aligned} c_Q &= n_Q \cdot o_Q \\ &= \left(\left(r - r_L\right) \cdot n_P\right) \cdot o_Q . \end{aligned} \tag{5}$$

Let the cost of delivering a segment over the p2p network be o_B. Then, the total cost of P2P delivery c_D will be

$$c_D = r_P \cdot o_B. \quad (6)$$

From (1), (3), (5), and (6), the theoretical system performance can be derived.

Note that minimizing c_B and c_D are usually the optimization goals of a streaming video system, because the cost of delivering a video stream is generally considered to dominate the system performance. However, the query overhead, i.e., c_Q, could also be quite considerable in the virtual channel platform. To show the impact of query overhead, let us consider the following example:

Let the length of each video segment be 5 minutes and let its coding bit rate be 1 Mbps. Thus, the size of each video segment is 5*60/8 = 37.5MB. Let a flooding query method similar to Gnutella with query TTL=7 be applied for locating the suppliers, and let a client averagely have 10 neighbors. Let the size of each query message be 1KB. When a client requests a video segment, the total bandwidth used to discover the video segment will be ($10^1 + 10^2 + 10^3 + 10^4 + 10^5 + 10^6 + 10^7$) * 0.001 (MB) = 11111.11 (MB). In this case, we must spend 11111.11MB of bandwidth to obtain only 37.5MB of video data, which is undoubtedly an inefficient service strategy. Therefore, in the following section, we propose a virtual stream mechanism to reduce the query message overhead.

Table 1. *Symbols*

c	the total system cost
c_B	the total cost of the backup service
c_Q	the total cost of the query messages
c_D	the total cost of the data delivered over the P2P overlay network
r	the total number of requests
r_L	the number of requests served by the local buffers
r_P	the number of requests served by other peers
r_B	the number of requests served by the backup service
n_P	the total number of peers that a query process involves
n_Q	the total number of query messages
o_S	the cost of delivering a video segment from the backup service
o_Q	the cost of delivering a query message
o_B	the cost of delivering a segment over the p2p network

3. OVERVIEW OF THE VIRTUAL STREAM MECHANISM

To reduce the query overhead caused by the extremely small segment size, we introduce a new concept called "virtual stream" on the lookup service. A virtual stream is a symbol of a set of video segments, and it is shared as a unit over the virtual channel platform. The virtual streams can be further divided into two types: (1) elemental virtual stream, and (2) beneficial virtual stream. Each video segment

is referred to as an elemental virtual stream, i.e., an elemental virtual stream contains only one video segment. A beneficial virtual stream is a series of video segments which are frequently accessed by user requests. The lookup service will associate an unique hash key with each virtual stream and record these hash keys into a hash table.

Once a client requests a virtual channel, the lookup service will (1) check all useable beneficial virtual streams and try to use them to replace the matched request sub-sequence of each user; (2) use the elemental virtual streams to replace the user request segments which were not matched by the beneficial virtual streams; (3) convert the request sequence to a series of hash keys associated with the virtual streams. By means of the transformation from individual video segments to beneficial virtual streams, the number of objects to be discovered in the P2P overlay can be significantly reduced. In the real implementation we assume that the lookup service will periodically determine the set of beneficial virtual streams because the available video segments will be updated with time. Furthermore, the number of beneficial virtual streams should also be limited to control the computation complexity of virtual stream replacement. Therefore, we let the lookup service periodically invoke virtual stream generation to select the most F popular request sub-sequences as beneficial virtual streams at the beginning of every "scheduling interval."

As a beneficial virtual stream should represent a popular request sub-sequence, determining the most popular request sub-sequences is critical for a lookup service. Let the virtual channel platform have n clients, and let each client C_i ($1 \le i \le n$) send out a request sequence q_i ($1 \le i \le n$), which is composed of d_i ($1 \le d_i \le y$) video segments, to the lookup service. (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 sub-sequence v' with y' video segments ($2 \le y' \le L$), i.e., $v' = \{U_1', U_2', \ldots U_{y'}'\}$. (Note that L is the maximum length of a virtual stream.) Then we define a term "match measure" MM as

$$MM(q_i, v') = \begin{cases} 1, & \text{if } v' \text{ is occurred in } q_i; \\ 0, & \text{if } v' \text{ is not occurred in } q_i. \end{cases} \tag{5}$$

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

$$MR(v') = \frac{\sum_{i=1}^{n} MM(q_i, v')}{n}. \tag{6}$$

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 sub-sequence v' is referred to as a popular sub-sequence and is treated as a beneficial virtual stream.

Based on the above concept, we can determine which request sub-sequences should be considered as beneficial virtual streams. However, the number of beneficial virtual streams should be tightly limited because the generation, matching, and replacement of virtual streams all could lead to considerable computation complexity and memory consumption. Here we introduce two approaches:

3.1 A Naïve Approach

A naïve approach is to list all possible sub-sequences, check their match ratio *MR*, and select the highest ones as the beneficial virtual streams. However, this naïve approach requires too large a memory space and computation overhead. For example, let a virtual channel 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 a total of 60 video segments (i.e., 5 hours). Therefore, the number of possible request sub-sequences in the worst case will be $10{,}000^{60} = 10^{240}$, because a video segment could be requested again and again by a client. To find the most popular sub-sequences, we must derive all the possible sub-sequences 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 number of possible sub-sequences will be:

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

which is an incredibly large number. Based on the above, we make two essential reductions for virtual stream generation: (1) the number of sub-sequences to be checked must be restricted, and (2) the length of each sub-sequence must also be restricted.

To reduce the query overhead, the lookup service should reschedule the request sequences and try to replace them 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 sub-sequences with the corresponding beneficial virtual streams. Undoubtedly, such a naïve scheduling approach has some problems: (1) large memory space and computation overhead, and (2) low match probability due to the irregularity of request sequences. By taking all these problems into account, we propose a new approach to speed up these processes.

3.2 The Proposed Approach

The proposed approach aims to reduce the computation overhead of both generation and scheduling of virtual streams in the naïve approach and improve the match probability of beneficial virtual streams. Although the interest-based mode

would lead to irregular request sequences and decrease the match probability, this problem might not be so serious in the real world because a user in the interest-based mode might not really care about the playout order of their demanded segments, i.e., rescheduling the playback order of these video segment could be acceptable. Therefore, we can let the lookup service rearrange the interest-based user request sequences to increase the total match probability of the beneficial virtual streams, thus the utilization of beneficial virtual streams can be improved.

On the other hand, to restrict the number of sub-sequences to be checked, we apply an interval batching mechanism on the lookup service to periodically regenerate the most valuable virtual streams. Consider the example we mentioned in the naïve approach again. Now we let the lookup service only deal with the request sequences which contain at most 12 segments. Therefore, the number of possible sub-sequences will be:

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

which is much smaller than the original number. However, the result is still too large to be stored in memory. So to reduce the number of possible sub-sequences to check, we further define a cost function called "estimated performance gain" to represent the performance gain of each sub-sequence. The sub-sequences with the largest performance gains will be selected as beneficial virtual streams. The estimated performance gain ($E(G)$) of a virtual stream is defined as below.

After the virtual stream is applied, we assume that the number 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 number of query messages n_Q' can be represented as:

$$n_Q' = ((r' - r_L') \cdot n_P).$$

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

$$E(G) = (n_Q - n_Q') \cdot o_Q - (r_B' - r_B) \cdot o_S. \quad (7)$$

Based on the equation (7), we propose a new algorithm for virtual stream generation as follows:

First, the lookup service finds the most M frequently accessed video segments in this scheduling interval. Then, it generates all possible sub-sequences by spanning the frequently accessed video segments. Since a request sequence can be rearranged, we only generate the sub-sequences by composing the segments as a nondecreasing order according to their publishing times. For example, let the popular segments in the current scheduling interval be A, B, and C, where A is published to the network before B and C. So the possible sub-sequences will be {A, B}, {A, B, C}, {B, C}, and {A, C}. Finally, the lookup service evaluates the

estimated performance gains $E(G)$ based on the equation (7) and selects the sub-sequences with the largest F gains as the beneficial virtual streams.

After the beneficial virtual streams are determined, the request sequences can be rescheduled with a rearranging strategy. Note that while rearranging a request sequence, each demanded video segment R_i cannot be scheduled to a time slot before its publishing time $t(R_i)$, i.e., cannot cause a "time violation." However, this procedure could also take a lot of computation overhead if the request sequence is irregular. To reduce the computation complexity, we let the lookup service sort the interest-based request sequences to a nondecreasing order on the segment publishing time $t(R_i)$. Here we prove that sorting the request sequences in advance can decrease the probability of causing a time violation, as below:

Consider a simple system that always supports only one video program at a time. Let R_i be the video segment requested at the i^{th} time slot. Let $t(R_i)$ be the publishing time of the segment R_i. (Note that if $t(R_i) > i$, a time violation will occur.) Let q' be a sorted request sequence which contains d segments, i.e.,

$$q' = \{R_1, R_2, \ldots, R_d\}, t(R_1) \le t(R_2) \le \ldots \le t(R_d).$$

Let a segment R_k cause a time violation at time k. Assume that R_k can be interchanged with another segment R_p so that both R_k and R_p will not cause a time violation. Thus, the possible results can be derived as the following two cases:

Case 1: $p < k$

Because $k > p$ and $t(R_k) > k$, we have $t(R_k) > k > p$. In this case, moving R_k to a previous slot, e.g., the p^{th} slot, will still cause a time violation.

Case 2: $p > k$

In this case, R_k will not cause a time violation, but R_p will. Since $t(R_p) \ge t(R_k)$ and $t(R_k) > k$. According to these, we now have $t(R_p) > k$. This implies that moving R_k to a subsequent slot, e.g., the p^{th} slot, will let the interchanged request R_p cause a new time violation at the k^{th} time slot.

Based on the above, we conclude that if a time violation occurrs in a sorted sequence, any interchanging will still cause a time violation. So to reduce computation overhead, we let the lookup service sort the interest-based request sequences in advance for virtual stream generation.

4. SIMULATION RESULTS

In this section, we evaluate the performance gain of the proposed virtual stream mechanism. As 0 illustrates, two platforms, in which one is a baseline platform and the other is an enhanced platform with beneficial virtual streams, are simulated to compare their system loads. Note that the baseline platform behaves just the same as a traditional P2P swarming system with an extensive number of small video programs.

Table 2. *Additional symbols*

$E(G)$	the estimated performance gain
n_Q'	the number of query messages after the virtual stream is applied
r_B'	the number of segments delivered by the backup service after the virtual stream is applied
r'	the total number of requests after the virtual stream is applied
r_L'	the number of requests served by the local buffers after the virtual stream is applied
F	the maximum number of beneficial virtual streams to be generated in each interval
v_i	the i^{th} beneficial virtual stream, where $1 \le i \le F$
q_i	the request sequence of the i^{th} client
d_i	the number of segments in the request sequence q_i
R_i	the i^{th} demanded segments in the request sequence q_i, i.e, $q_i = \{R_1, R_2, \ldots, R_{di}\}$
$t(R_i)$	the publishing time of the segment R_i

Table 3. *The two platforms for comparison*

Baseline platform	Without beneficial virtual streams
Enhanced platform (our scheme)	With beneficial virtual streams

In the core P2P overlays of both platforms, we let each client have 10 neighbors, and, to limit the query range of a request, let the TTL of each query message be 5 hops. Let the length of each video be one hour long, and let each video be divided into 12 five-minute video segments. The execution time of each simulation is 24 hours. Let the scheduling interval for virtual stream generation be one hour long. In each scheduling interval, the lookup service will determine at most 10 beneficial virtual streams according to the proposed approach. In addition, the maximum length of a beneficial virtual stream is five segments long. We use the term "normalized load" to compare the system performances of both platforms:

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

Since a user could change their viewing behavior at any time, we use a finite state machine to model the viewing behavior of each user, as illustrated in Figure 7. Under this finite state machine, a user can start from one of these five states (i.e., idle, live, review, serial-play, and interest-based), and then they can select to stay at the same state or go to another state in the next time slot.

Also, we assume that the original video programs of both platforms are from three TV channels, and the target audiences of these TV channels are different. That is, users in different viewing states have the different probabilities on accessing these three channels. Here we let the channel access pattern be the same as the example illustrated in 0. In this example, if a user is in the Live state, the probabilities that they will request video programs from the three channels are 50%, 30%, and 20% respectively. With the above viewing finite state machine and channel access pattern, the user viewing behavior can be modeled.

Now we evaluate the system performance of the baseline platform and the enhanced platform under different request patterns: (1) "live-major": with a high ratio of live viewing; (2) "serial-major": with a high ratio of serial-play viewing; (3) "random-major": with a 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 viewing model mentioned above. We then vary the number of clients from 10,000 to 100,000 to investigate the normalized load under these settings.

The simulation results illustrated in Figure 8 show that the system load of the serial-major can be improved the most, because the request sequences in serial-play mode easily generate both long and frequently accessed beneficial virtual streams. Besides, the system load of the interest-based mode still can be reduced by about 20%, although it is not as efficient as the serial-play mode. The live mode cannot be benefited by the virtual stream mechanism because each client receives the latest frame so that the virtual stream mechanism cannot be applied. Therefore, in the following simulations, we only focus on the viewing behaviors based on random-major and serial-major mode, and we modify two key parameters, i.e., the number of the beneficial virtual streams in a scheduling interval and the maximum length of a beneficial virtual stream, to investigate the corresponding service performance.

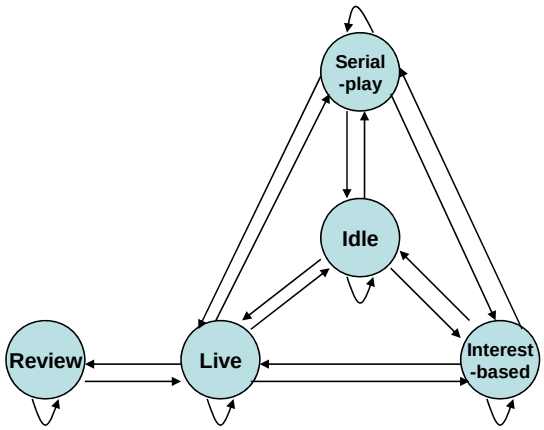

Figure 7. The finite state machine to model the user viewing behavior.

Table 4. *An example of channel cccess pattern*

	Channel 1	Channel 2	Channel 3
Live state	50%	30%	20%
Serial-play state	20%	70%	10%
Interest-based state	10%	30%	60%

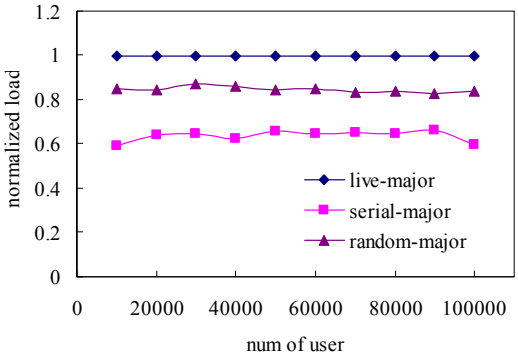

Figure 8. The impacts of request patterns.

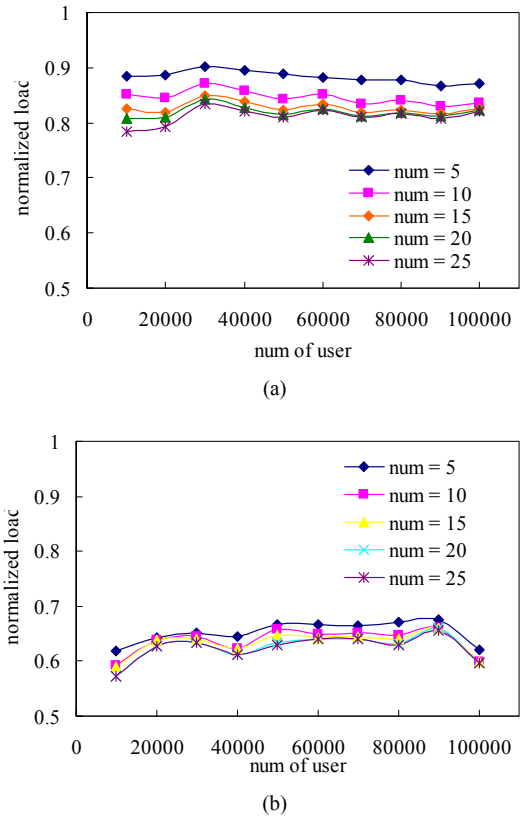

Figure 9. The impact of the number of generated beneficial virtual streams. (a) For random-major. (b) For serial-major.

Figure 9 depicts the simulation results by adjusting the maximum number of beneficial virtual streams generated in each scheduling interval time, where Figure 9(a) and Figure 9(b) are simulated with a random-major and a serial-major request pattern respectively. These figures demonstrate that the more beneficial virtual streams imply a lower normalized system load, and the request pattern in a serial-major mode has a larger improvement than that in a random-major mode. However, once the number of beneficial virtual streams reaches a bound, e.g., 20 streams if the number of users is 30,000, the improvement becomes insignificant. To derive the relation between the number of beneficial virtual streams and the performance gain will be one of our future works.

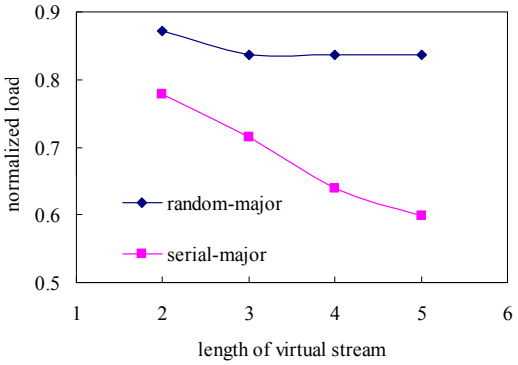

Figure 10. The impact of the maximum length of a virtual stream.

In the final simulation, we modify the maximum length of a beneficial virtual stream from two segments to five segments. Figure 10 illustrates the simulation results, demonstrating that a longer virtual stream implies a lower normalized system load. This figure also illustrates that the normalized system load of a serial-major request pattern has a greater improvement than that of random-major request pattern, which still conforms to the above simulation results.

5. CONCLUSIONS

We have proposed a virtual channel platform, which organizes DTV client devices to virtually support each user with a dedicated channel according to their desires. We have introduced a peer-to-peer overlay network for sharing their resources, such as their buffers and network bandwidth, 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 long video stream to reduce the number of query messages. The simulation results demonstrate that our virtual stream mechanism can significantly reduce the query overhead, thus 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

BitTorrent (2003). Retrieved from http://www.bittorrent.com.

Chorianopoulos, K., Lekakos, G., & Spinellis, D. (2003). The virtual channel model for personalized television. *Proc. EuroiTV2003*, 59-67.

Chorianopoulos, K., & Spinellis, D. (2004). Affective usability evaluation for an interactive music television channel. *Computers in Entertainment*, *2*(3), 14-18.

Chorianopoulos, K., & Spinellis, D. (2003). A metaphor for personalized television programming. *Proc. ERCIM Wor.: User Interfaces for All*, 187-194.

CoolStreaming (2005). Retrieved from http://www.coolstreaming.us.

Dana, C., Li, D., Harrison, D., & Chuah, C. N. (2005). BASS: BitTorrent assisted streaming system for video-on-demand. *Proc. IEEE MMSP*, 1-4.

Digital Video Broadcasting Project [DVB] (2007). Retrieved from http://www.dvb.org.

Hei, X., Liang, C., Liang, J., Liu, Y., & Ross, K. W. (2006). Insights into PPLive: A measurement study of a large-scale P2P IPTV system. *Proc. IPTV*.

KaZaA (2003). Retrieved from http://www.kazza.com.

LimeWire (2001). Retrieved from http://www.limewire.com.

Lv, J., Cheng, X., Jiang, Q., Ye, J., Zhang, T., Lin, I., & Wang, L. (2007). LiveBT: Providing video-on-demand streaming service over BitTorrent systems. *Proc. PDCAT*, 501-508.

Padmanabhan, V. N., Wang, H. J., Chou, P. A., & Sripanidkulchai, K. (2002). Distributing streaming media content using cooperative networking. *Proc. ACM NOSSDAV*, 177-186.

PeerCast (2001). Retrieved from http://www.peercast.org.

PPlive (2006). Retrieved from http://www.pplive.com.

Rowstron, A., & Druschel, P. (2001). Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proc. IFIP/ACM Int. Conf. Distributed Systems Platforms (Middleware 2001)*, *2218*, 329-350.

Ratnasamy, S., Francis, P., Handley, M., Karp, R., & Shenker, S. (2001). A scalable content-addressable network. *Proc. ACM SIGCOMM*, 161-172.

Stoica, I., Morris, R., Kaashoek, M., & Balakrishnan, H. (2001). Chord: A scalable peer-to-peer lookup service for Internet applications. *Proc. ACM SIGCOMM*, 149-160.

The Gnutella Protocol Specification (2001). http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf.

Vlavianos, A., Iliofotou, M., & Faloutsos, M. (2006). BiToS: Enhancing BitTorrent for supporting streaming applications. *Proc. IEEE INFOCOM*, 1-6.

Zhang, X., Liu, J. C., Li, B. & Yum, T.-S. P. (2005). CoolStreaming/DONet: A data-driven overlay network for efficient live media streaming. *Proc. IEEE INFOCOM*, 2102-2111.

Nien-Chen Lin received her B. S. degree in computer and information science from Tunghai University in 2004, and her M. S. degree in computer science from National Tsing Hua University in 2006.

She joined the Internet and Multimedia Application Technology Laboratory, Chunghwa Telecom Laboratories, in 2007 as an Associate Researcher. Her research interests involve multimedia delivery and peer-to-peer network communications.

Chen-Lung Chan received his B. S., M. S., and Ph. D. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1999, 2001, and 2006 respectively.

He joined the Computer and Communication Research Center, National Tsing Hua University, in 2006 as a Postdoc Researcher. His research interests include multimedia delivery, peer-to-peer over network, and wireless communications.

Jia-Shung Wang received his B. S. degree in mathematics from National Taiwan University in 1978, and his M. S. and Ph. D. degrees in computer science from National Tsing Hua University, in 1983 and 1986, respectively.

He joined the Computer Science Department, National Tsing Hua University, in 1986 as an Associate Professor and became a Full Professor in 1995. His current research interests cover several aspects of Multimedia Networking, Video Coding, and VLSI Design.