An Expanded Patching Technique using Four Types of Streams for True VoD Services

SookJeong Ha[1], IhnHan Bae[2], JinGyu Kim[3], YoungHo Park[3] and SunJin Oh[4]
[1] Department of Computer Engineering, Kyungpook National University, South Korea
[e-mail: sjha@knu.ac.kr]
[2] School of Computer and Information Engineering, Catholic University of Daegu, South Korea
[e-mail: ihbae@cu.ac.kr]
[3] School of Electrical Engineering, Kyungpook National University, South Korea
[e-mail: {kjg, parkyh}@knu.ac.kr]
[4] School of Computer and Communication System, Semyung University, South Korea
[e-mail: sjoh@semyung.ac.kr]
*Corresponding author: IhnHan Bae

Received July 22, 2009; revised September 7, 2009; accepted September 19, 2009; published October 30, 2009

Abstract

In this paper, we propose an expanded patching technique in order to reduce the server network bandwidth requirements to support true VoD services in VoD Systems. Double Patching, which is a typical multicast technique, ensures that a long patching stream delivers not only essential video data for the current client but also extra video data for future clients. Since the extra data may include useless data, it results in server network bandwidth wastage. In order to prevent a server from transmitting useless data, the proposed patching technique uses a new kind of stream called a linking stream. A linking stream is transmitted to clients that have received short patching streams, and it plays a linking role between a patching stream and a regular stream. The linking stream enables a server to avoid transmitting unnecessary data delivered by a long patching stream in Double Patching, so the server never wastes its network bandwidth. Mathematical analysis shows that the proposed technique requires less server network bandwidth to support true VoD services than Double Patching. Moreover, simulation results show that it has better average service latency and client defection rate compared with Double Patching.

Keywords: Server network bandwidth, true VoD, double patching, linking stream

DOI: 10.3837/tiis.2009.05.002

1. Introduction

Video-on-Demand (VoD) services, which transmit multimedia contents according to client requests, have become some of the most popular real-time multimedia applications available via the Internet. In a traditional client/server VoD system, if a client requests a video from a VoD server, the server accesses the video data in storage and transmits it to the client using its network bandwidth. Since the network bandwidth of a VoD server is limited, it is difficult to serve all clients immediately if many clients request videos asynchronously. In order to increase the number of clients for which a server provides true VoD (TVoD) services with limited server network bandwidth, multicast techniques such as Patching [1] and Double Patching [2] have been proposed. In unicast techniques, a server transmits a dedicated video stream for each client. Multicast techniques, however, offer efficient one-to-many data transmission by making multiple clients share the same video stream.

Patching is a multicast technique that enables a server to transmit only the beginning of the entire video data to clients and ensures that clients download the rest data of the video from an ongoing stream. By making multiple clients share an ongoing stream, Patching can reduce server network bandwidth requirements for TVoD services. Double Patching [2] ensures that a long patching stream delivers not only essential data for the current client but also extra data for future clients, so it significantly reduces the total amount of video data delivered by all streams.

In this paper, we propose an expanded patching technique using four types of streams (XP4S) to reduce server bandwidth requirements for TVoD services. In the proposed XP4S, when a server completes transmitting a patching stream, it initiates a linking stream for the clients that have received their respective short patching streams and shared the same patching stream. Since the server can compute the exact amount of data that the linking stream has to deliver by using the arrival time of the previous client request, the linking stream never delivers useless video data, unlike a long patching stream in Double Patching. Mathematical analysis using the performance model for Double Patching shows that the proposed technique requires less server network bandwidth to support true VoD services than Double Patching. We conduct a simulation study to compare the performance of our technique with that of Double Patching. The simulation results show that the proposed XP4S always improves the average service latency and client defection rate.

The remainder of this paper is organized as follows. We describe related multicast techniques for VoD services in section 2, and propose the XP4S in section 3. In section 4, we evaluate the performance of the proposed XP4S via mathematical equations derived to estimate server bandwidth requirements for TVoD services. In section 5, we compare the performance of the proposed XP4S with that of Double Patching by a simulation study and then conclude the paper in section 6.

2. Related Works

Recently, various techniques that make multiple clients share a stream have been proposed. Pyramid broadcasting [3], Skyscraper broadcasting [4], and Pagoda broadcasting [5] are techniques that repeatedly broadcast videos for large-scale VoD services. These techniques divide server network bandwidth into n logical channels, divide a single video file into n

different frame sizes, and repeatedly broadcast each frame on each assigned channel. By ensuring that the first frame has the smallest size and then ensuring that the next frame has a bigger size, they confine the worst client service latency to the size of the first frame regardless of the number of video requests. These broadcasting techniques, however, can only be used for very popular videos. In order to ensure that service latency is short or zero for all videos regardless of their popularity, video requests must be scheduled immediately [2].

Unicast techniques that transmit a new stream whenever a client requests a video are very simple and have no service latency. However, if the number of client requests increases rapidly, a server will suffer from the network bandwidth bottleneck. In order to alleviate the problems of unicast techniques, various multicast techniques such as Batching [6], Piggybacking [7], Patching [1], Optimal Patching [8], Scaling Patching [9], and Channel-Reservation Patching [10] have been proposed. Batching [6] gathers client requests for the same video during a period called a batching period, and then multicasts a single stream to the gathered clients. Batching enables a server to efficiently use limited server network bandwidth, but it cannot support TVoD services because it makes requests wait until a batching period ends.

2.1 Patching

Patching [1] enables clients to be served with no service latency by making them share an ongoing stream, so it can reduce server network bandwidth requirements for TVoD services. Multicast techniques [1][2][8][9][10] such as Patching assume that a client can download up to two streams simultaneously; it has two loaders that download data from streams and store it in its local buffer, and it has a video player that plays back the buffered data sequentially.

When a server receives the first client request for a video, it schedules a regular stream (R-stream) to deliver the entire data for the video. And then, if the server receives a new client request for the same video, it investigates the skew [1], which is the interval between the starting time of the latest R-stream and the current time, in order to schedule a new stream for the client request. If the skew exceeds a predefined time threshold, called a patching window, it schedules another R-stream to deliver the entire video data. Otherwise, it makes the client share the latest R-stream, so it schedules a new patching stream to deliver only the beginning of the entire video data that will be played back during the subsequent skew period. That is, the patching stream delivers the data that the new client has not received when the client starts receiving the R-stream. In this case, while the video player plays back the data, loader L_1 downloads from the patching stream and loader L_2 downloads the R-stream and stores the data in its local buffer. After the video player finishes playing back the patching stream, it continues to play back the buffered data and loader L_2 continues to download the R-stream simultaneously. By this means, the client can continuously play back the entire video data.

Since a server transmits an R-stream or a patching stream as soon as a request arrives, Patching can support TVoD services; since it enables multiple clients to share an R-stream by multicasting, it can significantly reduce server network bandwidth requirements for TVoD services. Optimal Patching [8] shows that the size of a patching window influences average server bandwidth requirements for TVoD services, and develops a performance model to derive the optimal patching window size.

2.2 Double Patching

We assume that the length of a stream means its playback time. As the skew between the latest R-stream and a new request becomes longer, the length of a patching stream also becomes

longer. In order to shorten the length of a patching stream, Double Patching introduces a long patching stream (L-stream). A client receiving an L-stream has to share the latest R-stream; the L-stream can be shared with future clients. It uses two time thresholds: a multicast window and a patching window. The multicast window is the minimum interval between two sequential R-streams; the patching window is the minimum interval between two sequential L-streams. We will use w_m to denote the size of the multicast window and w_p to denote the size of the patching window. The patching stream in Optimal Patching is called a short patching stream (S-stream) in Double Patching. The algorithm that a server uses to schedule a new stream for the current client is as follows:

- If the skew between the current client and the latest R-stream is greater than w_m, it schedules a new R-stream.
- Otherwise, it schedules a new L-stream /S-stream as follows:
  - If the skew between the current client and the latest R-stream/L-stream is less than or equal to w_p, it schedules a new S-stream to deliver the beginning of the entire video data that the client has not received from the latest R-stream/L-stream. In this case, to play back the entire video data, the client plays back data in the following order: S-stream, L-stream if the client has to share it, and R-stream.
  - Otherwise, it schedules a new L-stream to deliver the beginning of the video that the client has not received from the R-stream and the following data of the video that will be continuously played back during $2 \cdot w_p$ time units after playback of the beginning. In this case, the client first plays back the data from the L-stream and then the data from the R-stream.

Fig. 1 shows how to schedule streams for client requests in Optimal Patching and Double Patching. As shown in **Fig. 1**, the length of an L-stream is ($2 \cdot w_p$) longer than that of a corresponding patching stream in Optimal Patching. However, because w_p is much smaller than that in Optimal Patching, S-streams that are scheduled for clients sharing the same L-stream become very short. As a result, Double Patching can decrease the total amount of transmitted video data by 50% compared with Optimal Patching, and significantly reduce the server network bandwidth requirements [2].

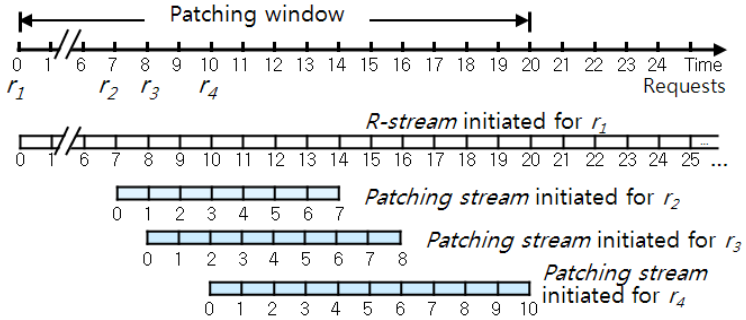
3. Expanded Patching Technique using Four Types of Streams

We will use $D[t_1, t_2]$ to denote the video data played back from time t_1 to t_2 assuming that the video is played back from time 0. We assume that the amount of data delivered by a stream means the playback time of the data. We will use w_m and w_p to denote the size of a multicast and patching window respectively. Multicast techniques such as Patching are applied to the requests of the same video. The server network bandwidth requirement for a server having many videos is the total bandwidth requirements for all videos. Therefore, we assume that the VoD servers discussed in this paper have only one video in order to simplify the problem, as in many patching techniques including Double Patching.

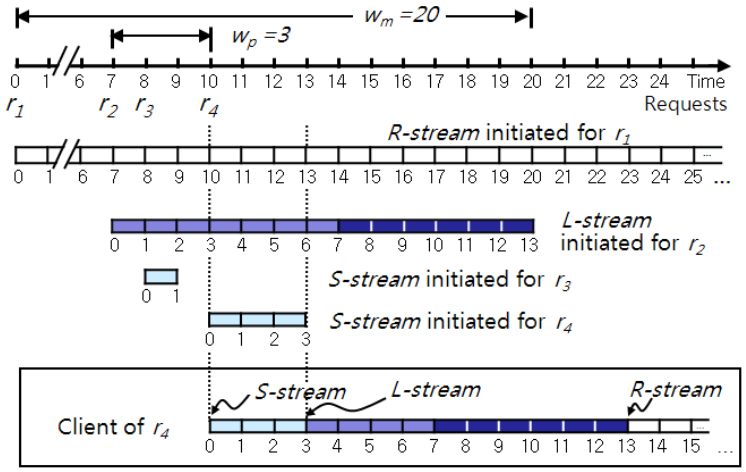
3.1 Motivation and Contribution

An L-stream plays an important role in shortening the length of an S-stream. An L-stream includes the beginning of the entire video data in order to cover the skew between the latest R-stream and the current client, and we will call this part essential data. In addition to essential data, an L-stream includes the following data that will be played back during $2 \cdot w_p$ time units after completion of the essential data, and we will call this extra data. The essential data of an

L-stream is for the current client; the extra data is for the future clients that have to share the L-stream.



(a) Streams in Optimal Patching



(b) Streams in Double Patching and continuous playback at the client station of request r_4

Fig. 1. Optimal Patching vs. Double Patching

The reason why an L-stream has to deliver extra data is as follows. Let's consider clients that receive their respective S-streams and have to share the same L-stream and the same R-stream initiated at t_L and t_R respectively. The last possible client c is the client whose request arrives at the end of the patching window of the L-stream t_L+w_p such as request r_4 in **Fig. 1(b).** Playing back the S-stream, client c downloads the L-stream. Client c cannot download the R-stream during the time it is downloading the S-stream and the L-stream, since it can download only two streams simultaneously, as described in section 2.1. As a result, it cannot download the R-stream during w_p. Moreover, since the request of client c arrives at the server w_p time units later after the L-stream is initiated, it has already missed the data delivered by the R-stream from time t_L to t_L+w_p. In order to ensure that client c downloads the missing data, Double Patching makes the L-stream deliver the extra data $D[(t_L-t_R), (t_L-t_R)+2\cdot w_p]$ as well as the essential data $D[0, (t_L-t_p)]$. $D[7, 13]$ of the L-stream in **Fig. 1(b)** corresponds to extra data.

The extra data of the L-streams, however, has following problems. Unless the last request within the patching window of an L-stream arrives exactly at the end of the window, the extra data always includes useless data. Every L-stream always delivers extra data, which are $2\cdot w_p$ units long, for any possible future request arriving at the end of the patching window of the

L-stream. However, the exact data that a client requesting a video at t_0 ($t_L<t_c\leq t_L+w_p$) cannot download from the latest R-stream is $D[(t_L-t_R), (t_L-t_R)+2\cdot(t_c-t_L)]$. Therefore, if the arrival time of the last request within the patching window of the L-stream is t_c, the end of the extra data $D[(t_L-t_R)+2\cdot(t_c-t_L), (t_L-t_R)+2\cdot w_p]$ results in server network bandwidth wastage. In the worst case, the entire extra data becomes useless if there is no request within the patching window of the L-stream.

Therefore, we propose a multicast technique that completely prevents a server from transmitting unnecessary video data and uses four types of streams: regular stream (R-stream), patching stream (P-stream), short patching stream (S-stream) and linking stream (LK-stream). The proposed patching technique is based on the following: at the end of the patching window of an L-stream, a server can identify the request arrival times of all clients sharing the L-stream; the transmission starting time of the L-stream's extra data is always after the end of the patching window of the L-stream. This means that the server can determine the exact amount of extra data for clients sharing the same L-stream, based on their request arrival times when transmission of the essential data is completed.

Based on the above, the proposed technique ensures that an L-stream delivers only essential data, so we will call this stream a patching stream instead of a long patching stream. Later, when the patching stream (P-stream) is completed, it schedules a new stream, called a linking stream, in order to deliver the necessary data for all the clients sharing the P-stream. Since an LK-stream never delivers unnecessary data, our technique completely prevents server network bandwidth wastage caused by the extra data of an L-stream. As a result, server network bandwidth requirements for TVoD services can be reduced. Using the same server network bandwidth, our technique always has better average service latency and client defection rate compared with Double Patching.

3.2 Proposed XP4S

An R-stream, an S-stream, w_m, and w_p in the proposed XP4S, have the same meanings as in Double Patching. A P-stream is scheduled in the same manner that Double Patching schedules an L-stream, except that it does not deliver extra data. In other words, a P-stream delivers the beginning of the entire video data that the current client has not received from the latest R-stream and it does not deliver extra data for possible future clients. If a client request is within the patching window of the latest R-stream/P-stream, an S-stream is scheduled to make it share the R-stream/P-stream. Let's consider a client *c* for which a new S-stream is scheduled and which has to share the latest P-stream and the latest R-stream. As described in section 3.1, since client *c* has to download the P-stream while playing back the S-stream, it cannot download the R-stream until it finishes downloading the S-stream. As a result, it misses the intermediate part of the R-stream delivered from the starting time of the P-stream to the ending time of the S-stream. It is an LK-stream that delivers such missing data and plays a linking role between the P-stream and the R-stream in order to enable client *c* to play back the entire video data continuously. Clients that have to download four types of streams such as client *c* play back the entire video in the following order:S-stream, P-stream, LK-stream and R-stream.

In XP4S, a VoD server can multicast a single LK-stream to all clients that have received their respective S-streams and shared the same P-stream. The reason is as follows. The shortest length of a P-stream is w_p+1 as with the length of the P-stream for request r_2 shown in **Fig. 4**. The largest skew between a P-stream and the last client that can share it is w_p. Therefore, when a server completes transmitting a P-stream, it already knows the arrival time of the last client sharing the P-stream. All clients sharing a P-stream have to receive their own LK-streams when the P-stream is completed. The beginning data of the LK-streams for the clients is the

same; the ending data is different. Therefore, the data delivered by the LK-stream for the last client includes all the data. Thus, it is certain that a server can schedule an LK-stream by using the request arrival time of the last client within the patching window of a P-stream after the P-stream is completed, and it can multicast the LK-steam to all the clients within the patching window of the P-stream. In order to guarantee that a client can play back an entire video continuously, a server must initiate an LK-stream as soon as it completes the related P-stream. It can be guaranteed by allocating only the channel used to transmit the related P-stream to the LK-stream immediately after completion of the P-stream.

Fig. 2 shows the stream-scheduling algorithm used by a server in XP4S. The server uses this algorithm to schedule an R-stream, P-stream, S-stream, or LK-stream in the following cases: it completes a P-stream; it has a free channel when a new request arrives; it has at least one video request in its waiting queue when a channel becomes free.

- t: current time the server schedules a new stream for clients that have requested video V
- l: length of video V
- *Client*: list of clients for which the server has to schedule a new stream at time t
- t_R and t_P: starting times of the latest R-stream and the latest P-stream respectively
- $t_R(P_t)$: starting time of the latest R-stream when a P-stream is initiated at time T
- $t_S(P_t)$: starting time of the S-stream for the last client sharing the P-stream initiated at time T
- C_R, C_P and C_S: channels transmitting an R-stream, a P-stream and an S-stream respectively
- C_{free}: free channel to be used to transmit a new stream
- R_T-*list*, P_T-*list* and S_T-*list*: lists of clients to which the server has to multicast the R-stream, the P-stream, and the S-stream initiated at time T respectively
- LK_T-*list*: list of clients to which the server has to multicast an LK-stream as soon as it completes transmitting the P-stream initiated at time T
- *service*(C_R, C_P, C_S, l_{LK}): service token that the server sends to *Client*, where C_R, C_P and C_S are channels transmitting an R-stream, a P-stream, and an S-stream, respectively, and l_{LK} is the length of an LK-stream to be transmitted on channel C_P

if (P-stream, initiated at T, on channel C is completed and LK_T–*list* ≠ *null*) {
    Schedule a new LK-stream to deliver $D[(T\text{-}t_R(P_T)), (T\text{-}t_R(P_T))+2\cdot(t_S(P_T)\text{-}T)]$;
    Multicast the LK-stream to LK_T-*list* on channel C;
}
else if (there is no R-stream or $(t\text{-}t_R) > w_m$) {
    Schedule a new R-stream to deliver $D[0, l]$;
    Set R_t-*list*=*Client*, $C_R=C_{free}$ and $t_R=t$;
    Send *service*(C_R, *null*, *null*, *null*) to *Client*;
    Multicast the R-stream to R_t-*list* on channel C_{free};
}
else if $((t\text{-}t_R) \le w_p)$ {
    Schedule a new S-stream to deliver $D[0, t\text{-}t_R]$;
    Set $C_S = C_{free}$ and S_t-*list* = *Client*;
    Append *Client* to $R_{t_R} - list$;
    Send *service*(C_R, *null*, C_S, *null*) to *Client*;
    Multicast the S-stream to S_t-*list* on channel C_{free};
}
else if $((t\text{-}t_P) \le w_p)$ {
    Schedule a new S-stream to deliver $D[0, t\text{-}t_p]$;
    Append *Client* to $R_{t_R} - list$, $P_{t_R} - list$ and $LK_{t_R} - list$;
    Set $C_S = C_{free}$ and $t_S(P_{t_P}) = t$;

```
        Set St-list = Client;
        Send service(CR, CP, CS, 2·(t-tP)) to Client;
        Multicast the S-stream to St-list on channel Cfree;
}
else { // (t-tP) > wp
        Schedule a new P-stream to deliver D[0, t-tR];
        Set Pt-list = Client;
        Set CP = Cfree, tP = t and tR(Pt) = tR;
        Append Client to R_{t_R} − list ;
        Send service(CR, CP, null, null) to Client;
        Multicast the P-stream to Pt-list on channel Cfree;
}
```

Fig. 2. Server's stream-scheduling algorithm in XP4S

When a client receives a service token *service*(C_R, C_P, C_S, l_{LK}) from a server, it plays back a video using the algorithm shown in **Fig. 3**. We assume that each client has two loaders and a video player having responsibility for downloading video streams and sequentially playing back buffered video data, respectively, as described in section 2.1.

```
if (CP = CS = lLK = null ) {
    While loader L1 downloads data from the R-stream on channel CR, the video-player plays it
    back simultaneously;
}
else if (CP ≠ null and CS = null) {
    While loader L1 downloads data from the P-stream on CP and the video-player plays it back
    simultaneously, loader L2 downloads data from the R-stream on CR and stores it in its local
    buffer;
    After the video-player finishes playing back the P-stream, it plays back the buffered data
    sequentially;
}
else if (CS ≠ null and CP = null) {
    While loader L1 downloads data from the S-stream on CS and the video-player plays back the
    data simultaneously, loader L2 downloads data from the R-stream on CR and stores it in its local
    buffer;
    After the video-player finishes playing back the S-stream, it plays back the rest data of the video
    in the buffer sequentially;
}
else if (CS ≠ null and CP ≠ null) {
    While loader L1 downloads data from the S-stream on CS and the video-player plays it back
    simultaneously, loader L2 downloads data from a P-stream on CP and stores it in its local buffer;
    When loader L1 completes downloading the S-stream, it switches to the R-stream on CR and
    stores the data in its local buffer;
    When loader L2 completes downloading the P-stream, it continues to download the LK-stream
    on CP and stores it in its local buffer;
    When the video-player finishes playing back the S-stream, it plays back the rest data of the
    video in the buffer sequentially;
}
```

Fig. 3. Client's video playback algorithm in XP4S

Fig. 4 shows four different streams for requests r_1, r_2, r_3 and r_4 in XP4S. Since w_p=6, the server schedules a P-stream to deliver $D[0, 7]$ for r_2. Since r_4 arrives within the patching window of the P-stream, it schedules an S-stream to deliver $D[0, 2]$. When it completes transmitting the P-stream at time 14, it knows that the last request within the patching window of the P-stream is r_4. Therefore, it multicasts an LK-stream that delivers $D[7, 11]$ to the clients of r_3 and r_4. The client of r_3 plays back $D[0, 1]$ from the S-stream, then $D[1, 7]$ from the P-stream, then $D[7, 0]$ from the LK-stream, and finally $D[9, l]$ from the R-stream.

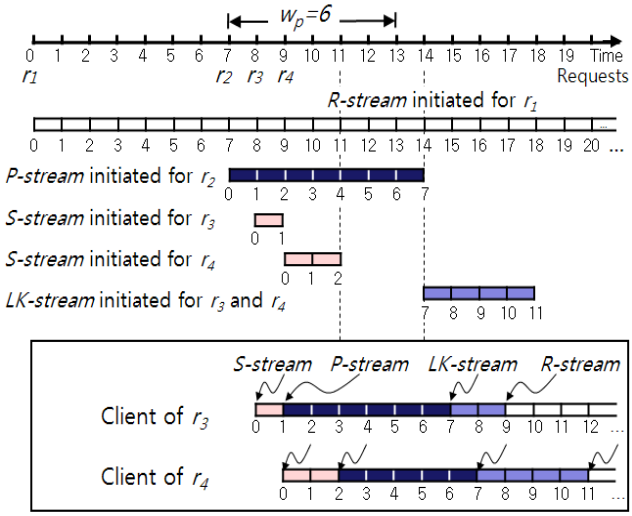


Fig. 4. Streams scheduled in XP4S and continuous playback of a video at client stations

4. Performance Evaluation

In this section, we evaluate the performance of the proposed XP4S based on the analytical performance model developed by Cai et al. in [2]. This model estimates the average server bandwidth required to support TVoD services when a server has a single video and client request arrivals are generated according to a Poisson process. In [2], an R-stream and the following L-streams and S-streams initiated before the next R-stream form a multicast group. Since an LK-stream, if necessary, is initiated as soon as the related P-stream is completed, we include the LK-streams initiated immediately after the P-streams in a multicast group in the same group. The S-streams initiated within the patching window of the same stream form a patching group, as shown in **Fig. 5**.

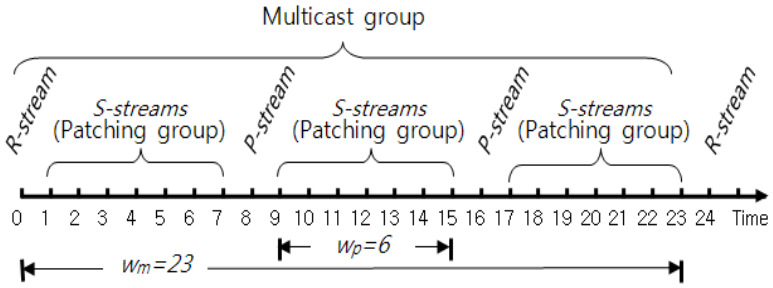


Fig. 5. Multicast and patching groups

Let D_R denote the amount of data delivered by an R-stream in a multicast group. Let D_P, D_{LK} and D_S denote the average total amount of data delivered by all P-streams, LK-streams and S-streams in the same multicast group, respectively. If v is the video playback rate, λ is the arrival rate of client requests, l is the length of the video (total playback time) and the time unit is the second, D_R, D_P, D_{LK} and D_S can be calculated as follows. Since one multicast group has only one R-stream,

$$D_R = v \cdot l.$$

The average request arrival interval is $I=1/\lambda$ and the average P-stream interval is $I_P=w_p+1$. The average number of P-streams in one multicast group is $C_P = \left\lfloor \frac{w_m}{I_P} \right\rfloor$. Since the amount of data delivered by a P-stream is equal to the skew between the latest R-stream and the P-stream,

$$D_P = v \cdot \sum_{n=1}^{C_P} (n \cdot I_P).$$

The data delivered by an LK-stream is determined by the arrival time of the last request within the patching window of a P-stream. Since the average number of requests arriving within the patching window of a P-stream initiated at t_P is $\left\lfloor \frac{w_p}{I} \right\rfloor$, the average arrival time of the last request within the patching window is $t_S = t_P + \left\lfloor \frac{w_p}{I} \right\rfloor \cdot I$. The average amount of data delivered by the LK-stream following the P-stream initiated at t_P is $2 \cdot (t_S - t_P) = 2 \cdot \left\lfloor \frac{w_p}{I} \right\rfloor \cdot I \cdot$ Since the average number of LK-streams in one multicast group is equal to the number of P-streams in the group, it follows that

$$D_{LK} = v \cdot C_P \cdot (2 \cdot \left\lfloor \frac{w_p}{I} \right\rfloor \cdot I) \cdot$$

On average, a multicast group has C_P+1 patching groups. Since the average total amount of data delivered by all S-streams in a patching group is the same, we consider the S-streams initiated from time 0 to w_p assuming that an R-stream is initiated at time 0. The length of data delivered by one of the S-streams is equal to the skew between the R-stream and the S-stream. If k S-streams are initiated from time t to $t+\Delta t$ and we assume that Δt is negligible, we can say that the average total amount of data delivered by k S-streams is approximately $v \cdot t \cdot k$. If we let $P(k, T)$ denote the probability that k S-streams are initiated during interval T, the average total amount of data delivered by the S-streams initiated from time t to $t+\Delta t$ is $\sum_{k=1}^{\infty} v \cdot t \cdot k \cdot P(k, \Delta t)$. If we divide the patching window into $\left\lfloor \frac{w_p}{\Delta t} \right\rfloor$ small intervals, the average total amount of data delivered by one patching group is approximately $\sum_{t=1}^{\left\lfloor \frac{w_p}{\Delta t} \right\rfloor} \sum_{k=1}^{\infty} v \cdot t \cdot k \cdot P(k, \Delta t)$. If $w_m > C_P \cdot I_P$, the last patching group in a multicast group includes the S-streams initiated during $w_m - C_P \cdot I_P$ before the

multicast window ends. The average total amount of data delivered by the S-streams in the last patching group is $\sum_{t=1}^{\left\lfloor \frac{w_m - C_P I_P}{\Delta t} \right\rfloor} \sum_{k=1}^{\infty} v \cdot t \cdot k \cdot P(k, \Delta t)$. Therefore, the average total amount of data delivered by all S-streams in a multicast group is

$$D_S = C_P \sum_{t=1}^{\left\lfloor \frac{w_p}{\Delta t} \right\rfloor} \sum_{k=1}^{\infty} v \cdot t \cdot k \cdot P(k, \Delta t) + \sum_{t=1}^{\left\lfloor \frac{w_m - C_P I_P}{\Delta t} \right\rfloor} \sum_{k=1}^{\infty} v \cdot t \cdot k \cdot P(k, \Delta t).$$

Since we assume that client request arrivals are generated according to a Poisson process, $P(k,T) = \frac{(\lambda T)^k e^{-\lambda T}}{k!}$. The average number of requests arriving during T is $\sum_{t=1}^{\infty} k \cdot P(k,T) = \lambda \cdot T$. If we set Δt equal to 1 second, it follows that

$$D_S = C_P \sum_{t=1}^{w_p} v \cdot t \cdot \lambda + \sum_{t=1}^{w_m - C_P I_P} v \cdot t \cdot \lambda.$$

The average R-stream interval, or the average multicast group interval, is w_m+I. Therefore, we can estimate the average server bandwidth required to support TVoD services in the proposed XP4S as follows:

$$B_{XP4S} = \frac{D_R + D_P + D_{LK} + D_S}{w_m + I}. \quad (1)$$

L-streams are scheduled in the same manner as P-streams, but they always deliver extra data as well as essential data. If D_L is the average total amount of data delivered by all L-streams in a multicast group, then

$$D_L = v \cdot \sum_{n=1}^{C_P} (n \cdot I_P + 2 \cdot w_p).$$

Therefore, we can estimate the average server bandwidth required to support TVoD services in Double Patching as follows:

$$B_{DP} = \frac{D_R + D_L + D_S}{w_m + I}. \quad (2)$$

From equation (1) and (2), we can find the optimal w_m and w_p for deriving the minimum network bandwidth B, and then use this value as the server network bandwidth. In order to compare the performance of the proposed XP4S with that of Double Patching, we use B as a performance metric. In order to find the performance gain of our XP4S compared to Double Patching, we first obtain the values of w_m and w_p for deriving B for Double Patching by equation (2). Then, we use the obtained values as the values of w_m and w_p for the XP4S. **Table 1** shows the parameters we used in our performance evaluation.

Table 1. Parameters used in the performance evaluation

Parameter	Range
Video length (l)	10~90 minutes
Client buffer size (b)	5~25 minutes
Average request arrival interval (I)	5~80 minutes
Video playback rate (v)	1.5 Mbps

Fig. 6 shows the effect of the average request arrival interval on B when l=90 and b=10 or 20. The proposed XP4S reduces B by 8.2% and 11.4% on average when b=10 and b=20, respectively.

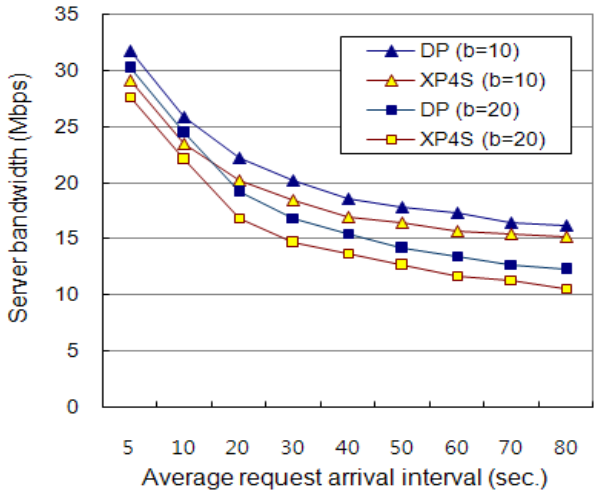


Fig. 6. Average request arrival interval vs. server network bandwidth requirement (B) for TVoD services

Fig. 7 shows the effect of the video length on B when b=10 and I=20 or 60. The longer the video length is, the larger the number of streams delivering data simultaneously at a given point of time. Thus, as the video length becomes longer, B becomes larger. As shown in **Fig. 7**, XP4S reduces B by 12.9% and 13.7% on average when I=20 and I=60, respectively.

Finally, **Fig. 8** shows the effect of the client buffer size on B when I=90, I=20 or 40. As shown in **Fig. 8**, XP4S reduces B by 10% on average.

5. Simulation Study

In this section, we compare the performance of the two patching techniques in terms of the client defection rate and average service latency, based on a simulation study. The client defection rate is the ratio of the number of client requests canceled by the clients to the number of total requests, and it represents the server throughput. The average service latency is the average period of time that a client request is held in the waiting queue of a server until the client gets served, and it represents the quality of service. We make the following assumptions. The video length l is 90 minutes, a client is watching a video sequentially from beginning to

end and the waiting time until it cancels its request varies over the range 10 to 90 seconds. Other parameters used in the simulation study are given in **Table 1**, and the total number of client requests is 100,000. As in section 4, we first decide the values of w_m and w_p for Double Patching and then apply the values to XP4S.

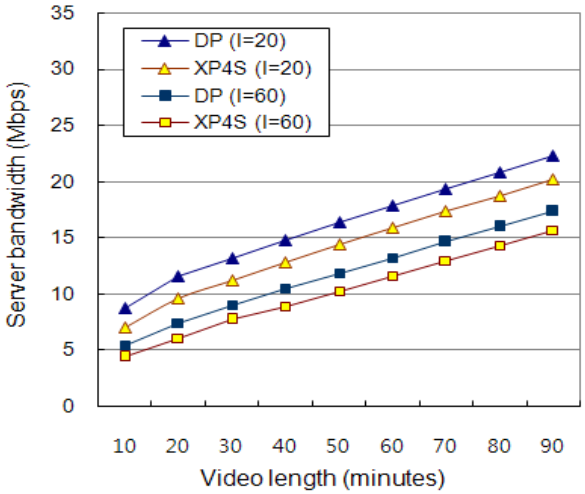


Fig. 7. Video length vs. server network bandwidth requirement (*B*) for TVoD services

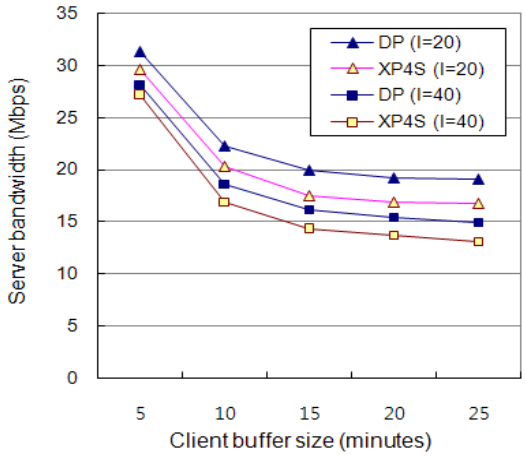


Fig. 8. Client buffer size vs. server network bandwidth requirement (*B*) for TVoD services

Fig. 9 and **Fig. 10** show the effect of the number of server channels, $\lceil B/v \rceil$, on the service latency and the defection rate when l=90, b=15 and I=10.

As shown in **Fig. 9** and **Fig. 10**, XP4S is always better than Double Patching. As the number of server channels decreases, the performance gain of our XP4S increases. This means that XP4S works better when the server load is higher. When the server has 16 channels, where 16 is equal to $\lceil B_{DP}/v \rceil$ that is determined from equation (2), our XP4S immediately serves 58,361 clients with no service latency, while Double Patching immediately serves 52,822 clients. In this case, XP4S improves the average service latency by 10.5% for all served clients.

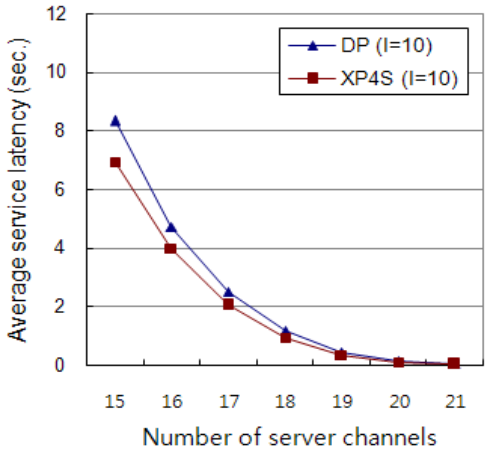


Fig. 9. Effect of the number of server channels on the average service latency

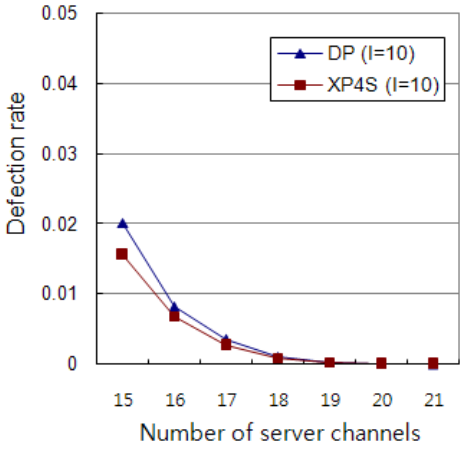


Fig. 10. Effect of the number of server channels on the client defection rate

Fig. 11 and **Fig. 12** show the effect of the number of server channels on the service latency and client defection rate when l=90, b=15 and I=30. As shown in **Fig. 11** and **Fig. 12**, XP4S is always better than Double Patching. When the server has 12 channels, where 12 is equal to $\lceil B_{DP} / v \rceil$ that is determined from equation (2), our XP4S immediately serves 72,275 clients, while Double Patching immediately serves 63,390 clients. In this case, XP4S improves the average service latency by 12.3% for all served clients.

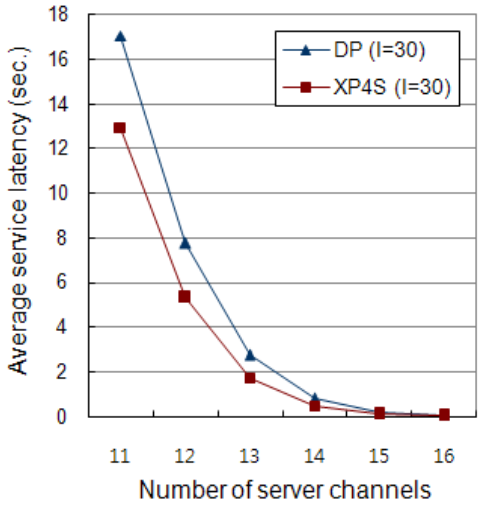


Fig. 11. Effect of the number of server channels on the average service latency

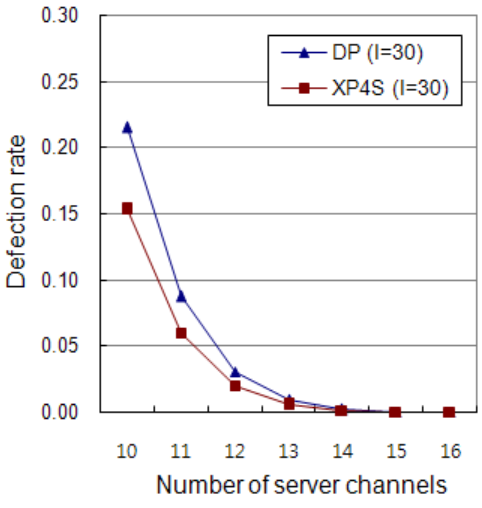


Fig. 12. Effect of the number of server channels on the client defection rate

We include **Fig. 13** and **Fig. 14** in order to show the other simulation results. **Fig. 13** shows the results when b=10 and the number of server channels is 15; **Fig. 14** shows the results when b=30 and the number of server channels is 11. As shown in the figures, the proposed XP4S is always better than Double Patching. We omit the results about the client defection rate, because the improvement is less than 1%.

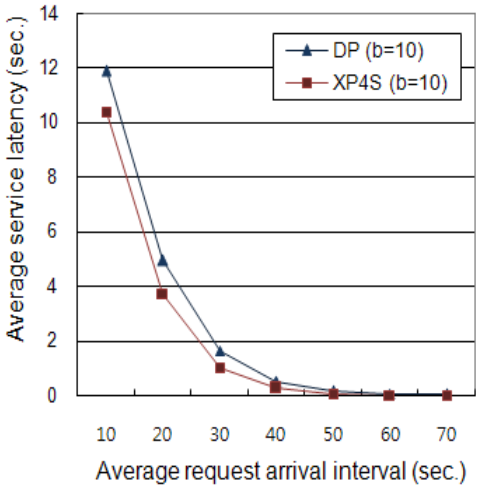


Fig. 13. Effect of the average request arrival interval on the average service latency

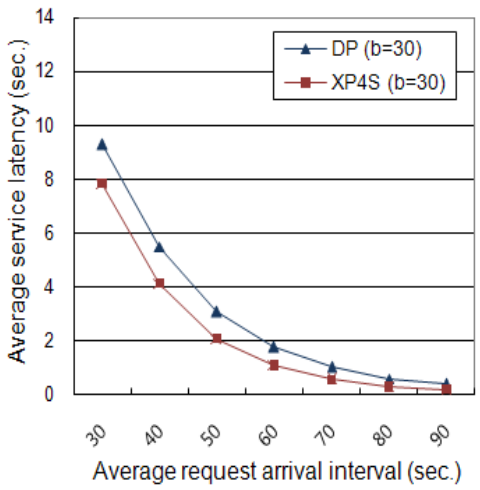


Fig. 14. Effect of the average request arrival interval on the average service latency

From the simulation study, we discovered the following. Although the average server bandwidth determined from equation (2) is not enough to support TVoD services to all clients, the proposed XP4S always improves not only the client defection rate and the average service latency but also the number of clients having TVoD services under the same conditions.

6. Conclusion

In this paper, we have proposed an expanded patching technique using four types of streams: R-stream, P-stream, S-stream and LK-stream. Although Double Patching improves on Optimal Patching by introducing an L-stream that delivers extra data so possible future clients can share it, the L-stream has a chance of delivering useless data. Using LK-streams, the proposed XP4S completely prevents the server network bandwidth wastage that can be generated by the extra data of Double Patching. Unlike an L-stream, a P-stream in XP4S delivers essential data with no extra data. Instead of the extra data, when the P-stream is completed, XP4S schedules an LK-stream to deliver the necessary data for clients whose requests have arrived within the patching window of the P-stream. A server can know what requests have arrived within the patching window of a P-steam when it completes transmitting it. Therefore, it can decide the exact amount of data that an LK-stream has to deliver by using the arrival time, so the LK-stream never delivers useless data. We have verified that XP4S requires less server network bandwidth to support TVoD services than Double Patching, by the performance model in section 4. Via the simulation study, we have verified that XP4S not only has a better average service latency and client defection rate but also a larger number of clients having TVoD Services compared with Double Patching. Our future works will improve the performance model so as to estimate the exact server network bandwidth requirements to support TVoD services, and expand XP4S so as to support interactive VCR functions.

References

[1] K. Hua, Y. Cai, and S. Sheu, "Patching: A Multicast Technique for True Video-on- Demand Services," *Proc. of ACM Multimedia*, pp. 191-200, 1998.

[2] Ying Cai, Wallapak Tavanapong, and Kien A. Hua, "A Double Patching Technique for Efficient Bandwidth Sharing in Video-on-Demand Systems," *Journal of Multimedia Applications and Tools*, Vol. 32, No. 1, pp. 115-136, 2007.

[3] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation-based Pyramid Broadcasting Scheme for Video-on-Demand systems," *Proc. of International Conference on Multimedia Computing and Systems*, pp. 118-126, 1996.

[4] K. A. Hua and S. Sheu, "Skyscraper Broadcasting: A New Broadcasting Scheme for Metropolitan Video-On-Demand Systems," *Proc. of the ACM SIGCOMM'97*, pp. 89-100. 1997.

[5] J. F. Paris, S. W. Carter, and D. D. E. Long, "Efficient Broadcasting Protocols for Video on Demand," *Proc. of SPIE's Conference on Multimedia Computing and Networking* (*MMCN'99*), pp. 317-326, 1999.

[6] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling Polices for an On-Demand Video Server with Batching," *Proc. of the 2nd ACM Multimedia Conference*, pp. 25-32, 1994.

[7] L. Golubchik, J. Lui, and R. Muntz, "Adaptive Piggybacking: Arrival Technique for Data Sharing in Video-on-Demand Service," *ACM Multimedia Systems*, Vol. 4, No. 3, pp.140-155, 1996.

[8] Y. Cai, K. Hua, and K. Vu, "Optimizing Patching Performance," *Proc. SPIE/ACM Conference on Multimedia Computing and Networking*, pp. 204-215, 1999.

[9] H.-Y. Lee, S.-J. Oh, S.-J. Ha, and I.-H. Bae, "Design and Evaluation of a Scaling Patching Technique for VOD Servers," *LNAI* 3214, pp. 219-226, 2004.

[10] S.-J. Ha and I.-H. Bae, "Determination of Reserved Channel Capacity for Popular Video in Video-on-Demand Systems," *Journal of KISS: Computer Systems and Theory*, Vol. 30, No. 5-6, pp. 223-231, 2003.

SookJeong Ha received the B.S. degree in computer science from Keimyung University, S. Korea, in 1988, the M.S. degree in computer science from Chungang University, S. Korea in 1990, and the Ph.D. degree in computer science from Catholic University of Daegu, S. Korea in 1998. She was a visiting professor in the School of Electrical Engineering and Computer Science, Kyungpook National University, Korea, from 2001 to 2005. She is currently a visiting professor in the Department of Computer Engineering, Kyungpook National University, Korea. Her research interests include wireless networks, mobile P2P systems, and multimedia systems.

IhnHan Bae received his Ph.D. degree in Computer Engineering from Chungang University, S. Korea, in 1990. He was a post-doctoral in Computer Science and Engineering, The Ohio State University, USA, in 1996~1997, and a visiting scholar in the Department of Computer Science, Old Dominion University, Norfolk, Virginia, USA, in 2002~2003. He is now an exchange professor in the Department of Computer Science, Old Dominion University, Norfolk, Virginia, USA. He is also a full professor in the School of Computer and Information Communication Engineering, Catholic University of Daegu, S. Korea. His research interests include distributed systems, multimedia systems, wireless mobile networks, ubiquitous computing, and mobile P2P systems.

JinGyu Kim received the B.S. degree in electrical engineering from Kyungil University, the M.S. and Ph.D. degrees in electrical engineering from Kyungpook National University, Daegu, S. Korea, in 1990, 1994, and 1998, respectively. In 2001-2008, he was an associate professor in the School of Electronics and Electrical Engineering at Sangju National University, S. Korea. He is currently an associate professor in the School of Electrical Engineering at Kyungpook National University, S. Korea. His research interests include non-thermal discharge plasma, application to the ecosystems, multimedia systems, and ubiquitous computing.

YoungHo Park received his BS, MS, and Ph. D degrees in electronic engineering from Kyungpook National University, Daegu, S. Korea in 1989, 1991, and 1995, respectively. He is currently a professor in the School of Electrical Engineering at Kyungpook National University. In 1996-2008, he was a professor in the School of Electronics and Electrical Engineering at Sangju National University, S. Korea. In 2003-2004, he was a visiting scholar in the School of Electrical Engineering and Computer Science at Oregon State University, USA. His research interests include computer networks, multimedia systems, and security.

SunJin Oh received the Bachelor of Engineering degree from Han Yang University, Korea, in 1981, and the Post Bachelor and the Master of Science degree in Computer Science from Wayne State University and University of Detroit, U.S.A. in 1987 and 1989 respectively. He completed the Ph.D. course in Oklahoma State University, U.S.A. and received the Ph.D. degree in Computer Science from Catholic University of Daegu in 1999. Currently, he is a professor at the department of Computer and Information Science in Semyung University, Korea. His research interests include Wireless Mobile Ad-Hoc Network, Vehicular Ad Hoc Network, Mobile Online Game, Embedded System Software, Distributed Multimedia Systems, and True VOD Systems. He is a member of the IEEE, IASTED, ACIS, IWIT, KIPS, KSII and KMMS.