Practical Patching for Efficient Bandwidth Sharing in VOD Systems

Sook-Jeong Ha*

ABSTRACT

Recursive Patching is an efficient multicast technique for large-scale video on demand systems and recursively shares existing video streams with asynchronous clients. When Recursive Patching initiates a transition stream, it always makes a transition stream have additional data for the worst future request. In order to share a VOD server’s limited network bandwidth efficiently, this paper proposes Practical Patching that removes the unnecessary data included in the transition stream. The proposed Practical Patching dynamically expands ongoing transition streams when a new request actually arrives at the server. As a result, the transition streams never have unnecessary data. Simulation result confirmed that the proposed technique is better than Recursive Patching in terms of service latency and defection rate.

Keywords: Bandwidth-sharing, Dynamic expansion, Multicast, Patching, Transition stream, VOD

1. INTRODUCTION

Video on Demand (VOD) is a critical technology for the multimedia applications such as home entertainment, electronic commerce, distance learning and digital video libraries[1]. In a typical VOD system, if a remote client requests a video that is stored on a VOD server, the server delivers the video data to the client. Since the network-I/O bandwidth of the VOD server is limited, the various multicast techniques such as Batching, Piggybacking, Patching, Transition Patching and Recursive Patching have been proposed to reduce communication traffic by sharing the same video data with asynchronous clients [1-5].

Simple Patching enables a new client to share future data, if it is possible, by caching an ongoing regular stream (R-stream) in the client’s local buffer. Therefore, the server can serve the client by transmitting a new patching stream (P-stream), which is significantly shorter than the R-stream, on a new channel [1]. To reduce the amount of video data in a new P-stream for a new client, Transition Patching enables the client to share not only an ongoing R-stream but also an ongoing transition stream (T-stream) that has to patch the R-stream [4]. Recursive Patching shares ongoing T-streams recursively and achieves significant reduction of service latency compared with Transition Patching [5]. Recursive Patching and Transition Patching, however, have the problem that the T-stream should have additional data to support uncertain future requests that may arrive at the server within the transition window of the T-stream. That overhead can be offset only if such uncertain requests actually arrive at the server within the transition window of the T-stream.

This paper proposes a multicast technique called Practical Patching that can improve Recursive Patching. While Recursive Patching always initiates expanded T-streams, Practical Patching dynamically expands ongoing T-streams when the expansion is actually needed for a new request.

The rest of this paper is organized as follows. Section 2 describes simple Patching, Transition...
Patching and Recursive Patching in detail. Section 3 introduces proposed Practical Patching, and section 4 describes the performance evaluation of the proposed technique in terms of defection rate and average service latency. Finally, section 5 describes some concluding remarks.

2. SIMPLE PATCHING, TRANSITION PATCHING AND RECURSIVE PATCHING

$v[t_1, t_2]$ is used to denote the video segment consisting of the video frames for a video that are played back at a client station during the period from time $t_1$ through time $t_2$, exclusive of time $t_2$. $skew(r_1, r_2)$ is used to denote the skew that is the time interval between the initiation of the stream for request $r_1$ and the initiation of the stream for request $r_2$. $D(t)$ is used to denote the video data needed to play back a video for $t$ time units. It is assumed that the playback duration of a video is $L$ time units. Simple Patching is motivated by the fact that a new client can join the latest ongoing regular multicast transmitting an entire video stream called the R-stream. Therefore, to reduce the amount of the data in the new stream for a new client, it tries to schedule and initiate a P-stream instead of a new R-stream. The P-stream is the beginning part of an entire video data. By using the P-stream and sharing the R-stream, it can service the new client without the latency, achieve true VOD service, thus has received considerable attention [5,6].

If a new request, $r_n$, for a video arrives and is scheduled at time $t$, Simple Patching initiates an R-stream or a P-stream for it on a new channel as follows. If there is no R-stream for the same video, the server initiates a new R-stream of $v[0, L]$. However, if the latest R-stream for the same video is ongoing for request $r_1$ and the time interval between current time $t$ and the initiation time of the R-stream is less than or equal to $rw$ called the size of the regular window, the server initiates a P-stream. That is, if $skew(r_n, r_n) \leq rw$, the server initiates a P-stream of only $v[0, skew(r_n, r_n)]$, which is the video data that the client has missed from the R-stream when he is scheduled. Otherwise, if $skew(r_n, r_n) > rw$, the server initiates a new R-stream. If a new request arrives at the server and can be scheduled within $rw$ time units after the initiation of the R-stream for the same video, it is said that the request is within the regular window of the R-stream and it means that the client can be served with a P-stream.

If a new client is served with a P-stream, he caches the subsequent data from the latest R-stream while he downloads and plays back the P-stream. Therefore, it is assumed that a client station, which is served by a Patching-like multicast technique, has the download bandwidth enough to receive two streams simultaneously. After the client finishes playing back the P-stream, he plays the cached video data and keeps on caching the R-stream. Therefore, the maximum skew enabling a P-stream is the size of the client's local buffer. The detail algorithms at the server station and the client station for simple Patching were described in [1], and are omitted here.

As the skew of a new request to the latest R-stream gets bigger, the amount of the data in a P-stream gets larger. To reduce the data in the P-stream, Transition Patching [4] introduces a new stream called the T-stream. In addition to the regular window, it uses the transition window that is smaller than the regular window. The size of the transition window, $tw$, is the minimal interval between two consecutive T-streams. A T-stream is initiated to patch the latest R-stream and a P-stream is initiated to patch the latest R-stream or T-stream. Once a T-stream is initiated, it tries to initiate P-streams that have to patch the T-stream instead of the R-stream. It results in the reduction of the video data in the P-streams.

If new request $r_n$ is out of the regular window of the latest R-stream for request $r_n$, then the server initiates an R-stream, else the server
initiates a P-stream or a T-stream as follows. If \( \text{skew}(r_x, r_n) \leq tw \), a P-stream of \( v[0, \text{skew}(r_x, r_n)] \) is scheduled as in simple Patching. If \( r_n \) is the first request that becomes out of the transition window of the R-stream, or \( \text{skew}(r_x, r_n) > tw \), a T-stream of \( v[0, \text{skew}(r_x, r_n)-2tw] \) is initiated to patch the R-stream. The reason why the T-stream has the additional data for \( 2tw \) time units will be discussed in section 3. Once a T-stream is initiated, the server tries to patch the T-stream instead of the R-stream. Thus, if \( r_n \) is within the transition window of the latest T-stream, then the server initiates a P-stream of \( v[0, \text{skew to the T-stream}] \) to patch the T-stream. However, if \( r_n \) is out of the transition window of the latest T-stream, the server initiates a new T-stream of \( v[0, \text{skew to the R-stream}+2tw] \) to patch the R-stream.

Fig. 1 illustrates the difference between simple Patching and Transition Patching. Simple Patching schedules P-streams for requests \( r_2 \) and \( r_3 \), respectively. Transition Patching, however, schedules a T-stream for \( r_2 \) because there is no ongoing T-stream and \( \text{skew}(r_1, r_2) > tw \). Later, because \( r_3 \) is within the transition window of the T-stream for \( r_2 \), or \( \text{skew}(r_2, r_3) \leq tw \), \( r_3 \) is served with a P-stream of \( v[0, \text{skew}(r_2, r_3)] \). While the client of \( r_3 \) downloads and plays back \( v[0, 3] \) from the P-stream, he caches \( v[3, 6] \) from the T-stream. When the client finishes playing back the P-stream at time 18, he releases the P-stream and begins to cache the R-stream. At the same time, the client plays back the cached \( v[3, 6] \), and keeps on caching the rest of the T-stream. From time 21 through time 33, the client continues to play back and cache \( v[6, 18] \) in the T-stream while caching the R-stream. When the client finishes playing back the T-stream at time 33, he plays back and caches \( v[18, L] \) in the R-stream. In this way, the client of \( r_3 \) can play back the entire video data continuously by assembling \( v[0, 3] \) from the P-stream, \( v[3, 18] \) from the T-stream and \( v[18, L] \) from the R-stream, in turn, as in Fig. 1(c). As the number of P-streams that have to patch a T-stream increases, the total amount of the data in all streams in Transition Patching reduces compared with simple Patching.

(a) Example of simple Patching

(b) Example of Transition Patching

(c) Client station of \( r_3 \) (Transition Patching)

Fig. 1. Comparison between simple Patching and Transition Patching.

Recursive Patching enables a client to patch a series of T-streams on upper levels to reduce the video data in P-streams. In \( k \)-phase Recursive Patching (kP-RP), where \( k \geq 2 \), a client may receive an R-stream, a P-stream, or \( (k-2) \) T-streams at maximum, but he is supposed to receive at most two streams at the same time. kP-RP uses \( (k-1) \) window sizes: \( rw, tw_1, tw_2, \ldots, tw_{k-2} \), where \( rw > tw_1 > tw_2 > \ldots > tw_{k-2} \). \( rw \) is the size of the regular window, and \( tw_i \) is the size of the transition window on level \( i \). 2P-RP corresponds to simple Patching using only \( rw \). 3P-RP corresponds to Transition Patching that uses \( rw \) and \( tw_1 \) as the size of the regular window and the size of the transition window, respectively.

An R-stream, P-streams that have to patch the R-stream, and T\(_i\)-streams that have to patch the R-stream are scheduled in the same way as in Transition Patching with \( tw = tw_1 \). However, once the server initiates a T\(_i\)-stream on level \( i \), the T\(_i\)-stream, kP-RP recursively applies Transition Patching with \( rw = rw \) and \( tw = tw_{i-1} \) to the T\(_i\)-stream and the subsequent requests that arrive within the transition window of the T\(_i\)-stream. Therefore,
T<sub>i</sub>-1-streams that have to patch the T<sub>i</sub>-stream are initiated and kP-RP recursively applies Transition Patching with $tw = tw_{i-2}$ to the T<sub>i</sub>-1-stream. In this recursive way, Recursive Patching tries to share T-streams on upper levels so as to reduce the video data in the P-stream that has to patch the T-stream. If a client is served with a P-stream being to patch the latest T<sub>i</sub>-stream, he has to cache the latest T<sub>i</sub>-stream, T<sub>i-1</sub>-stream, ..., T<sub>1</sub>-stream, and R-stream, sequentially. Fig. 2 shows the types of streams initiated by 4P-RP when it is assumed that every request arrives at the server every one time unit, $rw=35$, $w_0=11$, and $w_2=3$. R, T<sub>1</sub>, T<sub>2</sub>, and P represent the R-stream, the T<sub>1</sub>-stream, the T<sub>2</sub>-stream, and the P-stream, respectively. Each of requests arriving from 1 time unit to 11 time units is served with a different P-stream that has to patch the R-stream.

![Fig. 2. Example of 4P-RP.](image)

P2Cast [7] is an architecture based on a peer-to-peer approach and uses simple Patching to transmit a video stream. Unlike simple Patching, P2Cast makes clients buffer the video data to play back a server’s role, and for each session, the server forms an application-level multicast tree over the unicast-only network. P2Cast usually outperforms multicast-based patching in terms of defection rate when clients can cache more than 10% of the beginning data of a video and the server’s workload is high, but every client station in the same session is required to have a failure-tolerant capability.

3. PROPOSED PRACTICAL PATCHING TECHNIQUE

For the future requests that may arrive within $tw_i$ time units after the initiation of a T<sub>i</sub>-stream, Recursive Patching makes the T<sub>i</sub>-stream have additional data for $2tw_i$ time units in addition to $v[0, \text{ skew of the } T_i\text{-stream to the latest } T_{i-1}\text{-stream}]$. The reason is that a small “waste” may actually lead to significant savings of bandwidth later [4]. Now, let us consider the amount of the video data in the T-streams by using Fig. 1(b). Because the P-stream for $r_5$ is scheduled later than the T-stream for $r_2$ by $\text{skew}(r_2, r_5)$ time units, the client of $r_5$ has missed the R-stream more than the client of $r_2$ by $D[\text{skew}(r_2, r_5)]$. In addition, every client station can download at most two streams simultaneously in Patching-like systems. Hence, as long as the client of $r_5$ plays back the P-stream of $v[0, \text{ skew}(r_2, r_5)]$ on a channel and caches the T-stream on another channel, he cannot cache the R-stream simultaneously. When the client releases the P-stream at time $t$, he can begin to download the R-stream. After all, he has missed the R-stream more than the client of $r_2$ by $D[2 \text{skew}(r_2, r_5)]$. If the T-stream has the missed data, the client can download the missed data from the T-stream and play back the video continuously.

All requests that can be served with a P-stream that has to patch the T-stream must be within the transition window of the T-stream. The latest request among them is one arriving at the end of the transition window of the T-stream. The skew of the latest request to the T-stream is $tw$. Hence, Transition Patching always makes the T-stream have additional data of $D[2tw]$ in advance for the worst future request that may arrive at the end of the transition window of the T-stream. However, the overhead of $D[2tw]$ can be offset if there are P-streams that have to patch the T-stream.

Recursive Patching implies the problem as follows. If there is no request that has to patch the T<sub>i</sub>-stream, the additional data will be complete waste. Although such a request, $r_n$, arrives, the data that has to be appended to the T<sub>i</sub>-stream is not $D[2tw]$ but $D[\text{skew of } r_n \text{ to the } T_i\text{-stream}]$. As long as request $r_n$ does not arrive at the end of the transition window of the T<sub>i</sub>-stream, $D[2\cdot(\text{tw skew of } r_n \text{ to the } T_i\text{-stream})]$ is always waste.
Therefore, this paper proposes an efficient bandwidth-sharing technique called Practical Patching for VOD systems.

Proposed Practical Patching is a multicast technique for VOD services and can delete the unnecessary data included in T-streams. It initially schedules a T1-stream of \( v \{0, \text{ skew of the current request to the latest } T_{t-1}-\text{stream} \} \) with no additional data for the future requests. Then, when the server initiates a new P-stream or T\(_{t-1}\)-stream to patch the T\(_t\)-stream, k-phase Practical Patching (kP-PP) dynamically expands the latest T\(_t\)-stream, T\(_{t-1}\)-stream, ..., T\(_1\)-stream, which the new client has to patch. By expanding the T-streams when the expansion is necessary for the current request, Practical Patching can make every T-stream have necessary data.

To illustrate the correctness and implementation of proposed Practical Patching, Fig. 3 uses 3P-PP assuming that the minimum interval time between two requests is one time unit, \( \tau_w = 30 \), and \( \tau_w = 4 \).

The first T\(_1\)-stream, which has to patch the latest R-stream, occurs when the skew of the current request to the R-stream becomes larger than \( \tau_w \) for the first time. It is sure that the minimum data of a T\(_1\)-stream is \( D(\tau_w + 1) \). In Fig. 3(a), such a T\(_1\)-stream corresponds to one for request \( r_2 \). The arrival time of the latest request that can be served with a P-stream that has to patch the T\(_1\)-stream is \( (\tau_w + \tau_0) \) time units, that is, time \( \tau_w \).

When the server schedules request \( r_3 \), it does not finish transmitting the T\(_1\)-stream. That is, whenever the server initiates a P-stream that has to patch a T\(_1\)-stream, it is still transmitting the T\(_1\)-stream on an assigned channel and has not released the channel yet. Therefore, the server can expand the T\(_1\)-stream as follows. When it schedules the new request, it calculates the additional data of the T\(_1\)-stream. It expands T\(_1\)-stream by modifying the end of the T\(_1\)-stream to \( \text{(current end point} + (2 \cdot \text{skew of the new request to the } T_1\text{-stream}) \) and appending the additional data to the T\(_1\)-stream. It continues to transmit the expanded T\(_1\)-stream on the current channel. The final streams that the server transmits are shown in Fig. 3(b). By expanding the T\(_1\)-stream before the transmission of the T\(_1\)-stream is completed, Practical Patching can make sure that the server can continuously transmit the expanded T\(_1\)-stream on the assigned channel. To append the additional data stored on the hard disk to the T\(_1\)-stream, the server has additional disk-I/O overhead. However, the additional data is the data already included in the T\(_1\)-stream of Recursive Patching. Moreover, if there is no new request, like request \( r_4 \), within the transition window of the T\(_1\)-stream, the expansion will never occur and Practical Patching can remove the disk-I/O for \( D(2 \cdot \tau_w) \) included in Recursive Patching. Therefore, the disk-I/O overhead for the expansion in Practical Patching can be negligible.

![Diagram](image_url)

Fig. 3. Example of 3-phase Practical Patching.
T₁-stream, which the client has to download, by
\( D[2 \cdot \text{skew of } r_5 \text{ to the } T₁\text{-stream}]) \).

Fig. 4 illustrates how to expand T-streams in
4P-PP with \( tw=9 \) and \( tw=3 \). When a P-stream of
\( v(0, I) \) is initiated for \( r_5 \) to patch the T₁-stream
in Fig. 4(b), the client of \( r_4 \) has missed the
T₁-stream more than the client of \( r_3 \) by \( D[2 \cdot \text{skew}\{r_3, r_6\}] \). Therefore, the T₂-stream has to be
expanded by \( D[2\cdot\text{skew}\{r_3, r_6\}] \) to include the
missed data. As the result of expanding the
T₂-stream, the time when the client of \( r_4 \) can cache
the R-stream is delayed by \( 2\cdot\text{skew}\{r_3, r_6\} \) time
units, so the T₁-stream also has to be expanded
by \( D[2\cdot\text{skew}\{r_3, r_6\}] \) to include the missed data
in the R-stream. Finally, the expanded T₁-stream
is \( v(0, 10+29) \) for \( r_3 \),
a T₂-stream of \( v(0, 4+23) \) for \( r_5 \) and a P-stream
of \( v(0, I) \) for \( r_6 \) are initiated. Hence, for requests
\( r_2, r_3 \) and \( r_6 \), the total amount of saved data in
4P-PP is \( D[39–27] \), and the resource saving of
4P-PP over 4P-RP is about 30.8%.

![Diagram](image)

(a) Streams when \( r_5 \) is about to be scheduled

![Diagram](image)

(b) T-streams expanded for request \( r_5 \) and \( r_6 \)

Fig. 4. Example of 4-phase Practical Patching.

As mentioned above, every T-stream in
Practical Patching initially has the data necessary
to the current request, and then the expansion of
the T-stream occurs when a new request arrives
at the server within the transition window of the
T-stream. Therefore, every T-stream always has
the video data necessary to server the real
requests, and there is no waste data in the
T-stream. At the worst case that a new request
arrives at the end of the transition window of the
T-stream, the T-stream becomes equal to the
T-stream that is expanded in advance by
Recursive Patching.

4. SIMULATION AND PERFORMANCE EVALUATION

This section presents experimental measurements
in the simulation for performance evaluation of
Practical Patching. The parameters used in the
simulation are based on those in [45], and are listed
in Table 1. It is assumed that once a client starts
watching a video, he sequentially watches it until
the end of the video. Wong [3] has shown that
the performance improvement of kP-RP over
Transition Patching is getting less when \( k \) is getting
larger and determined the sizes of a regular
window and transition windows offline by
exhaustive search. Since it is important to compare
the performance of Practical Patching with that of
kP-RP in the same condition, only one video is
used and \( k \) and \( rw \) are fixed to 4 and 16 minutes
as default values, respectively. In order to find the
optimal pair, \( (tw_1, tw_2) \), the simulation was re-
peatedly performed varying the sizes of transition
windows. As a result, it was observed that \( (8, 4) \)
and \( (4, 2) \) minutes provided their own best service
latency to 4P-RP when the mean inter-arrival
time, \( m \), was 20~90 and 5~10 seconds,
respectively. The criteria for evaluation are the average
defection rate and the average service latency
that are commonly used to evaluate the performance of
VOD services. The defection rate is the percentage
of clients who cancel their own requests because
waiting time is too long, or the value of dividing
total number of requests by the number of canceled
requests [1]. The service latency is defined as the
waiting time from the time he requests a video
through the time he can play back the video.
Table 1. Parameters used in the simulation.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Server bandwidth</td>
<td>12 channels</td>
</tr>
<tr>
<td>Length of the video</td>
<td>90 minutes (L)</td>
</tr>
<tr>
<td>Client’s local buffer size</td>
<td>16 minutes (twi)</td>
</tr>
<tr>
<td>Number of requests</td>
<td>100,000 requests</td>
</tr>
<tr>
<td>Mean request inter-arrival time</td>
<td>5 ~ 90 seconds (Poisson distribution)</td>
</tr>
<tr>
<td>Defection time</td>
<td>10 ~ 90 seconds (Random distribution)</td>
</tr>
</tbody>
</table>

Fig. 5 and Fig. 6 show simulation results about the average deflection rates and the average service latency of 4P-RP and 4P-PP varying m from 5 to 90 seconds. To show the performance improvement of 4P-PP over 4P-RP, the simulation uses (8, 4) and (4, 2) as (tw1, tw2) minutes, which are the optimal pairs in 4P-RP when m is 20 ~ 90 and 5 ~ 10 seconds, respectively.

Proposed 4P-PP improves the deflection rate by 63.6% and the service latency by 60.4% on average, respectively. The simulation result that provides the best performance in 4P-PP is also plotted. 4P-PP(Optimal) represents 4P-PP with (tw1, tw2) = (4, 1), (tw1, tw2) = (6, 1), and (tw1, tw2) = (5, 1) minutes when m is 5 ~ 40, 50, and 60 ~ 90 seconds, respectively.

From the Fig. 5 and Fig. 6, we can see that the improvement percentage of 4P-PP over 4P-RP gets smaller as m gets smaller. The reason is as follows. As m gets shorter, the number of the requests that arrive within the transition window of a T-stream gets larger, and the waste data in the T-stream in 4P-RP gets less. From the simulation result, it is sure that proposed Practical Patching serves more clients with shorter service latency than Recursive Patching even under the condition that Recursive Patching provides the best performance. Moreover, 4P-PP can more improve the performance of the 4P-RP if the optimal pair (tw1, tw2) for 4P-PP is used.

Fig. 7 shows the simulation result about the average service latency of 4P-RP and 4P-PP varying the server bandwidth. In this simulation, m is fixed to 30 seconds and (tw1, tw2) is fixed to (8, 2) minutes, which provides the best performance to 4P-RP.

Proposed 4P-PP improves the deflection rate by 63.6% and the service latency by 60.4% on average, respectively. The simulation result that provides the best performance in 4P-PP is also plotted. 4P-PP(Optimal) represents 4P-PP with (tw1, tw2) = (4, 1), (tw1, tw2) = (6, 1), and (tw1, tw2) = (5, 1) minutes when m is 5 ~ 40, 50, and 60 ~ 90 seconds, respectively.

From the Fig. 5 and Fig. 6, we can see that the improvement percentage of 4P-PP over 4P-RP gets smaller as m gets smaller. The reason is as follows. As m gets shorter, the number of the requests that arrive within the transition window of a T-stream gets larger, and the waste data in the T-stream in 4P-RP gets less. From the simulation result, it is sure that proposed Practical Patching serves more clients with shorter service latency than Recursive Patching even under the condition that Recursive Patching provides the best performance. Moreover, 4P-PP can more improve the performance of the 4P-RP if the optimal pair (tw1, tw2) for 4P-PP is used.

Fig. 7 shows the simulation result about the average service latency of 4P-RP and 4P-PP varying the server bandwidth. In this simulation, m is fixed to 30 seconds and (tw1, tw2) is fixed to (8, 2) minutes, which provides the best performance to 4P-RP.

Fig. 8 shows the average service latency of 4P-RP and 4P-PP varying the sizes of transition windows when m and the server bandwidth are fixed to 30 seconds and 12 channels, respectively. Fig. 7 and Fig. 8 show that 4P-PP is always better than 4P-RP as expected.
5. CONCLUSION

This paper proposed Practical Patching that is a multicast technique to share a VOD server's limited network bandwidth efficiently. While Recursive Patching always makes T-streams have the additional video data for the worst request, Practical Patching dynamically expands the T-streams whenever such expansion is needed to serve a new request. The key of Practical Patching is that when the server is about to schedule a new request, the server is transmitting the T-streams, which the client of the request has to patch, on the already allocated channels. Therefore, the server does not have to worry about the allocation of a new channel to deliver the additional data as soon as it completes the transmission of the current T-stream. The server can expand the T-stream by modifying the end of the T-stream, appending the new additional data to the T-stream, and continuing to transmit the expanded T-stream on the current channel. Simulation results confirmed that Practical Patching could improve Recursive Patching in terms of deflection rate and service latency in VOD systems. The future research will be to develop a formula to help determine the optimal sizes of the regular window and transition windows for Practical Patching.

6. REFERENCES


Sook-Jeong Ha
She received the B.S. degree in computer science from Keimyung University, in 1988, the M.S. degree in computer science from Chungang University, in 1990, and the Ph.D. degree in computer science from Catholic University of Daegu, in 1998. Since 2001, she has been with School of Electrical Engineering and Computer Science, Kyungpook National University, as a visiting professor. Her research interests include mutual exclusion, wireless networks, and multimedia systems.