Design and Implementation of a Real-time Video Player on Tiled-Display System

Giseok Choe, Jeongsoo Yu, Jeonghoon Choi, Jongho Nang
Dept of Computer Science and Engineering,
Sogang University, Seoul, Korea
{brix, yjs, drumist, jhnang}@sogang.ac.kr

Abstract

This paper presents a design and implementation of realtime video player that operates on a tiled-display system consisting of multiple PCs to provide a very large and high resolution display. In the proposed system, the master process transmits a compressed video stream to multiple PCs using UDP multicast. All slaves(PC) receive the same video stream, decompress, clip their designated areas from the decompressed video frame, and display it to their displays while being synchronized with each other. A simple synchronization mechanism based on the H/W clock of each slave is proposed to avoid the skew between the tiles of the display, and a flow-control mechanism based on the bit-rate of the video stream and a pre-buffering scheme are proposed to prevent the jitter. The proposed system is implemented with Microsoft DirectX *filter technology in order to decouple the video/audio codec from the player. Experimental results on a tiled display system with 28 PCs show that the proposed system could display a video stream in a realtime without jitter and skew.*

1 Introduction

A tiled-display system consists of multiple display devices linked in a grid configuration to provide a very large display with a very high resolution, and is typically used in common cooperative environments of virtual spaces that feature the VR technology. In order to display a large graphic object or a video stream on this display system, the display on these tiles should be carefully controlled. There have been a lot of researches[1, 2, 3, 4] to devise a synchronization mechanism to control this display for playing the sequence of video frames. However, these previous researches assumed an uncompressed video stream, or used a complex TCP-based synchronization algorithm[1, 2] to temporally control the display of each video frame.

This paper proposes a realtime video player that can play a compressed video stream on the tiled display system without jitter and skew. There are two problems to be resolved to build such a video player system. One is how to divide and deliver the video stream to each PC so that it could display the designated part of video frame image, and the other is how to temporally synchronize these displaying operations of all PCs so that the whole video frame image is displayed on the tiled display temporally together. This paper proposes an algorithm in which the master process broadcasts the video stream in a compressed form, and the slaves (or players) decode the video stream , clip their own areas, and play back on their display tiles. This approach could reduce the loads for the network as well as the master process. Moreover, there is an advantage in that since identical stream is delivered to all players, UDP multicast can be used for efficient broadcasting. As for the synchonizing of display operations, we propose a simple synchronization scheme in which all players only reference their H/W clocks after the start time of the each player is synchronized. It is proved experimentally that this simple synchronization scheme is enough for realtime video play application on tiled display system and inherently scalable. This paper also proposes a control flow and buffering mechanism based on the bit-rate of video stream that helps to avoid the jitter of the realtime video playing. The proposed system is implemented with *Microsoft DirectX* filter technology [5] to prevent the codec-dependent problems frequently occurred in building an application based on the video player. Experimental results on the tiled display system consisting of 28 PCs show that the proposed schemes could display the video stream on tiled display system without the jitter and skew between tiles. Furthermore, this system operates regardless of the codec used in the video stream because all *DirectX* video/audio decoding filters could be embedded easily into the system.

2 Related Works

There are two technical aspects required for implementing video player on the tiled-display. One is the technology to deliver the video stream to multiple players in realtime, and the other is to synchronize the play back of the video frame that are partitioned and spread on the multi-

0-7695-2983-6/07 $25.00 © 2007 IEEE
DOI 10.1109/CIT.2007.36

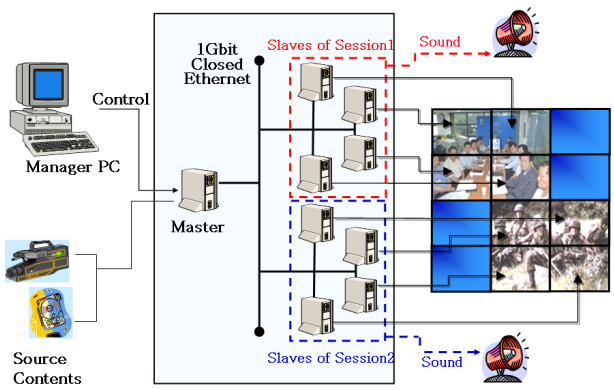


Figure 1. Overall System Architecture

ple PCs. There are some schemes[1, 2, 3, 4] developed and used in the previous video player systems on tiled display. Most systems have separate communication networks for broadcasting video or graphic objects and for sending the synchronization data. Moreover, realtime broadcasting over Gigabit Ethernet is generally implemented using the broadcast feature of the UDP protocol. On the other hand, synchronization is achieved using the reliable TCP protocol, and the corresponding packets are assigned with a high priority and regularly sent over the network for reduced delay and increased reliability. However, synchronization based on the regularly transmitted data can potentially result in errors caused by transmission delays among the PCs, and there is the problem of overheads associated with the delivery of the synchronization packets for each video frame. The approach proposed in this study does not require any synchronization message after the players are initialized, but could synchronize the playing of video frame precisely.

3 Design of Realtime Video Player

The realtime video player proposed in this paper, as shown in Figure 1, consists of a manager PC that initiates and manages the video playing session on multiple tiles, the master that multicasts the video stream to the slaves that participate in that session, and a set of slaves that decodes the received video stream and displays the part of video frame to tiles (or display) in a synchronized fashion. Note that there could be multiple sessions on the tiled display systems, but we assume that the slave is restricted to participate in only one session.

3.1 Partitioning and Delivery of Video Frame

There are three possible ways to partition and play a single video source on tiled-display. First is the master decompresses the video stream and gets a video frame in raw, partitions the video frame according to each area of the tiled-display, and transmits the partitioned video frames in raw format to each slave (or player) separately. This approach does not require the player to decode the video data, and there is an advantage of using a relatively simple player. However, delivery of uncompressed video requires a high network load. Second is the same as the first one but the master re-compresses the partitioned video frame before transmission in order to save the network bandwidth. In this case, compressing the partitioned images increases the system load, making it inappropriate for realtime transmission. More importantly, since these two approaches require that the master should send the partitioned video frames (in a raw or compressed form) to multiple slaves separately (via multiple unicastings), they are not scalable to the number of slaves participating the session. Third, proposed in this paper, is that the master multicasts the compressed video data and sends the image coordinate information to the slave regarding the areas to be displayed. The slave then decompresses the video frame, clips the decoded frame according its assigned the coordinate, and finally displays it to its tile. Although this approach requires that all slaves decode the same video frame independently so that the CPU resources of the slaves are wasted on the whole, it helps to reduce the CPU resources of the master because the video stream is delivered in the original compressed form, and to save the network bandwidths because the video stream could be multicasted. This approach also have an advantage that it can be applied regardless of the number of slaves participating the sessions.

3.2 Synchronization Mechanism

One of the ways of achieving synchronization for playing a video stream on multiple tiles is that the master regularly multicasts a kind of "synchronization" message to all slaves as used in [1, 2, 3, 4]. However, such an approach creates a lot of overheads for sending the synchronization messages and there is the difficulty of realizing accurate synchronization due to the difference in delay of the synchronization data transmission and processing. Furthermore, there should be a mechanism that the master knows that all slaves are ready to display the next frame that requires a lot of time and network resources.

In the proposed synchronization scheme, the slave references the stream time that is incremented by its own H/W clock periodically after initialized by the master, when displaying the decoded video frame to its tile. Let us explain this mechanism in more detail. Usually, some kinds of time stamps are attached to each presentation unit of multimedia data (for example, video frame and audio sample) when it is recorded. The presentation unit is displayed at the player when its time stamp is matched with the media stream time that is set to zero when the playing is started and incre-

mented periodically. If the media stream times of the slaves are started at the same time and they are incremented at the same rate, the display of partitioned video frames could be synchronized temporally because all slaves received the same video stream from the master.

Figure 2 shows the proposed synchronization algorithm that is used to display the video stream on tiled display synchronously. Let T^M_{base} and $T^{S_i}_{base}$ be the value of system clock (or H/W clock) of the master and i-th slave, respectively. When the session is initialized, they are first synchronized using Cristians algorithm[9]. After T^M_{base} and $T^{S_i}_{base}$ are synchronized, the master broadcasts the start time of the media stream time, T^M_{start}, to all slaves. The start time is determined by considering the transmission delay (d_i) and initial buffering time ($InitialBufferingTime : IBT$). Then, the master starts to multicast the video stream in compressed form to all slaves, and the slaves receive the video stream and start to decode and display it. Note that these two tasks are executed concurrently in order to display while receiving. The i-th slave display the video frame to its tile only when the current stream time ($T^{S_i}_{stream}$) reaches to the time stamp of j-th video frame, T^{frame}_j. Note that since the stream time of the slave is independently and periodically incremented by its own hardware clock (RTC - Real Time Clock), $T^{S_i}_{stream} \leftarrow$ ReadRTC() $-T^{S_i}_{start}$, and differences in the period of RTC between slaves are negligible, the stream times of all slaves could be the same all the time. It implies that the video frames could be displayed at the multiple tiles synchronously as shown in our experiments. This synchronization algorithm is simple and scalable because it does not require repeated synchronization messages for each video frame and the synchronization overhead is not a function of the number slaves participating the session.

3.3 Control Flow Mechanism

The media data transmission method could be roughly classified into "the burst mode" in which the transmission rate is irregular in time, whereas "the constant mode" in which the transmission rate is regular in time. Since "the constant mode" requires a lot of CPU resources in order to continuously transmit the media data, "the burst mode" transmission is deployed in the proposed system. That is the master periodically multicasts video data to slaves, and the amount of data to transmit at each period is only for one media sample (i.e., video frame). Since the video stream is usually VBR-coded, the amount of video data to be transmitted is different at each period. However, it could cause a buffer overflow at the slave. This problem is resolved in the proposed system by adjusting the size of socket buffer to be larger than the maximum data size of video frame in a compressed form. Unfortunately, this maximum data size of video frame is unknown until all video frames are trans-

```
//T^M_base and T^{S_i}_base: the value of system(or H/W) clock of the master and
i-th slave, respectively.
//k is the number of slaves in the session
//Seg_m is the presentation unit in a compressed form
//d_i: the transmission delay.
//M, S_i: the master and i-th slave, respectively.
procedure VIDEOPLAYERMASTER
    //Synchronize T^M_base and T^{S_i}_base using Cristian's algorithm[9]
    for ∀ slaves, S_i(1 ≤ i ≤ k), participating the session do
        T^M_base ← ReadRTC(); //Read the system clock
        SendMsg(S_i, T^M_base);
        t_tmp ← RecvMsg(S_i);
        T^M_current ← ReadRTC();
        //Compute the transmission delay
        d_i ← (T^M_current − T^M_base − t_tmp)/2;
        SendMsg(S_i, d_i);
    end for
    //Send the start time to all slaves
    T^M_current ← ReadRTC();
    T^M_start = T^M_current + avg(d_i) + IBT;
    Multicast(S_i(∀i, 1 ≤ i ≤ k), T^M_start);
    //Multicast the segment of video stream, Seg_m, to all slaves
    m ← 0;
    while not EOF(Seg_m) do
        ReadSeg(Seg_m);
        Multicast(S_i(∀i, 1 ≤ i ≤ k), Seg_m);
        m ← m+1;
    end while
end procedure

procedure VIDEOPLAYERSLAVE(S_i)
    // Synchronize T^M_base and T^{S_i}_base using Cristian's algorithm[9]
    T^M_base ← RecvMsg(M);
    T^{S_i}_1 ← ReadRTC(); T^{S_i}_2 ←ReadRTC();
    t_tmp ← T^{S_i}_2 − T^{S_i}_1;
    SendMsg(M, T_tmp);
    d_i ← RecvMsg(M);
    T^{S_i}_base ← T^M_base + d_i;
    T^M_start ← RecvMsg(M); //Receive the start time from the master
    j ← 0;
    parbegin
        //Do the tasks to receive the video segment and to
        decode/display concurrently
        do//Receive the video segment and insert to StreamingBuffer[]
            m ← 0;
            while not EOF(Seg_m) do
                Seg_m ← RecvMsg(M);
                WritetoBuffer(Seg_m); m ← m + 1;
            end while
        end do
        do//Get the video segment from buffer and decode/display
            T^{S_i}_current ← T^{S_i}_base+(ReadRTC() - T^{S_i}_1);
            Wait(T^M_start − T^{S_i}_current);
            T^{S_i}_start ← ReadRTC(); T^{S_i}_stream ← 0;
            j ← 0; m ← 0;
            while not EOF(Seg_m) do
                Seg_m ← ReadfromBuffer();
                F_j ← Decode(Seg_m);
                T^{frame}_i ← GetTimeStamp(F_j);
                T^{S_i}_stream ← ReadRTC() - T^{S_i}_Start;
                Wait(T^{frame}_j − T^{S_i}_stream); DisplayFrame(F_j);
                m ← m + 1; j ← j + 1;
            end while
        end do
    parend
end procedure
```

Figure 2. Delivery and Synchronization Algorithms of Master and Slave

mitted. In the proposed scheme, it is approximated with the following equation, where MAX^{bps} is the maximum bit rate of video stream that could be roughly estimated, and fps is the frames per second that is a constant in the video stream;

$$Maximum\_Media\_Sample\_Size = \frac{MAX^{bps}}{fps} \quad (1)$$

In the case of burst mode transmission, since there could be a transmission collision because a lot of data are transmitted at a burst and there could be multiple sessions being serviced in the network, it could lead the jitters in video playing. In order to avoid this jitter, another buffer is employed in the slave.The slaves initially buffer some video data for a pre-defined time interval (IBT) before starting the video playing. The amount of data to be buffered initially is computed as follows, where Avg^{bps} is the average bits-per-second of the video stream to transmit.

$$Amount\_of\_Initial\_Buffering = IBT \times Avg^{bps} \quad (2)$$

Although this initial buffering causes some latency in starting the video playing, it greatly helps to avoid the jitter as shown in our experiments. In the case of physically closed network as in our tiled-display environment where all PC are connected to the same physical network, the initial buffering time less than one second ($IBT < 1sec$) is enough to avoid the jitter as shown in our experiments.

4 Implementation with Microsoft DirectX Technology

When implementing video-related applications, the codec (encoder/decoder) used for compressing/decompressing the video stream should be available in an API level to control the open/play/pause/stop/record operations for video recording/playing. Furthermore, since there are a lot of codecs that are widely used in various video applications, the video player should embed the various multiple video/audio codecs to develop a practical application. Unfortunately, it is usually impossible in real environments. In order to avoid this problem, we have implemented the proposed realtime video player with *Microsoft DirectX* technology[5] in which various audio/video codecs as well as audio/video renders are available as independent engines called filters. By connecting various kinds of these filters, we could decouple the codec-related problems from the application development so that a lot of video-related applications could be developed easily.

Figure 3 shows the internal structure of the realtime video player on tiled display system, in which the filters that are developed in this research are filled with the gray color. The master process consists of the source filter and multicast filter. The source filter reads the video source and demultiplexes into audio and video streams. The multicast filter makes a packet for each presentation unit (video frame or a set of audio samples) with the header required for UDP multicasting, and multicasts it to all slaves that participate the session. Since the time stamps of the presentation units could be obtained from the *DirectX* source filters, they are packetized together in order to control the displays on slaves. Since the multicast filter is implemented by inheriting the "Base Render filter" of *DicrectShow*, it is activated only when the stream time reaches to the time that the time stamp of presentation unit indicates. It means that the multicast filter could transmit the video stream at a rate identical to the bit-rate of video (or audio) stream.

The player process at slave consists of the receiver filter, audio/video decoder filters, overlay filter, and renderer filter. The receiver filter reads the data over the UDP socket interface, performs de-packetization as well as buffering as shown at the 1st part of **parbegin** of VideoPlayerSlave() in Figure 2. The buffered audio and video data are forwarded to audio and video decoder filters, respectively, in order to generate audio/video data in raw format. The overlay filter clips the video frame, stretches it if required, and transforms it for edge-blending. The resulting image is delivered to renderer filter to be displayed on the tile. The render filter display the decompressed-, clipped-, stretched-video frame into its tile while synchronizing with other slaves using the algorithm presented at the 2nd part of **parbegin** of VideoPlayerSlave() in Figure 2.

Note that since the filters presented in Figure 3 run concurrently, a realtime streaming video playing could be provided if the audio/video decoding filters can decode the audio/video streams in realtime. Note also that, by substituting the video decoder and audio decoder shown in Figure 3, a lot of video streams compressed with various formats (for example, MPEG-1, MPEG-2, MPEG-4, WMV, ASF, AVI,) could be played on the tiled display system.

5 Experiments and Analyses

We have implemented the proposed realtime video player on tiled display system consisting of 28 PCs (Intel Dual Core 3.4GHz CPU, 1GB DDR2 memory,Gigabit LAN card, connected with 3COM 1GBit hub. The video samples with various resolutions(640 $\times$ 352, 1280 $\times$ 720, etc) and codecs(MPEG-1, MPEG-2, Xvid MPEG-4, WMV) are used to experiment the synchronization and flow control mechanisms of the realtime video player. Figure 4 shows the sample snapshots of the realtime video player on tilted display with multiple sessions.[1]

[1] A demo video could be found at http://mmlab.sogang.ac.kr/tdisplay/demo_high.wmv

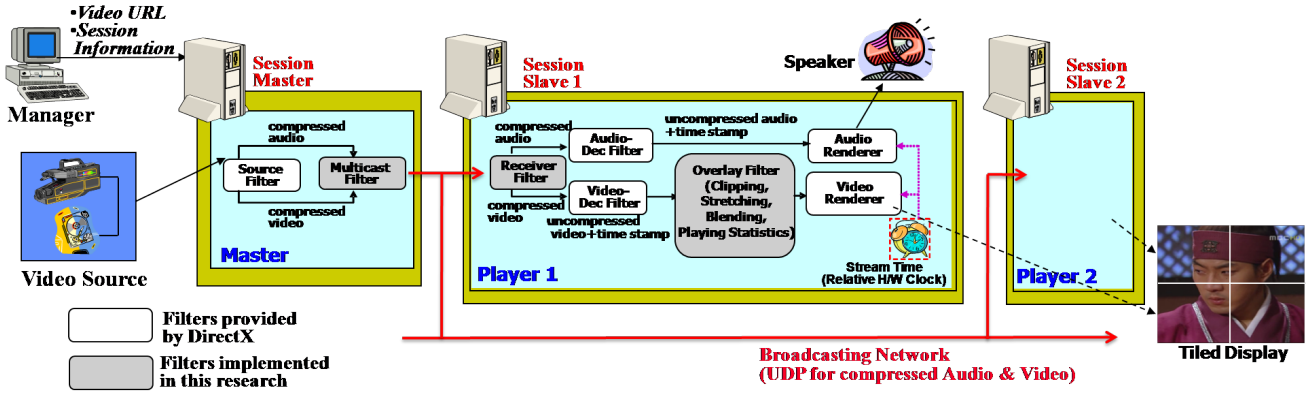


Figure 3. Internal Structures of Master and Slave Processes

Figure 4. Snapshot of the Video Player on Tiled Display System with 28 PCs

5.1 Experiments on Synchronization Mechanism

As explained in Section 3.2, the video player at slave S_i displays the uncompressed/clipped j-th video frame (F_j) only when it stream time($T^{S_i}_{stream}$) reaches to the time stamp of j-th frame (T^{frame}_j). In order to show that the stream times of all slaves are incremented synchronously and eventually all tiles are display synchronously, the stream times and sync offset (the difference between the time in time stamp and actual display time) of all players are measured and shown in Figure 5(a) and Figure 5(b), respectively. As shown in these experiments, the stream times are incremented synchronously and the sync offset are always zero. From these experiments, we can argue that the proposed synchronization mechanism works precisely. Actually, since the stream times are incremented by H/W clocks of PCs, it could be always the same although they are incremented independently.

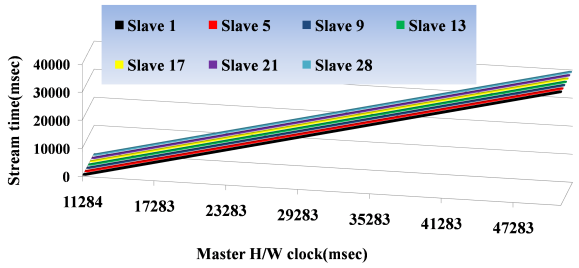
5.2 Experiments on Control Flow Mechanism

In order to avoid the jitter, the video stream should be buffered before the actual playing starts. If a lot of video streams are pre-buffered, the jitter could be avoided but the actual playing should be delayed for a long time. The problem of how many data streams should be buffered is generally the main design issue when implementing the video player operating in streaming mode. In the proposed scheme, the amount of video data to be pre-buffered is controlled by the parameter "IBT" as shown in Eq.2. As shown in Figure 6(a), if IBT is set to zero second (*i.e.*, no prebuffering), some jitters[2] are occurred when playing the video. However, if it is greater than 1 second, the video stream could be played stably without any jitter. Figure 6(b) show the jitter of the video playing when there are 14 sessions being serviced concurrently. It shows that there is also no jitter after it is initialized. Of course, the initial buffering time required to avoid the jitter depends on the network status and characteristics of VBR coding so that it is very hard to predict accurately. However, from our extensive experiments, we find out that pre-buffering of video stream for playing one second is enough to avoid the jitter in an environment that the tiled display system operates (*i.e.*, directly-connected closed network environment).

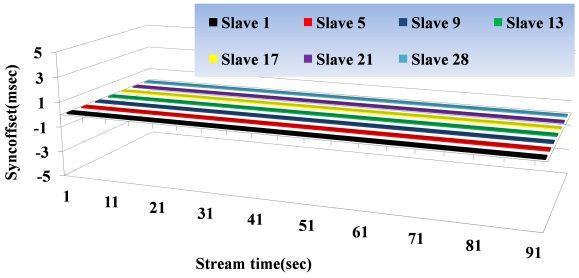
6 Concluding Remarks

The tiled display system was developed to provide a large and high resolution display. It is usually implemented with multiple PCs connected in a grid configuration. In order to display a video stream on this tiled display system, the problems of delivery of video stream to multiple PCs, synchronization of display tiles, and flow control should be resolved. In this paper, we proposed a delivery mechanism that the master multicasts the compressed video stream to multiple slaves and let the slave decodes and displays the part of video frame on their tiles independently, a synchronization mechanism that the slave references its own stream

[2]DirectX provide an interface to measure the jitter at time t_n that is the average jitter from the start time (t_0) to time t_n. That is the reason why there is a jitter at time t_n in graph although there is no jitter at time t_n

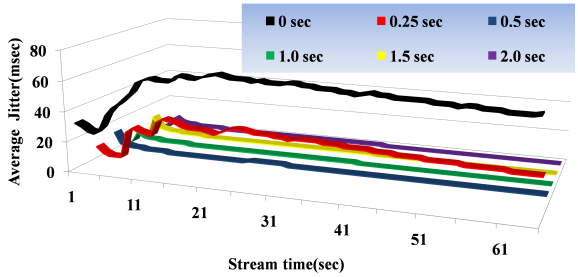


(a) Stream Times of all Slaves

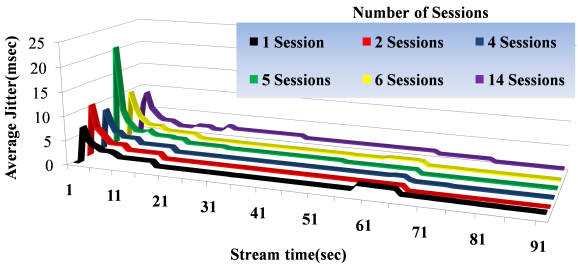


(b) Sync Offsets of all Slaves

Figure 5. Experiments on Synchronization Mechanism



(a) Jitters vs IBT



(b) Jitters when playing multiple sessions ($IBT = 1sec$)

Figure 6. Experiments on Control Flow Mechanism

time incremented by H/W clock after initialized, a mechanism that helps to avoid the jitters when playing the video in streaming mode. The proposed algorithms are implemented with *DirectX* technology, and experimented with a tiled display system with 28 PCs. The experimental results show that the proposed delivery, synchronization, and control flow mechanisms works precisely regardless of the video codec, the bit-rate of video streams, the number of slaves, and finally the number of sessions being serviced. The proposed system could be used to build a cost-effective tiled display system without any extra hardware for synchronization and control flow. The problem of how to move/resize the display area of video frame while playing is remained as a future work.

References

[1] University Illinois, *Scalable Adptive Graphics Environment* , http://www.evl.uic.edu/cavern/sage/ description.php, 2007.

[2] R. Singh, B. Jeong, L. Renambot, A. Johnson and J. Leigh, "TeraVision: a Distributed, Scalable, High Resolution Graphics Streaming System," in *Proceedings of the 2004 IEEE International Conference on Cluster Computing* , pp. 391-400, 2004.

[3] Y. Zhao and X. Zhang, "Design a secure and scalabel CVE: The Access Grid," in *Proceedings of the 2001 ACM/IEEE Conference on Supercomputing* , 59-59, 2001.

[4] H. Chen, D. Clark, Z. Liu, G. Wallace and K. Che, "Software Environments for Cluster-Based Display Systems," in *Proceedings of the 1st International Symposium on Cluster Computing and the Grid* , pp. 202-210, 2001.

[5] Microsoft MSDN, *DirectShow* , http://msdn2. microsoft.com/en-us/library/ms783323.aspx, 2007.

[6] E. Magana, J. Aracil and J. Villadangos, "Packet Video Broadcasting with General-Purpose Operating Systems in an Ethernet," in *proceedings of Multimedia Tools and Applications* , Vol.24, No.1, pp.5-28. Sep.2004.

[7] Ralf Steinmetz and Klara Nahrstedt, *Multimedia: Computing, Communications and Applications* , Prentice Hall, 1995.

[8] ETSI, *TS 126 234 v6.4.0 Release 6 Protocol and codecs* , 2005.

[9] A. Tanenbaum and M. Steen, *Distributed Systems Principles and Paradigms* , Prentice Hall, 2002.