# Multi-source Media Streaming for the Contents Distribution in a P2P Network

Sung Yong Lee, Jae Gil Lee, and Chang Yeol Choi

Kangwon National University
Department of Computer, Information and Telecomm. Engineering
192-1 Hyoja2-dong, Chuncheon 200-701, Republic of Korea
moota4@mmslab.kangwon.ac.kr

**Abstract.** For a P2P network, the contents distribution is a very important service because the contents provider is not fixed. And media streaming and file saving operations should be simultaneously carried out in order to distribute contents through the P2P network. In this paper, a P2P multi-source media streaming system which can replay the contents data during downloading is proposed and implemented. The system reduces the user response time and the number of simultaneous user increases more than two times.

## 1 Introduction

As P2P(Peer-to-Peer) based contents sharing programs are actively used on the internet, interest on the P2P network is being much more increased recently. And each peer has a duty to perform roles as a server as well as a client, therefore each peer takes the contents from the other peer and gives the contents to the other peer. At that time if the media data is received by only a specific peer user QoS is not satisfactory and a one-to-one P2P paradigm can't guarantee data rate and reliability needed by user because of frequent network escape of peer. To solve these problems multi-source downloads or multi-source streaming to receive data from several source peers are proposed [1] [2]. eDonkey [5] downloads data from several source peers, makes a temporary file, and enhances the speed by merge of temporary files into a file after receiving all data. GNUSTREAM [3] receives different data segments from each of source peers, then assembles and replays it. Like this, eDonkey and GNUSTREAM enhance transmission rate and mitigate the damage of the source peers escape by receiving data from multi-sources. But in the case of eDonkey, replay should be performed after downloading all the files, therefore response time becomes long. On the other hand, GNUSTREAM can't distribute contents on the P2P network because it is hard to share contents with other peers as it consumes media data after replay like a server-client streaming.

On the P2P network the contents distribution is a very important service because the contents provider is not fixed unlike CDN(Contents Delivery Network). And in the P2P media streaming, a request peer replays and saves data simultaneously, and after streaming it acts as a new source peer providing files to other peers. Therefore media streaming and file saving operations should be simultaneously carried out in order to distribute contents through the P2P network. If the number of peers which possess a specific file is small, the number of peers that participate in multi-source streaming is also small, so the number of peers that receive and provide specific data may be limited. As the existing contents sharing models can replay after downloading the data, the response time becomes long and the contents distribution is difficult.

In this paper, a P2P multi-source media streaming system which can replay contents during downloading is proposed and implemented. In the proposed system, as the request peer receive, replay and save the data files, it can play a role as a source peer as well as streaming. As a result, user response time can be reduced and fast contents distribution through network is possible by a mechanism transmitting only a part of media files.

## 2 Design and Implementation

### 2.1 Basic Operations

Every peer performs transmitting and receiving data simultaneously. The receiving peer requests a data block that divided media files in small size to a source peer, receives the other parts of media files from other peers in parallel, and replays it. It is possible to transmit the media when source peers have all parts of media file as well as when they have a part of a media file.

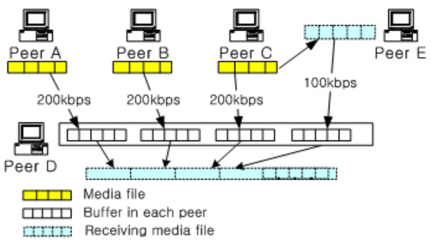


**Fig. 1.** Media data flow among peers

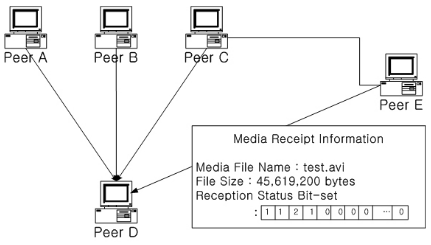

Peer A
Peer B
Peer C
Peer E
Peer D
Media Receipt Information
Media File Name : test.avi
File Size : 45,619,200 bytes
Reception Status Bit-set
: 1 1 2 1 0 0 0 0 ··· 0

**Fig. 2.** Exchanges for the media receiving states information

In the Fig. 1 Peer D is a receiving peer and plays as a client, Peer A, B, C are source peers and transmit a part of media file assigned to Peer D. Peer E receives data from peer C, stores and transmits the data to Peer D. Peer D writes data received from Peer A, B, C in the buffer block for each peer, stores it as a file, and replays the file. Peer E performs transmission and receipt simultaneously, and Peer D receives media-receipt-information from Peer E as shown in Fig. 2. And it confirms the states of media store, and by comparing it with media state information bit set, Peer D determines whether to request the next media or not.

## 2.2 Peer to Peer Communication

- Media information request : The requested file name is transmitted to the selected source peers.

```
j = 1 to N(Selected Source Peers's Number)
  if (Connection State == TRUE) then
    Message = PacketSize + Type(File_Info_Req) + FileName
Send message to selected peers that are connected
```

- Media information response : If there is an intact file, whether transmission is possible or not and file information are transmitted. For the receiving file, the file information and receipt state bit set(BitStream_Index) are together transmitted.

```
if (State == SEND_OPEN Number)
  if (State of Media file == Entire File) then
    Message = PacketSize+Type(File_Info_Ack)+State of File(Entire File)
            +File Name+FileSize
  else if (State of Media file != Entire File) then
    Message = PacketSize+Type(File_Info_Ack)+State of File(Not entire File)
            +File Name+FileSize+BitStreamIndex
Send message to request peer
```

- Media data request : A part of the media file to be transmitted is assigned to each peer and transmitted with the file name. After checking the receipt state bit set during the media receiving, it assigns and transmits the index of media file whose bit is ‘0’ to the source peer.

```
case state bit is 0: not yet requested
case state bit is 1: corresponding part has been completely received
case state bit is 2: corresponding part is being now received
for j=1 to N(Selected Source Peers's Num.)
  Message = PacketSize+Type(File_Stream_Req)+FileName(Request FileName)
          +Start Pointer+End Pointer
Request again after checking my Bit_Stream_Index
```

- Media data transmission : After the separating of the Start Index and the End Index from the media stream request packet, the assigned data is copied from a memory map file. The Start Index and the End Index are kept in the source peer until the transmission of the assigned parts completes. Here content size means the size of media data transmitted in a packet.

```
Message = PacketSize+Type(FILE_STREAM_ACK)+Send Point+Offset
        +Content Size+Media Data
Send message to request peer
```

- Media receipt state information update request : The request peer compares its own media receipt state bit set with the source peers, and it examines the block to request to the source peer. If there is no more block to request, it requests the media receipt state information update.

```
Message = PacketSize + Type(Update_bitStream_index_req)+ FileName
Send message to request peer
```

– Media receipt state information update transmission : If there is a request of the media receipt state information update, it converts the current media receipt state bit set into '0's and '1's and transmits it to the request peer.

```
Message = PacketSize+Type(Update_bitStream_index_Ack)+FileName+Bit_Stream_index
Send message to request peer
```

## 2.3 Client Module

The proposed system was implemented with Visual C++ and C on Windows XP. All the peers have the server module and the client module, the server module is a routine to perform commands requested by other peers and the client module performs the commands of source peers.

**Selection of the source peers.** To select the source peer it is examined whether the file can be transmitted or not immediately and whether the number of source peers exceeds the number of receipt peers connected by source peers in maximum or not. The network bandwidth and proximity with receipt peer are also examined. Available network bandwidth can be calculated with a TCP-friendly rate control algorithm(TFRC) [3].

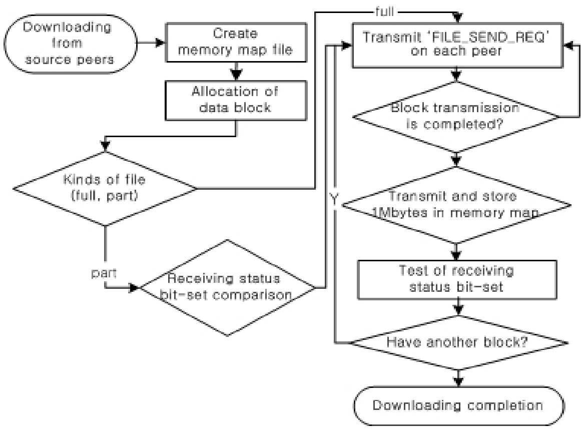

Downloading from source peers
Create memory map file
Allocation of data block
Kinds of file (full, part)
full
part
Receiving status bit-set comparison
Transmit 'FILE_SEND_REQ' on each peer
Block transmission is completed?
Transmit and store 1Mbytes in memory map
Test of receiving status bit-set
Have another block?
Y
Downloading completion

**Fig. 3.** Assignment of data block

**Assignment of data block for the transmission.** Figure 3 shows the procedure for assigning a data block to each source peer. First all, constants block size (1Mbytes) is assigned to each source peer. For source peers whose transmission finished early, media receipt state bit set is examined, and the earliest block among ‘0’ areas is assigned. The next data block is assigned to the fastest source peer.

The assignment of data blocks can be categorized into two fields as the file types. For the *mpg* file the initial part is assigned by 1Mbytes one after another. On the other hand in the case of the *avi* file which obeys the RIFF file standard, video/audio part is assigned by 1Mbytes one after another after receiving the header and *avi* index. An analysis [4] of file types requested by users in Gnutella said that more than 65% of the users have requested multimedia contents and among the contents requested *avi* files was the most frequent as 18.72%. Our system can also support *asf* type files as well as *avi*, *mp3*, and *mpg* types which are the most frequently requested.

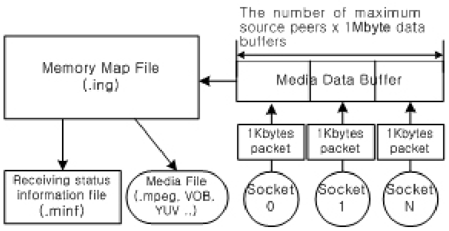

The number of maximum source peers x 1Mbyte data buffers
Memory Map File (.ing)
Media Data Buffer
1Kbytes packet
1Kbytes packet
1Kbytes packet
Receiving status information file (.minf)
Media File (.mpeg, VOB, YUV ..)
Socket 0
Socket 1
Socket N

**Fig. 4.** Media data merge

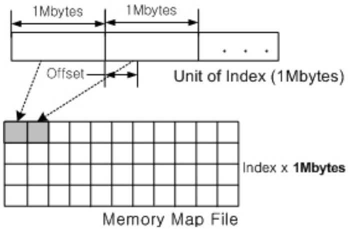


**Fig. 5.** Data transmission between the buffer and the memory map file

**Data merge and storing the file.** Every peer has a data buffer and a memory map file as shown in Fig. 4. The data merge and the file store are processes of assembling the media data from the source peers one after another, and the data transmitted by client socket is written in the data buffer block assigned to each source peer. When the data buffer is full, the buffer data is written in the corresponding location of memory map file. After a while the data is replayed by opening the memory map file. The data between the data buffer and memory map file is moved as shown in Fig. 5.

## 2.4 Server Module

The server module is composed of the file information transmission, peer connection, the data block assignment, and data transmission. Transmission of the file information is a process to know the identity of the file, whether the transmission is possible or not and the peer state after the request peer searches media, which can be divided into the information transmission of the downloaded file that is an intact and the information transmission of the downloading file itself. In the peer connection step, it should not exceed the maximum number of transmission peers. In the data transmission step, the transmission rate is limited by delay time between packets in order to protect the resources of the source peer. After calculating the file seek pointer in 1Mbytes data block index the corresponding 1Mbytes file is transmitted in 1Kbytes packets after delay time.

## 3 Experiments and the Results

A simple P2P network for experiments was organized as shown in Fig. 6. Network A and B are connected as to T3 class and the inside of network A and B is connected to 100Mbps Ethernet. The peers of each network are composed of Pentium IV, with more than 256MBytes DRAM and Windows XP.

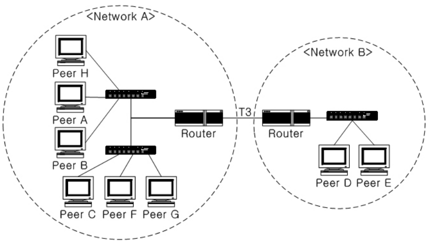


**Fig. 6.** Configuration of the testbed system

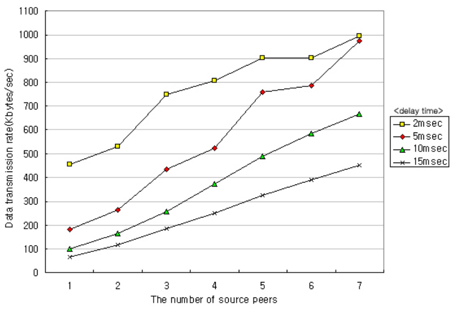


**Fig. 7.** Changes of the data transmission rate

### 3.1 Data Transmission Rates

The data transmission rates are measured as a time that receipt peer writes 30MBytes data in the file, increasing the number of source peers from 1 to 7, changing the delay time between data packets to 2, 5, 10, 15msec. As shown in Fig. 7, when the delay time is 2 msec, if increases the number of source peer from 1 to 7, the data transmission rate increased by 683%.

### 3.2 Media Streaming

The minimum number of the source peers and the delay time between packets of the source peer which enable files encoded with MPEG–1, MPEG–2, DivX to download and replay simultaneously are found. The relations between time variations needed to download all files and the number of source peers are observed. The results are shown in Table 1.

### 3.3 The Simultaneous Download and Upload

Fig. 8 shows an example system which several users can replay the media file simultaneously. The test file is amuro.mpg which size is 352 × 240, and the frame rates are 30fps, bit rates are 172KBytes/s. Even though there is only one media file originally, 4 peers could replay media files simultaneously through simultaneous download and upload. This feature increased the distribution speed of contents and the number of simultaneous users more than two times. Fig. 9 and Fig. 10 show the client and server view of intermediate node respectively, and Fig. 11 shows a client view of the bottom node.

**Table 1.** Characteristics of download and simultaneous replay for MPEG–1,2, DivX and Avi

| File name | Number of source peers | Delay time between packets | Data transmission rate(Kbytes/s) | Download time(sec) | File replay time(sec) |
|---|---|---|---|---|---|
| Amuro.mpg (MPEG–1, Low Resolution) | 1 | 5ms | 182.85 | 250 | 268 |
| | 2 | 5ms | 262.56 | 174 | |
| | 3 | 15ms | 187.88 | 243 | |
| Boa.mpg (MPEG–1, High Resolution) | 1 | 2ms | 455.11 | 147 | 259 |
| | 2 | 5ms | 262.56 | 255 | |
| | 3 | 10ms | 259.24 | 258 | |
| Sheri.VOB | 3 | 2ms | 749.26 | 392 | 407 |
| | 4 | 2ms | 808.42 | 364 | |
| Ani.avi (Divx4, mp3) | 1 | 15ms | 66.06 | 387 | 1203 |
| X-file (Divx3, mp3) | 1 | 5ms | 182.00 | 184 | 314 |
| | 2 | 15ms | 117.70 | 284 | |
| badboy.avi (Xvid, mp3) | 1 | 5ms | 192.00 | 520 | 622 |
| | 2 | 10ms | 190.00 | 526 | |
| | 3 | 15ms | 208.00 | 480 | |
| Firstlove.avi (Divx3, AC3) | 1 | 2ms | 455.11 | 340 | 629 |
| | 2 | 5ms | 262.56 | 589 | |
| | 3 | 10ms | 259.24 | 597 | |

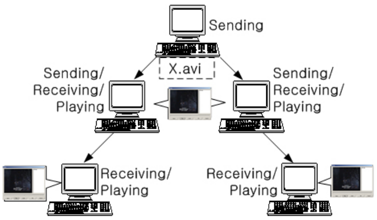


**Fig. 8.** System configuration

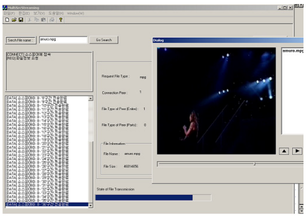
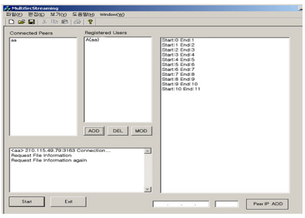


**Fig. 9.** Client view of an intermediate node



aa

A(aa)

```
Start:0 End:1
Start:1 End:2
Start:2 End:3
Start:3 End:4
Start:4 End:5
Start:5 End:6
Start:6 End:7
Start:7 End:8
Start:8 End:9
Start:9 End:10
Start:10 End:11
```

```
<aa> 210.115.49.79:3163 Connection...
Request File Information
Request File Information again
```

**Fig. 10.** Server view of an intermediate node

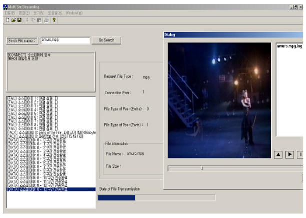


**Fig. 11.** View of a bottom client

## 4 Conclusions

The existing contents sharing programs had a shortcoming that the response time is long and it is not possible to confirm whether the contents is what the user wants or not during download. On the other hand, the existing researches on P2P multi-source media streaming did not considered the contents distribution services.

The proposed system can perform data transmission and receipt, file store and file replay simultaneously, therefore it can distribute contents between P2P peers easily. The system has an advantage to reduce the user response time and several features as it follows. First, it provides higher media transmission rates than that needed for file replay because it receives data from several peers. Experimental results show that the data transmission rate increased more than 683%. Second, the media can be replayed in real-time by saving received media as a file simultaneously. In addition, the contents distribution is also possible. Third, transmitting a part of media file which downloading peer has, makes fast distribution and diffusion of contents possible and the number of simultaneous user increases more than two times.

**Acknowledgments.** This work was supported by the Ministry of Information and Communications, Korea, under the Information Technology Research Center (ITRC) Support Program.

## References

1. D.Xu, M, Hefeeda, S.Hambrusch and B.Bhargava, "On Peer-to-peer Media Streaming", *IEEE ICDCS 2002*, pp. 363–371, July 2002
2. Reza Rejaie, Antonio Ortega, "PALS:Peer-to-peer Adaptive Layered Streaming", *Proceedings of the International Workshop on Network and Operation Systems Support for Digital Audio and Video*, June 2003
3. T. Nguyen and A. Zakhor, "Distributed Video Streaming Over Internet", *Proceedings of SPIE Conference on Multimedia Computing and Networking*, January 2002
4. Demertis Zeinalipour-Yazti, Theodoros Folias, "A Quantitative Analysis of the Gnutella Network Traffic", CS204 final project, California University, USA, July 2002
5. eDonkey, "http://www.edonkey2000.com"