# A GENERALIZED FRAMEWORK FOR CATV TRANSMISSION ON FUTURE BISDN

Geng-Sheng (G.S.) Kuo

Department of Information Management, National Central University
Chung-Li, Taiwan 32054, R.O.C.
TEL: +886 3 4263086 FAX: +886 3 4262309

*Abstract*

*The purpose of this paper is to investigate and construct a generalized framework for CATV transmission on future BISDN by using some new concepts and mechanisms of UNIX. The software developments have been made. In addition, some future works have been pointed out briefly.*

## INTRODUCTION

Clearly, the HDTV standard has already been decided in all-digital format, which is compatible with packet-oriented BISDN technologies. Hence, the HDTV technology can not only be used in broadcast networks, but also be used on the BISDN [8]. Furthermore, the BISDN is going to impact the CATV industry [10]. In the future, internetworking will construct one network in the world including BISDN, LANs, wireless networks, and so on. Due to frequency limitation on radio communications, BISDN will play the role of basic backbone for future communications.

The ultimate target of future communications will be the universally personal communications. Due to its established subscribers, CATV industry will keep the key role for information service provider on BISDN for business consideration. The information sources for a variety of future CATV-based multimedia information services will definitely be located on BISDN. To provide real-time-oriented CATV-based multimedia information services to large volume of users *simultaneously* is a must for the success of applications.

The purpose of this paper is to design a generalized framework for BISDN-based CATV information source (CATV server) serving the information services to a large volume of users simultaneously by using distributed client-server architecture with socket and fork mechanisms in UNIX.

## A GENERALIZED FRAMEWORK

Up to now, it might be one of the best solutions that the interfaces between the CATV-based terminal system set or multimedia workstation in user's side and information sources elsewhere are based on distributed client-server architecture [3]. In [4], a new better mechanism for distributed client-server architecture on the personal communications networks than the ones currently available has been designed. This new mechanism associating with the information management system designed in [1, 2] could play the role of basic framework for *interactive* customer control of future personal communications services [4].

On existing computer networks, UNIX is a well-accepted operating system. The socket mechanism of 4.3BSD UNIX [5] [6] is a special means of application program interfaces to a variety of different communication protocols [7]. The distributed client-server architecture needs socket as the means for interconnecting many client processes to some specific server process in order to provide corresponding users with the same information service *virtually* simultaneously. However, based on the current status of 4.3BSD UNIX and our experimental experiences on Sun workstation SparcStation SS10, the total socket number available for the server process is 256 and the number of child processes generated by one

parent process through fork mechanism is small too [9] [11].

In this paper, many study efforts and experimental experiences on the topic mentioned above have been made for shaping the proposed generalized framework. It is our conclusion that the multi-client multi-children-server single-parent-server architecture communicating with multiple pipes and sockets is the generalized framework for CATV transmission on future BISDN.

Parent server opens *P* pipes, generates *P* children in order to let every child own a private pipe. The value *P* is decided by MAXFORK defined in the following program. Then, it reads *K*-byte data from the file, sequentially sends them to its children with pipes. After the *K*-byte data have been send to all destinations, parent reads next *K*-byte data. The read-write procedure will repeated until parent reads the end of the file.

Every child opens *S* sockets, binds every socket to a unique port number. So, there is a port number range for every child, and the intersection of these ranges is null set. Then child enters a loop. Child receives data from its pipe, stores them in its buffer, and waits for a short time which is decided by `timeout` (a static `timeval` structure variable) to see if there are new client connections reaching. If there are indeed, child performs `accept()` function to establish connection with the client(s). Then, child sends the data in buffer to all connected clients with sockets. The loop procedure will not end until child can not read any data at all from its pipe.

The server, including parent part and child part, can be developed as the following program.

```
/*
 * Example of max client connection number testing
 */
#include "inet.h"

#define MAXFORK          3
#define MAXSD            3
#define BEGINPORT 10000


char *pname;
int  line[MAXLINE];

main(argc, argv)
int  argc;
char *argv[];
{
     int    childpid, pipefd[MAXFORK][2], i, n, f;
     int    sockfd[MAXFORK][MAXSD], acptflag[MAXFORK][MAXSD], newsockfd, clilen, maxfdp1;
     int    maxsd = MAXSD, readynum, cc;
     char   buffer[MAXFORK][MAXLINE];
     FILE   *fp;
     char   bufferpnt[MAXLINE];
     fd_set       writeto[MAXFORK], ready;
     static       struct timeval timeout;
     struct       sockaddr_in peer;
     int    peerlen = sizeof(peer);
     struct sockaddr_in  serv_addr, cli_addr;
```

```
pname = argv[0];
timeout.tv_sec = 0;
timeout.tv_usec = 5;


for(i=0; i<MAXFORK; i++) {
      if(pipe(pipefd[i]) <0)
            err_sys("can't creat pipe%d (No.%d)", i, pipefd[i]);
}

for(f=0; f<MAXFORK; f++) {

      if ((childpid = fork()) < 0 )
            err_sys("can't fork ");

      if (childpid == 0) {/* child */

            for(i=0; i<MAXFORK; i++) {
                  close(pipefd[i][1]);
                  if(i!=f) close(pipefd[i][0]);
            }
            printf("child(%d): pipe%d (No.%d) is ready for
              reading\n", f, f, pipefd[f][0]);


            FD_ZERO(&writeto[f]);

            for(i=0; i<MAXSD; i++) {

                  /* set the accept flags of the sockets off */
                  acptflag[f][i] = 0;

                  /*
                   * Open a TCP socket (an Internet stream
                   *  socket).
                   */

                  if ((sockfd[f][i] = socket(AF_INET, SOCK_STREAM,
                    IPPROTO_TCP)) < 0)
                        err_dump("socket() error! Max socket
                          No.= %d", i);


                  /* Bind our local address so that the client can
                   *  send to us.
                   */

                  bzero((char *) &serv_addr, sizeof(serv_addr));
                  serv_addr.sin_family    = AF_INET;
                  serv_addr.sin_addr.s_addr = htonl (INADDR_ANY);
                    /* INADDR_ANY _long)0x00000000 */
                  serv_addr.sin_port
                        = htons(f*MAXSD +i +BEGINPORT);
```

```
        if (bind(sockfd[f][i], (struct sockaddr *)
          &serv_addr, sizeof(serv_addr)) < 0)
               err_dump("child(%d): bind() socket%d
                 (No.%d) error!", f, i, sockfd[f][i]);

        listen(sockfd[f][i], 5);
        fprintf(stdout,"child(%d): socket%d (No.%d)
          (port %d) is listening !\n",f, i,
          sockfd[f][i], (f*MAXSD +i +BEGINPORT));

}

maxfdpl = sockfd[f][maxsd-1] + 1;
printf("child(%d): maxfdpl = %d \n", f, maxfdpl);

clilen = sizeof(cli_addr);

while(1) {

        for(i=0; i<MAXSD; i++)
               FD_SET(sockfd[f][i], &writeto[f]);

        readynum = select(maxfdpl, &writeto[f],
          (fd_set *) 0, (fd_set *) 0, &timeout );
        if(readynum < 0)
               err_sys("srever: select() error");

        if((n =readn(pipefd[f][0],buffer[f],
          MAXLINE))<=0) {
               printf("child(%d): closes pipe%d
                 (No.%d)\n", f, f, pipefd[f][0]);
               close(pipefd[f][0]);
               for(i=0; i<MAXSD; i++)
                      close(sockfd[f][i]);
               exit(0);
        }

        printf("child(%d) :\n", f);
        if(writen(1, buffer[f], n)<0)
               err_sys("child error: writen error on
                 screen\n");


        for(i=0; i<MAXSD; i++) {

               if (acptflag[f][i]) {
                      if(writen(sockfd[f][i],
                        buffer[f], n)<0)
                             err_sys("child(%d):
                              writen error on socket"
                              , f);
               }
```

```
                        if ((FD_ISSET(sockfd[f][i],
                         &writeto[f]))&&(!acptflag[f][i])) {

                           printf("child(%d): port %d is
                            selected !\n", f, (f*MAXSD +i
                            +BEGINPORT));

                           if((sockfd[f][i] =
                            accept(sockfd[f][i],
                            (struct sockaddr *) &cli_addr,
                            &clilen)) <0)

                              err_sys("child(%d):
                               accept() error", f);

                           else
                           {
                              acptflag[f][i] = 1;

                              if (getpeername(sockfd[f][i],
                               (struct sockaddr_in *)&peer,
                               &peerlen) < 0)

                                 err_sys("child(%d):
                                  getpeername() error", f);


                              fprintf(stderr,"child(%d):
                               select from %s\n", f,
                               inet_ntoa(peer.sin_addr));

                           }

                           if(writen(sockfd[f][i],
                            buffer[f],n)<0)
                              err_sys("child(%d): writen error
                               on socket", f);

                        }

                     }


            } /* end while: read from pipe, write to sockets loop*/

      } /* end child */

} /* end fork */

                        /* parent */

for(i=0; i<MAXFORK; i++)
      close(pipefd[i][0]);

if((fp = fopen("data", "r")) == NULL) {
```

```
            printf("error: fopen() ! \n");
            exit(0);
      }
      else printf("parent: open file ok, transmitting data ....\n");

      do {

            if(fgets(bufferpnt, MAXLINE, fp) == NULL) {
                  for(i=0; i<MAXFORK; i++) close(pipefd[i][1]);
                  fclose(fp);
                  printf("parent: close descriptors ...\n");
                  exit(0);
            }

            n = strlen(bufferpnt);

            for(i=0; i<MAXFORK; i++) {
                  if(writen(pipefd[i][1], bufferpnt, n)<0)
                        err_sys("parent: writen() error on pipe%d
                         (No.%d)\n", i, pipefd[i][1]);
            }

            sleep(1);

      } while(feof(fp) != EOF);
}
```

---

Client acquires a port number from command line, and binds its socket to that port in order to connect to server. It receives data from the socket and shows them on screen. The following program performs it.

---

```
/*
 * Example of client using TCP protocol.
 */
#include    "inet.h"
#define BUFLEN 1024

main(argc, argv)
     int             argc;
     char           *argv[];
{
     int              sockfd, port, n;
     char           buffer[BUFLEN];
     struct sockaddr_in serv_addr;

     /* name = argv[0]; */

     if(argc < 2) {
            printf("Usage: c ###      , ### indicates the port number\n");
            exit();
            }
```

```
        argv++; argc--;
        port = atoi(&argv[0][0]);


        /*
         * Fill in the structure "serv_addr" with the address of the
         * server
         * that we want to connect with.
         */
        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = inet_addr(SERV_HOST_ADDR);
        serv_addr.sin_port = htons(port);

        /*
         * Open a TCP socket (an stream socket).
         */
        if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
               err_sys("client: can't open stream socket.");
        else printf("client: socket() ok \n");


        /*
         * Connect to the server.
         */
        if (connect(sockfd, (struct sockaddr *) &serv_addr,
                    sizeof(serv_addr)) < 0)
              err_sys("client: can't connect to server");
        else   printf("client: connect() ok, socket is connected \n");

        str_read(sockfd);

        close(sockfd);
        exit(0);
}
```

The question is that: what is the maximum number of the clients that can be served by this server? It is dominated by the following two variables:

- *P*: the maximum number of child server processes generated by the parent server process on the same machine.
- *S*: the maximum number of socket descriptors that a process can open.

So, it is important to know the values of these two variables because the available client connection number is dependent on them.

## FUTURE WORK

Our first-cut results on this generalized framework for distributed client-server architecture on packet-oriented networks are very impressive, which might be beneficial to all CATV industry and multimedia applications by sharing the same public BISDN. Many important issues need to be studied. The comparison between high-speed protocol XTP and TCP/IP is needed for CATV-based information services. Further research efforts and experimental trials on this generalized framework are needed and have been planned in order to improve its functionality and performance.

## ACKNOWLEDGEMENT

I would like to thank Mr. Tsin-Hai Chang, one of my best students, for his contributions to the implementations of software programs in this paper.

## REFERENCES

[1] Kuo, G. S., "New Systems Concepts on HDTV System," in the 1991 NAB HDTV World Conference Proceedings, in Las Vegas, Nevada, on Apr. 15-18, 1991.

[2] Kuo, G. S., "Systems Architecture of Information Management System for Future HDTV System," in the 1992 NAB HDTV World Conference Proceedings, in Las Vegas, Nevada, on Apr. 13-16, 1992.

[3] Kuo, G. S., "Some Control Issues for Future HDTV System," in the Conference Proceedings on High Definition Video, International Symposium on Fiber Optic Networks and Video Communications, in Berlin, FR Germany, on Apr. 5-9, 1993.

[4] Kuo, G. S., "A New Framework for Customer Control of Future Personal Communications Networks," in the Fifth International Conference on Wireless Communications, in Calgary, Alberta, Canada, on Jul. 12-14, 1993.

[5] Stevens, W. R., UNIX Network Programming, Prentice-Hall, Englewood Cliffs, N.J., 1990.

[6] Sun Microsystems, Network Programming Guide, Revision A, Mar. 27, 1990.

[7] Ogle, D. M., Tracey, K. M., Floyd, R. A., and Bollella, G., "Dynamically Selecting Protocols for Socket Applications," IEEE Network, pp. 48-57, May 1993.

[8] Aoyama, T., Tokizawa, I., and Sato, K., "ATM VP-Based Broadband Networks for Multimedia Services," IEEE Commun. Mag., pp. 30-39, Apr. 1993.

[9] Kuo, G. S., "A Generalized Framework for HDTV Transmission on Future BISDN," in the Proceedings of International Workshop on HDTV '93, in Ottawa, Canada, on Oct. 26-28, 1993.

[10] Dukes, S. D., "Next Generation Cable Network Architecture," in the Proceedings of Technical Papers from the 42nd Annual NCTA Covention and Exposition, in San Francisco, California, on June 6 -9, 1993.

[11] Kuo, G. S. and T. H. Chang, "A Generalized Framework for HDTV-Based Multimedia Transmission on Future BISDN -(II)," in the Proceedings of NAB '94 MultiMedia World Conference, in Las Vegas, Nevada, on Mar. 20-24, 1994.