Design and Implementation of Firewire device driver on FreeBSD

Katsushi Kobayashi
Communication Research Laboratory, JAPAN
ikob@koganei.wide.ad.jp

Abstract

A Firewire device driver has been implemented on FreeBSD system. The driver provides IP network stack, native socket system interface, and stream device interface such as a DV video. The device driver shows enough performance on the IP over Firewire environment at 30Mbps. Also, DV video communication application using IP has been developed with the device driver and it enebles DV quality communication between US and Japan only with the consumer market products.

1 INTRODUCTION

Firewire, known as IEEE 1394 standard high-performance serial bus or iLink, has been designed as a packet based computer bus in the new age[1]. The market of the Firewire system is just breaking especially in audio visual area. To use a new kind of media not only Firewire, we have to designe and implement the device driver system and its associated environment including API. The Firewire specification considers that it is used for too many purposes, e.g., interface to storage devices, to audio visual equipments, to preripharal devices and between other computers. So it is difficult to categolize Firewire into legacy UNIX device types as either a network or a peripheral. In this paper, we present a Firewire driver development effort and discuss its way.[1]

[1] The device driver we mentioned here can obtained from following URL:
`ftp://ftp.uec.ac.jp/pub/firewire`

2 Overview of Firewire System

2.1 Firewire System

IEEE 1394 standard high-performance serial bus, often called “Firewire” or “iLink”, is a standard interface designed to meet a lot of requirements of the new generation. The standard covers the whole system of Firewire from its physical layer to the layer of network management function. Firewire is capable of high network bandwidth; 100, 200, 400, 800, 1600 and 3200Mbps, permits connecting their heterogeneous bandwidth equipment on the same system, supports hot plug-in and -out on its working environment, and also provides both best effort and bandwidth guaranteed communication within one network media. From these advanced features, Firewire has the potential to integrate into only one bus system every peripheral interface of computers as SCSI, every network interface as Ethernet, every processor bus system as VME, and processor interconnect on multi processor systems.

In the IEEE 1394 standard, two types of media are defined as physical layer devices, i.e., backplane environment and cable environment. Today’s market supplies only cable environment, while the products of backplane environment have not appeared yet. So, hereafter we only mention cable environment IEEE 1394 system.

Typical Firewire system consists of device nodes and cables, and its network topology is a tree structure. 2^{48} memory space, including the Firewire specific control registers, is assigned to each device and all the devices connected are mapped into unified memory space. One Firewire network can be connected up to (2^6 - 1 = 63) device nodes. When using a Firewire network bridge device, up to (2^{10} - 1 = 1023) network can be interconnected. Therefore, in the specification, almost 2^{16} device nodes are allowed on a single system and every device is mapped into 2^{64} memory space. Every communication action is brought with 125 μ second (8kHz) time slice whose value corresponds to the fairness unit in the Firewire system. The time slice unit is also divided into 6144 time slot. By allocating the number of the slot to each application, the network resource on the bandwidth guaranteed communication is managed. Three data transfer modes are defined i.e., Asynchronous request, Isochronous stream, and Asynchronous stream. Asynchronous request provides the function to communicate between the devices specifying the address in the packet with the memory accessing action, e.g., read, write, and transaction update same as memory devices. Isochronous stream mode provides the bandwidth guaranteed communication way using broadcast-like transmission not sent to specific devices. The stream has a feature of 6 bits channel identifier in the packet header. Asynchronous stream mode is defined in IEEE 1394.a standard, a supplemental specification of the original. This mode provides the best effort basis communication in a broadcast style. Due to so many requirements from today's media as bandwidth resource control, hot plug in/out, lower jitter transmission, and a large number of devices on one system, the whole specification of the Firewire became a large and complicated one. To support every function defined, a lot of effort is required compared with other network media.

IEEE 1394 standard series just defined a raw packet level communication protocol. When application uses Firewire, higher level protocols are also required such as IP over 1394, IEC61883 for real-time communication protocol, or SBP-2 for applying SCSI function[2, 3]. The standardization processes for their protocols have been accelerated in recent years.

The Firewire device driver has been developed for the major operating systems, such as Window 98, Windows NT and Linux, by distributing the Firewire equipments.

2.2 IP networking

IP over Firewire standardization effort has been proceeded in IP1394 working group in IETF[4]. Although the fundamental architecture such as the selection of the transfer mode in each occasion has been reached consensus, the status of its standard is still on a draft stage and details of architecture will possibly be changed.

Two transmission modes, asynchronous request and asynchronous stream, are adopted by IP1394. In the case of IP unicist and ARP response, asynchronous request packet will be sent to the specific node. In other communication modes such as IP multicast, broadcast, and ARP request, asynchronous stream mode will be used. The stream channel assignment protocol are also defined.

1500 bytes value is adopted for the MTU size for IP packet on the IP over Firewire standard. On the lower speed communication modes such as 100 and 200Mbps, this 1500 bytes MTU is large compared with the link level MTU size in the Firewire network. Moreover, a Firewire device must ensure communication, even when the device only has poor input buffer compared with the link level MTU. The size of link level MTU on the Firewire may be more restricted and changed in some condition. If the smaller MTU issue in the intermediate link prove with the IP fragmentation, it leads the end-to-end network throuput to a poor one. Because a loss of fragmented packet causes unused packet sent. IP1394 group solved this issue by the link fragment mechanism; i.e., the fragmentation and assembly of the packet larger than MTU is done within the local media.

2.3 A/V device

Firewire is adopted as the standard of digital interface for audio visual device, since it provides lower jitter, bandwidth guaranteed transmission within the network level, and also provides lower equipping cost. Today, Firewire applications are one of the most popular concerns for the audio visual products such as various types of video camcorders and VCRs. The protocol structure of the audio visual interface consists of 2 layers. The lower layer corresponds to a generic communication protocol for real-time stream media using isochronous stream. The higher one is designed as an adaptation protocol to each media type as NTSC, PAL, HDTV, MPEG and MIDI[5]. Some of these protocols are well known as DV video and the protocol set is approved as an IEC standard, IEC61883. In the real-time media system connected with the packet based network, the jitter of packets is the most important factor. If a larger jitter of packet is accepted, the system becomes more expensive because of the necessity of larger input buffer for absorbing the fluctuation. To avoid the problem, the specification requires strict timing in the order of μ second. To support such strict real-time media packet timing, the computer system is recommended to take account of the hardware level not only the software.

2.4 Peripherals

SBP (Serial Bus Protocol) is designed as a SCSI adaptation protocol to Firewire and its standardization process is ongoing in ANSI. SBP is expected as the most major protocol for a storage device interface. However, the standards for computer peripheral are already established, e.g., SCSI and IDE apart from audio visual world. The Firewire application in computer peripherals has not been popular yet, even if Firewire has some advantages.

AV/C protocol is an audio visual device control protocol supposing the cooperation with auch as DV video mentioned above, and provides audio visual equipment specific protocol set as "Play", "Rewind", and "Record". AV/C protocol only uses the raw Firewire functions in a simple way without any legacy protocol adaptation such as SCSI.

3 Integrating BSD system

The goal of our effort is to provide the Firewire environment on BSD system independent of the Firewire hardware. The device driver we developed is divided into two parts, i.e., the common part of the Firewire system and the device dependent part. We describe both of them in this section. We implement some feature of the Firewire into the kernel level due to the indispensable to use it. The new kernel implemented feature concerning native Firewire are Bus Manager, Isochronous Resource Manager, and CSR register includes related functions,

3.1 Native Firewire socket

The driver supports native mode Firewire socket, or N1394. The most part of N1394 is written referring to NATM code that is ATM native mode socket system included from FreeBSD 3.0 release[6]. Both Asynchronous request and Isochronous communication modes can be used. The N1394 socket structure is shown in Fig. 1.

Unused fields in Fig 1, `sn1394_lch`, `sn1394_ltag`, and `sn1394_mode` remain due to backward compatibility previously released.

Isochronous stream

This mode provides the function for sending and recieving isochronous packet at a little programing cost. In this mode, user creates communication end point of socket, connects it to the isochronous channel specified by the N1394 socket structure, and communicates in the conventional datagram manner as UDP(Fig. 2).

Our Pentium MMX 233Mhz computer can recieve the stream of consumer DV video, whose bandwidth is about 28Mbps using the isochronous interface. Although it also has a enough performance to send the video stream, the video device connected with Firewire cannot display due to the the reason of described later.

Asynchronous request

This mode provides asynchronous request function. In this mode, user creates communication end point of socket, connects it, sends the asynch packet user prepared, and waits for the responce from the target device(Fig. 3). When using this mode, user must ensure the data structure of the asynchronous request

```
struct sockaddr_n1394 {
    u_int8_t    sn1394_len;           /* length                */
    u_int8_t    sn1394_family;        /* AF_N1394              */
    char        sn1394_if[IFNAMSIZ];  /* interface name        */
    u_int8_t    sn1394_fch;           /* isochronous channel   */
    u_int8_t    sn1394_lch;           /* parameter unuse       */
    u_int8_t    sn1394_ftag;          /* tag of ishchronous    */
    u_int8_t    sn1394_ltag;          /* parameter unuse       */
    u_int8_t    sn1394_mode;          /* parameter unuse       */
    u_int8_t    sn1394_flags;         /* specify socket status */
    u_int8_t    sn1394_sped;          /* transmission speed.   */
};
```

Figure 1: N1394 socket structure

```
....
#include        <net1394/netfw.h>
...
main(){
        int s;
        char ifname[] = "lynx0";
        struct sockaddr_n1394 sfw;
        u_long recvbuf[512/4];
        int i, len;

        if((s = socket(AF_N1394, SOCK_DGRAM, PROTO_N1394ISO)) < 0 ){
                perror("socket");
                exit(1);
        }

        sfw.sn1394_family = AF_N1394;
        sfw.sn1394_lch = 63;
        sfw.sn1394_ltag = 0x1;
        bcopy(ifname, sfw.sn1394_if, IFNAMSIZ);

        if(connect(s, (struct sockaddr *)&sfw, sizeof(sfw)) < 0){
                perror("connect");
                exit(1);
        }
        while(1){
                len = recv(s, recvbuf, 512, 0);
                ......
        }
}
```

Figure 2: Code example to use Isochronous stream with socket

packet oneself in a different way from the isochronous case.

3.2 Firewire BSD and IP

Our implementation of IP over Firewire is partially compliance to the specification described in `draft-ietf-ip1394-08.txt`. The part of difference from the original are:

- Isochronous stream mode is used insted of Asynchronous stream in ARP and IP broadcast
- MCAP function is not implemented.

Firewire is a broadcast capable media and IP over Firewire adopts broadcast based ARP function same as Ethernet. So, the usage of the IP adaptation of Firewire does not require special mechanism. User can use IP network with following familier operation:

```
# ifconfig lynx0 10.0.0.1 netmask 255.255.255.0
```

The improvement of the IP implementation is now stopping. So, our implementation does not accord with the latest IP1394 draft. However, since the basis of the specification has not been changed, the modification to accord with the latest one may not be a difficult.

3.3 Audio Visual device support

IEC61883 real-time media protocol requires strict packet timing in the order of μ second. This timing condition can be satisfied with transmitting the isochronous packets at every 125 μ second time slice. Almost all Firewire devices support such a transmission with its programmable DMA function. Moreover, the protocol requires to write 24.576 MHz clock based time stamp into the specific packet header for the clock synchronization between the sender and the receivers. The generic BSD socket system cannot send packets ao as to satisfy these strict timing and cannot support the special treatment of the time stamp.

We prepare another device system for play out real-time stream media in addition to a network socket interface, which is named "dv". The purpose of "dv" device is mainly for play out use, since the packet timing issue is not serious in the receiving. The "dv" device system treats a bunch of isochronous packet sequence as a minimum transmission unit. Every transmission operation on the device must be done with the bunch of packets. This treatment reduces the interrupt occuring at the end of DMA operation compared with the socket system if the number of packets in a bunch is large. To avoid the fluctuation in the system, the output queue is implemented and the length of the queue can be changed up to 12. The application and the kernel only copy the bunch to the DMA data entry and kicks a DMA action after the previous queued data is sent out. When the driver kicks a DMA action, only the first packet's time stamp field in the bunch is overwritten by a certain value. Then, the application must arrange the packet requiring the time stamp to become the first in the packet group. On the DV video format, the unit of bunch corresponds to the one video picture frame.

The "dv" device system provides two methods for stream packet play out. One is to use `write()` system call and the other is `mmap()` system call. Since both system calls do not kick the DMA operation itself, the application must tell the start of the operation to the kernel using `LYNX_DV_TXSTART` ioctl.

Use of `write()` system call

This mode is conventional use of "dv" device. This mode provides isochronous stream transmission with low program cost. In this mode, user open the device, load the data into a buffer and kick the DMA action(See `LYNX_DV_TXSTART`). This system call does not return the success code until the system comes by an empty buffer after sending out the previous data. This function satisfies the strict packet transmitting timing only at the kernel level. So, user will not care about the strict timing issue and it provides easiest way to make isochronous stream. When using this mode, user must be ensured that each packet is stored into 512 bytes boundaries in the output buffer and the first quadrat data of every packet corresponds to the header of the isochronous packet including the packet

```
....
#include        <net1394/netfw.h>
...
main(){
        int s;
        char ifname[] = "lynx0";
        struct sockaddr_n1394 sfw;
        u_long sendbuf[512/4], recvbuf[512/4];
unsigned long long addr;
        int i, len;

        sfw.sn1394_family = AF_N1394;

        bcopy(ifname, sfw.sn1394_if, IFNAMSIZ);

        if((s = socket(AF_N1394, SOCK_DGRAM, PROTO_N1394ASY)) < 0 ){
                perror("socket");
                exit(1);
        }

        if(connect(s, (struct sockaddr *)&sfw, sizeof(sfw)) < 0){
                perror("connect");
                exit(1);
        }
/*
 * Offset 0x0001fffff0000018 represents destination network/node/address
 * network = 0x0,
 * node    = 0x1,
 * address = 0xfffff0000018(SPLIT_TIME_OUT_HI register)
 */
        addr =  0x0001fffff0000018ull;

/* To make a packet for Read request for data quadret */
        sendbuf[0] = htonl(0x00000040 | ((addr >> 32) & 0xffff0000 ));
        sendbuf[1] = htonl(((addr >> 32) & 0x0000ffff ));
        sendbuf[2] = htonl(addr & 0xffffffff);

        send(s, sendbuf, 12, 0);

/* Wait for a response from destination */
len = recv(s, recvbuf, 512, 0);
        .......
}
```

Figure 3: Code example to use Asynchronous request with socket

size, tag, channel number, tcode and sync information(Fig. 4). The nbytes field of `write()` specifies the total size of the isochronous packets bunch.

Use of `mmap()` system call

This mode provides the accessing way of the "dv" device buffer both for sending and for receiving allocated in the kernel space. In the mmaped area of "dv" device, the control entry is placed before the memory space of output/input buffer. When using this mode, user manupurates the control entry in the "dv" specific manner additionally to the buffer management on the write system call. The data structure of the mapped "dv" device is shown in Fig. 5. In this mode, user open the device, map the "dv" device into the application space, access the space as conventional memory and kick the DMA actions(See `LYNX_DV_TXSTART/LYNX_DV_RXSTART`)(Fig. 6).

The size of both output/input buffers above is calculated as following:

$$size = sizeof(u\_int32\_t) \times N_q \times L_p \times N_p$$

where N_q, L_p and N_p represent the number of attached queue, the size of the maximum packet, and the number of packets included in the buffer respectively. In our driver, N_q, L_p and N_p are set to permanent value as 16, 512 and 300 respectively.

Also, `ioctl()` supports the following functions.

- `LYNX_DV_TXSTART`, `LYNX_DV_TXSTOP`
  Start and stop the DMA action for transmitting
- `LYNX_DVRX_START`, `LYNX_DVRX_STOP`
  Start and stop the DMA action for recieving
- `LYNX_SSIGNAL_WR`, `LYNX_GSIGNAL_WR`
  Set/get the process to raise signals, when transmitting DMA action is finished.
- `LYNX_GSIGNAL_RD`, `LYNX_GSIGNAL_RD`
  Set/get the process to raise signal, when recieving buffer is filled.
- `LYNX_GFRAMESIZE`
  Get the amount of size of memory the device attached
- `LYNX_DV_SYNC`
  Set the queue length of transmission buffer. If the fluctuation factor is large in transmitting, a large value should be settled.

We have released an DV application software that transmits DV video over IP using the function[8]. This application accomplishes DV quality communication only using the equipment on the consumer market. In SC98, we have presented the DV quality video communication between US and Japan with Transpac, Startap, and vBNS links. The application consumes over 30Mbps bandwidth including IP packet header in NTSC and it is not feasible to use such a bandwidth eater now. However, we believe that such volumes of bandwidth will be obtained easily in the near future, because the improvement of the high-speed link technology is too fast.

3.4 Suppoted Device

Our device driver supports two types of the Firewire chipset, i.e., Texas Instrument's PCILynx and Adaptec AIC5800[7, ?]. Since AIC5800 is derived from Apple Firefire chipset, the chipsets of the same series, e.g, Sony's chipset may work with a bit modification of the device driver code. Both chipsets only support the speed up to 200Mbps due to earlier product and does not support Asynchronous stream transmission defined in IEEE 1394.a itself. On PCI-Lynx driver, our driver performes about 30Mbps TCP transmission performance with 100Mbps mode in netperf. This value shows that our driver has good performance compared with theoretical limit as 32Mbps. In some cheap PC configuration, the DMA operation stops in failure probabily due to the poor PCI bus performance.

We are planning to develop the driver code for second generations's chipset as OHCI and PCILynx2 that supporting up to 400Mbps, asynchronous stream mode.

4 Conclusion

We developed a Firewire device driver on the FreeBSD system. Of cource, the API specification and the driver we presented in this paper is not complete. It is still an open issue which type of UNIX

```
....
#include <machine/lynx.h>
...
main(){
        int d, qlen, dummy, frame;

#define MAXFRAME 12

        u_long sendbuf[MAXFRAME][512/4*300];
        u_long datalen[MAXFRAME];

        d = open("/dev/dv0", O_RDWR);

/* change queue size */
if( ioctl(d, LYNX_DV_SYNC, &qlen) < 0 ) {
             err(1, "LYNX_DV_SYNC");
}

/* kick playout */
        if( ioctl(d, LYNX_DV_TXSTART, &dummy) < 0 ) {
             err(1, "LYNX_DV_TXSTART");
        }

        ......

frame = 0;
while(1){
frame++; if( frame == MAXFRAME ) frame = 0;

         .....

         if( write(d, sendbuf[frame], datalen[frame] * 512) < 0 ){
                 perror("write");
         }
}

        ......
}
```

Figure 4: Code example to use `write()` on “dv”

```
struct dv_data{
        /* Maximum size of output queue */
        u_int32_t n_write;
        /* Buffer now user locking, device does not send this data */
        u_int32_t a_write;
        /* Buffer now kernel locking, user must not access this data */
        u_int32_t k_write;
        u_int32_t write_done;
        /* Number of valid packet data in the buffer */
        u_int32_t write_len[16];
        /* Offset of buffer for writing */
        u_int32_t write_off[16];
        /* Maximum size of input queue */
        u_int32_t n_read;
        /* Buffer now user locking, device does not write this data */
        u_int32_t a_read;
        /* Buffer now kernel locking, this entry is not stable */
        u_int32_t k_read;
        u_int32_t read_done;
        /* Number of valid packet data in the buffer */
        u_int32_t read_len[16];
        /* Offset of buffer for reading */
        u_int32_t read_off[16];
};
...
isochronous data buffers for output
...
isochronous data buffers for input
...
```

Figure 5: Control entry on “dv”

```
....
#include <machine/lynx.h>
...
main(){
int d, size;
        u_int32_t *dvdata;
        d = open("/dev/dv0", O_RDWR);

/* obtain the amount of maximum buffer size incl. control, output and input */
        if(ioctl(d, LYNX_GFRAMESIZE, &size)){
                exit(1);
        }

        if((dvdata = (u_int32_t *)mmap((caddr_t) 0, size,
PROT_WRITE|PROT_READ, 0, d, (off_t)0)) < 0 ){
              err(1, "mmap");
        }
.....
/* Manipulate output/input buffer */
}
```

Figure 6: Code example to use `mmap()` on “dv”

system call accords with Firewire programing. Especially, the socket implementation of asynchronous request must be reconsidered. Because it is not a light programming effort to satisfy various mode of the asynchronous requests function. Many DV video equipment cannot display complete picture probably due to packet timing, even if using our “dv” implementation. A complete video picture appeares in the limited configuration only. We are trying to solve the problem investigating the packet behaviour. The device having a Firewire interface will be more distributed and a variety of devices will increase. However, our developing capacity is limited. So, we want to cooporate with other UNIX development effort, e.g, implementing SBP with SCSI experts.

There is also other device driver development efforts on LINUX system. And, some UNIX vender announces the Firewire support on its OS as SGI IRIX. We are planning to make compatibility with other implementations, to reduce porting effort between UNIX systems.

References

[1] IEEE Computer Society, “IEEE Standard for a High Performance Serial Bus”,IEEE Std. 1394(1996)

[2] International Electrotechnical Commission, ”Consumer audio/video equipement Digital interface”,IEC 61883(1998)

[3] American National Standard for Information systems, ”Serial Bus Protocol 2”,ANSI NCITS 325(1998)

[4] P. Johanson, “IPv4 over IEEE1394”, Internet Draft draft-ietf-ip1394-ipv4-8.txt(1998)

[5] HD Digital Video Conference, “Specifications of Consumer-Use Digital VCRs using 6.3mm magnetic tape”(1995)

[6] C. D. Cranor, “Integratig ATM Networking into BSD”, See `http://www.ccrc.wustl.edu/pub/chuck/`

[7] Texas Instruments, "1394 to PCI Bus Interface/TSB12LV21APGF Functional Specification" (1998)

[8] Akimichi Ogawa *et al., "Design and implemnentation of DV Stream over Internet", Proc. of Internet Work Shop 99(IWS99) (1999)*