Design and Implementation of Synchronization of WPAN Devices for Multimedia Playback

By Xiaofei Tian

September 3rd , 2010

Committee Members:

Supervisor	**Dr. Martin Jacobsson**
Responsible professor	**Prof. dr. ir. Ignas Niemegeers**
Member	**Dr.ir. Anthony Lo**
	Dr. Christian Doerr

TU Delft

Acknowledgments

This thesis is the result of a 9 months master project. At the moment when the project is finished, I would like to thank my mentor dr. Martin Jacobsson for giving me the chance to participate this project. Thanks for the time he spent on discussing the related works with me. I have learned a lot from his patient explanations and professional programming skills.

I would like to thank prof.dr.ir. Sonia Heemstra de Groot and dr. Ertan Onur for their kind suggestions when I was looking for my thesis work.

I would like to thank all the instructors of telecommunications department of TU Delft. Thanks for giving me a wonderful academic life during the last two years.

At last, I would like to say thanks to my friends and my family, especially to my dear mother. Your continuously support and encouragement made me survived here.

Abstract

With the development of projector technology, portable projectors called pico-projectors are emerging on the market. To achieve a high quality projection, multiple pico-projectors can be used to display the same video, where accurate synchronization is required.

The target of this thesis is to design and implement a Linux-based application which can synchronize multimedia playback over a WPAN with high accuracy. By analyzing the synchronization requirements, the synchronization problem is decomposed into playback synchronization and clock synchronization. To achieve a synchronized playback based on synchronized clocks, we explore several open source projects and eventually build a synchronized player which can play video with frame-by-frame level synchronization. For clock synchronization, the principles of a typical computer clock and the most popular time synchronization – NTP are studied. Based on that, a new time synchronization protocol with quick-start and anti-interference properties is designed.

We also conduct a survey to test if the application meets the users' requirement of synchronized playback. The final results show that our application provides a synchronized playback with the accuracy which is acceptable for most users.

Content

Chapter 1 Introduction

1.1 Project Overview

With the development of projector technology, portable projectors called pico-projectors are emerging on the market. A pico-projector has a size that fit inside a typical mobile phone. Early pico-projectors were pure projectors with the same function as traditional projectors. Currently, they are being embedded into portable devices, such as mobile phones and digital cameras. With its portability and the data communication ability of such devices, new applications can be designed around the projection function.

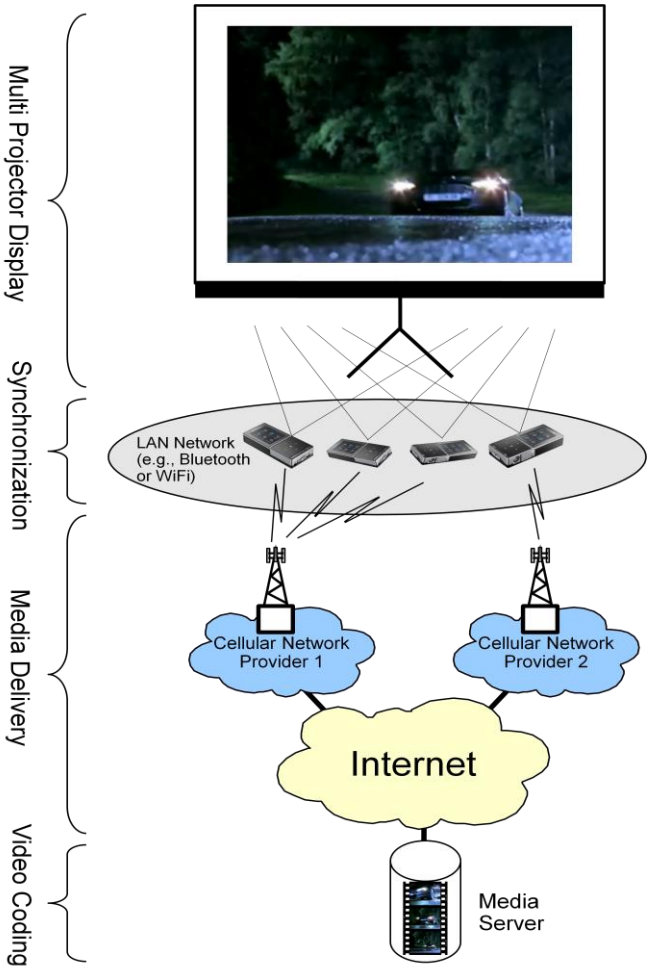


Figure 1.1

Compared with a normal projector, the resolution and illumination are worse. To compensate this disadvantage, multiple pico-projectors can be utilized in the following approaches:

1) Tiling

Multiple pico-projectors can be used to build a video wall, where each pico-projector is in charge of a specific part of the screen. Therefore, the resolution is increased. Thereby, a movie in full HD format can even be presented by pico-projectors.

2) Superposition

Multiple pico-projectors can also be used in the way of superposition. All the pico-projectors are in charge of the whole screen, but a specific pico-projector only projects specific video frames. For example, if two pico-projectors are used, one of them projects odd frames and the other one projects even frames. Therefore, the illumination is increased.

Besides, this application can be extended. In the future, devices could request the multimedia content from a media server located in the Internet via cellular networks. The media content are preprocessed in the media server and streamed to the devices. Figure 1.1 shows such a scenario.

To achieve either of the approaches above, the playback needs to be synchronized. Therefore, a WPAN is used for the synchronization. This thesis focuses on the synchronization of playback over a WiFi [1] network. The goal is to achieve synchronized playback by using two computers through designing and implementing an application on the Linux platform.

1.2 Final Target

The synchronization of multimedia playback is an entertainment application. Typical target users are the people who have no background knowledge about

computer and telecommunications technology. Therefore, the whole application should be easily set up. Figure 1.2 shows our experimental setup.

Figure 1.2 The experimental setup

For the users, they are happy when synchronized screens appear shortly after the application starts. Long time waiting should be avoided. Thus, the application should have a quick-start property. The maximum start time should be controlled within several tens of second.

Since the project is aiming to have video streaming via cellular networks in the future and the number of wireless devices is significantly increasing, the interference of the WPAN signal cannot be ignored. Therefore, the application should have the ability to cope with a relative strong interference.

The last and the most important requirement is synchronized playback. The application should have a good synchronization accuracy that is acceptable by the users. Therefore a bound of the synchronization accuracy should be set. Within this bound, the playback can be considered in sync by most users. To find the bound is a challenge. Then I conduct a users' feeling survey. Through the survey, we find the users' acceptance threshold for the playback synchronization is around 8 ms. Thus, we set 8 ms as the final accuracy target of the synchronization over WiFi networks. The details of the survey will be

described in Chapter 6. During the implementation of the application, I tried to minimize all error factors.

1.3 Synchronization Basics

What people are interested in is when the frames eventually be put on screen regardless of how long time the frames are being processed inside the video player. In order to achieve a synchronized playback among devices connected over a WPAN, the moment when a frame appears on the screen, must be exactly the same on all devices. Several factors are involved in achieving that, such as, the playback rhythm, local clock synchronization and system response time synchronization. The most simple and reasonable approach is to make all the factors above synchronized rather than use additional delay in one factor to compensate out-of-sync in other factors. Let us briefly look into every factor:

1. The first factor is the playback rhythm. All video formats are based on picture frames. A video player sending sequential frames in turn onto the screen gives people the feeling of dynamic image due to the persistence of vision effect. The frames must appear on the screen in a fixed rhythm in order to truly reflect the content of the video. Playback that is too fast or too slow must be avoided. In almost all the video formats, timestamps are included in every frame that gives the player the information about when to put the frames onto the screen. Usually, timestamps are the relative time with respect to the beginning time of the video. Once the playback starts, the video player records the starting time of the playback, then convert all the relative timestamps into absolute timestamps. The player compares the absolute timestamps with the local clock to decide when to put the frames to the screen. Thus, the local clock determines the rhythm of the playback.

2. The second factor is the synchronization of the local clocks. From the playback rhythm factor, we know that, the players use the absolute

timestamps to present each frame. Hence, to have synchronized local clocks is the prerequisite for realizing synchronized playbacks. Here, both the frequency of the clock, i.e. the speed of the clock, and the accumulative offset error, i.e. the instant indication of the clock, need to be the same. Clock frequency synchronization guarantees the playbacks have the same rhythm. Clock offset synchronization ensures that there is no time offset between the two playbacks when they have the same rhythm.

3. The third factor is the response time of the system. The response time is the time a system takes to react to a given input. Actually, the time difference between the response times of any two modern electronic devices are getting smaller and smaller, especially, when they are equipped with multi-core processors and have multi-threading ability. However, we are going to make an application which is various devices oriented. We cannot make the hypothesis that the devices are from the same vendor, with the same hardware configuration and have the same software installed. Therefore this factor still needs to be considered, otherwise, we will risk having playbacks which are out of sync although the local clocks are synchronized.

We can see that, to realize synchronized playbacks, several related factors need to be taken into consideration. Now, let us do some modeling to make the problem clear.

First of all, we need to make some definitions. Let us call the moment when a frame eventually appears on screen, the *presentation time of the frame*. The moment when the player starts to play is called *trigger time*. Each frame has its own presentation time. However, the trigger time can be discussed in different scales depending on the approach we use to make the player. If we are using a ready-made video player, the trigger time represents the time when the player is triggered to start. If we can manipulate the video player and operate it

at a frame-by-frame level, e.g. using some codec APIs, the trigger time represents the time when a specific frame is sent to the screen.

Based on the concepts of trigger time and presentation time, we can define the response time. Generally speaking, the response time is the period of time between the trigger time and the presentation time. Again, the concept of response time can be defined in different scales. In the situation which a ready-made player is utilized, the response time is the time period between the trigger time of the player application and the presentation time of the first frame. In this case, we have only one trigger time and one response time. However, for an arbitrary frame in the frame-by-frame case above, it should refer to the period of time between the trigger time of a frame and the corresponding presentation time. Now, the numbers of trigger times and that of presentation times are equal to the number of frames.

Figure 1.3 gives the timing diagram of the case using a ready-made video player for a two devices playback circumstances and Figure 1.4 shows that of doing the playbacks at a frame-by-frame level.

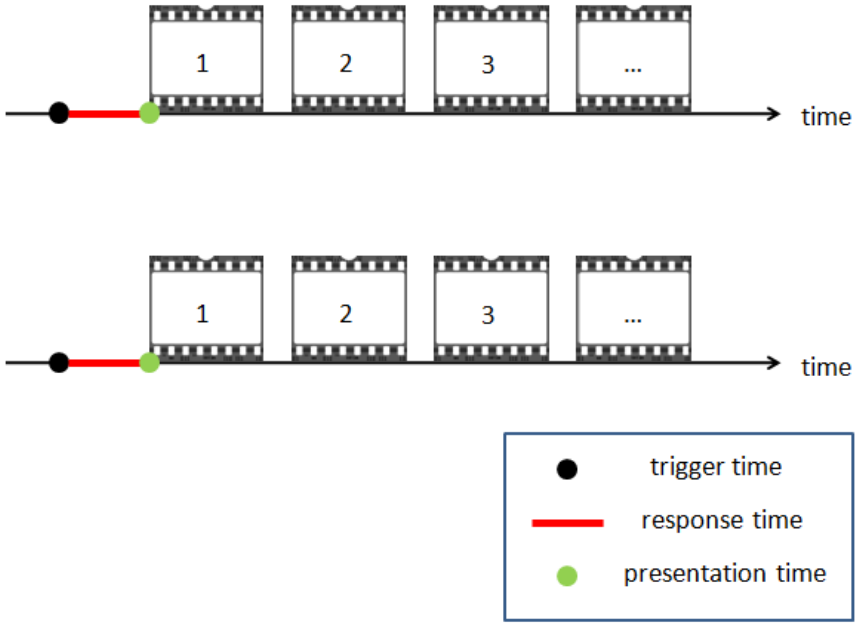


Figure 1.3 Using ready-made player for synchronized playback

For the case of Figure 1.3, we can notice that, the player has the ability to maintain the rhythm of the playback based on the local clock. To make the two playbacks synchronized, the corresponding trigger times must be aligned and the two response times must have the same length. Since on each device, the player can only know its local time as time reference, the synchronization of the local clocks is a requirement of making the trigger times and the response times in sync.

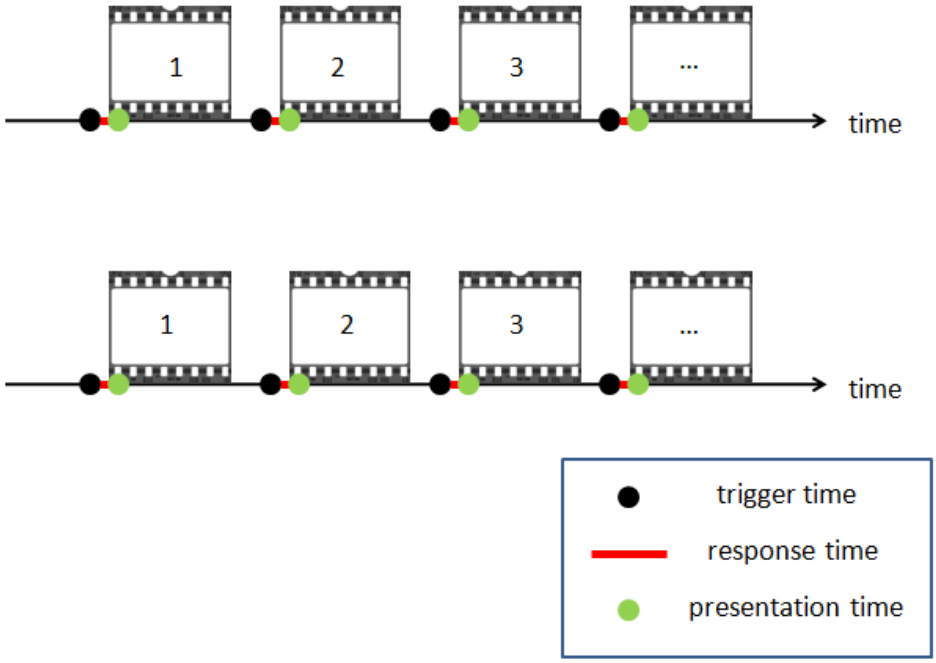


Figure 1.4 Using frame level tool for synchronized playback

In Figure 1.4, we are at the frame level. In this situation, to achieve synchronized playback, we need to make all the corresponding trigger times and all the corresponding response times aligned on the time axis. Due to the same reason as above, the clocks of the devices still need to be synchronized to supply reliable absolute time references.

The two situations above clearly show that, no matter what kind of tool we use to make the video player, to achieve synchronized playbacks, two aspects must be guaranteed. One is the clock synchronization; the other is synchronization of playback based on synchronized clocks. The latter one can

be divided into trigger time synchronization and response time synchronization. To synchronize the trigger times is not a difficult task as long as we have the clocks well in sync. Hence, the key of synchronization of playbacks is to make sure that response times have the same length.

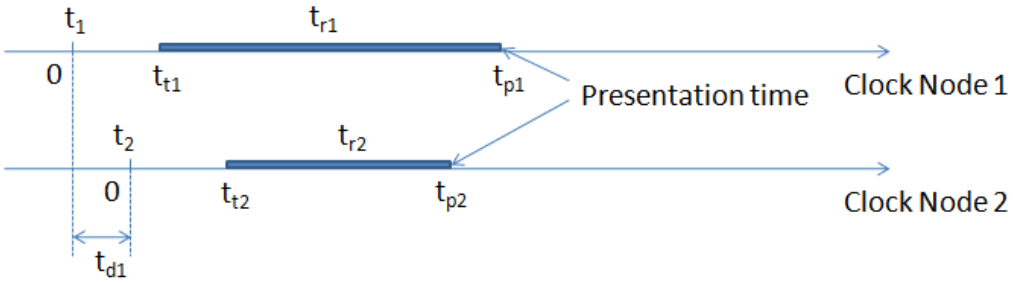


Figure 1.5 Abstracted timing diagram

Figure 1.5 gives an abstraction of the problem. t_1 and t_2 correspond to the zero points of the time axes which show the relative clock offset of the clocks. t_{t1} and t_{t2} are the trigger times. At these times, the devices received their playback instructions. The clock offset is,

$$t_{d1} = t_1 - t_2$$

The difference between t_{t1} and t_{t2} is the same as the clock offset. Hence, we can easily synchronize the trigger time if we have synchronized clocks (t_{d1} is small).

t_{r1} and t_{r2} are the response times of the devices respectively. The response time difference is,

$$t_{d2} = t_{r1} - t_{r2}$$

t_{p1} and t_{p2} are the final presentation times. The presentation time difference can be as bad as the sum of the two time differences,

$$t_{dt} \leq t_{d1} + t_{d2}$$

Let us go back to the beginning of this section, people can only say that the

playbacks are in sync when the absolute presentation time of the playbacks are the same. That means, the presentation time difference, i.e. the total time difference is close enough to zero. Thus, the task is to minimize the total time difference to a tiny level which is not noticeable by people. Since t_{dt} is bound by the sum of t_{d1} and t_{d2}, an easy way to minimize t_{dt} is to minimize both t_{d1} and t_{d2}. So far, our final job has been divided into two parts, the minimization of clock offset and the minimization of the response time difference, i.e. the synchronization of clocks and the synchronization of playbacks. In the following chapters, these two aspects will be discussed in detail.

1.4 Outline of the Thesis

The thesis is organized as follows: Chapter 2 will describe how a synchronized playback is achieved based on synchronized clocks and how the response time is minimized by exploring several open source projects. Chapter 3 will give the principles of a typical computer clock and the most popular time synchronization – NTP. Besides, the performance of NTP will be studied as well. In Chapter 4, the design and the implementation of a new time synchronization protocol – Fast Clock Adjustment Protocol will be given. Chapter 5 will show the measurement setup and the measurement results of the time synchronization protocols. Chapter 6 will describe how the users' feeling survey is conducted, as well as the analysis of the survey results. Chapter 7 will conclude the thesis and will give the suggestions for future works.

Chapter 2 Synchronized Playbacks

As mentioned in section 1.3, achieving the minimization of clock offset and the minimization of the response time difference is a reasonable approach to realize synchronized playbacks. In this chapter, we will focus on the minimization of the response time difference. In details, we will discuss how to use various tools to make a video player with the ability to play something in sync on multiple devices based on synchronized local clocks. During the development, we explored two open source projects which are related to media playback. They are VLC and FFmpeg.

2.1 Utilizing VLC

The VideoLAN project was initially started in 1996 in the French engineering school École Centrale Paris by students there [2]. Their original intention was to watch television on their computers based on media steaming. Two programs, VLS (VideoLAN Server) and VLC (VideoLAN Client), were planned to be made. After the negotiation with the school's Director, the license of the project was agreed to change to open source (GPL). From then on, VLS has been subsumed into VLC and programmers from all over the world joined the development of the project.

Today, VLC is an open source cross-platform multimedia player and multimedia playback framework. It consists of a media player and a development library – libVLC. The player can be used with a GUI interface. It can also be used in command line mode with or without the GUI interface. The libVLC library has some API functions which can used to set up the connection between VLC and another program. Then the programmer can use VLC's functionality in his own program, for example, making a new multimedia player with a totally newly designed interface and VLC's playback functionality.

2.1.1 Using VLC Console Tool for Synchronized Playback

Now that VLC has a multimedia player, the first attempt was directly using VLC's player in command line mode without any interface.

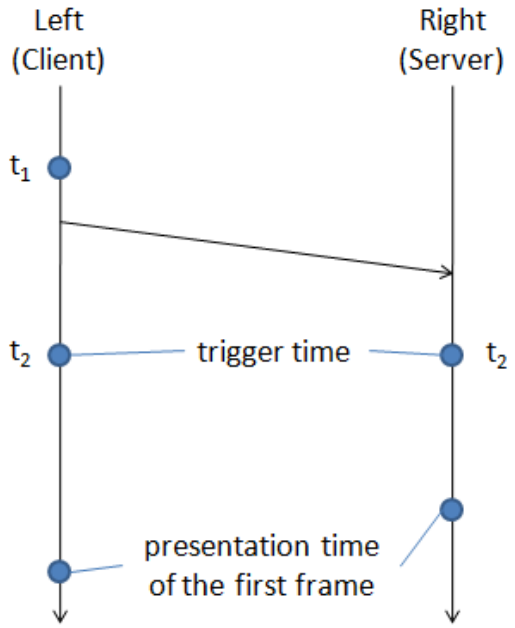


Figure 2.1 Using VLC command line tool

Figure 2.1 is the timing sequence chart of the first attempt. The two computers, 'Left' and 'Right', are now working in a client/server mode. Their clocks have already been synchronized by a time synchronization protocol. What we need to do is to make the two computers start to play at the same absolute time. In another word, synchronize the trigger times of the player applications. Strictly speaking, we are expecting that the first frame of the video clip on both computers would appear on the two screens at the same time when we make the applications start at the same time. So, next, we will make the trigger times synchronized.

The server keeps working in a listening state. The playback can only be initiated by the client. When the client plans to play the video, it firstly captures a timestamp of the current time t_1, and then adds a fixed period of time to the

just captured timestamp. Thus, it generates a new timestamp t_2 of a moment in the near future. The newly generated timestamp t_2 denotes the trigger time of the player application on both the client and the server. The client then sends the new timestamp to the server to inform when to start the application. After sending the timestamp message, the client will suspend the current thread execution for a while until t_2. At the server side, when it receives the timestamp, it will capture a local timestamp immediately. Then a comparison between the received timestamp and the local timestamp is made. The time difference between those timestamps is the period of time which the server needs to suspend its thread execution for. When the pre-set trigger time is reached, both the client and the server will stop their thread suspension and launch the VLC media player application. So far, the trigger times are synchronized.

It is worth to notice that the fixed period of time we added to the first timestamp should be chosen carefully. If it is set too long, the user would feel that the program is slow to start. And if we set it too short that is shorter than the transmission time of the timestamp message, the comparison of the timestamps at the server will have a negative value. As a result, the launch of the VLC will start at the client before the timestamp message arrives at the server and the playback will be out of sync.

Recall the definition of the response time in section 1.3. Here we are encountering the case of using a ready-made player. The response time is the period of time between the launch of the VLC media player application and the moment when the first frame of the video appear on the screen. However, the response time strongly depends on the hardware configuration and the current CPU load. Under the worst situation, a slower computer has a response time longer than a faster computer by several hundred milliseconds.

The reason for such a huge response time difference is that we are utilizing a

ready-made player; we lose the control of the application after the trigger time. The VLC player application has its own scheduling mechanism internally to initialize the player base on the current CPU load. There are a series of jobs needed to be done during initialization, such as creating a new thread for playback, loading the media file into memory, setting up a proper display format for the screen, etc. Each of the steps would have an independent unpredictable response time. The whole initialization procedure sums up all the small response times which amplifies the uncertainty of the response time. In a test we did with two different computers, we noticed this approach doesn't meet our real-time requirement.

2.1.2 Using the VLC API Functions for Synchronized Playback

By analyzing above, we know that the first approach to make a synchronized player failed due to leave too many series jobs to the system. Hence, we need to go deeper into the execution of the initialization of the application and try to control the whole initialization procedure step by step. Then the uncertainty of the response time will be reduced. Since VLC has a development library which provides external APIs for other application to use most of the VLC's features, the second attempt was to use the VLC API functions to decompose the initial procedure of the player application.

There are dozens of functions in the libVLC library. To start the playback, a few API functions need to be called to accomplish the initialization. However, what we are interested in is the actual play function. Figure 2.2 shows the timing sequence chart of the second attempt.

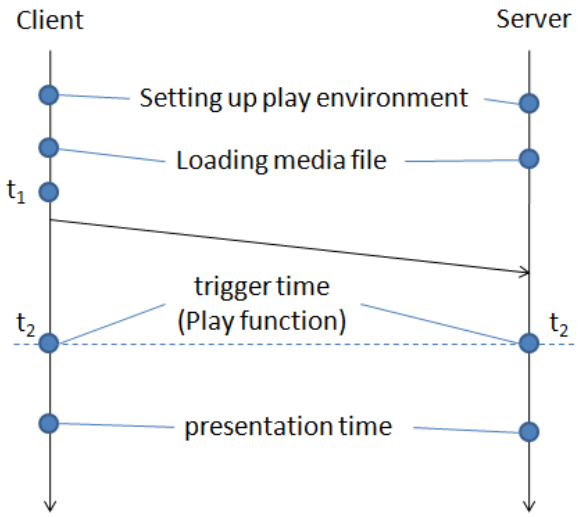


Figure 2.2 Using VLC API functions

In the figure, we can notice that is almost the same as the one using the ready-made player. The only difference is that some functions, such as setting up the playing environment, and loading the media file, have been executed before the client captures its first timestamp. Those functions were included in the initialization and executed after the trigger time in the first attempt. Here, the timestamp for the trigger time is used just before the execution of the play function. Hence, the response time is the period of time between the trigger time of the play function and the moment when the first frame appears on the screen.

Compared to the first attempt, the response time of the second attempt is only a small part of that of the first one. In theory, it should be significantly smaller. Since we are using the API functions, we measure the response time. Let's capture two measurement timestamps. One is got just before the execution of the play function, or we can use the timestamp of the trigger time instead for simplicity. The other one is captured when the play function returned from VLC. And we make the play function run for a fixed period of time. Then the time difference between the two timestamps should be a little bit longer than the running time of the play function. Thus, the exceeded part is the response time.

Figure 2.3 shows the measurement results (PDF) for the response time of repeating the initialization for 50 times. The playback duration was 1 second.

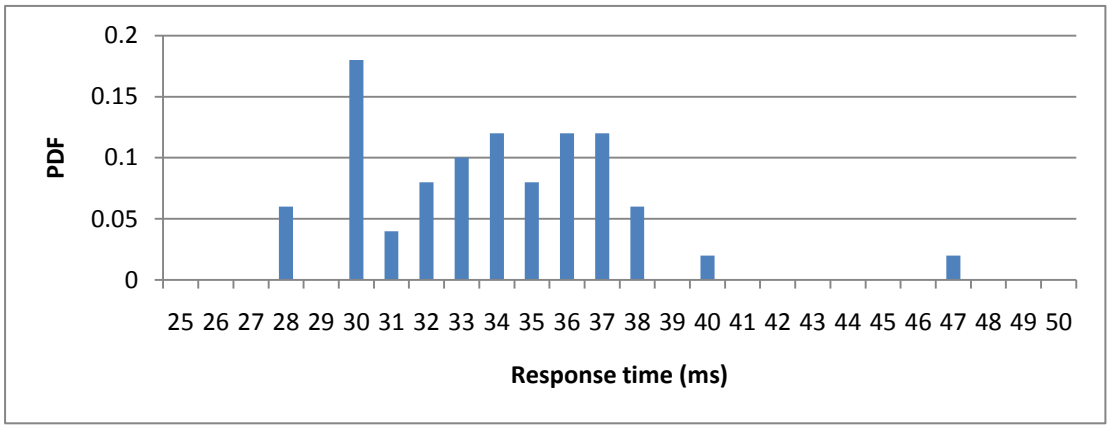


Figure 2.3 Result for response time using VLC APIs

The result shows that the response times are mainly distributed in the range between 27 ms and 41 ms already on the same system. It means they are still affected by the CPU load and other unpredictable events. Thus, we need to decompose the response time further to a frame-by-frame level and for this we need to switch from VLC to FFmpeg.

2.2 FFmpeg

Most multimedia playback systems can be considered as a layered stack structure. The following figure is a simple example.

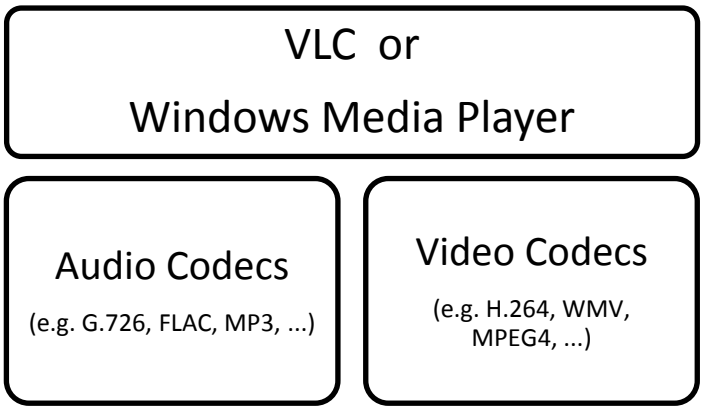


Figure 2.4 Layered playback system structure

It consists of 2 layers. The lower layer is the codec layer. It is in charge of

encoding and decoding the media streams. It is the fundamental and core function of the playback system. The upper layer is the application layer. It utilizes the service of the codec layer to build the application's own features.

According to this layered structure model, VLC works on the application layer. The best we can do in the application layer is to use the application's API functions which was our second attempt. In order to decompose the response time further, some manipulations on the codec layer is required. Inside VLC, the encoding and decoding works rely on a codec library with the name FFmpeg [3].

FFmpeg is a cross-platform open source project. It also supplies API functions, however, at the codec level. There are two main libraries in FFmpeg: libavcodec and libavformat. The former one contains the codecs and the latter contains the file format handling. These two libraries work together with several other assistance libraries.

Since FFmpeg is a codec layer library, we need to use the services it supplies to make a totally new media player application. However, to make a player with complete popular functions is a giant project. In order to implement synchronized playback on time, we will only deal with the video streams in the multimedia files and not audio. The only two functions of the player that we will implement are synchronized play and stop. So it can be called 'Synchronized Video Player'. However, the mechanism of the synchronization for audio playback is almost the same as that for video.

Some concepts with respect to multimedia data will be involved in the following discussion:[4]

Container - The multimedia file itself is a container. It contains the metadata and all the encoded media data.

Stream – Media data is stored in container in the form of streams. A

stream is a succession of encoded data element. Normally, a multimedia file has at least one video stream and one audio stream.

Frame – A data element in the stream is called one frame.

Packet – A packet is a piece of stream. Depending on the data format, a packet can be decoded into a part of a frame, a complete frame or a few complete frames.

2.2.1 Synchronized Playbacks Using FFmpeg APIs

Based on the concept above, our synchronized video player works in the following way:

```
00 INITIALIZATION
01 OPEN video_stream
02 WHILE video_stream IS NOT FINISHED
03      READ packet FROM video_stream INTO frame
04      IF frame NOT COMPLETE GOTO 02
05      SYNCHRONIZED PLAY frame
```

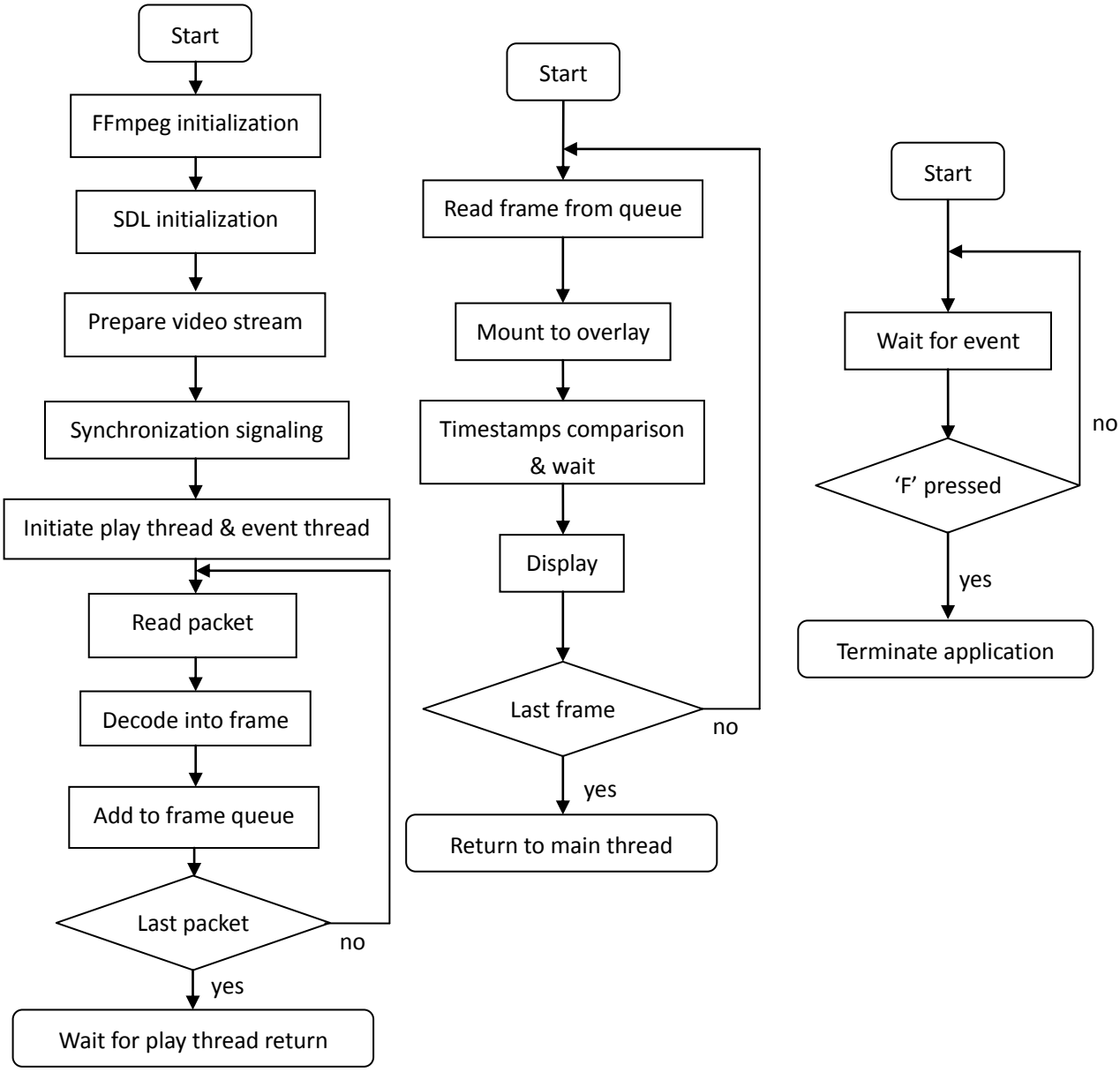
Actually, line 01 should be part of line 00. Here I separate Line 01 from line 00 since the player will only deal with the video streams. The code is written in a multithreading style. The decode function and the play function work in two independent threads since those two functions are asynchronous. But then, we prefer to use multithreading rather than use multi-process because the two functions still need to share the same memory space. The play function requires the decoded data from the decode function. If multi-process technique is used, each process has its own memory space. Then inter-process communications are required for delivering the decoded data to the play function. Obviously, it’s more complicated than making the two functions share

the same memory space. Thus, multithreading is the choice.

The speed of the playback is determined by the video format and is recorded in the metadata. (25 and 30 frames per second are typical values.) However, the speed of the decoding is determined by the computer's computational capability. It will decode packets into frames as fast as possible.

In order to display the decoded frames on screen, another library with the name SDL (Simple DirectMedia Layer) [5] is also used. It provides low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and the 2D video framebuffer. Thus, we can use it to implement the input and output interfaces of the player application. We use its video functions to display decoded frames and use its keyboard event functions to terminate the application.

Our whole synchronized player application also works in client/server mode and consists of two separated applications for the 'Left' and 'Right' computers. The general structures of the two applications are basically the same. Both of them have three threads, a decode thread, a play thread and an event handling thread.



a. decode thread b. play thread c. event thread

Figure 2.5 Flow chart of the player using FFmpeg APIs

Figure 2.5 is the flow chart for one of the applications. The only difference between the client and the server is the Synchronization Signaling part. It will be discussed later.

2.2.2 The Decode Thread

In the decode thread, which is also the main thread, all the initializations and

preparations are done. The timestamp indicated the display time of the first frame is generated. And, the packets from the video stream are decoded.

-Initialization FFmpeg and SDL

```
av_register_all();
avcodec_register_all();
SDL_Init(SDL_INIT_VIDEO | SDL_INIT_AUDIO | SDL_INIT_TIMER);
```

All the formats and codecs which are supported by FFmpeg are registered. When a media file is loaded, FFmpeg would detect its format and choose a proper codec for decoding. Then SDL is also initialized by using its own initializing function.

-Prepare video stream

```
if(av_open_input_file(&(is->pFormatCtx), filename, NULL, 0, NULL)!=0)
    printf("Couldn't open file!\n");
if(av_find_stream_info(is->pFormatCtx)<0)
    printf("Couldn't find stream information!\n");
```

First, the application opens the media file and reads the header of the file to find out the format and stream information. Then it writes the information into the pFormatCtx structure.

```
for(i=0; i<is->pFormatCtx->nb_streams; i++)
    if(is->pFormatCtx->streams[i]->codec->codec_type==CODEC_TYPE_VIDEO)
    {
        is->videoStream=i;
        break;
    }
if(is->videoStream==-1)
    return -1; // Didn't find a video stream
 is->video_st=is->pFormatCtx->streams[is->videoStream];
```

nb_streams is the number of the streams in the file including audio and video streams. The loop above checks the type of each stream to find the video

stream we need.

```
 pCodec=avcodec_find_decoder(is->video_st->codec->codec_id);
if(pCodec==NULL)
{
        fprintf(stderr, "Unsupported codec!\n");
        return -1; // Codec not found
}
// Open codec
if(avcodec_open(is->video_st->codec, pCodec)<0)
      return -1; // Could not open codec
```

Then FFmpeg tries to find a proper codec for the found video stream and open it.

```
is->timebase=av_q2d(is->video_st->time_base);
```

Another important thing about the stream is the time base. Time base is a quantity that indicates the speed of the playback. It is a rational number with the unit of second. In each frame, there is a field called PTS (Presentation Timestamp). The PTS are natural numbers, which count time bases. Thus, the product of time base and a PTS is the presentation time of the frame as an offset compared to the first frame. The function above converts the fraction form time base into a real number.

```
screen = SDL_SetVideoMode(is->video_st->codec->width,
is->video_st->codec->height, 0, 0);
```

Finally, the display size of the screen is set to match the resolution of the video stream.

-synchronization signaling

This part is different between client and server.

```
gettimeofday(&tv,NULL);
tv.tv_sec++;
n = sendto(sockfd, &tv, sizeof(tv), 0, (struct sockaddr *)&rightaddr, sizeof(rightaddr));
```

In the client, a current timestamp is captured. Then a one second lag is added to the timestamp. Thus the timestamp now indicates a moment in the near future which is the planned display time of the first frame. Finally, the timestamp is sent to the server in a UDP packet to inform the server the display time.

```
n = recvfrom(sockfd, &tv1, sizeof(tv1), 0, (struct sockaddr *)&leftaddr, &leftaddr_len);
```

In the server application, the execution is suspended and waiting for the timestamp message from the client. Once the message is received, it will continue to execute. At this point in time, both the client and the server have the same timestamp for presenting the first frame.

-Initiate the play thread and the event thread

```
SDL_CreateThread(play_thread,is);
SDL_CreateThread(event_thread,NULL);
```

Here, the other two threads are initiated. They will run in parallel and are scheduled by the CPU.

-The decode loop

A loop is used to decode packet into frames.

```
while(!av_read_frame(is->pFormatCtx, &packet))
{
    if(packet.stream_index==is->videoStream)
    {
        ...
        pts_convert(is->timebase,packet.pts,&tv_tmp);

        avcodec_decode_video(is->video_st->codec, tmp_frame->pFrame,
        &frameFinished, packet.data, packet.size);
        if(frameFinished)
        {
            timeradd(&tv,&tv_tmp,&(tmp_frame->pts));
```

```
                SDL_LockMutex(is->p_mutex);
                    while(is->qsize>=MAX_Q_SIZE)
                     {
                          SDL_UnlockMutex(is->p_mutex);
                          usleep(1000);
                          SDL_LockMutex(is->p_mutex);
                     }
                ...//Add to frame queue
                SDL_UnlockMutex(is->p_mutex);
            }
        }
        av_free_packet(&packet);
    }
```

Firstly, a packet is read from the video stream, whose return value is the enter condition of the loop. If the read packet is the last one in the stream, the loop stops. Secondly, inside the loop, the packets should be checked if it's from a video stream since the read packet may be from an audio stream. *pts_convert()* converts the natural number form packet's timestamp into a real number timestamp in the unit of second. Now, the timestamps are relative to the beginning of the playback. Then, *avcodec_decode_video()* actually decode the packets into frames. After that, the decoded frame will be added to a frame queue which is a linked list. Each element of the queue consists of a timestamp of the frame and the data of the frame. Here, the timestamp in the queue element is converted to absolute time by adding the relative timestamp and the timestamp captured in the synchronization signaling part.

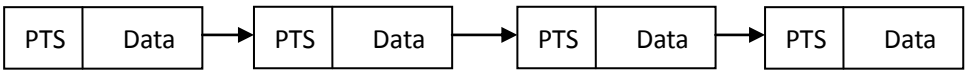


Figure 2.6 The frame queue

The frame queue is also read by the play thread. Since it is a queue, it has a FIFO data structure. The decode thread only adds elements to the rear end of the queue and the play thread only get elements from the front end. Then, there may be a risk that both the two threads use the queue simultaneously. To

avoid that, I use a mutex lock.

Assuming that the decode speed is faster than the playback speed, the length of the queue continuously grows without any limitation. The application will run out of memory eventually. So, the frame queue has a maximum length *MAX_Q_SIZE*. When the length reaches the maximum value, the decoding will be suspended for a while until the queue length is reduced below the maximum value.

The decode loop continuously executes until the all the packets of the stream is decoded or the user terminates the application.

2.2.3 The Play Thread

The play thread presents the decoded frames on the screen based on their presentation timestamps and the internal computer clock.

```
bmp=SDL_CreateYUVOverlay(is->video_st->codec->width,
                         is->video_st->codec->height,
                         SDL_YV12_OVERLAY,screen);
```

At the beginning, a SDL overlay is introduced. An overlay is a frame carrier. A decoded frame needs to be mounted to an overlay, and then it can be displayed on the screen.

```
for(;;)
{
        ...
        SDL_LockMutex(is->p_mutex);
        if(is->q_first_end==NULL) break;
        tframe=is->q_first_end->pFrame;
         tpts=is->q_first_end->pts;
        is->q_first_end=is->q_first_end->next;
        SDL_UnlockMutex(is->p_mutex);
         ...
        img_convert_ctx = sws_getContext(is->video_st->codec->width,
                                         is->video_st->codec->height,
                                         is->video_st->codec->pix_fmt,
                                         is->video_st->codec->width,
                                         is->video_st->codec->height,
```

```
                                        PIX_FMT_YUV420P,SWS_BICUBIC,
                                        NULL, NULL, NULL);
        sws_scale(img_convert_ctx,tframe->data,tframe->linesize,0,
                is->video_st->codec->height,pict.data,pict.linesize);
        sws_freeContext(img_convert_ctx);
        ...
        gettimeofday(&tv_now,NULL);
        timersub(&tpts,&tv_now,&tv_sleep);
        TIMEVAL_TO_TIMESPEC(&tv_sleep, &ts);
        nanosleep(&ts,NULL);
        SDL_DisplayYUVOverlay(bmp, &rect);
        av_free(tframe);
        ...
}
```

Then a loop is introduced for the playback. First, the thread retrieves a frame from the queue to a temporary queue element *tframe*. Second, the decoded data of *tframe* is mounted to the predefined overlay. Third, a new timestamp (*tv_now*) of current time is captured. The absolute PTS of *tframe* should indicate sometime in the near future, because the decode speed is faster than the play speed. By comparing *tv_now* and the absolute PTS of the frame, the period of time until the presentation of *tframe* is easily calculated. Then the play thread suspend for that period of time. After that, *SDL_DisplayYUVOverlay()* eventually put the decoded frame onto the screen. Finally, the temporary frame is released. If the queue has more frames to play, the loop will continue to run.

2.2.4 The Event Thread

```
for(;;)
{
    SDL_WaitEvent(&event);
    switch(event.type)
    {
    case SDL_KEYDOWN:
        if(event.key.keysym.sym==SDLK_f){
            SDL_WM_ToggleFullScreen(screen);
            kill(0,SIGINT);}
        break;
```

```
        default:
            break;
        }
}
```

There are lots of events predefined by SDL for the control of the application. Here, we only wait for *SDL_KEYDOWN* event. When the key 'f' is pressed, the application will be terminated.

2.2.5 Response Time Measurements

The description above shows how the synchronized player works. Recalling the definitions and discussions in the previous sections, we can find the trigger time and the response time for this implementation. Now, we are working in a frame-by-frame level. Hence, we have multiple trigger times and response times. The final presentation of the frames is done by the *SDL_DisplayYUVOverlay()* function. So the trigger time is the moment that the function is executed.

To accurately measure the response time is not practical. However, we can measure the time difference between the execution moment and the return moment of the function. It is a little longer than the actual response time, so it gives us the upper limit. We call it pseudo response time.

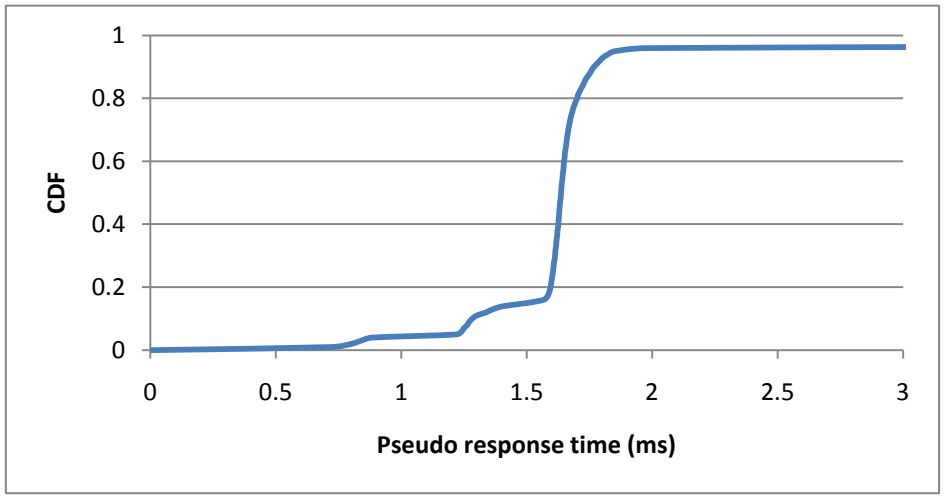


Figure 2.7 CDF of pseudo response time

Figure 2.7 is the measurement result of the pseudo response time of a video clip with 5434 video frames. Thus, there are 5434 measurement samples. The 95-percentile value is 1.85 ms. Since the precision requirement for the synchronized playback is 8 ms, this approach leaves around 6 ms to the clock synchronization protocol as the maximum synchronization error in the worst case. Thus, using FFmpeg APIs to make a synchronized player is a feasible approach.

2.3 Preparation of the Media Files

For synchronized playback in the approach of tiling, a normal media file needs to be processed by splitting the screen into several parts, for example, two parts in our project. A ready-made application named ffmpeg is available in the FFmpeg project. This application can process media file in many approaches, such as decoding and encoding through different codecs, setting aspect ratio and changing resolution. We used the following commands to process media files for our application:

```
$ffmpeg –i video.avi –cropright 640 –s 427x480 –sameq left.avi
$ffmpeg –I video.avi –cropleft 640 –s 427x480 –sameq right.avi
```

The ffmpeg application firstly decodes the input file into raw frames. Then, it encodes the decoded frames according to the user's requests to generate the output file. The meanings of the arguments are as follows:

- '-i' indicates the media file to be processed.
- 'left.avi' and 'right.avi' are the file names of the output files.
- '-cropright' and '-cropleft' indicate how to crop the screen of the input file. Our input file has a resolution of 1280x720 and we need the input file to be spitted into two parts. Therefore, we cut off the right part of the screen by 640 pixels for 'left.avi' and the left part of the screen by 640 pixels for 'right.avi' as well.

- ‘-s’ indicates the resolution of the output file. The resolution of the cropped screen is 640x720. Since our pico-projector has a resolution of 640x480 (VGA), we need to reduce the resolution of the cropped screens. To keep the aspect ratio constant, we set the resolution of the output file in 427x480.
- ‘-sameq’ tells the ffmpeg application to keep the image quality of the output file the same as the input file.

Chapter 3 Clock Synchronization

In this chapter, we will focus on the synchronization of clocks. Firstly, basic working principle and synchronization principles of computer clock will be introduced. Secondly, we will discuss the most widely used time synchronization protocol – NTP (Network Time Protocol) [6].

3.1 Computer Clocks

Inside a typical computer, time is stored in a specific register in the form of discrete timestamps. The addition operation applied to the register makes the time tick on. The number of bits the register uses is hardware dependent and determines the range of the time value and the resolution of the time. Using more bits could widen the range or increase the resolution. The origin of the time is called ‘epoch’. The epoch of our Linux-based development environment is the midnight (0 hour) of January 1, 1970.

The addition operation to the register is done in an interrupt service routine. The interrupt is triggered by a timer chip automatically. Therefore the clock advances. The value added every interrupt is called a ‘tick’.

The accuracy of a clock is the closeness between the indication of its own and that of the time reference. We use the concept of accuracy to describe the quality of a clock. However, it is impossible to find any two clocks with exactly the same frequency that makes clocks advance, since they may use different numbers of bits, different ticks, and even different oscillator frequencies in the CPUs. Since time is a cumulative quantity, a tiny frequency difference will result in a large indication difference. For example, a frequency error of 0.0012% would cause an indication error of about 1 second per day. Therefore, we use a fine measure, PPM (Part per Million) to describe the clock accuracy, i.e. the frequency error. 1 PPM is 0.001% (1E-6). In the example above, 0.0012% equals to 12 PPM. Normally, a computer clock has a frequency error of tens of

PPM.

3.2 Clock Synchronization Principles

Any clock has a finite accuracy with respect to the true time. In clock synchronization, a clock could not be more accurate than its reference clock. The purpose of a time synchronization protocol is to make a clock as close to its reference clock as possible. To make a clock synchronized with its reference clock, we need to analyze the relations between the two clocks.

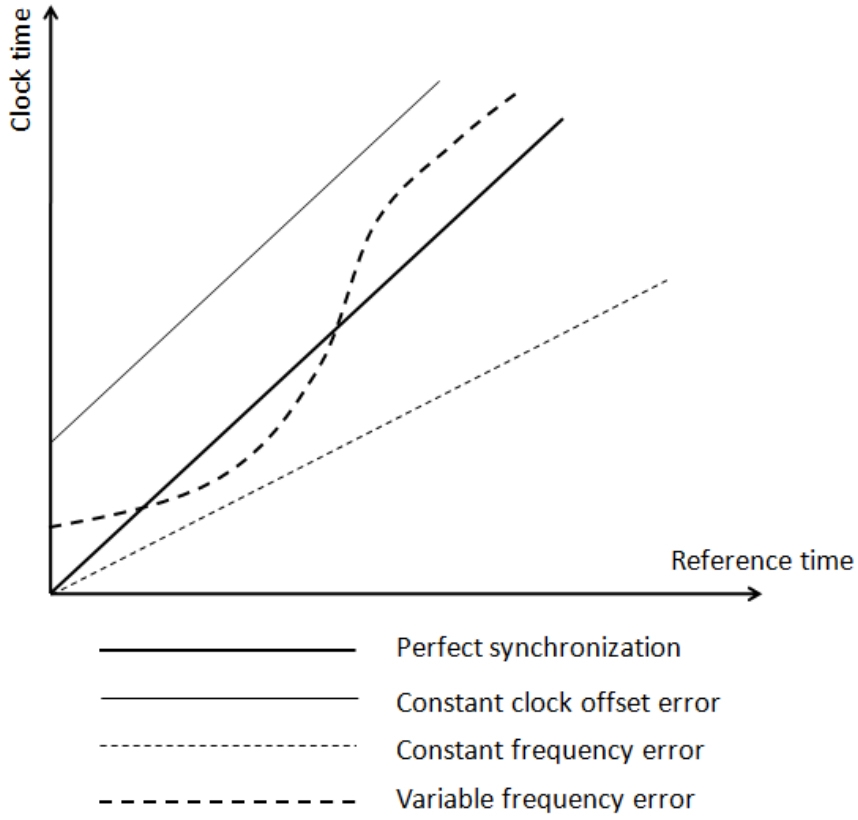


Figure 3.1 Clock relations classification [7]

Figure 3.1 classifies the relations without synchronization into several categories.

- Perfect synchronization

  This is the ideal case where the two clocks are perfectly synchronized. The line also shows the target of the synchronization protocol.

- Constant clock offset error

  Two clocks with exactly the same frequency and different indications setting result in a parallel straight line with respect to the ideal line. In this case, a step operation applied to the clock would make the clocks synchronized.

- Constant frequency error

  If the frequencies of the clocks have a constant difference, there is a constant intersection angle between the result line and the ideal line. Then the clock offset is linearly increasing. To synchronize these two clocks, both the frequency and the clock need to be reconfigured.

- Variable frequency error

  This is the common case since the frequency of an oscillator is affected by the change of temperature and other environmental effects. However, this effect is notable only in a long-term observation. In a short period of time, the variety of the frequency is tiny. Therefore, in a short-term observation, we can consider the frequency as constant.

To sum up, a time synchronization protocol uses a combination of frequency adjustment and clock offset adjustment to reconfigure the clock based on the relation between the two clocks. However, frequently using a big step to correct the clock offset error should be avoid since that can disturb some application. For example, after a huge backward step, the generation time of some newly generated files may represent some time in the future; applications may not recognize a file generated in the future. Instead, gradual adjustment with frequent small steps is a better way to correct the clock offset error which reduces the risk of making the system fall into chaos.

A time synchronization protocol should have a way to collect the information of the relations between the clocks. Obviously, that procedure would cost some

time. We call the time from a protocol starts until the clocks are in sync the start time of the protocol. It is clear that more frequent communications between the computers lead to a shorter start time.

3.3 NTP

NTP stands for Network Time Protocol and is an Internet protocol for time synchronization [6]. It is one of the oldest Internet protocol and is designed by Dave Mills of the University of Delaware.

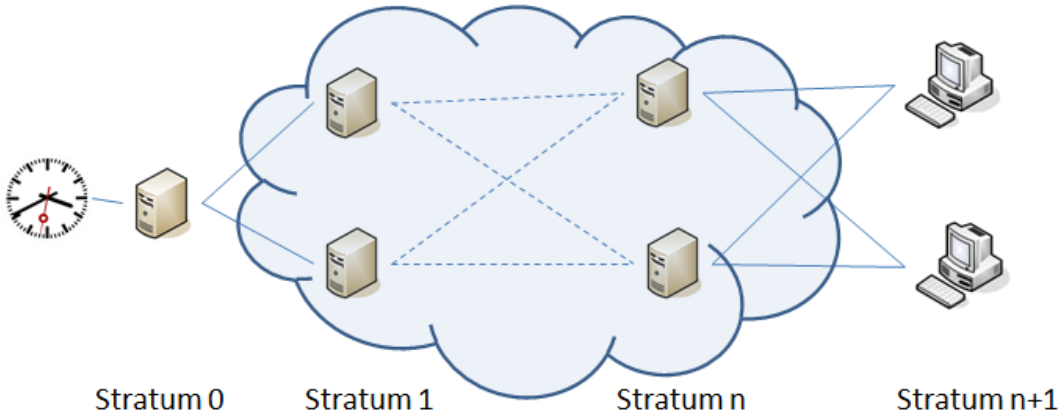


Figure 3.2 NTP time synchronization network

NTP works based on a network with layered structure (Figure 3.2). Each layer is called a stratum. A computer in stratum n uses one or few computers in stratum n-1 as its synchronization reference. Stratum 0 computers have the most accurate timing devices in the network. Normally, they have very accurate clocks, such as atomic clocks and GPS clocks.

The reference computer provides time source, therefore it is a time server. The computer which is being synchronized is the client. The synchronization consists of several packet exchanges. In each exchange, the client sends a request; then the server sends back a reply. After a few exchanges, the client would have the information about relation between the two clocks. With that information, corresponding operations (mentioned in Section 3.2) would be applied to the client's clock. Detailed descriptions of the packet exchange will

be discussed in Chapter 4.

3.3.1 The Clock Discipline Algorithm of NTP

A NTP client use the information obtained from time packet exchange to discipline its local clock. The discipline algorithm can be abstracted as in Figure 3.3,

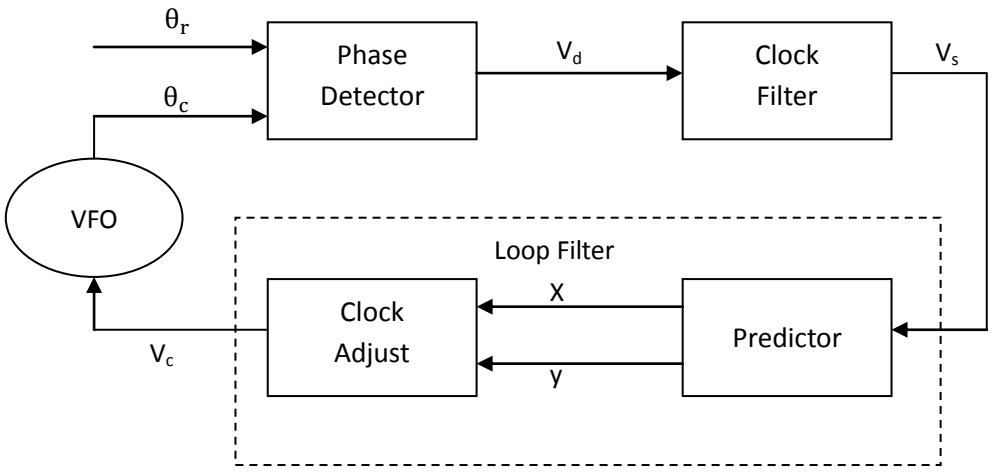


Figure 3.3 Clock discipline algorithm [8]

This is a typical PLL (Phase Lock Loop). The meaning of each module is as follows:

- Phase Detector

  Time can be considered as phase since time move on due to crystal oscillating. Therefore, θ_r is the phase of the server's clock; and θ_c is the phase of the client's local clock. The phase detector compares the two phases and output the difference V_d. The inputs of the detector are the clock times, and the output is the clock offset. Therefore, the packet exchange procedure is functioning as the phase detector.

- Clock Filter

  As mentioned above, a computer may have multiple reference clocks. Clock filter would use the clock selection; clustering and combining algorithms to combine the clock offset data to generate a more accurate

phase difference V_s.

- Loop Filter

  In the loop filter, a predictor uses V_s to generate a phase correction x and a frequency correction y for the Clock Adjust module.

$$x = V_s$$

$$y = \frac{V_s \tau}{T_c^2}$$

  where τ is packet exchange interval and T_c is called the loop time constant. As mentioned before, a huge clock time step should be avoid. Therefore, the clock adjust module generate clock corrections gradually with respect to x and y at rate $\frac{1}{T_c}$.

  The expression of y results in a transfer function [9]

$$F(s) = \frac{\omega_c^2}{s^2}\left(1 + \frac{s}{\omega_z}\right)$$

  Where $\omega_c = \frac{1}{T_c}$ and $\omega_z = \frac{1}{2T_c}$. s is the complex frequency. From elementary theory, this is a linear, time-invariant transfer function. Hence, it drives both the frequency and clock offset error to zero [8].

- VFO (Variable-frequency Oscillator)

  The VFO here represents the local clock of the client. It uses the corrections V_c which are generated by the Loop Filter to adjust its frequency and clock time.

3.3.2 Performance of NTP

NTP is designed for time synchronization over Internet and for long-term usage. Therefore, it occupies as small bandwidth as possible to exchange the time packets since it is not necessary to guarantee a short start time and that saves bandwidth. However, it has some disadvantages. We know that, the

more frequently the time packets are exchanged, the faster the clocks can synchronize. The packet exchange interval of NTP can be configured by users. Normally, the exchange interval can be set within a range from 16s to 1024s. It means, in the most frequent case, a NTP client exchanges time packets with its server every 16s. This means that it may take hours to reach synchronization.

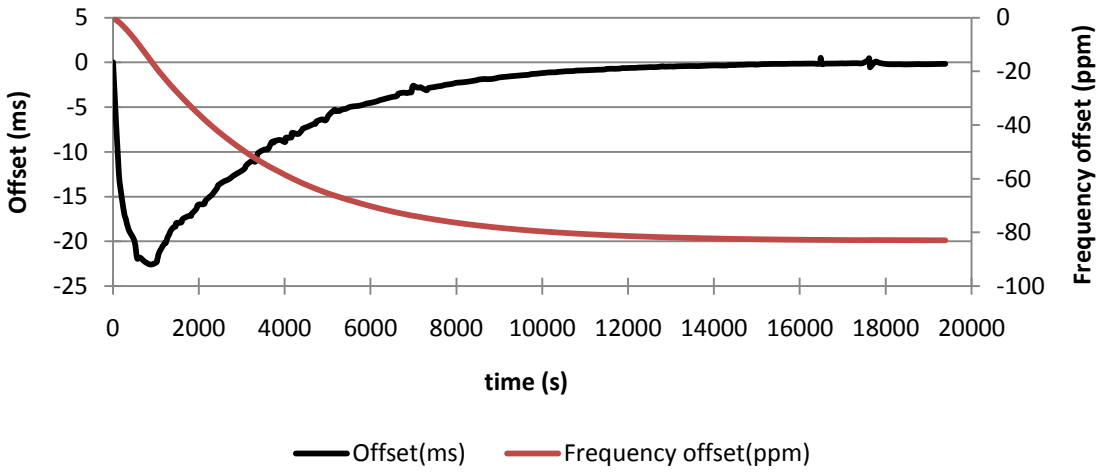


Figure 3.4 Initial run of NTP

Figure 3.4 shows a typical case of the initial run of NTP. With the help of the transfer function of the software loop filter, both the clock offset error and the frequency error is driven to zero eventually. Each clock has an own frequency origin. Thus, the frequency offset converges to around -80 ppm instead of 0 ppm. We assume that the frequency offset of the server's clock is 0 ppm. Therefore, the difference between the frequency origins is about 80 ppm.

The two clocks starts with no clock offset error but frequency error. At the very beginning, the clock offset increases significantly. That is because both the clock offset and the frequency are adjusted gradually. At the beginning, there is still a noticeable frequency difference between the two clocks. At this time, the clock offset is under adjustment as well. However, the frequency difference is too large. The clock offset adjustment cannot cope with the large frequency

difference. Thus, the clock offset continuously increases. With the frequency adjusted, the frequency difference decreased. When the frequency difference is small enough, the clock offset starts to decrease gradually.

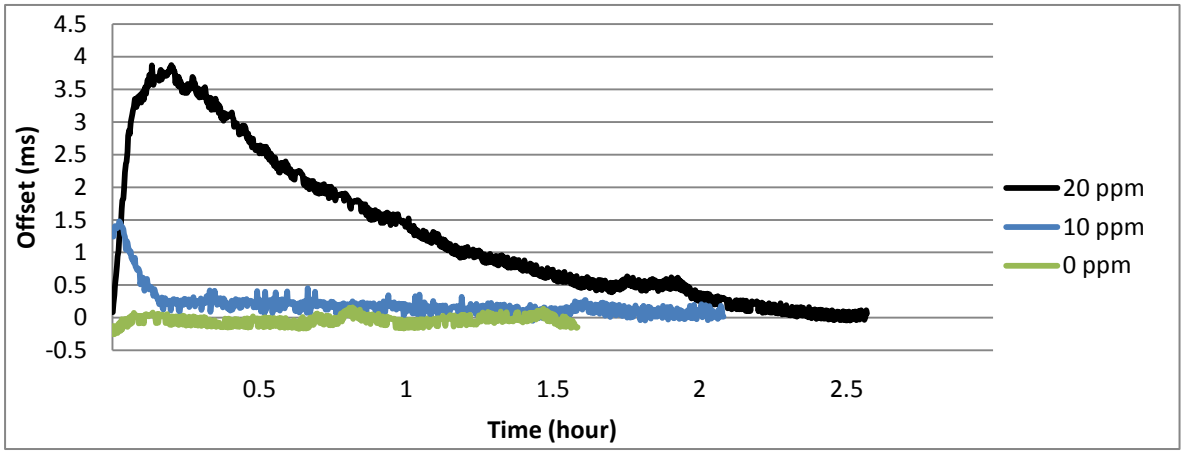


Figure 3.5 Frequency difference impact on initial run

We can notice that, the start time is mainly spent on eliminating the frequency. Figure 3.5 gives the clock offset error of the initial runs with three different initial frequency differences. Larger initial frequency difference results in a longer start time.

However, the accuracy of NTP after it reaches the steady state is great. The following table gives some statistical result for the clock offsets in Figure 3.5 after they became steady.

Table 3.1 Clock offset error statistics in steady state of NTP

Mean	Median	Variance	Standard Deviation	95-percentile
0.06 ms	0.05 ms	0.003 ms	0.05 ms	0.15 ms

To summarize, NTP has a long start time when there is a large initial frequency difference. However, thanks to its NTP's PLL algorithm, it has good synchronization accuracy in the steady state.

3.3.3 Suggestions for Improving the Performance

Our project requires a time synchronization protocol that synchronizes the clocks quickly. NTP's long start time does not meet that requirement. NTP is designed for using over Internet. However, in our project, a dedicated WiFi network is used for synchronization. Bandwidth resource is not too critical in our case. Thus, we can make an assist protocol for NTP. The assist protocol is launched before the NTP starts. It uses a larger bandwidth to exchange more time packets. Thus, it could collect the information about the relations between the clocks as soon as possible. Then a fast estimation is used to estimate the frequency difference and step the frequency of the client's clock to eliminate the frequency error. After that, NTP starts with near zero initial frequency and offset differences. Therefore, the total start time should be significantly reduced.

With the idea above and the time packet exchange mechanism of NTP, we can even design a new light weight time synchronization protocol. The new protocol could achieve quick start and accurate synchronization by more frequent time packets exchange than NTP does. The principles of the new protocol will be detailed in the next chapter.

Chapter 4 Fast Clock Adjustment Protocol

The clock synchronization precision of NTP gradually increases. So it works well in a long-term usage. However, what we need is a time synchronization protocol with quick-start and anti-interference properties. In this chapter, a Fast Clock Adjustment Protocol is introduced.

As mentioned in Section 3.3, time synchronization protocols discipline clocks according to the samples which are generated from exchanging timestamps between the server and the client.

In theory, if two clocks with different oscillating frequencies run without synchronization, the clock offset between them would drift apart linearly. So, the basic idea of the Fast Clock Adjustment Protocol is to generate some samples on the clock offset – time plane and use linear regression algorithm to estimate the drift rate and the instant clock offset. With that information, the client's clock can be adjusted.

The more samples the client has in a unit time, the more information about the server's clock it gets. We can exchange more timestamps than NTP does by reducing the exchanging interval. And then, the client could be acquainted with the server's clock as early as possible. Thus, the quick start can be realized.

4.1 Timestamp Exchange

Timestamp exchanging is the way to generate the samples on the clock offset – time plane. The server's clock is the synchronization reference for the client. The client could use the samples to trace the server's clock.

4.1.1 PCAP (Packet Capture)

The PCAP is a packet capture library for unix and Linux [10]. It is able to capture any type of packet on the network and provide related information about the packet and the capturing such as the capture time. We use its API

functions to capture the synchronization messages and use the capture time as the timestamps.

4.1.2 Timestamp Exchange

The timestamp exchanging procedure is divided into rounds. After each round, a sample is generated at the client by using the information collected during the last round. The samples will be used to estimate the feature of the server's clock later.

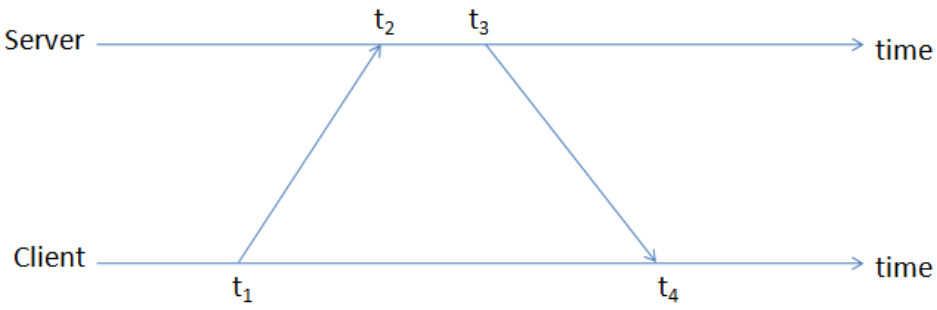


Figure 4.1 A timestamp exchange round

Figure 4.1 shows the process of a round. The client sends a packet to the server. At both sides, PCAP captures the packet and records the capturing times. After that, the server sends another packet back to the client as a response. PACP records the times as well. Thus, we have four timestamps. They can be used to calculate the instant clock offset as follows:

$$\begin{cases} t_2 - t_1 = \theta + \delta_1 \\ t_3 - t_4 = \theta - \delta_2 \end{cases}$$

where θ is the clock offset and δ is the packet transmission delay. It is the absolute time between the two capturing of a packet transmission. If the two transmissions have the same delay, i.e. $\delta_1 = \delta_2$, they are symmetric. So, the clock offset is expressed as,

$$\theta = \frac{t_2 - t_1 + t_3 - t_4}{2}$$

However, the forward and the backward transmissions are not always symmetric due to different delays. So we need the information about the delay as well to distinguish the reliability of the samples.

$$\delta = \frac{t_2 - t_1 - t_3 + t_4}{2}$$

So far, we have all the information to generate a sample. A sample is like follows:

time	offset	delay

The offset field and the delay field are calculated above. The time field can be filled using the time of any of the four timestamps. This will introduce some error. But the error can be ignored since each round of timestamp exchanging is accomplished within a short period of time.

4.2 Samples Read-out Technique

The four timestamps generated during the forward and backward packet transmissions are distributed at the client and the server. They need to be gathered at the client to generate the sample.

The packets have two missions. One mission is to be transmitted as a probe to be captured by the PCAP to generate the timestamps. Another mission is carrying the request information from the client to the server and fetching the captured timestamps at the server back to the client.

The packets are UDP packets and the formats are the same for all transmissions:

Int	int	struct Timestamp	struct Timestamp
direction	seqnum	timestamp A	timestamp B

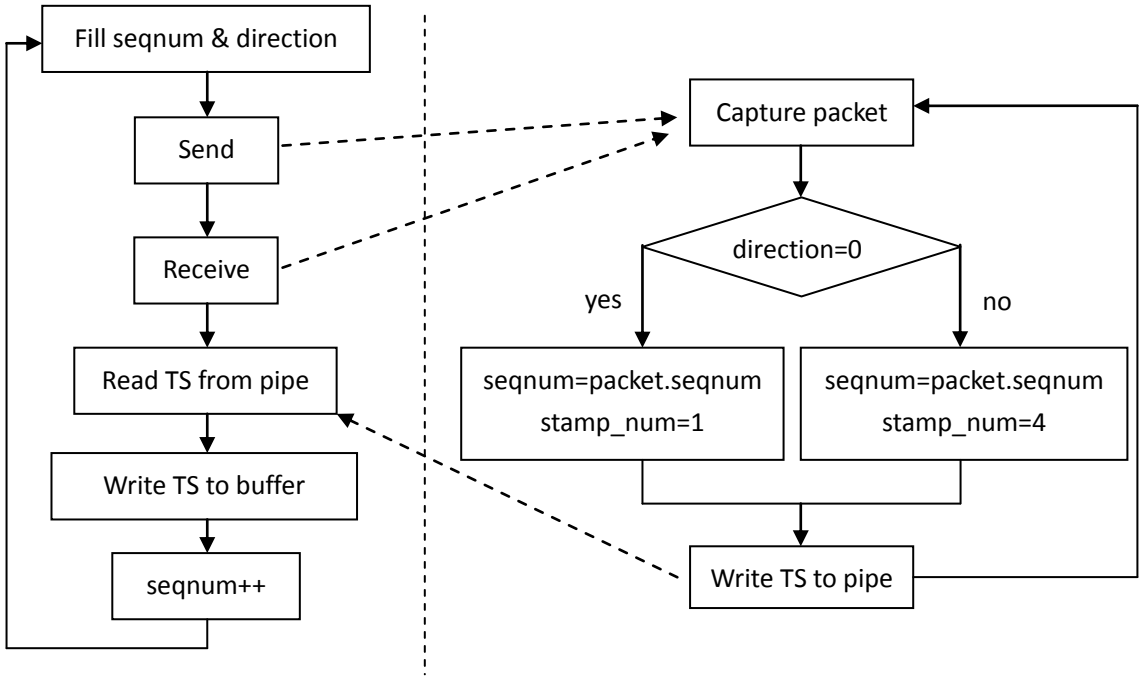
The *direction* field is an integer. It indicates whether the packet is a request packet or a response packet. The *seqnum* field is also an integer. It gives the sequence number of the round. The two timestamps belongs to a custom structure Timestamp. The definition is as follows,

```
struct Timestamp
{
    int seqnum;
    int stamp_num;
    struct timeval tv;
}
```

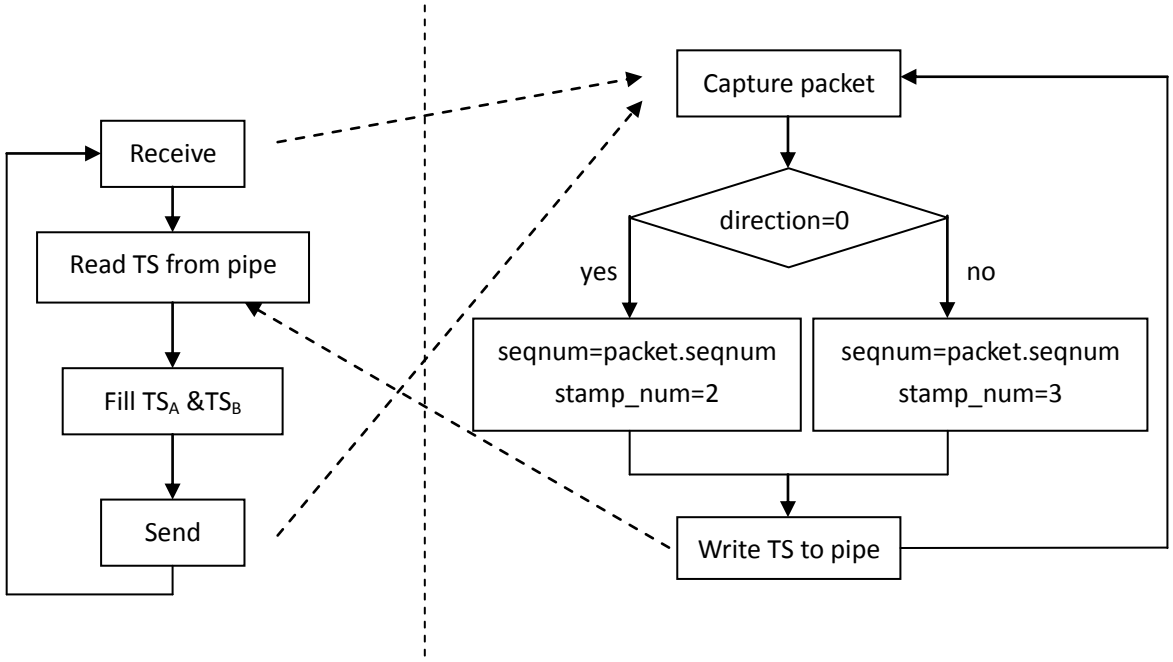
In Timestamp, *seqnum* is also the sequence number of the round. *stamp_num* indicates which of the four timestamp it is since there are four timestamps generated in one round. *tv* is the time data of the timestamp.

Figure 4.2 is the flow chart that shows how the timestamps are gathered at the client. On both sides, the packet transmission and the packet capturing are running in two processes. The IPC (Inter-process communication) is achieved by using unix pipes.

When the client transmits a packet, only the first two fields are used. The server needs to be informed the round sequence number and the packet capturing process needs the type of the packet and the round sequence number to fill *stamp_num* and *seqnum*. The two timestamp fields are filled with two empty Timestamp structures.



a) Client



b) Server

Figure 4.2 Timestamps gathering

On the server side, all the four fields are filled. The functions of the first two fields are the same as the client side. The latter two fields are used to carry the timestamps captured in the last round back to the client.

In the main process, i.e. the packet transmission process, after receiving from

the server and reading from the pipe, all the timestamps are gathered in a sample buffer. The buffer has a format like,

Sequence Number	1	2	3	4	…	99	100
T1	101	1108	2110				
T2	5004	6007	7008				
T3	5009	6016					
T4	112	1116	2122				

The buffer can hold the timestamps of the last 100 rounds and is used circularly. The four timestamps within the same round sequence number cannot be gathered to the buffer in the same round. The table above is an example of pattern shows that. T1 and T4 are captured at the client, so they can be moved to the buffer within the current round. T2 is captured at the server before the server sends the response packet. Normally, T2 can be gathered within the same round as well. However, T3 is captured after the server sent the response packet, so T3 can reach the buffer in the next round at the earliest. T2 also has the risk of reaching the buffer after the current round if it has not been written to the pipe before the server sends the packet. Anyway, the application on the server would read the captured timestamps in turn from the pipe and send two timestamps back to the client. When all of the four timestamps with the same sequence number are gathered, they can be used to calculate a sample. However, the sequence number they have represents several rounds ago. That is the reason for using a buffer to temporarily store the timestamps.

4.3 Adjusting the Clock Using Linear Regression

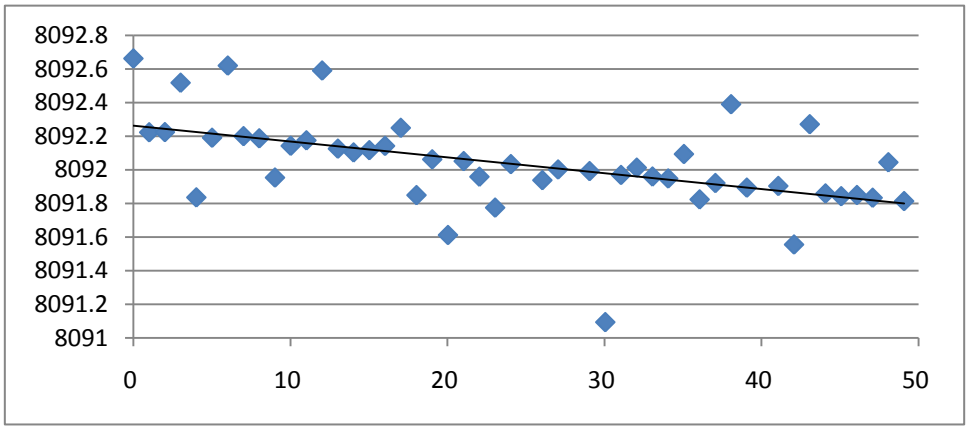
With the samples, we can estimate their trend line one the clock offset – time plane. Since we can assume that the clock offset drifts apart linearly, the trend line should be a straight line, so we can use linear regression to estimate the line. The trend line has the form of $y = \hat{\beta}x + \hat{\alpha}$, where the y-axis represents the clock offset and the x-axis represents the time. The formula for the estimation is,

$$\hat{\beta} = \frac{\sum_{i=1}^{n}(t_i - \bar{t})(\theta_i - \bar{\theta})}{\sum_{i=1}^{n}(t_i - \bar{t})^2}$$

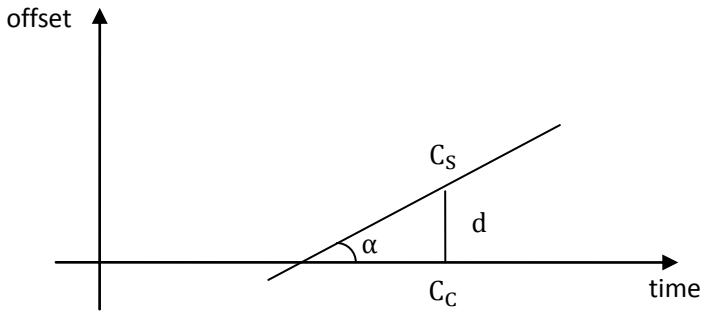
$$\hat{\alpha} = \bar{\theta} - \bar{t}\hat{\beta}$$

where $\hat{\beta}$ is the estimation of the slope, $\hat{\alpha}$ is the estimation of the intercept, t_i is the time of the i^{th} sample and θ_i is the offset of the i^{th} sample. The estimated line is updated every 10 samples in order to trace the variation of the transmission environment.

Now we can use the trend line to adjust the client's clock. The operations we can apply to the clock are to set the clock's indication and to change its running frequency by using some system calls.



(a)



(b)

Figure 4.3 Linear regression example

Figure 4.3 (a) is an example of the estimated trend line and (b) is the abstraction of (a). We can obtain two aspects of information from the curve.

- The instant offset. Point C_C represents the current time on the client; point C_S represents the current time on the server. C_C is the projection of C_S on the time axis. So the distance d between them is the current clock offset.
- The clock frequency difference. The indications of the two clocks drift apart because there is a constant frequency difference. So the estimated slope ($\hat{\beta} = \tan\alpha$) reflects the frequency difference.

To synchronize the clock, firstly, we step the clock forward or backward according to the current offset. See the flow chart in Figure 4.4.

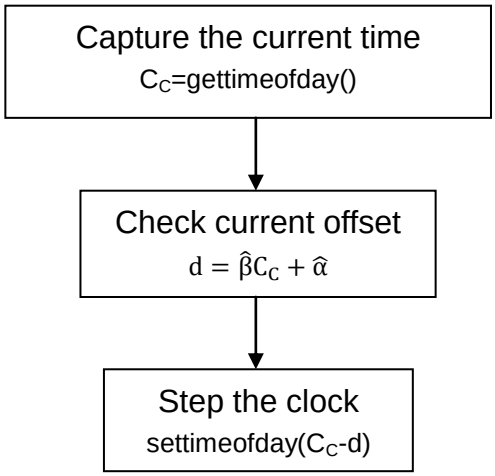


Figure 4.4 Adjusting the clock offset

Secondly, we adjust the frequency to make it the same as the server. The system call *adjtimex()* is the tool. As mentioned before, the frequency is presented in the unit of ppm. *adjtimex()* uses a scaled ppm as the unit for the frequency. And it has a tolerance value for the frequency adjustment to limit the frequency changing range. The value is 500 ppm and can be read by *adjtimex()* in the unit of scaled ppm. On the offset – time plane, both the axes are in the unit of millisecond. So, the variation of the frequency should be,

$$\Delta f[\text{scaled ppm}] = \frac{\text{tolerance}[\text{scaled ppm}] * \hat{\beta} * 1000000}{500[\text{ppm}]}$$

After the offset and frequency adjustment, the two clocks have the same frequency and no offset. Thus, they are synchronized. However, it is unnecessary to adjust the offset and the frequency at the same time. For the purpose of synchronized playback that the clocks indicate the same time is more important. Hence, we can apply the offset adjustment more frequently than the frequency adjustment, which can increase the efficiency of the protocol to some extent. Actually, I set a threshold of 1 ms for the clock offset. In each timestamps exchange round, the client checks the offset. If the offset exceeds the threshold, the offset adjustment is applied. The frequency adjustment is applied only when the sample buffer is full (after 100 samples).

4.4 Utilization of Old Samples

After the clock adjustment, the running features of the client's clock are changed. The old samples in the buffer cannot be used for the estimation any more. However, to increase the efficiency of the protocol, we can change the old samples' positions as well according to the adjustments. The target is to make the old and the new lines collinear.

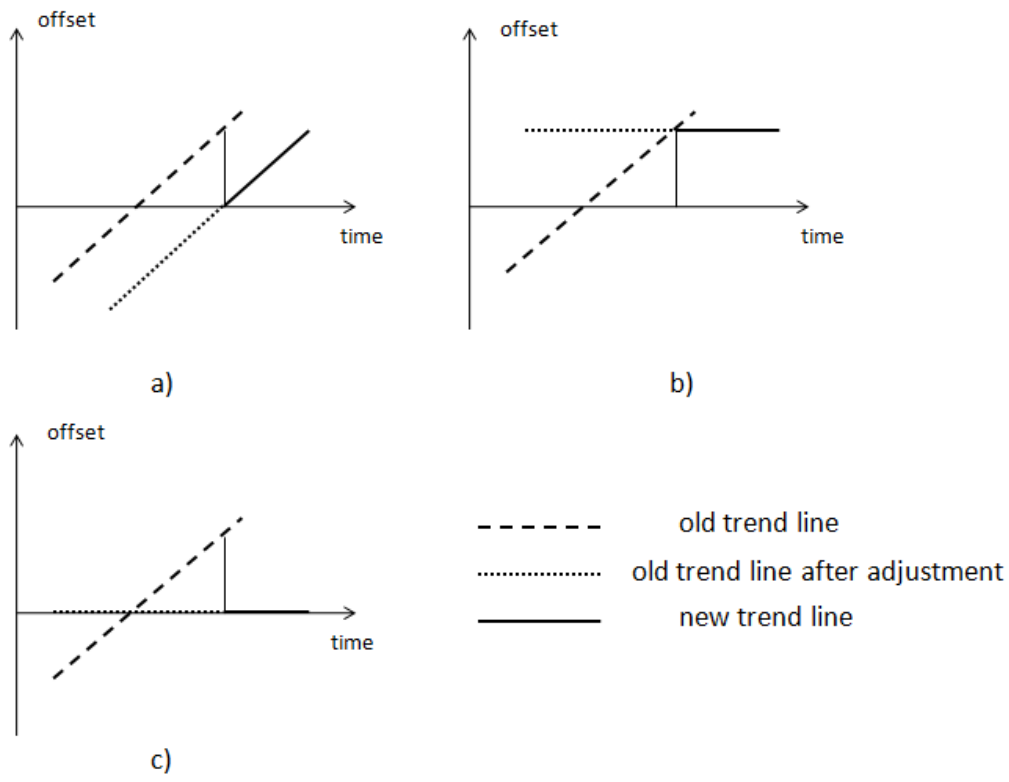


Figure 4.5

The Figure 4.5 shows three cases after the clock adjustment. For clarity, we only show the trend lines without the samples.

a) Applying only offset adjustment.

   After the offset adjustment, the new trend line will be parallel to the old line, however, with different intercepts. Applying the same offset value to the old samples as just used in the adjustment makes the two lines collinear.

b) Applying only frequency adjustment.

The frequency adjustment introduces an intersection angle between the two lines. Rotating the old line by $\tan^{-1}\hat{\beta}$ makes it collinear with the new line.

c) Applying both adjustments.

This case is the combination of the previous two. To make the two lines collinear, firstly, we rotate the old one by $\tan^{-1}\hat{\beta}$; secondly, the offset value just used is applied to the rotated line.

4.5 Error Analysis

All the discussions in the previous sections are based on the assumption that the two packet transmissions have the same delay, i.e. $\delta_1 = \delta_2$. That is the case when there is no interference during the transmission and it works well. However, we cannot guarantee that the transmission environment is always idle. Especially, when the video is streamed to the computers, there should be some interference in the environment. So we need to take that into account.

When $\delta_1 \neq \delta_2$, the expression of the clock offset becomes,

$$\theta = \frac{t_2 - t_1 + t_3 - t_4 + \delta_1 - \delta_2}{2}$$

So the offset has an error of $\frac{\delta_1 - \delta_2}{2}$. With that error, the samples are more dispersive on the offset – time plane. That introduces more estimation error of the trend line. The ideal approach to eliminate the error is to find the asymmetry, i.e. the value of δ_1 and δ_2. Unfortunately, that approach is infeasible. What we can obtain from the four timestamps is just the sum of them,

$$\delta = \frac{t_2 - t_1 - t_3 + t_4}{2} = \frac{\delta_1 + \delta_2}{2}$$

Now we try to minimize the error. We know that,

$$\begin{cases} \delta_1 > 0 \\ \delta_2 > 0 \\ \delta_1 + \delta_2 > |\delta_1 - \delta_2| \end{cases}$$

Hence, if $\frac{\delta_1+\delta_2}{2}$ is small enough, the impact of $\frac{\delta_1-\delta_2}{2}$ on the offset can be ignored. That is the reason why we introduce a delay field to the sample structure. Generally speaking, a sample with a small delay value is considered as a good sample; a sample with a large delay value is a poor one. If the poor samples are removed, then the error of the offset is minimized and then the estimation error is also minimized. Figure 4.6 shows a comparison between the estimations of the same sample set with and without removing the poor samples.

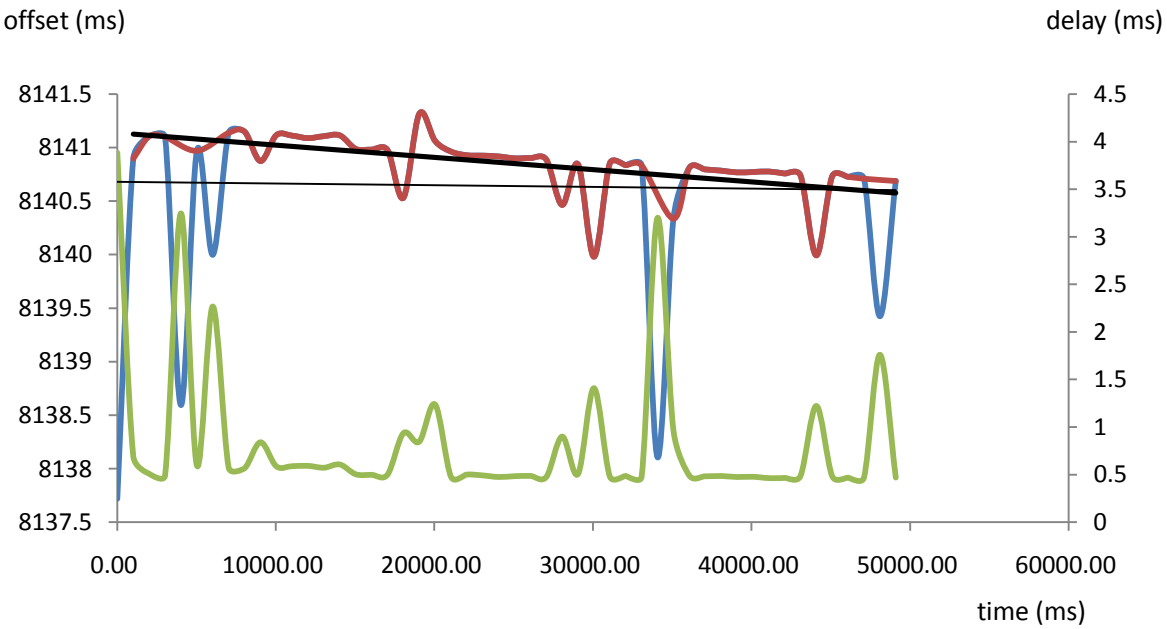


Figure 4.6

(blue: complete sample set, red: sample set without poor samples, thin black: trend line of blue, thick black: trend line of red, green: corresponding delay $\frac{\delta_1+\delta_2}{2}$)

4.5.1 The Valid Samples Filtering Threshold

Now we need a criterion to select good samples. Figure 4.7 shows a typical delay distribution with a strong interference.

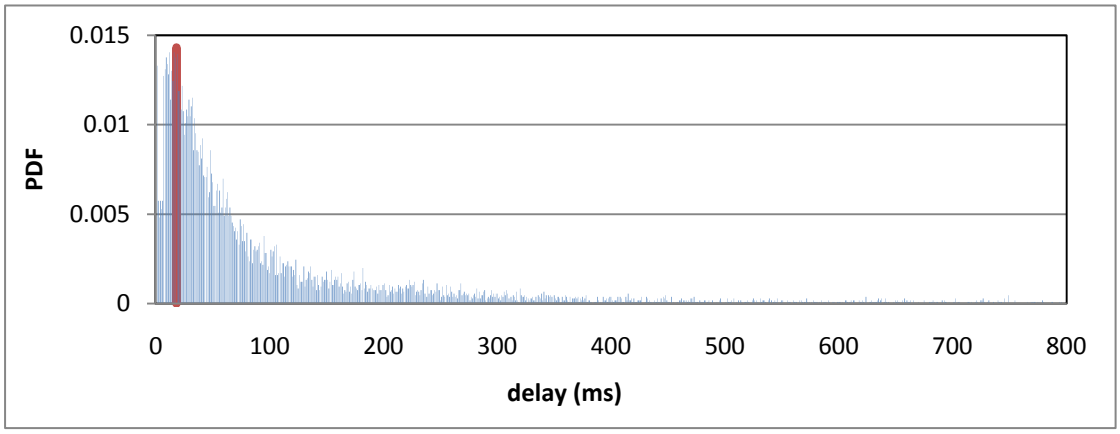


Figure 4.7 Delay distribution

The distribution always has a long tail. We use the 20-percentile of the delay (the red line in Figure 4.7) as the threshold to select samples. All the samples with a delay value smaller than the threshold is considered valid. The threshold is updated periodically together with the update of the estimated line to trace the variation of the channel. See Figure 4.8.

We only use the valid samples to estimate the trend line. Thus the impact of the asymmetric packet transmission is minimized.

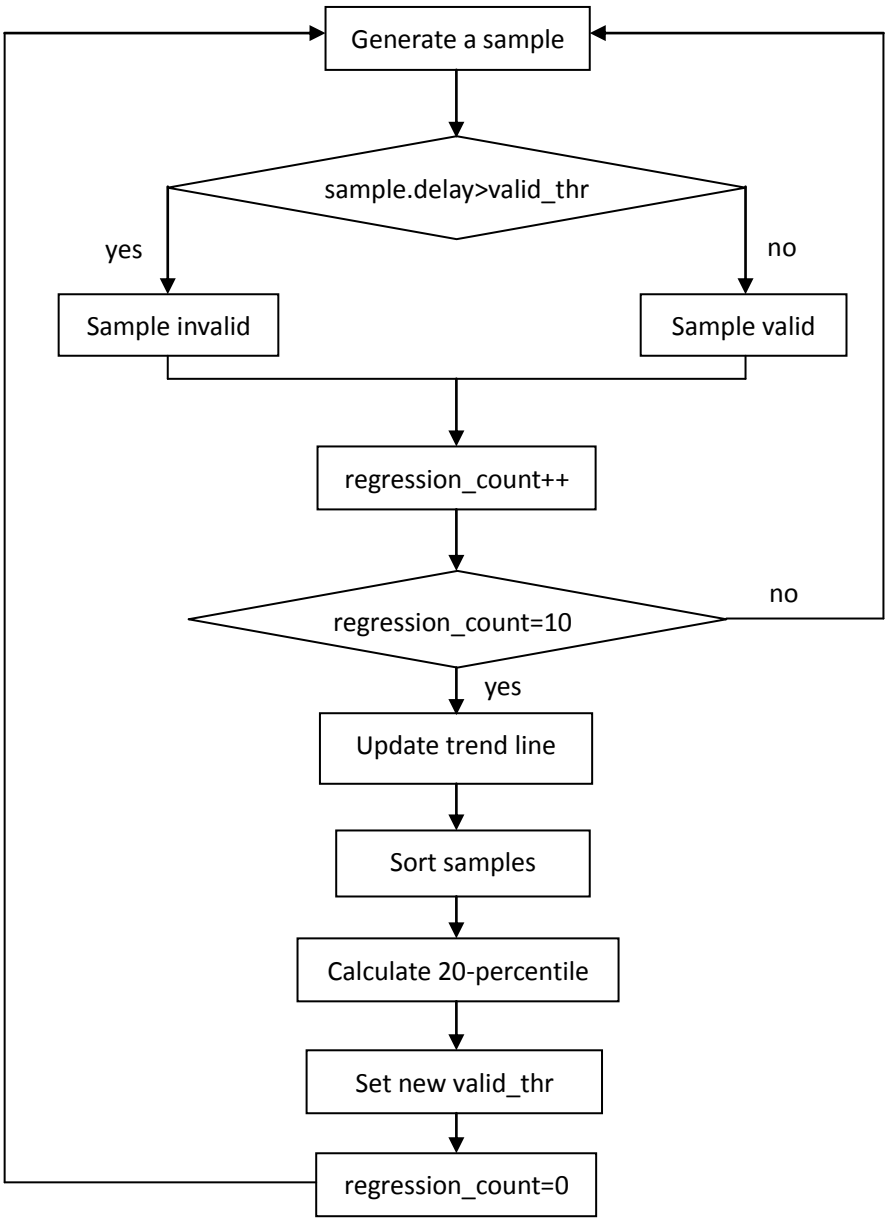


Figure 4.8 Update the valid threshold

4.5.2 Quick Start

In the initial phase of the protocol, the client needs to collect enough samples for the first estimation. Originally, the packet transmission rate of the client is 1 packet per second. It means the client could obtain 1 sample per second on average which is a relative high rate for time synchronization protocol. Since the usage of the bandwidth does not matter in our dedicated WiFi network, we make the client collect enough samples for the initial estimation in a short period of time by increasing the sample rate. The quick start is achieved.

However, if there is interference during the transmissions, the samples become unreliable. Hence, introducing a sample selecting mechanism to the initial phase is still necessary. However, in the initial phase, the number of samples in the buffer is quite few. Sorting the samples with respect to the delays and using the 20-percentile of the delay to filter the samples are not reasonable. Instead, the samples are selected in the following way in the initial phase (see Figure 4.9).

At the beginning, the client always uses 1.5 times the minimal delay it has encountered so far as the valid threshold and counts the number of the valid samples. When there are enough valid samples, the initial phase is finished and the client executes the first estimation. In an environment without or with light interference, that can be achieved quickly.

In an environment with strong interference, the delay values are dispersed. The client is unable to collect enough valid samples for a long time. Hence, it is possible that sample buffer becomes full (100 samples). When that happens, the samples are sorted and filtered by the 20-percentile delay value. This sets an upper bound for the length of the initial phase. Thus the quick start property is guaranteed even when there is severe interference.

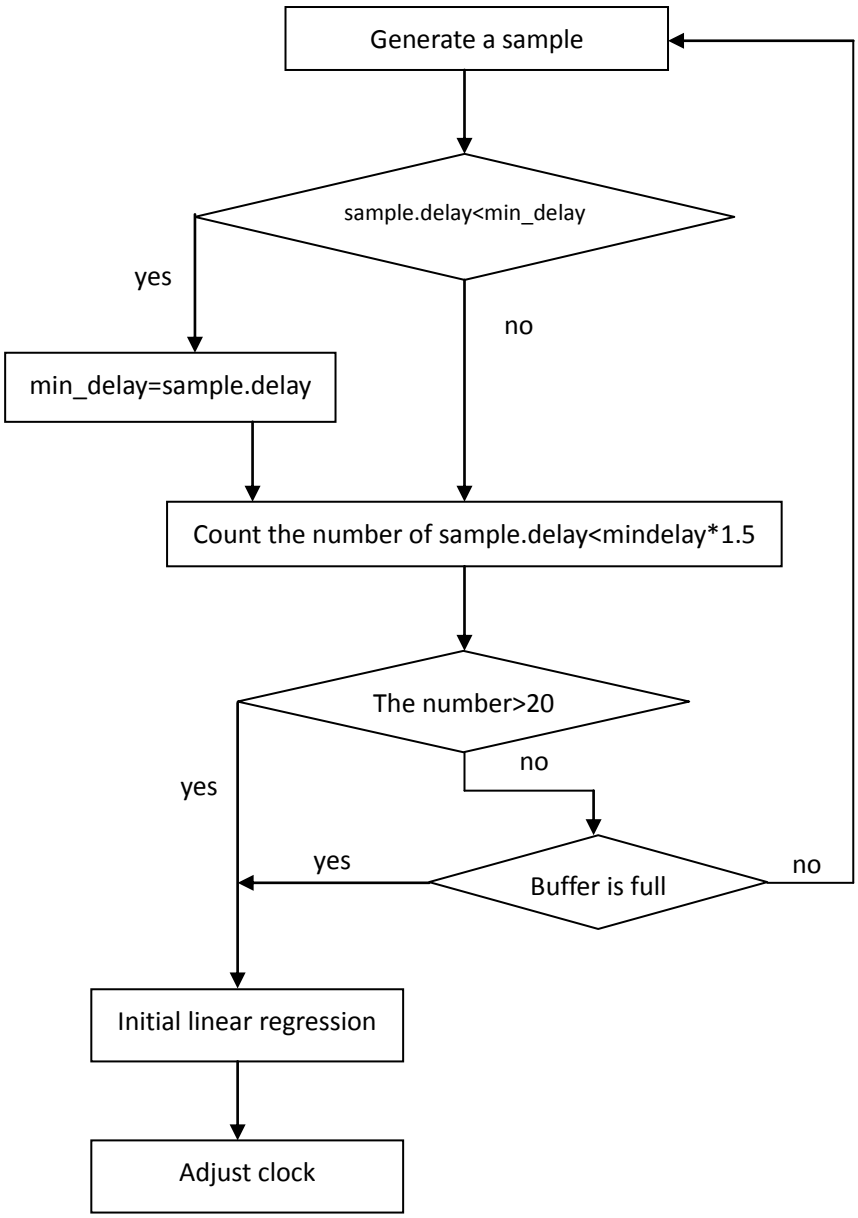


Figure 4.9 Sample filtering in initial phase

4.5.3 Anti-interference (Approaches for Increasing Precision)

The interference makes the samples disperse on the offset – time plane. The samples are distributed within a band on the plane and the interference increases the width of the band.

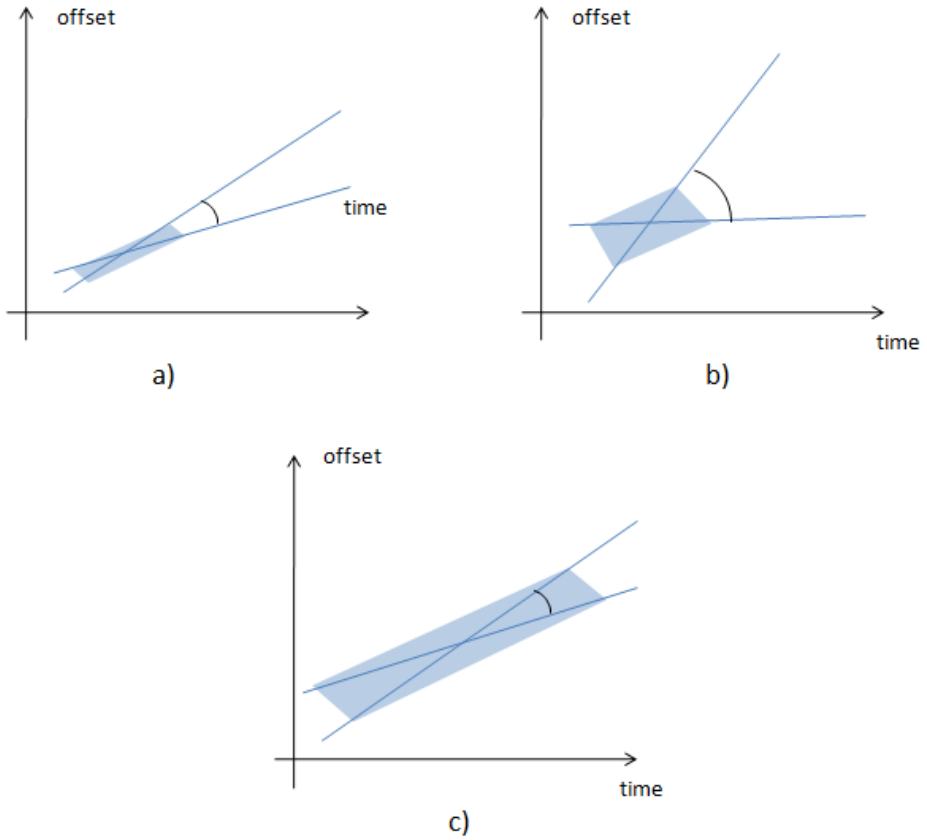


Figure 4.10 Anti-interference

Figure 4.10 shows how a wider band can affect the accuracy of the regression line. From the figure we can see that, the regression line reflects the trend of the sample band. With a narrower band (4.10a), the estimated slope is limited within a smaller range. However, with a wider band (4.10b), the range for the estimated slope is increased. The line could rotate in a wider range. Then the estimation error of the slope is increased. If the length of the band can be increased (see 4.10c), the possible range for the regression line to rotate will be reduced to the same level as a narrow band has. Then the estimation error is reduced.

To increase the length of the band means to use more samples for the estimation. Obviously, a larger buffer is needed and there are two approaches to fill the larger buffer:

1) Using a longer observation time.

The buffer size was 100 samples originally. For instance, we can use a larger buffer with 400 samples. If the transmission rate is kept at 1 packet per second, 400 seconds is needed to fill the whole buffer. To fulfill the quick start requirement, the first estimation is done when we get the first 100 samples. So the precision gradually increases after the first estimation. When the whole buffer is full, the precision reaches the highest level. By using this way, the precision of the synchronization is a little bit poor at the beginning. The advantage is the saving of bandwidth resources.

2) Increasing the packet transmission rate

Another option is to increase the transmission rate. For instance, we can use a rate of 4 packets per second to generate the samples. Then, the initial estimation can be done within the same period of time as the original version. The advantage of this approach is that the precision reaches the highest level after the first estimation, however, at the cost of utilizing more bandwidth resources.

In our final version of the protocol implementation, we use a buffer with the size of 400 samples and we name it the main buffer. Besides, we introduce a secondary buffer with the size of 400 samples as well to increase the observation time in a more efficient way. The generated samples are filled into the main buffer. Whenever the last position of the main buffer is filled, we sort all the samples in the main buffer with respect to the delays and copy the best 40 samples into the secondary buffer. The packet transmission rate is 4 samples per second. Therefore, the samples in the whole buffer cover an observation time of $\frac{400}{4} * \frac{400}{40} = 1000$ seconds.

Chapter 5 Measurement Results

In this chapter, the measurement results of the Fast Clock Adjustment Protocol and the comparison with NTP are presented. Firstly, the tools used to measure the performance of the time synchronization protocols are described.

5.1 Test Environment Setup

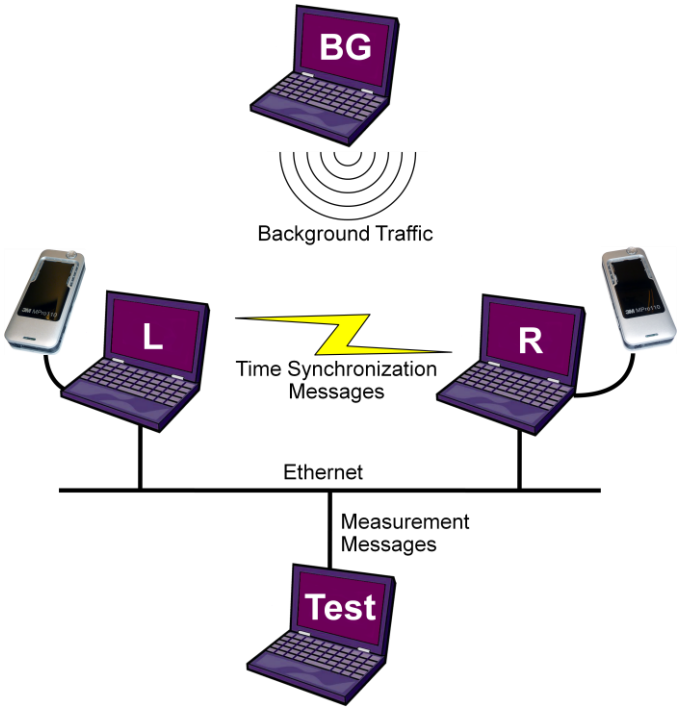


Figure 5.1 The test environment

Figure 5.1 shows the basic configuration of the test environment of this project. Since the pico-projectors we currently have are the 1st generation products, they are just tiny conventional projectors. Therefore, two computers are used as the data processing part and communication part representing future devices with embedded pico-projectors. We name the two computers 'Left' and 'Right' as indicated in the figure. The two computers set up an ad-hoc Wi-Fi network in order to exchange time synchronization messages and synchronized playback signaling messages. The test video clips are stored

locally in the two computers.

A third computer (BG) is on the same Wi-Fi network and is introduced as an interference source to simulate background traffic. The interference source continuously generated background traffic in different intensities. The purpose of intentionally introducing background traffic is that we need the time synchronization protocol to perform well not only in an interference free environment, but also in a complex radio propagation environment with interfering nodes.

Finally, a fourth computer (Test) is used as the test-bed controller for testing the time synchronization protocols. Specifically speaking, the test-bed controller measured the time difference, i.e. the clock offset, between the clocks from an onlooker's point of view. The test-bed controller basically is working based on capturing and comparing timestamps. The delay of a message between either Left or Right and the test-bed controller should be very small in order to achieve a high precision measurement. Thus, the measurement messages are exchanged via a dedicated Ethernet due to its delay is rather small and predictable. The principles of the generation of background traffic and the test-bed will be detailed in Chapter 5.

Since the interference source is using its WLAN interface to generate background traffic and the test-bed is using its wired Ethernet interface to measure the time difference, the two jobs above can be undertaken by the same computer via its different network interfaces. So, in practice, three computers and two pico-projectors are made use of during the testing.

5.2 Test-bed

To test the performance of time synchronization protocols, a test-bed is required. The test-bed should have the ability of measuring the time difference between clocks. It should take timestamps of the clocks respectively at the same absolute time. Since the PCAP library [10] can provide the capture time

of a received packet with a very high accuracy, we use it as the tool to build the test-bed.

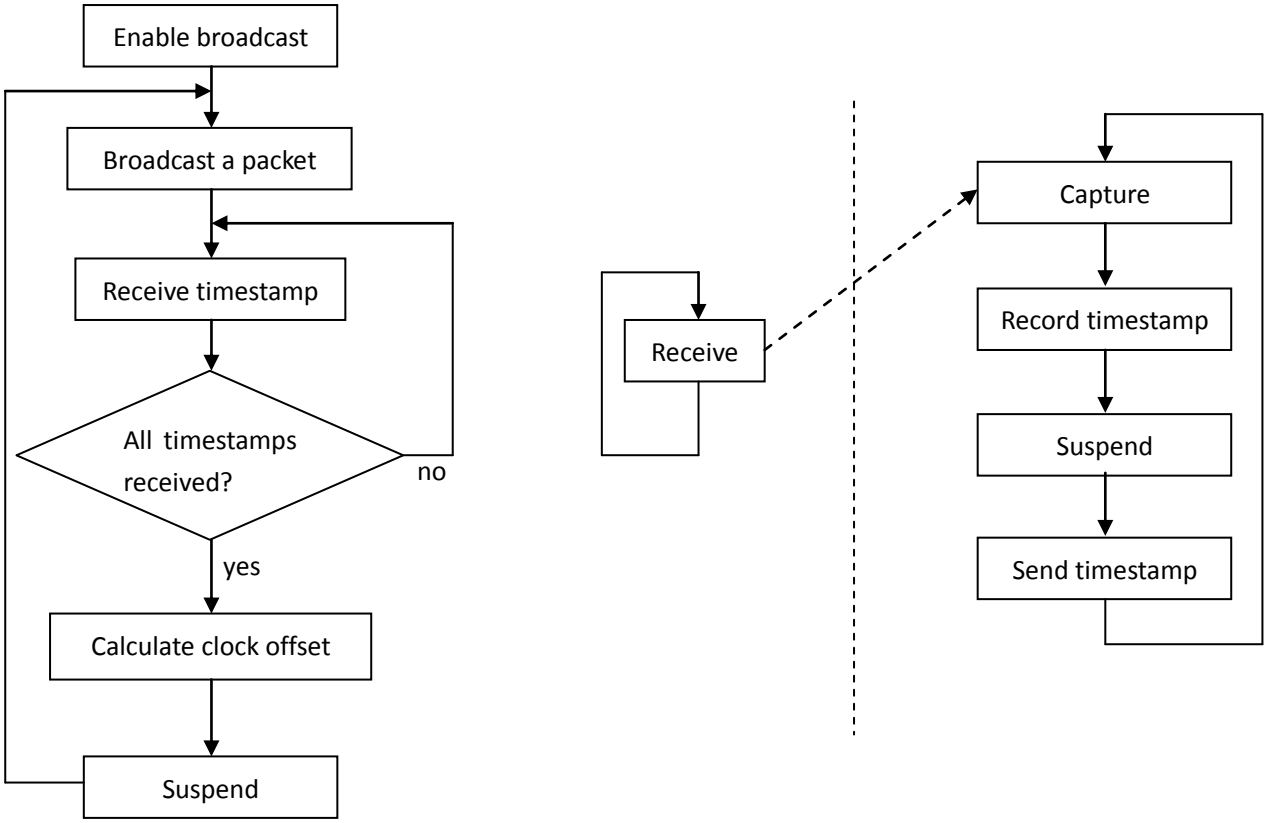
The application of Figure 5.2 (a) is the test-bed server application, which runs on ‘Test’; and that of Figure 5.2 (b) is the client application runs on all computers being tested, such as ‘Left’ and ‘Right’.

The test-bed server application continuously sends broadcast message to the Ethernet link. The reception of a packet does not have any unpredictable delays, such as carrier sense or retransmissions. The received packet is handed over to upper layer immediately where PCAP captures packets in the MAC layer. A packet’s capture time is therefore very close to the real reception time. Besides, the propagation delay of Ethernet is tiny and deterministic. Therefore, using broadcast packets over Ethernet guarantees that the packets capturing at different computers are taken place at almost the same absolute time. After the transmission of a broadcast packet, the test-bed server waits for the test-bed clients to send back the captured timestamps.

A test-bed client application is a multi-process application. In the main process, the application continuously receives the broadcast packets from the test-bed server application. When a test-bed client application receives the broadcast packet, PCAP which works in the other process captures the packet and records the capture time. Then, the capturing process will be suspended for a predefined period of time. After the suspension, the timestamps are sent to the server. Each client is assigned a unique integer identifier and a unique suspension time. The suspension time of a specific client in the unit of second equals to the numerical value of its identifier. The reason for this is that the clients must send the timestamps back to the server in a predefined order. Thus, the server can distinguish the timestamps of different clients.

When the timestamps of all the clients are received, the server calculates the time difference between the timestamps which is the clock offset of the two

clocks.



(a) Server (b) Client

Figure 5.2 Flow chart of the test-bed

5.3 Interfered Channel Simulation

To test the anti-interference capability of the protocols, we need to intentionally simulate background traffic. The idea of the simulation of interference is to generate some traffic to occupy the WiFi channel. The occupation will introduce additional transmission delay when transmitting packets because of carrier sense and retransmissions. Therefore, we use the WiFi NIC (Network Interface Controller) of 'Test' to continuously broadcast UDP packets to the WiFi network. See Figure 5.3.

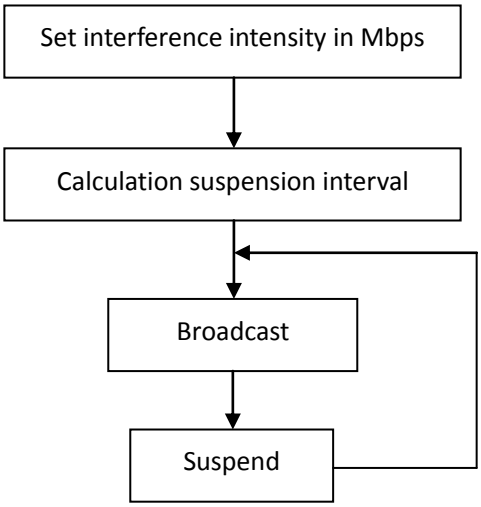


Figure 5.3 Flow chart of the interference application

By default, the MTU of a WiFi MAC frame is 1500 bytes on most systems. To avoid fragmentation, the broadcasted packets have a fixed length which equals to the MTU. The size of an IP header is 20 bytes [11], and that of a UDP header is 8 bytes [12]. Therefore, the broadcasted packets have a payload of 1472 bytes.

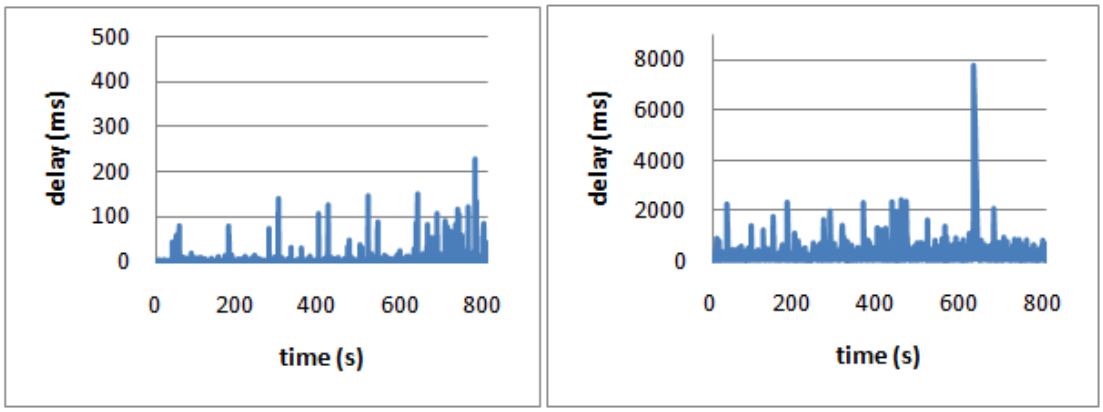
The interference application adjusts the interference intensity by changing the broadcasting interval. The interval is calculated by $\frac{8*1500\ \text{bytes}}{\text{Interference intense in Mbps}}$.

We can present the effect by interference on packet round-trip delay by using the following formula:

$$\text{Round trip delay} = t_2 - t_1 - t_3 + t_4$$

Where t_1, t_2, t_3 and t_4 are the captured timestamps during timestamps exchange in Section 4.1.2.

Figure 5.4 shows the delay under different interference intensity.



(a) (b)

Figure 5.4 Round-trip delay

Through Figure 5.4, we can notice that the delay values increased with the increment of the interference intensity. Figure 5.5 is the CDF of the delay which clearly shows the impact of the interference on transmission delay.

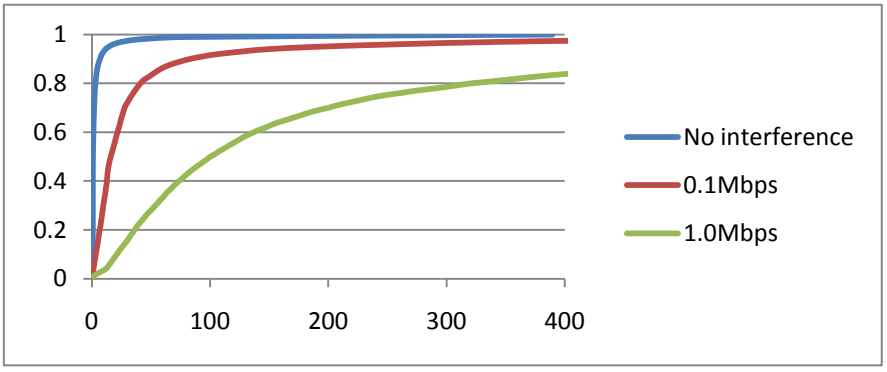


Figure 5.5 CDF of delay in three interference levels

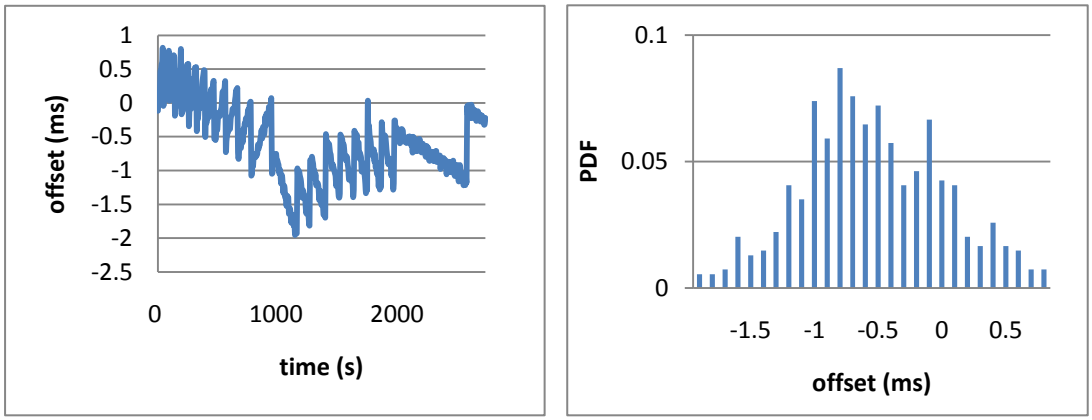
5.4 Measurement Results

In this section, the measurement results of the FCAP (Fast Clock Adjustment Protocol) are presented in order to show its synchronization accuracy. The measurements took place under three different interference levels which tests its performance under different circumstances. For each interference level, a comparative measurement of NTP was also done.

In each measurement, firstly, the clock offset is presented in the time domain. Secondly, a corresponding histogram of the offset is given. At the end of this

section, some statistics are given in order to show the numerical results.

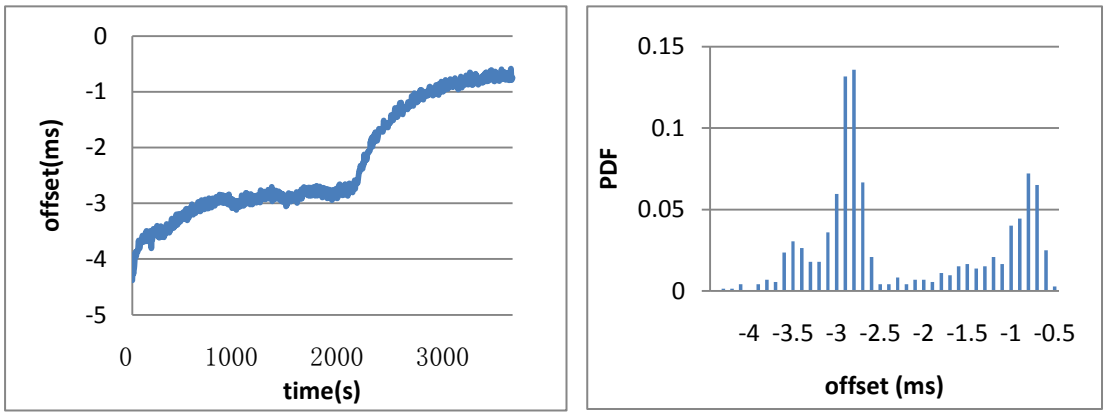
5.4.1 Without Interference



(a) (b)

Figure 5.6 Measurement for FCAP without interference

Figure 5.6 shows the measurement results of FCAP when the channel is near idle. It clearly shows that the clock frequency is adjusted whenever the sample buffer is full (1000 s). Clock offset is adjusted whenever it exceeds the step threshold (1 s).

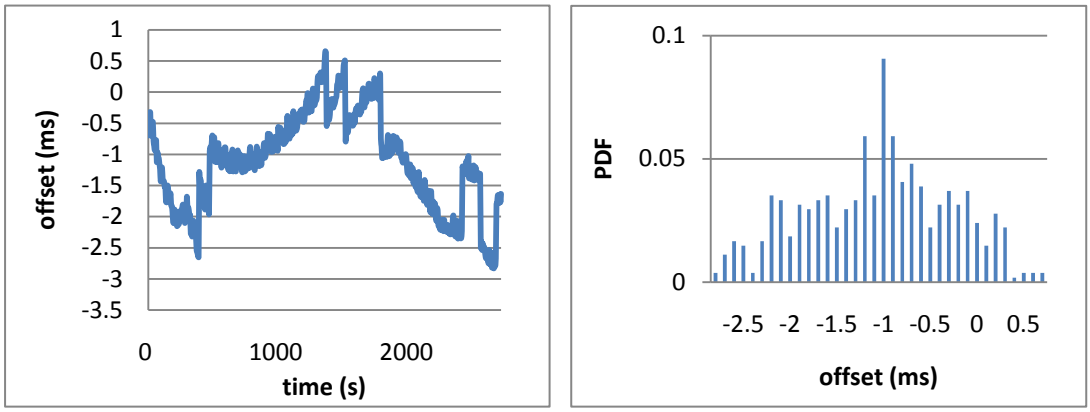


(a) (b)

Figure 5.7 Measurement for NTP without interference

Figure 5.7 shows the measurement results of NTP without interference. NTP starts with a clock offset error of 4 ms. The clock offset is continuously driven towards 0 ms. However, the start time is longer than FCAP.

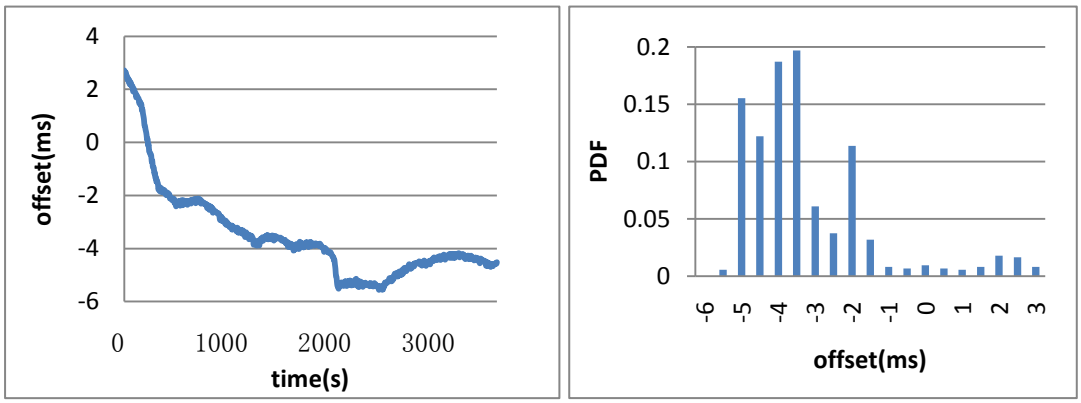
5.4.2 With 0.1Mbps Interference



(a) (b)

Figure 5.8 Measurement for FCAP with 0.1Mbps interference

Figure 5.8 shows the measurement results of FCAP with weak interference (0.1 Mbps). Since the WiFi channel is occupied by the background traffic, the transmission delay of the synchronization message is increased and the estimation error becomes greater. Therefore, the maximum clock offset error in Figure 5.8 is larger than that in Figure 5.6.

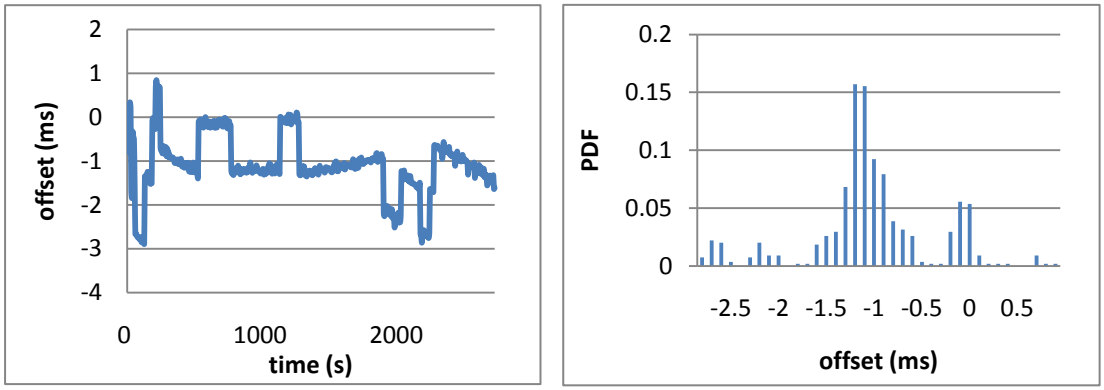


(a) (b)

Figure 5.9 Measurement for NTP with 0.1Mbps interference

Figure 5.9 shows the measurement results of NTP with weak interference. Due to the interference, the clock offset drifts apart again after it converges to almost 0 ms.

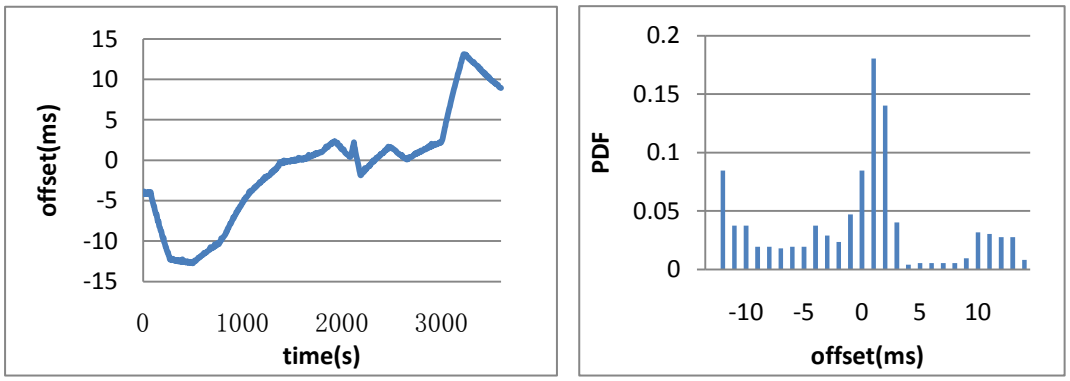
5.4.3 With 1.0Mbps Interference



(a) (b)

Figure 5.10 Measurement for FCAP with 1.0Mbps interference

Figure 5.10 shows the measurement results of FCAP with strong interference (1.0 Mbps). Since FCAP has an observation time of 1000 s, it can collect enough valid samples although there is strong interference. Therefore, the performance of FCAP under this circumstance is not too much worse than with weak interference.



(a) (b)

Figure 5.11 Measurement for NTP with 1.0Mbps interference

Figure 5.11 shows the measurement results of NTP with strong interference. At the beginning, the clock offset diverges due to the initial frequency difference. After the clock offset converges to 0 ms at around 2000 s, the strong interference drives it to diverge again. The maximum clock offset error caused by the strong interference is significantly larger than that under the weak

interference.

5.4.4 Statistics

Table 5.1 Statistics for the clock offset errors

Protocol	Interference (Mbps)	Mean	Median	Variance	Standard Deviation	95-percentile
FCAP	0.0	0.69	0.66	0.19	0.44	1.51
NTP	0.0	2.28	2.80	1.02	1.01	3.58
FACP	0.1	1.15	1.04	0.52	0.72	2.40
NTP	0.1	3.71	3.90	1.52	1.23	5.38
FCAP	1.0	1.11	1.14	0.39	0.62	2.57
NTP	1.0	5.04	2.46	21.68	4.65	12.52

Table 5.1 gives the statistical results of the clock offset errors. We define the clock offset error at time t as $|C(t) - S(t)|$, where $S(t)$ is the clock time of the server and $C(t)$ is the clock time of the client.

5.5 Results Analysis

From the measurement results in the previous sections, we can make the following observations,

(i) Start time

FCAP can make the clocks synchronized shortly after the protocol initiated regardless of offset and frequency error as well as the interference level due to its quick start property. However, NTP's start time depends on the initial conditions. The initial frequency difference and the initial clock offset are totally random values. Therefore, the start time of NTP is unpredictable.

(ii) Anti-interference

Comparing the data in Table 3.1 and Table 5.1, NTP synchronizes the clocks more accurate in its steady state when no interference existing in the channel. For both protocols, the accuracy goes down with the increase of the

interference intensity. However, the accuracy of NTP deteriorates more significantly. Since the timestamp exchange interval of NTP is much larger than that of FCAP, when interference exists, NTP has more probability to get a worse sample. Therefore, the clock adjustment can be driven in the wrong direction. From Figure 5.9 (a), we can notice that, when there is light interference (0.1Mbps), the maximum clock offset error of NTP is around 5ms which has almost reached the bound of our project requirement. In the case of strong interference, the maximum clock offset is even worse.

However, FCAP has a stronger anti-interference probability due to its short timestamp exchange interval. In all cases, the clock offsets are controlled within 3 ms.

(iii) Shape of the offset curve

NTP has a smoother clock offset curve than FCAP. NTP uses a PLL algorithm to adjust the clock, which avoids big steps in the adjustment. In order to meet the quick start requirement, FCAP sets up a step threshold for the clock offset. Therefore, the offset curve presents a zigzag pattern. The step threshold is currently set to 1ms. It means that the amplitude of each step is around 1ms. For the applications and the users, 1ms is not a huge value and it will not be conscious of.

To summarize, NTP has excellent synchronization accuracy in its steady state and when the channel is idle. Therefore, it is suitable for long-term usage. However, for an application, which needs to start quickly and be resistant to interference, FCAP is a better choice.

Chapter 6 Users' Feeling Survey

We also conducted a survey to test if the application meets the users' requirement of synchronized playback.

6.1 Survey Method

The general idea of the survey is to show the synchronized playbacks to the users and get the feedback of their feeling about the synchronization and how much synchronization error that can be tolerated.

In order to conduct the survey as accurate as possible, we need to make sure the synchronization error is under our control. It means we can manually set the presentation time difference between the screens. Therefore, we use wired Ethernet to synchronize the clocks. Since the transmission delay of Ethernet is only a few hundreds of microseconds, the clocks could be synchronized very accurately.

In the playback application, an additional delay is applied to the play thread. When a frame is about to be presented, the thread is suspended for an additional period of time and the additional time is manually set every time. Therefore, the playbacks can be out of sync at a specific time difference set by us.

With the preparations above, the survey was conducted as follows. The survey was divided into 30 tests. For each test, a time difference value is picked randomly for the playback. The users have binary options. What they need to answer is just whether they feel the playback in sync or not. They would mark '1' for in sync and '0' for out of sync. Ideally, after all the 30 tests, a user should have a result as presented in Figure 6.1 if the tolerable level is 10 ms.

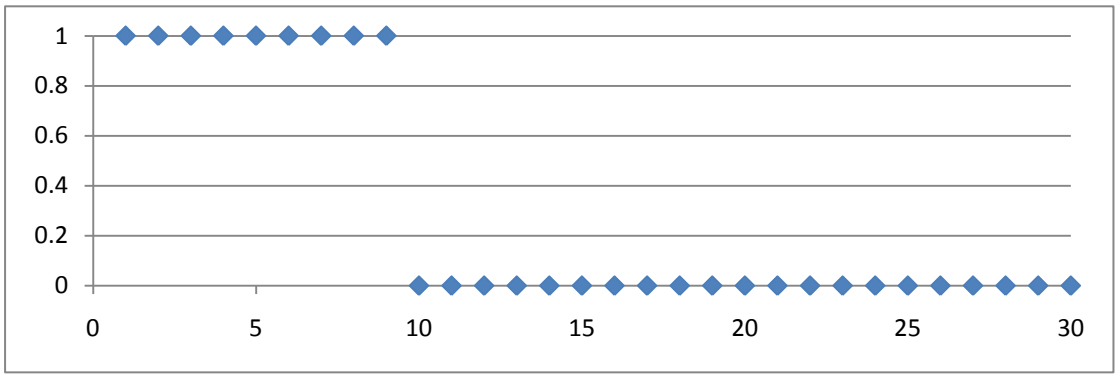


Figure 6.1 Ideal survey result of a user

If there are four users, by accumulating the answers of corresponding time difference values of all the users, we get the chart in Figure 6.2 which reflects the general feeling of all the users. The maximum value in the chart equals to the number of the users.

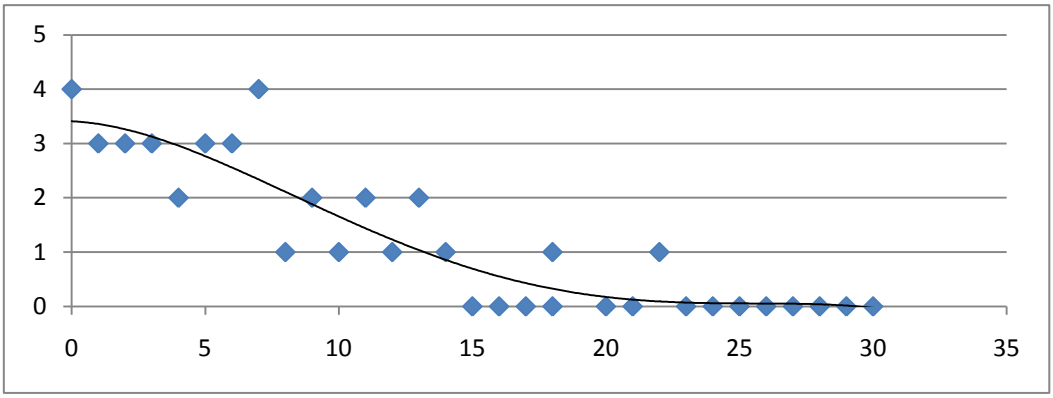


Figure 6.2 Example of final survey result

By applying polynomial regression, we can find the trend line of the chart. The final acceptance threshold can be found by analyzing the regression line. The analytical method will be described in Section 6.3.

6.2 Factors Affecting the Results

During the survey, I found that the survey results were affected by the following two factors:

(i) The user's sensitivity

To reflect user's true feeling on synchronization, they should have no

background knowledge about this project. Therefore, they judge the playback in sync or not totally based on their feelings. We call this approach the normal method. During the survey, I found that different people have different sensitivities. To find an acceptance threshold by using the method in the previous section, a bigger number of users are required. However, for our project, that is not practical.

People's sensitivities are proportional to the familiarities of the video content. To make users more familiar with a video could increase their sensitivities to a universal level. This approach eliminates the impact of background knowledge of the result. Playing the same video to the user for several times could increase the familiarities. Obviously, the threshold identified by the new method would be stricter. If our application meets this stricter threshold, it also meets the threshold generated by the normal method.

(ii) The content of the video

The video content also affects the users' sensitivity. Objects on the screen with different velocities and brightness result in different feelings. A video with complex content will make different users focus on different details on the screen. However, a video with simple content will make users' attentions focused.

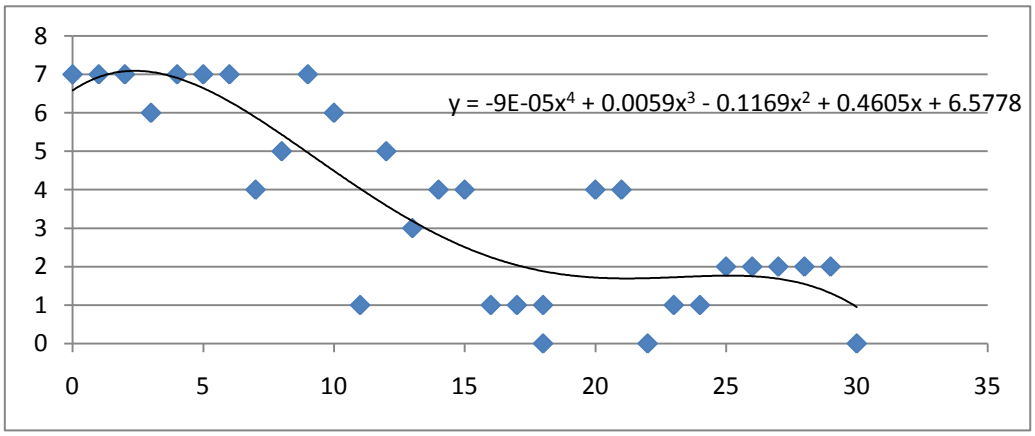
6.3 Final Results

Based on the two factors above, I choose two HD and wide-screen videos with different complex level to do the survey. One of the video is a clip of a concert with actresses and flashing light on the screen (Figure 6.3a), which is a relative complex video. The other video introduces the universe with rotating stars on the screen (Figure 6.3b), which has simpler content.

(a) (b)

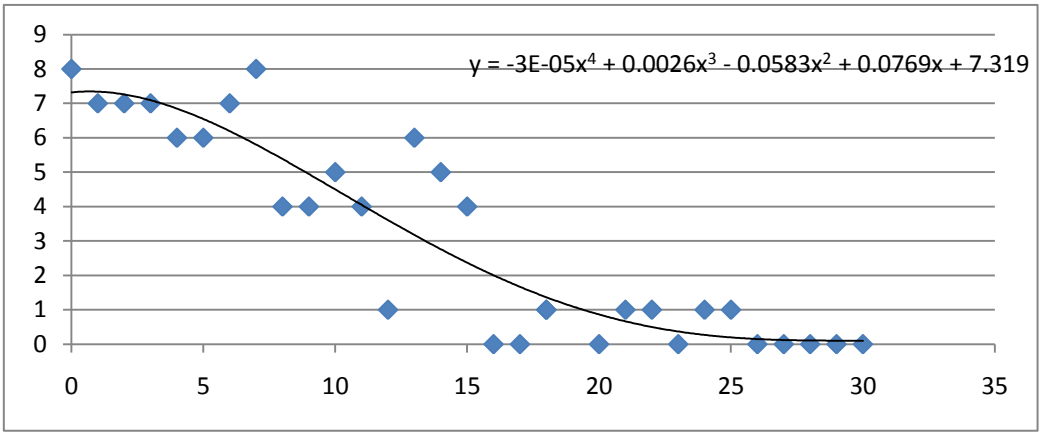
Figure 6.3 Screenshots of test videos

The number of users was 8. All of them are students of Tilburg University. Therefore, the results are as follows,



max=7.0883, $\frac{\sqrt{2}}{2}\max = 5.0122$, threshold=8.858

Figure 6.4 Survey result of a complex video



max=7.345, $\frac{\sqrt{2}}{2}\max = 5.1937$, threshold=8.516

Figure 6.5 Survey result of a simple video

The curves in the Figure 6.4 and 6.5 are the regression line generated by polynomial regression. By using the definition of half-power bandwidth in electric circuit theory, we can find the acceptance threshold [13]. The time difference with a corresponding value on regression line which is $\frac{\sqrt{2}}{2}$ of the maximum value is the threshold.

Both of the results give the threshold around 8ms. Therefore, we can say that a synchronized playback with a time difference smaller than 8ms can be considered in sync by most users.

Chapter 7 Conclusions and Future Work

In this chapter, all the results of the project are concluded. Based on the conclusions suggestions for the future works are given.

7.1 Conclusions

In this thesis, we have studied the synchronization of multimedia playback on WPAN devices over a WiFi network.

Firstly, the final target of the project is raised. The project required an application that could achieve a synchronized play of videos with quick-start and anti-interference properties.

Secondly, we analyzed the whole synchronization problem in detail. The whole problem was decomposed into two main factors, the synchronization of the playbacks and the synchronization of the clocks. Both the factors were discussed in detail.

In the playback synchronization, the challenge is to minimize the response time. We explored the open source project VLC and utilized its command line tool and its APIs to make a synchronized player. However, the response times of those tools exceed the requirement of our project. Therefore, we switched to a lower layer tool – FFmpeg to play the video with frame-by-frame level synchronization. Its response time is limited well within a 2ms range, which meets the requirement.

The basic principles of computer clocks and clock synchronization were introduced. Then we analyzed the most popular time synchronization protocol NTP and tried to use it to synchronize the clocks for our project. However, it is more suitable for long-term usage, since it cannot meet the requirements of quick-start and anti-interference although it has an excellent performance when it reaches its steady state. In order to meet all the requirements, a new

time synchronization protocol, the Fast Clock Adjustment Protocol, was designed. It uses a linear regression algorithm to estimate the clock frequency difference and clock offset. Its short timestamp exchange interval enables itself to have the properties of quick-start and anti-interference.

Thirdly, we measured the performance of FCAP and NTP and analyzed the measurement results.

Finally, a users' feeling survey was conducted to find the tolerable level of out-of-sync. In the survey, we found that different people have different sensitivity about the synchronization of the playback. Their sensitivities are also affected by the content of the video. In order to find the acceptance threshold of most users, a huge size of the samples set is required. Therefore, we used a higher standard to do the survey to guarantee that the threshold we found is accepted by most of the users. Through the survey, we eventually found the threshold to be 8ms, which is greater than the maximum synchronization error of FCAP (3 ms). Therefore, the application meets the synchronization requirement of most of the users.

7.2 Future Work

Although synchronized playback is achieved, there are still some aspects of our project worthy to be improved in the future.

(i) Synchronization of multiple clocks

In the thesis, we focused on the synchronization of two clocks. If more devices are used to build a video wall, the synchronization of multiple clocks is required. For multiple clocks, the synchronization can be done in a more accurate way. We know that the synchronization error comes from the uncertainty of the transmission delay. The uncertainty is because of the carrier sense and the retransmission mechanism when transmitting a packet. In the multiple clocks case, if the server broadcasts a packet, all the clients should receive the

packet at exactly the same time regardless of carrier sense and retransmissions. The principle is the same as implementing the test-bed in Chapter 5. By exchanging the reception timestamps, the clients' clocks can be synchronized more accurately.

(ii) Virtual clock

FCAP continuously adjusts the local clock of the client. However, the local clock is the only time source for all the local applications. Frequent adjustments may affect the execution of other applications. A better way is to set up a dedicated virtual clock for the playback application. The time synchronization protocol adjusts the parameters of the virtual clock, but not the real local clock. The virtual clock is only used for the synchronized playback application.

(iii) Video streaming

In the project, the video files were stored locally. In the future, the devices may request the video from a media server as well. Therefore, the video streaming and splitting are other parts which are worthy to be implemented.

References

[1] IEEE 1999 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. ANSI/IEEE Std 802.11, ISO/IEC 8802-11: 1999.

[2] VideoLAN, http://www.videolan.org

[3] FFmpeg, http://www.ffmpeg.org

[4] S. Dranger, An ffmpeg and SDL Tutorial, http://dranger.com/ffmpeg

[5] Simple DirectMedia Layer, http://www.libsdl.org

[6] D. L. Mills, "Network time protocol (version 3) – specification, implementation and analysis," IETF RFC 1305, Mar. 1992.

[7] J. Ridoux and D. Veitch, "Principles of Robust Timing over the Internet," *Commun. ACM*, vol. 53, no. 5, pp.54-61, 2010.

[8] D. L. Mills, "Adaptive hybrid clock discipline algorithm for the Network Time Protocol," *IEEE/ACM Trans. Networking 6, 5*, 505-514, October 1998.

[9] D. L. Mills, "Improved algorithms for synchronizing computer network clocks," *IEEE/ACM Trans.Networks* , 245-254, June 1995.

[10] TCPDUMP/LIBPCAP, http://www.tcpdump.org

[11] J. Postel, "Internet Protocol," IETF RFC 791, Sep. 1981.

[12] J. Postel, "User Datagram Protocol," IETF RFC 768, Aug. 1980

[13] Wikipedia, http://en.wikipedia.org/wiki/Bandwidth_(signal_processing)