Desktop Video Recording and Streaming Techniques

Individual Report by Hu Yucheng

Supervisors:
Dr. Ronald Chung

Project members:
Hu Yucheng
Abstract

This project is going to develop low CPU consumption desktop video recording and streaming applications. With these two applications, manager can record and see what their employees are doing during their work time. Similar techniques have been broadly used in game broadcasting websites like, Panda TV, Twitch.
# Contents

Contents ........................................................................................................................................... 2

List of figures .................................................................................................................................. 4

1 BACKGROUND ............................................................................................................................. 5

1.1 ONLINE LIVE STEAMING ......................................................................................................... 5

1.2 MONITORING SOFTWARE ........................................................................................................ 8

1.4 OBJECTIVES .............................................................................................................................. 9

2. Introduction to the project ........................................................................................................... 10

2.1 HIGH TIME CONSUMPTION IN DEVELOPING ........................................................................ 10

2.2 USE OF OPEN SOURCE LIBRARY .............................................................................................. 11

2.3 HIGH REQUIREMENT OF EFFICIENCY ....................................................................................... 11

2.4 SCOPE ....................................................................................................................................... 12

2.5 Deliverables ................................................................................................................................ 12

3 Methodology .................................................................................................................................. 14

3.1 OPEN SOURCE SOLUTION FFmpeg ........................................................................................... 15

3.1.2 Capture the desktop ................................................................................................................. 17

3.1.3 One to many monitoring ........................................................................................................... 17

3.1.4 H264 ...................................................................................................................................... 18
3.1.5 RTP ..................................................................................................................18
3.1.6 Playback Technique ..........................................................................................19
3.2 WCF .......................................................................................................................20
3.3 The failed trial – using application framework ....................................................20
3.3.1 REGISTER SEQUENCE ..................................................................................20
3.3.2 WORKING SEQUENCE ..................................................................................21
3.3.3 TERMINATION PROCESS ..............................................................................22
3.3.4 PLUG-IN ARCHITECTURE ............................................................................23
3.3.5 APPLICATION FRAME ...................................................................................25
4 The Implementation: ...............................................................................................26
4.1 documents overview .............................................................................................26
4.2 Server files : ..........................................................................................................27
4.3 Client files: ............................................................................................................31
5 Conclusion and future work ....................................................................................55
List of figures

1. Douyu panda huya’s snapshot
2. Screenshot of douyu
3. THE MAPPING OF MONITORING SOFTWARE MARKET
4. Screenshot of the project
5. ffserver.conf
6. server.bat
7. mpeg.bat
8. mpeg.bat
9. REGISTER SEQUENCE DIAGRAM
10. WORKING SEQUENCE DIAGRAM
11. TERMINATION SEQUENCE DIAGRAM
12. PLUG-IN ARCHITECTURE
13. Overview of the project documents
14. VtServer
15. vtClient
16. vtClient.cpp
17. vtClientDlg.cpp
18. vtServer.cpp
19. vtServerDlg.cpp
1 BACKGROUNG

This technique can be applied in situations. Analyzing both cases can help find the market entry point for the project.

1.1 ONLINE LIVE STREAMING

The first one is the online live streaming. In 2014, Twitch came into people’s sight, and the broadcaster used the technique provided by Twitch to broadcast their game. This new communication has not only attracted millions of young people to the website but also created tons of opportunities to both the players and the watchers. Meanwhile, China’s live-streaming websites, such as DouyuTV and ZhanqiTV, also started their business in China by initially focusing on the game live streaming. At present, they have already become a burgeoning cottage industry, which offers money-making opportunities and even stardom to their mostly female hosts. Also, this is a new alternative for millions of viewers to online dramas. The technique used by these websites is the same as what we implement in this project. The difference is that we have no need to worry about scaling at architectural and organizational level.
What is the relation between this project and online live streaming? Similar techniques have been used. A screenshot of DouyuTV (see figure 1) can give us some intuitive feelings. The host’s computer is a recorder, while the audience's computer is like a monitor. The movement on the host’s computer can be recorded and transmitted to the website. Then, the audiences can see what the host is doing on his desktop via the website.
Figure 2. Screenshot of Douyu

This is a huge market, in which various institutions has invested huge amount of money.

Because in the current Internet era, the most important thing is to occupy the user's time.

Regarding each person only having 24 hours a day, if you can take as much user's time as possible, you would have more chance to generate payment from the user directly or indirectly.

Now the market in the flow of several products ranked bucket fish tiger panda battle flag.

Then is there a chance for us to enter as a technology company in this highly mature development market? I think there is. Fine operation is in the first priority for a company in fiercely competing market. Under the same traffic, if you can deliver a technique that requires
less bandwidth and clearer video quality, you have much more chance to win this competition.

Therefore, this project has huge chance to sell to the existing platform.

1.2 MONITORING SOFTWARE

The second one is the monitoring software. The companies install a desktop video that records application on every computer as well as a streaming application on the manager’s computer. Then, the managers can see what the employees are doing during their work time.

Indeed, this is a quite efficient way in restricting the manners of the employees.

The existing monitoring software and their main features can be shown in the following Figure 2.

![Figure 3. THE MAPPING OF MONITORING SOFTWARE MARKET](image-url)
The first row lists the monitoring software in the market, and the first column lists the features.

Nearly all of the software has many features. Besides, it can be seen that the price of all the software is really high. In some user cases, however, only one or two features are needed. Thus, the users are unwilling to pay so much for the needless features. Maybe, there is a need for the monitoring software that can be customized. Hence, users can choose and pay for the features they need. This refers to a great market entry point for this project. After finding this, I set the goal of my project to develop a user-friendly and highly customized monitoring software.

1.4 OBJECTIVES

The objective of this project is to develop a user-friendly monitoring software which has:

High customization

High scalability

Low CPU consumption

Low delay

Customization: The users can choose the features they need and download the corresponding versions. The intermediate goal of this project is to have two features, including real time recording and remote recording, in the feature pool. This will be done by
next April. The ultimate goal of this project is to have more than 20 features in the feature pool. Then, the users can choose the feature they need and download the corresponding version of software.

Scalability: The first version of the project will be finished before the next April. Later, the new features will be added to the first one. New features can be added to the existing version easily. Also, the new features added will not influence the previous ones.

Low CPU consumption: The software will have low CPU consumption. Regarding the CPU consumption, no specific standard is set up now. To lower the CPU consumption, continuous testing will be carried on in February.

Low delay: The monitor will receive the real-time movement on the recorder’s desktop. Due to the limitation of network and hardware, there are some delays. However, it will be controlled to less than 0.25 seconds.

2. Introduction to the project

2.1 HIGH TIME CONSUMPTION IN DEVELOPING

Assume that there are 10 features in the feature pool, there would be more than 100 different combinations of different features. If the application is developed in a traditional way like the
previous work did, it would consume large amounts of resources. If the cost to develop such a comprehensive system is initially too high, or if the timeline to deployment is too long, the beautiful thing about custom software will no longer be beautiful. Thus, the project needs a fast developing path.

2.2 USE OF OPEN SOURCE LIBRARY

The recorder and monitor will be implemented based on the open resource FFMpeg and its subtool FFplay. Since this is the first time of working with open source, a good understanding of the open resources will be needed.

2.3 HIGH REQUIREMENT OF EFFICIENCY

Indeed, this is a project that focuses more on implementing than researching. There are already some similar applications in the market, such as airplay, chrome cast, and twitch, which do really good job. The meaning of this project lies in the sharpening of desktop programming skills and self-learning skills. With all the other applications developed by a big team, the biggest problem of the project is the efficiency. The first version with real-time recording and remote recording should come out before February. Furthermore, to optimize the functionality of the software, the tests would be taken in the last two months. No chance for constructional mistakes is allowed. Before the implementation, a good user case study and a clear product design will be needed.
2.4 SCOPE

The programming language of this project is C++, which is a requirement of this project. The programming platform is Virtual Studio, and it can offer a set of helpful tools for the developing.

This project, which is a combination of existing techniques, takes the mature and popular plug-in architecture. Other fundamental functionalities are implemented with the help of open source library FFmpeg and its subtool FFplay. Also, it includes making improvement to these techniques so as to let them work better with frame.

These two applications support live broadcasting whose delay would be controlled to less than 25 milliseconds. However, the specific time interval delay broadcasting is not supported. Also, the video streaming application will support all the main video formats. Besides, the resolution ratio can be adjusted according to the user’s preference. Most of all, this project supports one-to-many monitoring, meaning that one recorder can monitor unlimited recorders.

2.5 Deliverables

Developed two clients, one is client side, a server side. Input client ip address in the server side. Start the client and server side, you can then receive client-side terminal transmission over the signal. A server can simultaneously monitor multiple clients, while the current...
interface has only six display interface. If you need more, it is very convenient to expand. At
the same time of recording, the video will be saved in the c drive storage folder, then you can
easily see the video.

![Figure 4. Screenshot of the project](image)

Usage:

Client setting:

1. Copy the folders ffmpeg and video_save into the root directory of disk C;

2. Bind IP to port through an adjustment of the setting of c:\ffserver.conf;

3. Change the IP and port in the file c:\mpeg.bat into the IP and port of the client;

4. Open the vtClient (recall it using alt+f7 after the window has been hidden);
Client setting:

1. Copy the folders ffmpeg and the video_save into the root directory of Disk C;

2. Open the solution, and change the source code at the top of the vtServerDlg.cpp;

The IP and port in data array chart g_szSource[8][255] are both port and IP of the various video channels;

Set up the parameter in the g_nVedioCounts = 2; // the number of all the connected video channels;

3. Press F5 to start compiling. After refreshing, it will show the different video channels which will get recorded and stored in folder c:\vedio_save. Double-click the small video image to allow zooming in.

### 3 Methodology

The open-source FFmpeg is usually the program choice used in the implementation of functions. The program solution to XXX is utilized to develop and call specific functions. However, the combination of the functions is altered. I initially intended to come up with a framework to isolate the functional properties and improve the extendibility thereby creating
changes. However, it was realized that the complexity in the development of framework prototype was underrated especially by the virtue that it was a non-open framework. After a given period of development, I retained the functions dependency on FFmpeg and did an introduction of the MCF of Windows to the framework. For a period of two months, I dedicated my time and resources to the research on the MCF concept and completed the development through the MCF. Despite the result, I believe that it was a desirable and valuable attempt to code development. Hence, the discussion in Methodology is required to involve the two schemes so as to represent the change of the ideas I had to deepen the understanding.

3.1 OPEN SOURCE SOLUTION FFmpeg

FFmpeg is a set of open source audio and video tools for recording, converting, and streaming multimedia content. Moreover, it supports nearly every digital format and codec known, from the old and obscure to the cutting edge. The toolset is highly portable - available on most operating systems and platforms.

The FFmpeg project distributes four major applications:

FFmpeg - A command-line utility that can be utilized to process, convert, or manipulate media.

This tool refers to the foundation of the other applications.

FFplay - A simple media player.
FFserver - A streaming media web server.

FFprobe - A stream analysis tool.

FFmpeg is built with multiple self-contained libraries, which provide discreet functionalities that can be included into the other applications. These features include codec encoding and decoding, compression, image scaling, re-sampling, and format conversion.

Set up the FFserver

Edit the FFserver configuration file

```
Port 5555 #Assign port number
BindAddress 0.0.0.0 #Assign ip
MaxHTTPConnections 2000 #Maximum Http connections
MaxClients 1000 #Maximum Client connections
MaxBandwidth 10000 #Maximum bandwidth
CustomLog - #Diaries
NoDaemon

<Feed feed1.ffm>
    File c:\ffmpeg\tmp\feed1.ffm #File set up for each input
    FileMaxSize 100M #Maximum size for the saved file
    ACL allow 127.0.0.1 #Ip allowed for input
</Feed>

<Stream test.fly>
    Format flv #Video stream format
    Feed feed1.fmm #The source of the video stream
    VideoFrameRate 5 #Video frame rate
    VideoBitRate 512 #Video bit rate
    VideoSize 960x560 #Video frame size
    #OptionVideo flags +global_header
    NoAudio
    #AudioBitRate 24 #Audio bit rate
    #AudioChannels 1 #Audio channel
    #AudioSampleRate 44100 #Audio sample rate
    #OptionAudio flags +global_header
</Stream>
```

Figure 5. ffserver.conf

Start FFserver, input command via terminal
c:\ffmpeg\ffserver.exe -f c:\ffmpeg\ffserver.conf

Figure 6. server.bat

Start Ffmpeg


Figure 7. mpeg.bat

3.1.2 Capture the desktop

The project use Win32 GDI-based screen capture device provided by Ffmpeg. This device allows to capture the entire window or a certain part of the desktop.

The grab the entire desktop using Ffmpeg:

ffmpeg -f gdigrab -framerate 6 -i desktop out.mpg

The grab a 640*480 region at position 10,20

ffmpeg -f gdigrab -framerate 6 -offset_x 10 -offset_y 20 -video_size vga -i desktop out.mpg

3.1.3 One to many monitoring

Through the MCF to create multiple dialog, each dialog will correspond to a client ip, from the corresponding ip collection of information flow. To achieve one to many monitoring.

sprintf(szCmd,"c:\ffmpeg\ffplay.exe %s",g_szSource[0]);
sprintf(szCmd,"c:\\ffmpeg\\ffplay.exe %s",g_szSource[1]);

sprintf(szCmd,"c:\\ffmpeg\\ffplay.exe %s",g_szSource[2]);

3.1.4 H264


Figure 8. mpeg.bat

The following command implements the sending H.264 bare stream "desktop monitoring video" to address rtp: //233.233.233.223: 6666

3.1.5 RTP

The following command sends H.264 bare stream "chunwan.h264" to address rtp:

//233.233.233.223: 6666


Note 1: -re is on behalf of the rate according to the frame sent

Note 2: -vcodec copy to be added, otherwise ffmpeg will re-encode the input H.264 nudity.

Note 3: The rightmost "> test.sdp" is used to store the output information of ffmpeg to form an sdp file. This file is used for RTP reception. When not "> test.sdp", ffmpeg will directly output
the sdp information to the console. Adding ":> test.sdp", you can directly save these sdp
information into text.

Play the RTP carrying the H.264 bare stream

Ffplay test.sdp

3.1.6 Playback Technique

The video is stored in the video_save document, so when the playback button is clicked, the
response video stored in the document will be called out.
3.2 WCF

Windows Communication Foundation (WCF) is a framework for building service-oriented applications. Using WCF, we can send data as asynchronous messages from one service endpoint to another. Using this foundation, we can easily establish the destop clients. So I choose this as the the key framework in this project.

3.3 The failed trial – using application framework

3.3.1 REGISTER SEQUENCE

The first process involved in a new connection is the register process, and it can help set up a stable connection between the monitor and a particular recorder. The process can be shown in figure 3.
This diagram gives the sequence of the register process. It is triggered by the recorder on clicking the starting register button. After the click of button, the initialization can be done. Then, it will send the register signal to the monitor via the TCP long connection. After receiving the register signal, the monitor will automatically assign a port number to this request and send the number back to the recorder. In the end, the register process will succeed.

### 3.3.2 WORKING SEQUENCE

After the success of register, the monitor and recorder will start the working procedures. This process can be shown in Figure 4.
Following a successful register, the monitor will start monitoring. Then, the recorder will generate FFmpeg launch parameters and create a FFmpeg process. After the success of process creation, the recorder will obtain RTP information and send the record working signal to the monitor. Meanwhile, the FFmpeg process will start recording the screen and do the encoding. After receiving the recorder working signal, the monitor will generate the RTP description file and create the FFplay process, which will receive the video transferred from the recorder.

### 3.3.3 TERMINATION PROCESS

A termination process will be generated to terminate the monitoring. The detailed sequence of this process can be shown as follows.
The monitor sends the signal of terminating monitoring to the recorder. After receiving the signal, the recorder will terminate the FFMpeg process and return the terminate status to the monitor. Then, the monitor will terminate the FFplay process and refresh the status of the recorders. Under monitoring, the corresponding record will disappear from the table of recorders.

### 3.3.4 Plug-in Architecture

For achieving the goal of high customization and scalability, the project will develop a plug-in architecture. From a high-level view (see figure 5), we can regard that the plug-in architecture is formed by four parts:

- Application framework
An application framework is a software library that provides the fundamental structure to support the development of applications for a specific environment. An application framework acts as the skeletal support to build an application. During the development of applications, it can lessen the general issues. Through the use of code that can be shared across different modules of an application, this is achieved. It acts like a commander. Even though it knows which step the program will take, it does not know what each step is. Plug-in interface, also
known as plug-in contract, exists in the form of a service interface. All the plug-ins of the system will implement the unified interface specification of the system framework. As a result, this can help the application facilitate an effective organization and management of the plug-ins.

Plug-in is the software component that adds features to the program. It is loaded to the application under the requirements of plug-in interface.

Public function library contains the functions used by both of the framework and the plug-ins.

In this architecture, each system function module, common user interface, and icon can be developed as plug-ins to enhance the versatility of functional modules. The modification of functional modules will not affect the normal operation of the other plug-in modules, reduce the difficulty of system maintenance, and improve the system scalability. Also, when the features are implemented in the form of plug-ins as separate components, it can be added and removed from the application without influencing the other features.

### 3.3.5 APPLICATION FRAME

The application frame chosen for this project can be developed by a coder for personal use. I asked him for the permission to use his frame. This frame is composed of four modules:

- Transport Module
Session Module

Service Module

Application Module  Transport module is based on Asio library and it takes asynchronous mode. The main function is to deal with the establishment and monitoring of connections, as well as the data's transmission and reception.

Session module is mainly responsible for the maintenance of the connection state, including heartbeat, reconnection, and message routing.

Service module is primarily responsible for the abstract decoupling of the application, as well as the encoding and decoding of the message.

Application module mainly implements a set of plug-in system specifications. Also, it is responsible for the maintenance of plug-in's management and the application's life cycle.

4 The Implementation:

4.1 documents overview

This is the folder of the program. The major implementation programs for the monitoring end and the client are the vtServer and the vt-Client. Ffmpeg is the resource from the open-source
program. The ffmpeg has its applications in the project, and mainly consists of necessary dll files. The Video-save is usually the file that stores the surveillance pictures and usually needs to be copied directly into the root of disc C. The debug stores the monitoring end and the compiled client. Other files are usually get generated automatically upon compilation.

Figure 13. Overview of the project documents

4.2 Server files:

vtServer.vcproj

It is normally the major file of the VC++ project which is generated through the use of application wizard.

It includes some of the components such as the information about the platform, the project functionality and configurations chosen through the application wizard and the VC++ version data of the generated files.

vtServer.h
The vtServer.h is the major head file of the application. It includes the states CvtServerApp application class and other projects' head files (e.g., Resource.h).

vtServer.cpp

This involves the source file of the major application and usually includes the application class CvtServerApp.

vtServer.rc

This involves the list of all Microsoft Windows sources that are used in the application. It includes the cursors, icons, and bitmaps stored in the RES child directory. The files can be edited directly in Microsoft Visual C++. Additionally, the project resources are usually located at 2052.

res\vtServer.ico

This includes the file of application icons. It is usually included in the main resource file (vtServer.rc)

res\vtServer.rc2

This element involves a scenario where the file consists of resources that cannot be edited in Microsoft Visual C++. All resources which cannot be edited using the resource editor should be stored in the res\vtServer.rc2 file.
vtServerDlg.h, vtServerDlg.cpp - dialog

These files include the CvtServerDlg class, which explains the behaviors which can be conducted by the application’s main dialog. The template of this dialog is usually contained in the vtServer.rc, which can get edited in the Microsoft Visual C++.

Other functions:

ActiveX control

These applications support the use of ActiveX control.

Support Print and Print Preview

The application wizard has conducted the generation of codes for print setting and print preview and printing by calling the other member functions of the CView class in MFC library.

Other standard files:

StdAfx.h, StdAfx.cpp

These files are used for the generation of the pre-compiled type file (StdAfx.obj).

And the precompiled head (PCH) file (vtServer.pch).

Resource.h
This is the standard head file and usually involves the definition of the new resource’s ID.

Microsoft Visual C++ usually reads and update this file.

vtServer.manifest

The Windows XP scans the applications by use of the inventory file applications because it is dependent on the parallel programs of specific versions. This information can be useful in program loading from the cache or in loading confidential information from the various applications. The inventory of the application can be externally installed together with the executable files of the application. The file of manifestation is similarly included to enhance subsequent publications. Additionally, it may be included as a resource in the executable file.
Figure 14. VtServer

4.3 Client files:

vtClient.vcproj
It is normally the major file of the VC++ project which is generated through the use of
application wizard.

It includes some of the components such as the information about the platform, the project
functionality and configurations chosen through the application wizard and the VC++ version
data of the generated files.

vtClient.h

The vtClient.h is the major head file of the application. It includes the states CvtClientApp
application class and other projects’ head files (e.g., Resource.h).

vtClient.cpp

This involves the source file of the major application and usually includes the application class
CvtClientApp.

vtClient.rc

This involves the list of all Microsoft Windows sources that are used in the application. It
includes the cursors, icons, and bitmaps stored in the RES child directory. The files can be
edited directly in Microsoft Visual C++. Additionally, the project resources are usually
located at 2052.

res\vtClient.ico
This includes the file of application icons. It is usually included in the main resource file (vtClient.rc)

res\vtClient.rc2

This element involves a scenario where the file consists of resources that cannot be edited in Microsoft Visual C++. All resources which cannot be edited using the resource editor should be stored in the res\vtClient.rc2 file.

vtClientDlg.h, vtClientDlg.cpp - dialog

These files include the CvtClientDlg class, which explains the behaviors which can be conducted by the application’s main dialog. The template of this dialog is usually contained in the vtClient.rc, which can get edited in the Microsoft Visual C++.

Other functions:

ActiveX control

These applications support the use of ActiveX control.

Support Print and Print Preview

The application wizard has conducted the generation of codes for print setting and print preview and printing by calling the other member functions of the CView class in MFC library.
Other standard files:

StdAfx.h, StdAfx.cpp

These files are used for the generation of the pre-compiled type file (StdAfx.obj).

And the precompiled head (PCH) file (vtServer.pch).

Resource.h

This is the standard head file and usually involves the definition of the new resource’s ID.

Microsoft Visual C++ usually reads and updates this file.

vtClient.manifest

The Windows XP scans the applications by use of the inventory file applications because it is dependent on the parallel programs of specific versions. This information can be useful in program loading from the cache or in loading confidential information from the various applications. The inventory of the application can be externally installed together with the executable files of the application. The file of manifestation is similarly included to enhance subsequent publications. Additionally, it may be included as a resource in the executable file.
The vtClient.h is the major head file of the application. It includes the states CvtClientApp application class and other projects' head files (e.g., Resource.h). vtClient.cpp involves the source file of the major application and usually includes the application class CvtClientApp.

vtClient.rc involves the list of all Microsoft Windows sources that are used in the application. It includes the cursors, icons, and bitmaps stored in the RES child directory. The files can be edited directly in Microsoft Visual C++.

#include "stdafx.h"

#include "vtClient.h"
#include "vtClientDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// CvtClientApp

BEGIN_MESSAGE_MAP(CvtClientApp, CWinAppEx)

ON_COMMAND(ID_HELP, &CWinApp::OnHelp)

END_MESSAGE_MAP()

// CvtClientApp structure

CvtClientApp::CvtClientApp()
{
    // put all the important initialization in InitInstance
}

CvtClientApp theApp;

// CvtClientApp initialization

BOOL CvtClientApp::InitInstance()
{
    INITCOMMONCONTROLSEX InitCtrls;
    InitCtrls.dwSize = sizeof(InitCtrls);

    // Set it as a public tool which can be used by all applications.
    InitCtrls.dwICC = ICC_WIN95_CLASSES;
    InitCommonControlsEx(&InitCtrls);
CWinAppEx::InitInstance();
AfxEnableControlContainer();
SetRegistryKey(_T("HKU_FYP"));
CvtClientDlg dlg;
m_pMainWnd = &dlg;
INT_PTR nResponse = dlg.DoModal();
if (nResponse == IDOK)
{
    // “Yes” to close the communication framework
}
else if (nResponse == IDCANCEL)
{
    // “Cancel” to close the communication framework
}
return FALSE;
}

Figure 16. vtClient.cpp

#include "stdafx.h"
#include "vtClient.h"
#include "vtClientDlg.h"
#include <atlstr.h>
#ifdef _DEBUG
#define new DEBUG_NEW
#endif
#define SHOW_OPTION SW_HIDE

// "About" dialog
class CAboutDlg : public CDialog
{

public:

CAboutDlg();

// Dialog figures
enum { IDD = IDD_ABOUTBOX };   

protected:

virtual void DoDataExchange(CDataExchange* pDX);   // DDX/DDV support

// Realisation
protected:

DECLARE_MESSAGE_MAP()

};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{

}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{

}
CDIg::DoDataExchange(pDX);

}
BEGIN_MESSAGE_MAP(CAboutDlg, CDIg)

END_MESSAGE_MAP()

// CvtClientDlg dialog

CvtClientDlg::CvtClientDlg(CWnd* pParent /*=NULL*/)
    : CDIg(CvtClientDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CvtClientDlg::DoDataExchange(CDataExchange* pDX)
{
    CDIg::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CvtClientDlg, CDIg)

ON_WM_SYSCOMMAND() 

ON_WM_PAINT()

ON_WM_QUERYDRAGICON()

//}}AFX_MSG_MAP

ON_BN_CLICKED(IDC_BUTTON1, &CvtClientDlg::OnBnClickedButton1)

ON_WM_HOTKEY() 

ON_BN_CLICKED(IDC_BTN_HIDE, &CvtClientDlg::OnBnClickedBtnHide)
ON_WM_CLOSE()

ON_BN_CLICKED(IDC_BUTTON2, &CvtClientDlg::OnBnClickedButton2)

END_MESSAGE_MAP()

// CvtClientDlg information handling process

BOOL CvtClientDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // IDM_ABOUTBOX 必须在系统命令范围内。
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        BOOL bNameValid;
        CString strAboutMenu;
        bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
        ASSERT(bNameValid);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }
}
Set the icon for communication framework

SetIcon(m_hIcon, TRUE);    // Set big icon
SetIcon(m_hIcon, FALSE);   // Set small icon

// Initialization
::RegisterHotKey(GetSafeHwnd(),1,MOD_ALT,VK_F7);
SetDlgItemText(IDC_EDIT1,_T("5555"));
SetDlgItemText(IDC_IPADDRESS1,_T("1.1.1.1"));
return TRUE;  // Unless setting the focus to control, return TRUE

void CvtClientDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDIalog::OnSysCommand(nID, lParam);
    }
}
// Add minimize button to the dialog

void CvtClientDlg::OnPaint()
{

    if (IsIconic())
    {

        CPaintDC dc(this);
        SendMessage(WM_ICONERASEBKGND,
        reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

        // Make the icon centered in the work area rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }

    else
    {
        CDialog::OnPaint();
    }
}
//get the cursor

HCURSOR CvtClientDlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CvtClientDlg::OnBnClickedButton1()
{
    //Start service
    WinExec("c:\ffmpeg\server.bat",SHOW_OPTION);
    Sleep(3000);
    WinExec("c:\ffmpeg\mpeg.bat",SHOW_OPTION);
}

void CvtClientDlg::OnHotKey(UINT nHotKeyId, UINT nKey1, UINT nKey2)
{
    ShowWindow(SW_SHOW);
    CDialog::OnHotKey(nHotKeyId, nKey1, nKey2);
}

void CvtClientDlg::OnBnClickedBtnHide()
{
    ShowWindow(SW_HIDE);
}
void CvtClientDlg::OnClose()
{
    if (MessageBox(_T("确定退出吗"), _T("提示"), MB_YESNO | MB_ICONWARNING) == IDNO)
        return;
    system("TASKKILL /F /IM ffmpeg.exe");
    system("TASKKILL /F /IM ffserver.exe");
    CDialog::OnClose();
}

void CvtClientDlg::OnBnClickedButton2()
{
    system("TASKKILL /F /IM ffmpeg.exe");
    system("TASKKILL /F /IM ffserver.exe");
}

Figure 17. vtClientDlg.cpp

The vtServer.h is the major head file of the application. It includes the states CvtServerApp application class and other projects’ head files (e.g., Resource.h). vtServer.cpp involves the source file of the major application and usually includes the application class CvtServerApp. vtServer.rc involves the list of all Microsoft Windows sources that are used in the application. It includes the cursors, icons, and bitmaps stored in the RES child directory. The files can be edited directly in Microsoft Visual C++.
BEGIN_MESSAGE_MAP(CvtServerApp, CWinAppEx)
  ON_COMMAND(ID_HELP, &CWinApp::OnHelp)
END_MESSAGE_MAP()

// CvtServerApp 构造
CvtServerApp::CvtServerApp()
{
}

// The only object of CvtServerApp
CvtServerApp theApp;

// CvtServerApp initialization
BOOL CvtServerApp::InitInstance()
{
  INITCOMMONCONTROLSEX InitCtrls;
  InitCtrls.dwSize = sizeof(InitCtrls);
  InitCtrls.dwICC = ICC_WIN95_CLASSES;
  InitCommonControlsEx(&InitCtrls);
  CWinAppEx::InitInstance();
  AfxEnableControlContainer();
  SetRegistryKey(_T("HKU_FYP"));
  CvtServerDlg dlg;
  m_pMainWnd = &dlg;
INT_PTR nResponse = dlg.DoModal();

if (nResponse == IDOK)
{
}

else if (nResponse == IDCANCEL)
{
}

return FALSE;

Figure 18. vtServer.cpp

char g_szSource[8][255] = {//IP address of each source of client(this should be changed
//accordingly)
  "http://162.105.75.115:5555/test.flv",
  "http://127.0.0.1:5555/test.flv",
  "http://192.168.1.103:5555/test.flv",
int g_nVedioCounts = 6; // Total amount of sources to monitor

class CAboutDlg : public CDialog
{

public:

    CAboutDlg();

    // Dialog figures

enum { IDD = IDD_ABOUTBOX };

    protected:

    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support

    // Realization

    protected:

    DECLARE_MESSAGE_MAP()

};

CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{

}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{

    CDialog::DoDataExchange(pDX);

}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)

    END_MESSAGE_MAP()


// CvtServerDlg dialog

CvtServerDlg::CvtServerDlg(CWnd* pParent /*=NULL*/)
    : CDialog(CvtServerDlg::IDD, pParent)
{
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CvtServerDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
}

BEGIN_MESSAGE_MAP(CvtServerDlg, CDialog)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    //})AFX_MSG_MAP
    ON_BN_CLICKED(IDC_BUTTON1, &CvtServerDlg::OnBnClickedButton1)
    ON_WM_CLOSE()
    ON_STN_DBLCLK(IDC_VEDIO1, &CvtServerDlg::OnStnDblclickVedio1)
    ON_BN_CLICKED(IDC_BUTTON2, &CvtServerDlg::OnBnClickedButton2)
    ON_STN_CLICKED(IDC_VEDIO2, &CvtServerDlg::OnStnClickedVedio2)
    ON_STN_CLICKED(IDC_VEDIO3, &CvtServerDlg::OnStnClickedVedio3)
    ON_STN_CLICKED(IDC_VEDIO4, &CvtServerDlg::OnStnClickedVedio4)
//}}AFX_MSG_MAP
ON_STN_CLICKED(IDC_VEDIO5, &CvtServerDlg::OnStnClickedVedio5)

ON_STN_CLICKED(IDC_VEDIO6, &CvtServerDlg::OnStnClickedVedio6)

END_MESSAGE_MAP()

// CvtServerDlg information dealing process

BOOL CvtServerDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);

    if (pSysMenu != NULL)
    {
        BOOL bNameValid;
        CString strAboutMenu;
        bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
        ASSERT(bNameValid);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX, strAboutMenu);
        }
    }
}
SetIcon(m_hIcon, TRUE);    // set big icon
SetIcon(m_hIcon, FALSE);    // set small icon
return TRUE;

void CvtServerDlg::OnSysCommand(UINT nID, LPARAM lParam)
{
    if ((nID & 0xFFF0) == IDM_ABOUTBOX)
    {
        CAboutDlg dlgAbout;
        dlgAbout.DoModal();
    }
    else
    {
        CDetl::OnSysCommand(nID, lParam);
    }
}

void CvtServerDlg::OnPaint()
{
    if (IsIconic())
    {
        //Paint...
    }
}
CPaintDC dc(this);

SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

int cxIcon = GetSystemMetrics(SM_CXICON);

int cyIcon = GetSystemMetrics(SM_CYICON);

CRect rect;

GetClientRect(&rect);

int x = (rect.Width() - cxIcon + 1) / 2;

int y = (rect.Height() - cyIcon + 1) / 2;

dc.DrawIcon(x, y, m_hIcon);

}

else
{

CDialog::OnPaint();

}

}

HCURSOR CvtServerDlg::OnQueryDragIcon()
{

return static_cast<HCURSOR>(m_hIcon);

}

void CvtServerDlg::OnBnClickedButton1()
{

//Refresh the sources

char file[1000];

CStringA str,strTime;

CTime t = CTime::GetCurrentTime();


char szCmd[500];

for(int i = 0;i<_nVedioCounts;i++)
{
    // int i = 1;

    str = _szSource[i];

    str = str.Right(str.GetLength()-7);

    str = str.Left(str.GetLength()-14);

    str += strTime;

    sprintf(szCmd,"c:\ffmpeg\sdf.exe %s %ld",g_szSource[i],GetDlgItem(IDC_VEDIO1+i)->GetSafeHwnd());

    WinExec(szCmd,SHOW_OPTION);//play in small window

    sprintf(file,"c:\ffmpeg\ffmpeg.exe -i %s c:\vedio_save\%s.avi",g_szSource[i].str);

    TRACE("%s\n",file);

    WinExec(file,SHOW_OPTION);//Save

}
void CvtServerDlg::OnClose()
{
    g_bRun = false;
    CDialog::OnClose();
}

void CvtServerDlg::OnStnDbclickVedio1()
{
    //details about the first source
    char szCmd[500];
    sprintf(szCmd,"c:\ffmpeg\ffplay.exe %s",g_szSource[0]);
    ::WinExec(szCmd,SHOW_OPTION);
}

void CvtServerDlg::OnBnClickedButton2()
{
    //Playback the videos
    ::ShellExecute(NULL,_T("open"),_T("explorer"),_T("c:\vedio_save"),_T("c:\vedio_save"),SW_SHOW);
}

void CvtServerDlg::OnStnClickedVedio2()
{
    //details about the second source
    char szCmd[500];
void CvtServerDlg::OnStnClickedVedio3()
{
    // details about the third source

    char szCmd[500];
    sprintf(szCmd,"c:\ffmpeg\ffplay.exe %s",g_szSource[2]);
    ::WinExec(szCmd,SHOW_OPTION);
}

void CvtServerDlg::OnStnClickedVedio4()
{
    // details about the forth source

    char szCmd[500];
    sprintf(szCmd,"c:\ffmpeg\ffplay.exe %s",g_szSource[3]);
    ::WinExec(szCmd,SHOW_OPTION);
}

void CvtServerDlg::OnStnClickedVedio5()
{
    // details about the fifth source

    char szCmd[500];
    sprintf(szCmd,"c:\ffmpeg\ffplay.exe %s",g_szSource[4]);
void CvtServerDlg::OnStnClickedVideo6()
{
    // details about the sixth source
    char szCmd[500];
    sprintf(szCmd,"c:\ffmpeg\ffplay.exe %s",g_szSource[5]);
    ::WinExec(szCmd,SHOW_OPTION);
}

Figure 19. vtServerDlg.cpp

5 Conclusion and future work

Market research: despite the definition of the project theme beforehand, I spent some quality
time on the market research for the project and got interested in identifying the future
development’s potential direction. This gave me the motivation to carry out the research and
complete the project well.

Engineering ability: during the process of undertaking the research, I need to wholly learn
about the technical schemes that are there on the ground, draw inspirations from the
available experiences and come up with a solution that matches well my technical requirements and ability.

Initially, I intended to use the framework in solving the problem. However, this approach was very challenging to apply during the subsequent implementation process. This made me move to the use of MCF. I was also made aware of the significance of the open source schemes in computer development. In the system there is a corresponding implementation scheme in the Ffmpeg for each of the important functions needed. This is greatly beneficial to the private developers and the small and medium size companies. I am able to gain the first-hand experience without necessarily participating in the real world project.

Computer science can be compared to an endless wall consisting of various bricks. Each of us makes a contribution to this wall, by doing the best we can to make the world a better place despite the language and geographical barriers.

In future, I intend to have the technique well implemented on the mobile terminal. From the market research, it can be noted that the sphere of the market potential is shifting to the mobile terminals. In the mainland China, the mobile game broadcasting nowadays acknowledged as one of the most next promising opportunity for start-ups. Both the existing broadcasting platforms such as Douyu, Xiongmao and Huya) and the new ones such as the Choushou broadcasting platform are trying to accumulate their own resources in this regard. There is a bright prospect for the firms which specialize in the delivery of relevant
technological devices. The market research that I have conducted on the mobile games is described below as some of the sources of impetus for the continued work in this regard.

Mobile game is a rising market in China. According to iiMedia Research, in 2016, the user base of mobile game has climbed to 523 million with market revenue 78.3 billion CNY. The user base of watching mobile game live webcast has risen to 313 million. We argue that it is because of multiple phenomenal mobile games released in 2016 – Ying Yang Shi, Wang Zhe Rong Yao and Clash Royale. The increasing popularity has attracted well-known professional gaming players, who already have large number of audiences. The players began to webcast new mobile games on mostly Huya TV, Douyu TV, Chushou TV and Panda TV. It produces a new trend in gaming industry – “mobile gaming + webcast”. It has significant advantages over traditional games:

1) It has shorter development periods and smaller initial investment. These features are favorable for venture capitals as such more development teams are willing to devote in mobile games.

2) It requires a smaller development team. Traditional PC games require 12-30 member’s initial development team, but mobile games only require 3-5 member’s development team. It encourages small start-ups.
3) It has a more open platform to operate. In order to be franchised, traditional games should pay the "franchise fee". However, to operate mobile game on smart phones, it is free and open.

Screen recording application is essential in supporting mobile game webcast. It should ensure webcast qualities in 1) no delay in voice especially in large-size games. 2) high image qualities. 3) support different format conversions. Thus this is still an opportunity window for the technique-driven startup in this area.