IEICE TRANS. INF. & SYST., VOL.E92–D, NO.12 DECEMBER 2009

PAPER

Codec-on-Demand Based on User-Level Virtualization

Youhui ZHANG[†a)], ***Member and*** **Weimin ZHENG**[†], ***Nonmember***

SUMMARY At work, at home, and in some public places, a desktop PC is usually available nowadays. Therefore, it is important for users to be able to play various videos on different PCs smoothly, but the diversity of codec types complicates the situation. Although some mainstream media players can try to download the needed codec automatically, this may fail for average users because installing the codec usually requires administrator privileges to complete, while the user may not be the owner of the PC. We believe an ideal solution should work without users' intervention, and need no special privileges. This paper proposes such a user-friendly, program-transparent solution for Windows-based media players. It runs the media player in a user-mode virtualization environment, and then downloads the needed codec on-the-fly. Because of API (Application Programming Interface) interception, some resource-accessing API calls from the player will be redirected to the downloaded codec resources. Then from the viewpoint of the player, the necessary codec exists locally and it can handle the video smoothly, although neither system registry nor system folders was modified during this process. Besides convenience, the principle of least privilege is maintained and the host system is left clean. This paper completely analyzes the technical issues and presents such a prototype which can work with DirectShow-compatible players. Performance tests show that the overhead is negligible. Moreover, our solution conforms to the Software-As-A-Service (SaaS) mode, which is very promising in the Internet era.

key words: *codec, user-level virtualization, on demand*

1. Introduction

The number of videos that can be accessed through the Internet keeps on growing rapidly. As well, a desktop PC or kiosk is usually available at work, at home, and in some public places. However, how to play the ubiquitous videos conveniently on any PC is an open problem. That is, most users may have an annoying experience: after waiting a long time for a video to download, the user is informed that the video cannot be played because there is no suitable codec on the host.

Some mainstream video players have solutions for this situation. For example, Windows Media Player (WMP) can automatically download a suitable codec to play a new video, and RealPlayer has the same function. However, because of the diversity of codec types, they cannot solve this problem completely and the "cannot play" issue still arises frequently. Therefore, there are some open-source players famous for their capabilities to deal with many different codecs.

We performed a test to show the decoding capability of the mainstream media players. More than eighty video files were selected randomly from the Internet to be played by five players, WMP 10.0*, RealPlayer 11, QuickTime 7.0, MPlayer v1.0rc2 [1] and VLC 0.9 [2] on a freshly-installed Windows XP SP2 host. Each player tried to show all videos before it was uninstalled and the next player was installed and tested successively. Successful decoding meant that both the video and audio streams could be decoded smoothly. The results were interesting:

- MPlayer behaved perfectly as it could decode all videos; the exception was that the audio streams of two files could not be handled. The runner-up was VLC: about 94% of the files could be decoded.
- WMP behaved not badly: more than 85% of the files were decoded smoothly; for RealPlayer, the successful ratio was about 77%; QuickTime performed poorly: the ratio was only 30%. All players were configured as "downloading codec automatically".

It appears that the two open source players (MPlayer and VLC) performed much better that their commercial counterparts. However, neither of them has the function to download a needed codec automatically when an unknown video is met, which means that their extensibility is limited and they are not very user-friendly.

For a skilled user, locating and installing a suitable codec or player is easy. However, for an ordinary user, it is difficult to understand the technical codec-related terms, let alone find and install it.

Moreover, in some computing environments, the user may be not the owner of the host and then she has to run players on a locked-down PC without the administrator privilege, which means she cannot install any new software.

We believe an ideal solution should have the following features:

- User-friendliness. That is, the user need not know anything about codec location, installation, and so on. No administrator privilege is needed.
- Program transparency. Current media players should not need to be rewritten to have the new feature.

Manuscript received October 16, 2008.
Manuscript revised April 8, 2009.

[†]The authors are with the Department of Computer Science and Technology, Tsinghua University, Beijing, 100084, China.
a) E-mail: zyh02@tsinghua.edu.cn

DOI: 10.1587/transinf.E92.D.2422

*The latest version is WMP 11. However, it does not affect the credibility of our test because the new features of WMP 11 are focused on the flexibility in managing functions as mentioned at http://www.microsoft.com/windows/windows-vista/features/media -player-11.aspx. It still needs administrator privilege to install an on-demand codec.

- High adaptability. At least for the mainstream media players on most desktop PCs, it should "just work".

The most difficult problem is to provide these features in a compatible way, so that the existing players and codecs can be reused. This paper presents such a solution. It enables the player software to run in a user-mode virtualization environment that intercepts some resource (registry, files/directories, environment variables, etc.) accessing API calls from the player to learn whether a suitable codec exists on the local host or not. If needed, the codec is downloaded from the Internet into the same virtualization environment. Subsequently the resource-accessing API calls are intercepted and some of them will be redirected to the downloaded resource.

Therefore, from the viewpoint of the media player, the necessary codec is available and it can handle the video smoothly. The key point is that, during the whole process, neither system registry nor system folder is modified. Thus, the principle of least privilege [3] is maintained; in addition, the host system is left clean, which minimizes the possibility of conflicts between software.

On the other side, our solution expresses some of the principles of SaaS; that is, the software can run on demand across the Internet without installation and without any special privilege; users' intervention has been decreased to the minimum.

SaaS is believed as a promising delivery and usage mode for software, which was usually used by some enterprise-level software and now it has moved closer to common users. The automatic download mechanism of WMP and RealPlayer can be regarded as a step to the software-as-a-service mode. But because they are based on the traditional DirectShow framework, their solution is not complete—the administrator privileges are still needed. Our solution bridges the gap between the SaaS mode and the legacy software (including codecs)—using the user-level virtualization technology, we have implemented such a solution on the Windows OS, which can work successfully with WMP, RealPlayer, some third-party players, and existing codecs; no source codes of them are modified.

This paper focuses on the offline mode of video playback. Another mode, online, is becoming more and more popular. Some discussion of these two modes will be presented in Sect. 2.

The main contributions of this paper include:

- It proposes a user-friendly codec-on-demand mechanism using user-level virtualization technologies.
- Within the existing Windows video-playback framework, it provides a compatible solution that can apply to DirectShow-based players transparently.
- Based on some mainstream media players and a popular free codec package (FFDSHOW [4]), a functional prototype has been implemented and tested. The results show that this solution performs well in decoding, and its overhead is negligible.

2. Related Work

2.1 On-Demand

On-demand is a feature which responds to the user's desire for instant gratification and immediate use. It is regarded as a friendly and economical service by IT users. Some types of on-demand service include:

- Video-on-demand is a type of video or movie service which allows the viewer to access the media immediately upon subscription, such as streaming Internet or pay-per-view television offerings.
- On-demand computing most frequently refers to a type of computing service where the actual software is presented to the user, once a subscription to the service is successfully processed.
- On-demand software is typically delivered by an application service provider. This type of service offering is also frequently referred to as Software as a Service.

As mentioned in Sect. 1, although video-on-demand is a convenient service, users often encounter the "cannot play" problem because of the lack of a suitable codec. Therefore it appears that combining video-on-demand with software-on-demand would be a smooth solution, which means that a suitable codec could be automatically downloaded on demand with the video. This paper focuses on that idea.

Of course, online video-playback is also a popular mode. The most famous example is YouTube using Adobe Flash technology [5] to provide the video streams. Adobe Flash adopts a proprietary file format and its recent release supports H264 and AAC codecs. Users can watch videos in this mode on diverse hosts because the Adobe Flash Player and web browser plug-in are widely available. However, in this mode, existing videos have to be converted into the given format before delivery. Furthermore, it is unimaginable that the online mode will totally replace the offline mode.

2.2 Multimedia Playback

For desktop PCs, two mainstream media players, Windows Media Player and RealPlayer, dominate the market and both support downloading codecs automatically. For example, WMP can be set to "downloading codec automatically" by selecting a predefined menu option. However, this setting requires the user to be an administrator or a member of the administrators group, as is mentioned in [6]. Furthermore, the diversity of codecs often prevents the auto-download mechanism from finding a suitable codec. A similar problem applies to RealPlayer as well.

Therefore, there are many third-party media players and codec packages; some of them are famous for their abilities to deal with diverse codecs. For instance, MPlayer

can run on many systems and play most media files. However, it does not support the auto-download mechanism. Another instance is FFDSHOW, which is an open-source codec for decoding/encoding many video and audio formats. For a skilled user, locating a suitable codec or player and installing it is not a difficult job. However, for most ordinary users, it is difficult.

Online mode is promising. From the technical viewpoint, the playback mechanisms of both modes are similar. When a video is to be watched, the container program (for the offline mode, it is a multimedia player, while for online it is often a browser) always has to first look for a suitable codec. For the online mode, as mentioned in Sect. 2.1, because the codec formats adopted by the content providers are usually limited, it appears that this mode can decode videos smoothly in most cases.

2.3 User-Level Virtualization

Unlike hardware-level virtual machine technologies (like Xen [7] and VMWare [8]), user-level technologies have a virtualization layer between the operating system and application programs. Every virtualization environment shares the same execution environment as the host machine, and only keeps any divergences from the host in the VM's local state. Therefore, such an environment can have small resource requirements and thus introduce very limited overhead.

User-level virtualization can make some interesting things happen, including running a Windows application without installation, as mentioned in [9]. There are some existing similar user-level implementations, such as Progressive Deployment System (PDS) [10] and Desktop2Go [11].

PDS is a virtual execution environment and infrastructure designed specifically for deploying software on demand while enabling management from a central location. PDS intercepts a selected subset of system calls on the target machine to provide a partial virtualization at the operating system level. This enables the software's install-time environment to be reproduced virtually.

Desktop2Go is a similar solution. In addition to the user-level virtualization, p2p transportation is employed to help the user access his personalized desktop applications, configurations, and data on-the-fly.

2.4 Virtualization on Vista

Windows Vista provides a virtualization mechanism [12] for folders and the registry, called Folder/Registry Virtualization. For example, Registry Virtualization can redirect operations from the global registry store to a per-user location. As mentioned in reference [12], this is an interim application compatibility technology, and Microsoft intends to remove this form of virtualization from future versions of the Windows operating system. It looks like Vista does not provide such APIs for developers. A more important issue is that under Vista (with this feature enabled) Windows Media Player still needs the administrator privilege to make the downloaded codec work [13], as does RealPlayer.

Based on these descriptions, this paper proposes an application-independent solution for Windows that adopts user-level virtualization to implement codec-on-demand that integrates the codec with players to work without explicit installation. It can work on the NT/2000/2003/XP/Vista platforms.

3. Design Philosophy

For Windows systems, the Microsoft DirectShow [14] application programming interface is a widely-used media-processing architecture. Using DirectShow, applications can perform video and audio playback. DirectShow divides video playback into a sequence of steps known as filters. Each filter has input and/or output hooks to connect the filter to others in order to implement different complex functions. There are three main types of filter:

- Source filters: These provide the source streams of data.
- Transform filters: These transform data that is provided from another filter's output, e.g., decompressing a video frame.
- Renderer filters: These render the data.

During the video playback process, the player searches the Windows Registry for registered filters and connects the filters together to provide file parsing, video/audio decompressing and rendering, and playing the target video. More details can be found in [14].

Another out-of-date but still used streaming media technology is Video for Windows (VFW) [15]. It was replaced by the July 1996 release of its COM-based successor ActiveMovie, which became part of DirectShow in 1997.

As mentioned in Sect. 1, the player is running in a user-level virtualization environment and some API calls from the player that access resources (registry, files/directories, environment variables, etc.) will be intercepted by this environment. Thus the following issues should be resolved to realize this technology:

- How to intercept APIs.
- How to learn which codec is needed for a given movie.
- How to judge whether the needed codec is available locally or not.
- How to locate the needed codec across the Internet.
- How to use the codec without explicit installation or modifications of the system registry and system folders (which means it can work without the administrator privilege).

For simplicity, the following discussion will focus on on-demand transform filters but the principle can also be used for source filters.

3.1 How to Intercept APIs

API interception means intercepting calls from the target ap-

plication to the underlying running system and reinterpreting the calls. It is usually used to instrument and extend existing OS and application functionality without access to the source code.

Detours [16], a library developed by Microsoft Research Institute, is used to intercept those APIs accessing the system registry and files/folders. In detail, interception code is applied dynamically at runtime—Detours replaces the first few instructions of the target API with an unconditional jump to the user-provided detour function. This detour function can either replace the target or extend its semantics by invoking the target API as a subroutine.

Detours are inserted at execution time. The code of the target function is modified in memory, not on disk, thus facilitating interception of binary functions at a very fine granularity. Unlike the system hook mechanism, the procedures in a DLL can be detoured in one execution of an application, while the original procedures are not detoured in another execution running at the same time.

Based on Detours, we have built Wrapper APIs that inject a wrapper DLL into the target player's virtual address space, as described in [16]. For example, when an application uses an interpreted WIN32 API to access a file, the wrapper API will be called first.

3.2 How to Learn Which Codec Is Needed

Because all file-access APIs are intercepted, when the media player opens and reads a movie file using the CreateFile and ReadFile APIs, the intercepting code can beforehand parse the file format and locate the corresponding field of the compression codec. For example, one field (fccHandler) of the *strh* structure of the widely-used AVI file header indicates the codec to be used.

In detail, it is a sequence of four bytes (FourCC) used to uniquely identify data formats. For example, XVID stands for the MPEG-4 XVID codec; DIV3 stands for DivX MPEG-4 and 3IV2 is used for "3ivx Delta 4.0." Complete information can be obtained from [17].

3.3 How to Judge Whether the Needed Codec Is Available Locally

DirectShow is based on the Component Object Model (COM) and any DirectShow-compatible codec should be a COM object. Therefore, all codecs are managed as ordinary COMs, which means that they are registered in the system registry and each is referenced by a unique CLSID (CLaSs IDentifier).

In detail, a COM object of ActiveMovie Filter Class Manager (its CLSID is 083863F1-70DE-11d0-BD40-00A0C911CE86) is provided by Windows, which manages all registered DirectShow codecs. In the system registry, codecs' information are gathered under their subkeys, including all codecs' CLSIDs.

Moreover, DirectShow provides an interface to enumerate media types supported by any transform filter and each media type is described by a unique GUID (Globally Unique IDentifier). Therefore, we have to map a media FourCC to the related GUID. Fortunately, Microsoft reserved a range of 2^{32} GUIDs for representing FourCCs: these GUIDs are all of the form XXXXXXXX-0000-0010-8000-00AA00389B71 where XXXXXXXX is just the FourCC code. For example, the GUID for YUY2[†] is 32595559-0000-0010-8000-00AA00389B71.

Thus, to judge whether the needed codec is available locally or not, the first step is to enumerate all codecs' CLSIDs and create their instances. For each instance, all media types supported are also enumerated, and can be compared with the given FourCC.

For VFW codecs, the look-up method is straightforward: the COM mechanism is also used to locate the suitable codec. One COM object (its CLSID is 33D9A760-90C8-11D0-BD43-00A0C911CE86) gathers all VFW codecs' information under its registry key, including a value named FccHandler, which has the same meaning as the corresponding field of the *strh* structure mentioned in the previous section.

3.4 How to Locate the Needed Codec Across the Internet

Here we just prove its feasibility from a technical viewpoint.

Fundamentally, the content producer or publisher could give hints—which can be embedded in some reserved fields of media files—on how to find the suitable codec.

Another practical solution, which is used in this design, is to employ one of the many third-party network resources to locate a suitable codec. For instance, FFDSHOW is an open source package of DirectShow codecs for decoding/encoding many video and audio formats. Its web site declares that FFDSHOW can decode 3ivx, DivX, FFDS, FFV1, H264, Indeo 3, Theora, TSCC, XviD, and so on. Thus, through gathering information from such web resources, most needed codecs can be found. An unknown but distinctive media player, GOM player [18], is an example. Its web site collects many codec installation files, so that the user can be guided by GOM to access the proper codec when the "cannot play" problem occurs. However, user intervention and administrator privileges are still needed during this process.

One helpful contribution would be to construct a web forum so that any skilled user could voluntarily upload useful information on the relationship between video FourCC values and suitable codecs. Such grassroots behaviors have been proved successful many times in the Internet's growing history.

However, this is a tactic rather than a technical issue and how to construct such a web site is beyond the scope of this paper.

The last "how to" is the key point of our design, and is described completely in the next section.

[†]A Packed YUV Format.

4. Codec on Demand

The issue is how to use a codec without installation and leave no modifications in the system registry and folders.

Most Windows applications need to be installed before they can run normally. Even "green" applications that can work without installation often save their customizations into the system registry and/or into configuration files located in some system folders. Therefore, for a user with only guest privileges, it is usually impossible to install any new software (including a codec).

In our view, an application contains two parts: Part 1 contains all of the files, folders, registry keys, and environment variables created by its installation process; Part 2 is its customizations produced at runtime.

To realize the design, we have to enable Part 1 to be downloadable and make the application run in a virtualization environment where it can access Part 2 in an isolation mode. We employ user-level virtualization to obtain these functions.

This work consists of two tasks, the installation snapshot and the runtime system design.

4.1 Installation Snapshot

To make Part 1 downloadable, we have to capture the modifications made by the installation process of a codec. There are two main sites of modifications: the registry and files/folders. Some existing tools can be used to catch them. The one we adopt is InstallWatch [19]. It is a system monitoring tool that tracks changes to the computer's hard disk, registry, and .ini files when a new application is being installed.

In our implementation, a target codec is installed on a clean Windows system. During this process, InstallWatch is used to log the files created or modified by this installation, as well as the registry items and their contents. Then, the files/folders created or updated are copied to a special folder, called the private folder. Similarly, the contents of the modified registry keys are collected to be stored in a separate file, the private registry file.

Both the private folder and the private registry file are packaged into a compressed file, which is placed on our web site and can be downloaded on demand. After downloading it will be unpacked into a writable folder (like the system temp directory) and the directory hierarchy is kept intact.

4.2 Runtime System

The second issue is how to make the downloaded resource accessible by the player's executable file. API Interception is also employed to do this.

When the player is injected, the interception code deals with resource access requests first: if one of the registry keys of Part 1 is to be accessed, the injected code can return the corresponding value from the private registry file; otherwise the original API will be called to visit the system registry. The same flow is also used for visits to files and folders.

In detail, the captured installation snapshot can be divided into six categories:

- Added registry set. It contains the entries created by the installation.
- Modified registry set. It contains the entries whose values or sub-keys have been modified or deleted.
- Deleted registry set. Those entries deleted by the installation are included, so that the entries in this set will not be accessed at runtime.
- Added file set. This is similar to the added registry set, including new files and new folders created by the installation.
- Deleted file set. This is similar to the deleted registry set.
- Modified folder set. For any file or folder in the added/deleted file set, its parent folder will be included in this set.

Take registry accesses as an example—when the registry key of ActiveMovie Filter Class Manager is opened and enumerated by the player using *RegOpenKeyEx* and *RegEnumKeyEx*[†] APIs (which are usually used for the codec query), the following steps will be executed sequentially:

- Because this registry key is in the modified set (as codecs should add sub-keys there), both the private registry and the system registry should be queried and the results will be merged before return. Therefore from the viewpoint of the media player, the needed codec does exist on the host.
- Then it will try to open the corresponding CLSID key of the codec to locate the codec file (usually a .DLL or .ax file). This item is definitely in the added set so that it will be queried just in the private space.
- Finally, the codec file will be loaded. The actual codec file location often differs from that described in the registry value, because the environment of the snapshot is usually different from the actual running host. Then, access redirection will be achieved by our interception code for File APIs.

It is necessary to note that the codec file itself will access its registry items for configurations. Because it is loaded in the target player's virtual address space, these accesses are intercepted by the inserted code and thus it can work without any explicit installation.

More principles of the access process can be found in [11]. In summary, the philosophy is that any modification is always saved in the private space while any query will return the combination of results from both registries. In addition, if there is any duplication, the private value has higher priority. For APIs that access the file system, a similar philosophy is adopted because folders can be regarded as registry

[†]*RegOpenKeyEx* is used to open an existing registry key and *RegEnumKeyEx* enumerates all sub-keys of the open key.

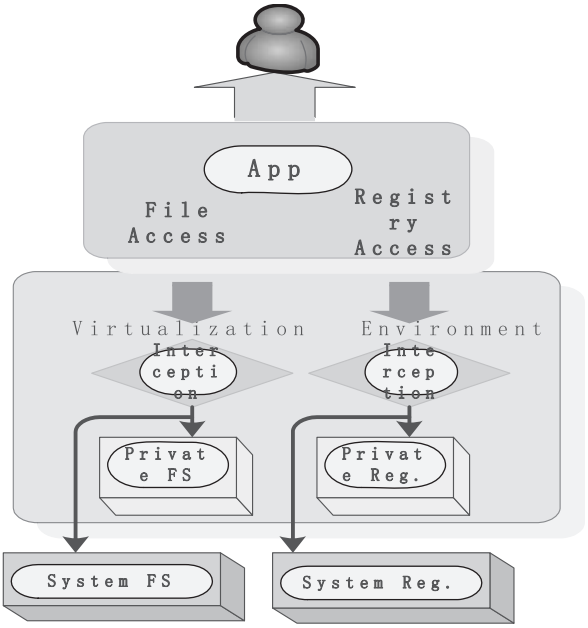


Fig. 1 The virtualization environment.

keys and files can be regarded as values.

This means that the configurations and files of a codec can be isolated from the host environment. In addition, any modification happening at runtime will be stored in the private registry file or the private folder, instead of the system's default position. Therefore no trace will be left in the system registry and folders. The environment is illustrated in Fig. 1.

5. Implementation and Tests

According to the above design, we have implemented such a solution on the Windows platform.

The frontend of our prototype contains an executable file and two DLL files that can be stored on the user's portable storage device or can be downloaded from our web site; no installation is needed (this means the executable can be used with the Guest privilege). Besides the interception functions, the main component of the frontend is a private registry, which is a complete registry system that provides access APIs just as the Windows OS does. It works like a small subset of WINE [20], an open source implementation of the Windows API on top of UNIX.

The executable is used to launch the target media player and then inject a wrapper DLL into the target's virtual address space.

The backend is a web server where installation snapshots of codecs are stored. Thus, a codec can be downloaded if the frontend demands it. The code used in our test is FFDSHOW, which is a free codec package containing many DirectShow decoding filters for decompressing DivX, XviD, H.264, FLV1, WMV, MPEG-1 and MPEG-2, MPEG-4 movies, as well as numerous other video and audio formats.

The snapshot size of FFDSHOW (version 1.0.5.2036; build time is 06/07/2008) is about 8.36 MB and its zip version is less than 3.5 MB. So, the downloading time is very short, especially compared with that of a large video. The size of its private registry is about 46 KB; it contains two types of keys:

- ffdshow-related: some items about the configurations of ffdshow itself;
- CLSID-related: CLSIDs of FFDSHOW codec objects, managed by the ActiveMovie Filter Class Manager COM object.

The work flow of the whole playback process of our solution is illustrated in Fig. 2.

5.1 Function Test

In the test, the frontend should access the web server for the FFDSHOW snapshot across the Internet. Hence the placement of the server(s) is decisive for the access performance. Here we assume some edge server(s) can be found to provide the download service, so that the web server is located in the CERNET[†] as well as the frontend. This is a common case now: the Content Delivery Network has been widely used for software downloading, and as claimed by Akamai, the world leading CDN provider, most visit requirements can be fulfilled by some edge server(s) just a single hop away.

On the frontend side, a freshly-installed Windows XP system was used as the test environment. Three media players, Windows Media Player (version 10), RealPlayer (version 11) and Media Play Classic (version 6.4.9), were used for playback. Media Player Classic is open source software primarily based on the DirectShow architecture, and it can automatically use DirectShow transform filters. Therefore, it was selected as the representative of open-source players since neither VLC nor MPlayer supports DirectShow.

In addition, ten AVI files (compressed by XVID, DIVX 5/6, 3ivx D4, Chinese AVS, TechSmith Camtasia, and FFDS codecs) have been tested. All tests were performed with guest privileges.

1) On the test host, none of the video files could be decoded by any player because no suitable codec was installed.
2) When each player was launched by our frontend, all videos could be decoded after a few seconds—the inserted code contacted the web server to download the installation snapshot of FFDSHOW and unpacked it into the current user's temporary folder. Then the information of six sets (mentioned in Sect. 4.2) and registry contents were loaded, and all API calls accessing

[†]CERNET (Chinese Education & Research Network) is the second largest network backbone in China. Its backbone bandwidth has been up to multiples of 10 Gbps and regional bandwidth up to multiples of 2.5 Gbps. Now there are about 1,500 universities and institutions connected and more than 20 million end users.

files and registries were handled by the inserted code as described in Sect. 4. From the viewpoint of the media player, the necessary codec existed, while neither system registries nor system folders were modified.

The results of playback are summarized in Table 1. There are more than 80 types of format supported by FFDSHOW and it is very difficult to test them all. But all codecs in FFDSHOW are compatible with the DirectShow architecture so that we believe our solution can deal with them gracefully.

5.2 Performance Test

On the other hand, application run times are also key usability metrics of our prototype, which includes two extra overheads: the latency caused by the API interception and the startup time delayed by the download process.

As mentioned in [16], the former overhead is very limited: less than *2.5 %* for *one* intercepted system call. Compared with the whole playback process, this overhead will be much less because only a small fraction of system calls are intercepted in our case.

Therefore, we focus on the startup latency.

In the test environment as described by Sect. 5.1, the average access time for the requested snapshots (including the judgment processing time, the downloading time and the decompression time) was about *16.2* s. Compared with the long video playback time, this latency is also trivial.

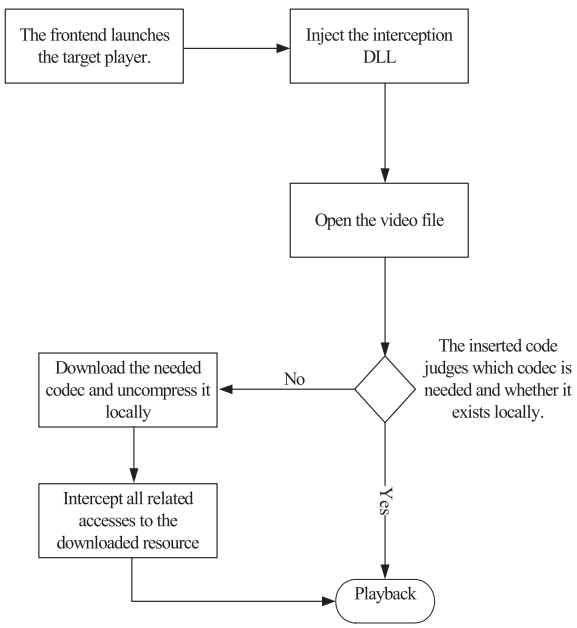


Fig. 2 The work flow of the whole playback process.

Table 1 Playback effects.

Codec	FourCC	Windows Media Player	RealPlayer	Media Player Classic
XVID	XVID	OK	OK	OK
DIVX 5	DX50	OK	OK	OK
DIVX 6	DX50	OK	OK	OK
3ivx D4	3IVX	OK	OK	OK
AVS	CAVS	OK	OK	OK
TechSmith Camtasia	TSCC	OK	OK	OK
FFDS	FFDS	OK	OK	OK

6. Discussion

6.1 Security and Privacy

The principle of least privilege, which refers to the concept that all users should launch applications with as few privileges as possible, is widely recognized as an important design consideration in enhancing the protection of data and functionality from malicious behavior. Our solution enables the downloaded codec to work without the administrator privilege, and is therefore considered to improve system security.

On the other side, this solution only writes modifications to temporary folders on the host. This isolation can keep the local OS pristine and maintain the user's privacy.

6.2 Other Platforms

For Linux, because it is a more open platform and several open-source software are used as the mainstream media players, to integrate such a codec-on-demand function is very straightforward. The problem is that, because Linux lacks a general application programming interface for media-streaming like DirectShow in Windows (Simple DirectMedia Layer [21] looks like a possible candidate, but it works on a lower level, only providing access to audio, keyboard, mouse, joystick, and 3D hardware), application-specific codes have to be written for different players.

6.3 Data Is the Center

Based on the traditional view regarding programs as the center, data is completely separated from code, and programs (including codecs) should be explicitly delivered, installed, and configured, usually with administrator privileges. We believe that, for an ordinary user, this mode is not good enough because she has to be involved in the whole process even though she just wants to see a movie.

On-demand software (also frequently referred to SaaS) is a software distribution model in which applications are hosted by a service provider and made available to customers over a network on demand. It is regarded as a promising mode and used more and more widely today. However, legacy software cannot be used in this mode directly. Thus, our solution constructs a bridge between legacy codecs and the on-demand usage mode, so that the user can play back diverse videos on-demand with neither interventions nor administrator privileges.

One similar latest work is Xax [22]. Xax is a browser

plug-in that enables developers to leverage existing tools, libraries, and entire programs to deliver feature-rich applications across the Internet. For Xax, the source code of legacy applications and the existing tool chain have to be modified. Therefore, this solution can leveraging the legacy code efficiently, not the applications directly.

A similar "cannot play" problem exists in many other fields. Some existing research has proposed the idea of integrating code with data. For instance, VXA [23] is an archival storage system that uses virtual machines to deal with compressed data archives against changes in data compression formats. It packages executable decoders into the compressed archives along with the compressed data itself.

Inspired by this, we believe it is very interesting in the next step to propose a general format of multimedia files that can integrate codec with data, not only for ubiquitous playback, but also to be future-proof. That means to preserve the usability of digital content so that they can be accessed smoothly in the future even when the internal format is totally out-of-date.

7. Conclusions

How to decode various movies on any compatible PC without users' intervention and without administrator privilege is a real problem.

Our analysis shows that on one side, there are plenty of player and codec resources across the Internet; on the other side, because of implementation problems and/or lack of sufficient privileges, it is inconvenient for average users to make use of them.

Therefore, based on user-level virtualization technologies, we have implemented a mechanism to link suitable codecs and media players on demand.

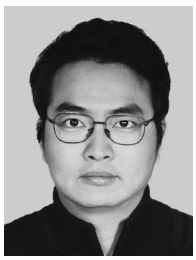
The solution only depends on Windows system APIs and the DirectShow framework; the latter is used by most Windows-based players. Therefore, our solution is believed to work well with most players on Windows.

Acknowledgments

The work was supported by the High Technology Research and Development Program of China under Grant No. 2006AA01Z111 and the National Natural Science Foundation of China under Grant No. 60773147.

References

[1] http://www.mplayerhq.hu/
[2] http://www.videolan.org/vlc/
[3] http://en.wikipedia.org/wiki/Principle_of_least_privilege
[4] http://sourceforge.net/projects/ffdshow-tryout
[5] http://en.wikipedia.org/wiki/Adobe_Flash
[6] http://www.microsoft.com/windows/windowsmedia/player/faq/codec.mspx
[7] XenEnterprise, http://www.xensource.com/products/xen_enterprise/
[8] VMWare, http://www.vmware.cn/
[9] Y. Yu, F. Guo, S. Nanda, L.-C. Lam, and T.-C. Chiueh, "A feather-weight virtual machine for Windows applications," Proc. Second ACM/USENIX Conference on Virtual Execution Environments (VEE'06), June 2006.
[10] B. Alpern, J. Auerbach, et al., "PDS: A virtual execution environment for software deployment," Proc. First ACM/USENIX International Conference on Virtual Execution Environments, March 2005.
[11] Y. Zhang, X. Wang, and L. Hong, "Portable desktop applications based on P2P transportation and virtualization," Proc. 22nd Large Installation System Administration Conference (LISA '08) USENIX Association, San Diego, CA, Nov. 2008.
[12] http://msdn.microsoft.com/en-us/library/bb530198.aspx
[13] http://windowshelp.microsoft.com/Windows/en-US/Help/3d0c5a49-8f61-45cc-8a7d-38c4695ba9291033.mspx
[14] http://msdn.microsoft.com/en-us/library/ms783323(VS.85).aspx
[15] http://msdn.microsoft.com/en-us/library/ms713492(VS.85).aspx
[16] G. Hunt and D. Brubacher, "Detours: Binary interception of Win32 functions," Proc. Third USENIX Windows NT Symposium, July 1999.
[17] http://www.fourcc.org/
[18] http://www.gomlab.com
[19] http://tejasconsulting.com/open-testware/feature/installwatch.html
[20] http://www.winehq.org/site/docs/wineusr-guide/index
[21] http://www.libsdl.org/
[22] J.R. Douceur, J. Elson, J. Howell, and J.R. Lorch, "Leveraging legacy code to deploy desktop applications on the Web," Proc. 8th USENIX Symposium on Operating Systems Design and Implementation, CA, USA, Dec. 2008.
[23] B. Ford, "VXA: A virtual architecture for durable compressed archives," Proc. 3rd USENIX Conference on File and Storage Technologies, Dec. 2005.



Youhui Zhang is an Associate Professor in the Department of Computer Science at the University of Tsinghua, China. His research interests include portable computing, network storage and microprocessor architecture. He received his Ph.D. degree in Computer Science from the same university in 2002.

Weimin Zheng is a Professor in the Department of Computer Science at the University of Tsinghua, China. His research interests include high performance computing, network storage, parallel compiler.