Computer Science and Application 计算机科学与应用, 2020, 10(6), 1166-1179
Published Online June 2020 in Hans. http://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2020.106121

Hans汉斯

Research on Mechanism Optimization of USB Redirection Transmission Based on SPICE Protocol

Liang Cheng

City College, Kunming University of Science and Technology, Kunming Yunnan
Email: ynchengliang@126.com

Received: May 22nd, 2020; accepted: Jun. 3rd, 2020; published: Jun. 10th, 2020

Abstract

With the development of cloud computing and virtualization technology, desktop cloud based on SPICE (Simple Protocol for Independent Computing Environment) has been widely applied in different scenarios. In view of the limited performance of USB redirection data transmission based on SPICE protocol, we analyzed the factors affecting data transmission performance of USB redirection recognition process, and proposed an optimization scheme. The scheme improved overall performance of the USB redirection recognition process, by optimizing the USB controller, data compression and disk read/write IO. The research results showed that the scheme effectively reduced the waiting time of USB device recognition, improved the performance of USB redirection and enhanced the user experience.

Keywords

Desktop Cloud, SPICE Protocol, USB Redirection, Transmission

基于SPICE协议的USB重定向通道传输机制优化研究

程　良

昆明理工大学，城市学院，云南　昆明
Email: ynchengliang@126.com

收稿日期：2020年5月22日；录用日期：2020年6月3日；发布日期：2020年6月10日

摘　要

随着云计算和虚拟化技术的不断发展，基于SPICE (Simple Protocol for Independent Computing

文章引用: 程良. 基于 SPICE 协议的 USB 重定向通道传输机制优化研究[J]. 计算机科学与应用, 2020, 10(6): 1166-1179.
DOI: 10.12677/csa.2020.106121

Environment)的桌面云被广泛应用在各种应用场景。针对目前基于SPICE协议的USB重定向数据传输性能受限的问题，详细分析了影响USB重定向识别过程数据传输性能的因素，提出了优化方案。方案通过优化USB控制器、数据压缩和磁盘读写IO三个方面，提升了USB重定向通道识别过程的整体性能。实验对USB重定向识别的过程进行验证，结果表明，该方案有效地减少了USB设备识别的等待时间，改善了USB重定向的性能，提升了用户使用体验。

关键词

桌面云，SPICE协议，USB重定向，传输

1. 引言

随着云计算技术和虚拟化技术的不断发展，桌面云作为云计算的典型应用之一，可以通过瘦客户端或者其他任何与网络相连的设备来跨平台访问应用程序以及整个客户桌面，具有易于管理、安全性高、应用环保及成本低等特点，被广泛应用在桌面集中管理、信息安全、远程办公等各种应用场景中。桌面云因其具有“桌面统一、资源共享”的特点，能为高校教学和办公提供高效灵活的环境。该技术的应用为师生提供了个性化的桌面服务，使机房和实验室的教学安排更加灵活，促进了虚拟化实验资源的共享，极大地提高了 IT 资源管理的水平和效率[1] [2] [3]。

但与传统物理 PC 机桌面相比，桌面云对 USB 设备的支持尚不成熟。桌面云在使用中暴露出高速 USB 设备读取性能受限方面的问题。USB 外接设备凭借数据传输速率快、携带便利、即插即用等显著优势，已经成为当前使用最广泛的计算机外设[4] [5]。因此，在桌面云实际应用中，如果不能有效解决 USB 重定向通道传输性能的问题，终端用户也就不会热衷于桌面虚拟化技术。目前，在桌面云环境下实现对 USB 设备的支持，一般来说有两种方式，一种是基于 USB 总线的虚拟化重定向，又称为 USB 端口重定向，通过替换 USB 总线驱动实现，比如文献[6] [7]提到的方法；第二种是 USB 设备驱动的重定向，又称为设备重定向，即替换掉 USB 设备驱动。与 USB 端口重定向相比，USB 设备重定向技术的整个工作流程中少了很多的代理环节，所以在整个数据传输过程，无论是数据的传输效率，还是网络延时都会比 USB 端口重定向技术更优，也更适合数据流量较大，同时对延时也比较敏感的外设[8]。本文所研究的基于 SPICE 协议的 USB 重定向正是基于后者实现。虽然基于 SPICE 协议的桌面虚拟化技术发展十分迅速，国内外各大公司的产品也相继问世，但是，桌面云在 USB 识别方面还是存在着一些问题[9]，如识别速度偏慢等。由于虚拟化环境下虚拟机的客户系统访问主机的 USB 设备的 I/O 性能存在问题，文献[10]选择 QEMU [11]作为开发的基础，对直接硬件访问进行尝试，通过改进 QEMU 的体系结构，借助 USB 设备访问接口库，实现了虚拟机中客户系统直接访问真实 USB 接口设备，来提高虚拟机 I/O 性能。文献[5]为了解决 USB 重定向过程中高速 USB 设备读取性能受限等问题，提出了 USB 映射与网络映射相结合的融合方法，对特定类型、高速的 USB 设备进行自动识别并采用响应机制，提升了 USB 设备的读写性能和网络效率。但是，此方案的缺点在于需要改变 USB 设备的使用习惯。USB 设备重定向实质就是通过网络将位于终端机上的物理 USB 设备和位于虚拟桌面服务器端的虚拟 USB 设备之间建立连接，并将对于 USB 设备的 URB 请求及应答进行重定向[12]。文献[12]中指出通常以 SPICE 协议为基础的 USB 重定向技术大多都是

运用了 USBRedir 和 LibUSB 等技术，依靠 SPICE 的服务端和客户端的 Spice-GTK [13]来实现。这些原生的实现方式，并没有对 USB 重定向数据传输性能问题做优化处理，在极端情况下，会影响用户使用体验，甚至造成无法正常使用的情况。

因此，为了让用户在使用桌面云时，能够获得接近传统物理 PC 机使用 USB 存储设备的体验，本文针对 USB 重定向识别过程数据传输的性能，结合实际应用场景，以 SPICE 协议为基础，通过优化 USB 重定向识别过程整体性能，来有效提高 USB 存储设备重定向识别速度，进而提高用户使用桌面云的整体体验。

2. SPICE 架构

桌面云运用虚拟化技术，把计算资源进行池化，并通过网络为多形态的用户终端提供资源池中的应用程序或者传统 PC 桌面[4]。桌面虚拟化技术的核心是虚拟桌面传输协议，在保证和改善用户交互体验质量时发挥着重要作用[14]。SPICE [15]是一个开源的虚拟桌面协议，提供了动态自适应、高性能的虚拟桌面服务，实现了远端主机和设备(键盘、鼠标等)的接入。SPICE 主要由 SPICE 协议[16]、SPICE 服务端、SPICE 客户端以及 SPICE 相关的 QXL 设备组成，在虚拟机内部还有 QXL 驱动。如图 1 所示，SPICE 使用 QEMU 作为虚拟化层，QEMU 作为中间件与 KVM 和 SPICE 进行交互。虚拟机运行于 QEMU 进程里，SPICE 服务端作为一个基于 libspice 写的静态库被 QEMU 编译进去。SPICE 协议通过各种不同的通道实现客户端和服务端之间的通信，利用不同的通道传输不同类型的数据，每一种通道用来专门负责一种类型的数据传输与通信，每个通道中的内容都可以通过相应的图形命令数据流或代理命令数据流传输[17]。SPICE 服务端以动态连接库的形式与 KVM 虚拟机整合，通过 SPICE 协议与客户端进行通信。SPICE 协议架构如图 1 所示。

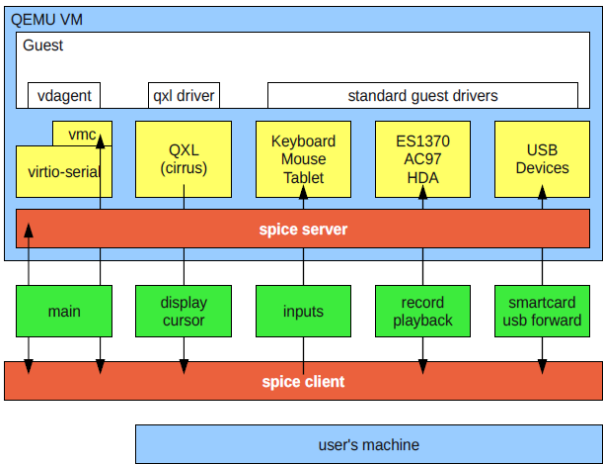

Figure 1. Structure of SPICE protocol
图 1. SPICE 协议架构

SPICE 协议定义了以下几种通道，每个通道针对一个远端设备，通道支持 SPICE 自定义：

1) 主连接通道(main)：主要的 SPICE 会话通道；
2) 显示通道(display)：处理图形命令，接收远程显示更新；
3) 输入通道(inputs)：发送鼠标和键盘事件；
4) 光标通道(cursor)：接收指针形状和位置；
5) 播放通道(playback)：从服务器接收音频流；

6) 录音通道(record)：在客户端捕获音频；

7) USB 通道：将 USB 设备重定向到 Guest 操作系统中。

3. 基于 SPICE 协议的 USB 重定向识别原理和性能分析

基于 SPICE 协议的 USB 重定向是将客户端本地的 USB 设备映射到 QEMU 虚拟机中的过程，这样虚拟机在 USB 驱动程序的配合下，就可以像访问本地 USB 设备一样，访问客户端的 USB 设备。本地客户端将 USB 外设通过网络方式，映射到服务端的虚拟化层；服务端虚拟机管理程序 QEMU 通过创建虚拟的 USB 重定向字符设备，供虚拟机中的驱动层来访问。

当客户端本地的 SPICE 客户端连接到服务端 QEMU 虚拟机时，其会同 spice-server 建立通道连接，如果客户端有 USB 设备需重定向，则会同服务端 spice-server 建立 USBRedir 通道来收发数据，服务端 QEMU 会模拟出 Redirect 字符设备，同 spice-server 通道关联，这样从通道接收到的数据就会提交给 Redirect 字符设备，然后由控制器提交到虚拟缓冲区供虚拟机读取；如果 Redirect 字符设备有数据需要发送，则写入到对应的 spice-server 通道，由 USBRedir 通道来负责将数据发送到客户端。基于 SPICE 协议的 USB 设备重定向架构图如图 2 所示。

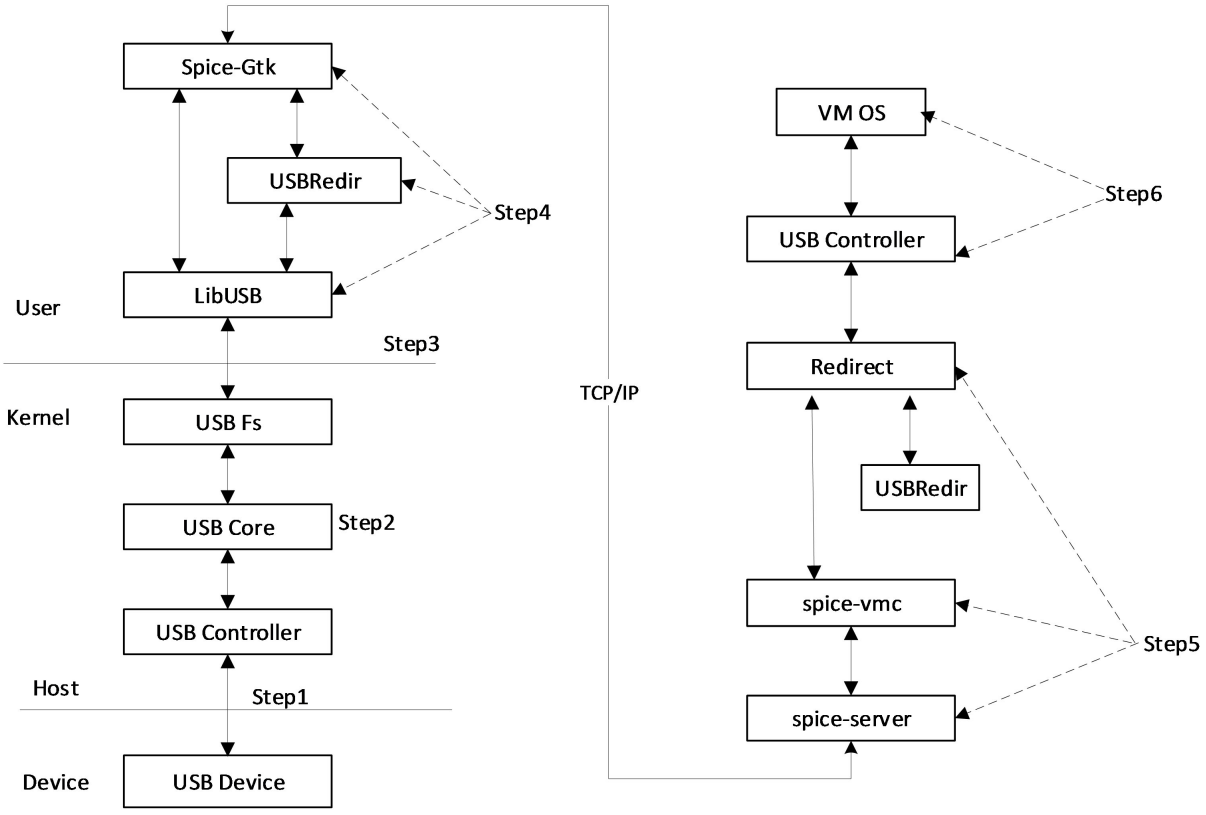

Figure 2. USB redirection architecture based on SPICE protocol
图 2. 基于 SPICE 协议的 USB 重定向架构图

从图 2 可以看出：

1) 客户端 USB 设备同内核态的控制器进行交互，二者之间通过端点 endpoint 建立逻辑上对应的管道 pipe 来通信；

2) 内核态经过 USB 核心处理负责进行驱动的配置和读取操作，向上提供了接口给 USB Fs；

3) 用户态则通过 LibUSB 的 API 来配置和读写操作；

4) Spice-Gtk 则直接调用 LibUSB 相关接口来初始化和注册对应的事件处理函数，然后通过 USBRedir 模块来读取数据，并经过协议封装后，通过 SPICE 的 vmc 通道发送给服务端；

5) 服务端 spice-server 同 spice-vmc 模块进行耦合，从 spice-server 接收到的数据，通过 qemu-char 字符设备接口写入到虚拟的字符设备 Redirect 中，然后经过 USBRedir 部分的协议解析后，提交到虚拟缓冲区，供 USB 控制器处理；

6) USB 控制器获取的数据则写入到 Redirect 字符设备中，然后经过 USBRedir 协议的处理后，调用 spice-vmc 的接口，最终通过 spice-server 的通道发送给客户端。

在传统 PC 上使用 USB 时，USB 设备直接连接 PC USB 控制器，识别速度快，用户体验好，USB 设备的数据传输不会造成其它影响，而对于基于 SPICE 协议的桌面虚拟化应用来说，USB 设备是通过客户端和主机的网络重定向到虚拟机中的，由于各种延迟的存在导致 USB 识别速度普遍偏慢，极大的影响用户使用体验。本文从 USB 重定向识别过程出发，分析基于 SPICE 协议的 USB 重定向识别原理和传输数据的性能，进而提出相应的优化方案。

3.1. 基于 SPICE 协议的 USB 重定向识别原理

USB 识别过程也就是重定向开始的过程，当客户端检测到有 USB 设备插上或者有 USB 已经存在，且该 USB 设备允许重定向，则首先会通过 LibUSB 的接口打开该设备，获取到设备句柄，然后调用 USBRedir 提供的 usbredirhost_set_device 接口来走重定向逻辑，此时，USB 重定向已经开始。

usbredirhost_set_device 会根据设备句柄获取到设备结构，然后会调用 LibUSB 的接口来获取设备描述符和配置描述符，再调用 LibUSB 的 libusb_set_auto_detach_kernel_driver 或 libusb_detach_kernel_driver 接口将其从内核态 detach，最后调用 libusb_claim_interface 对设备进行宣告，至此，/dev/下对应的设备会消失，彻底由 LibUSB 来接管。

接下来会对 USBRedir 进行适当的配置，包括端点信息、最大包长等。

最后会复位设备，获取设备的速率模式，再将设备的接口描述符有关信息、端点有关信息写入到队列，SPICE 中的 USB 重定向通道会负责对该队列中的数据发送到服务端。

至此，服务端的虚拟机中设备管理器中已经可以看到相应的 USB 设备了。

后续过程，磁盘控制器会进行磁盘的读写，才能看到对应的 USB 磁盘。

3.2. 基于 SPICE 协议的 USB 重定向识别过程数据传输性能分析

对于 USB 设备，根据其设备类型的不同，主要有 SCSI/HID/UVC 等几种协议。SCSI 协议主要用于 USB 存储设备(U 盘、USB 硬盘、USB 读卡器等)，HID 作为人机接口设备采用的协议(键盘、鼠标等)，UVC 主要用于 USB 视频设备(Camera、DV 等)。

在虚拟机中通过 Bus Hound USB 抓包工具分析确实有大量的数据会通过 Data In 发送到主机，为了确定其数据内容和采用的协议，这里通过 Wireshark 来抓包，并且对 USB 协议进行分析，可以知道，大量的数据传输是主机需要通过 SCSI command 命令来读取 U 盘对应的 Flash 扇区导致，即主机需要知道 U 盘的扇区相关的信息，才能显示该磁盘。可以参考文档《SCSI Commands Reference Manual.pdf》。

通过 Wireshark 抓包，对 request 请求报文(如图 3 所示)和 respond 响应报文(如图 4 所示)的协议分析可以验证，此过程确实采用 SCSI 的 CDB Read 命令来读取了 LUN 上的大量的设备块信息，且数据量较大。

为了方便对整个数据传输过程中的理解，这里给出了客户端 USB 大容量存储设备数据流图，如图 5 所示：

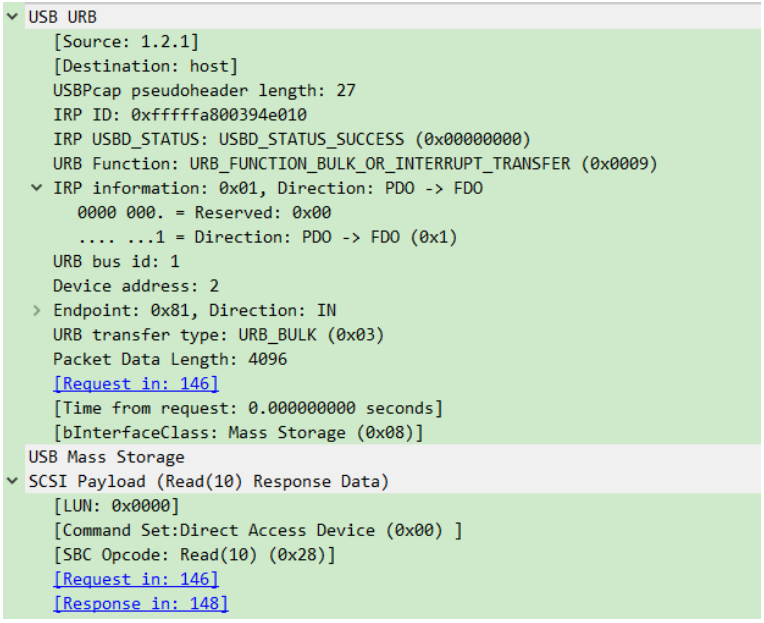

Figure 3. URB request message
图 3. URB 请求报文

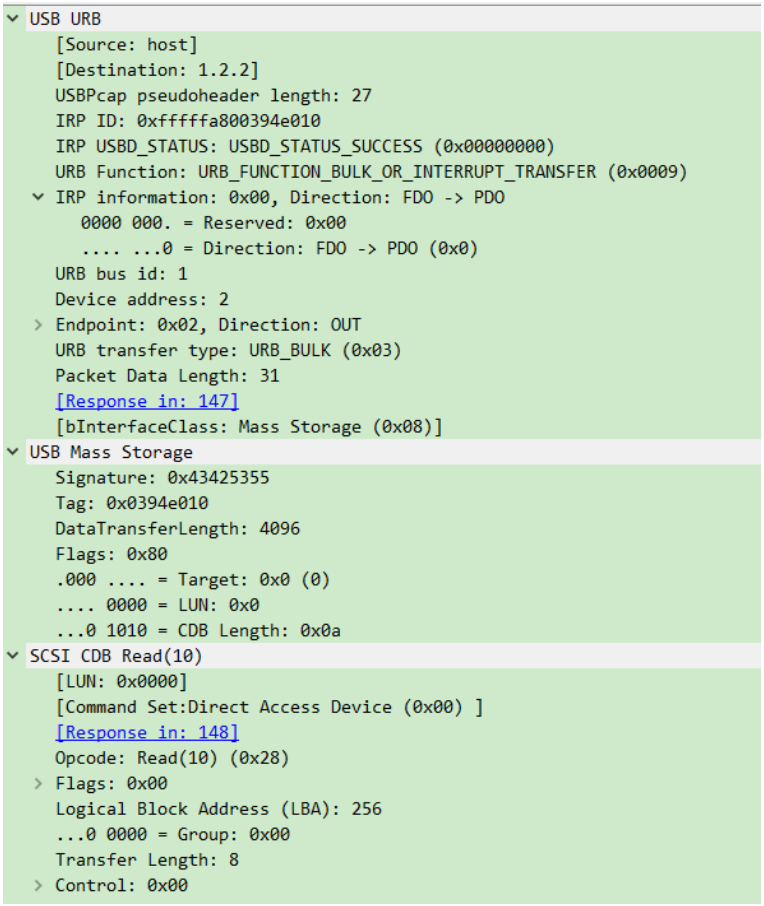

Figure 4. URB response message
图 4. URB 响应报文

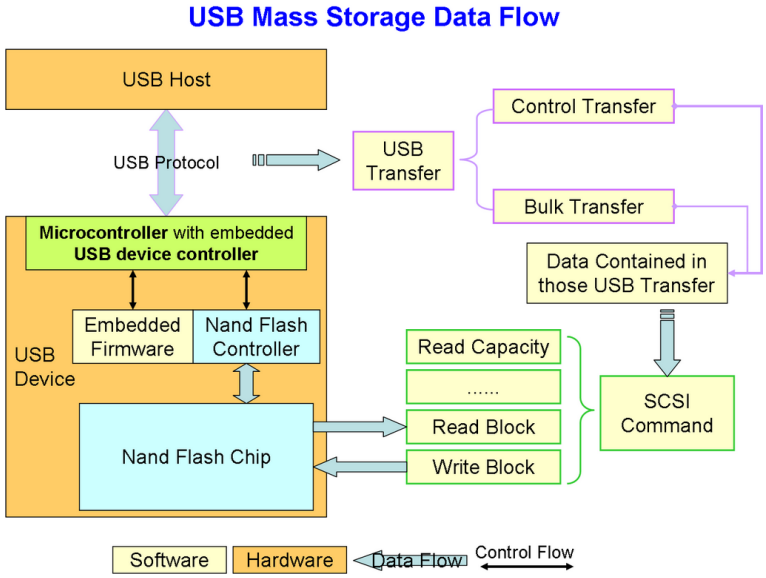

Figure 5. USB mass storage data flow
图 5. USB 大容量存储设备数据流图

3.2.1. 协议层面分析

从抓包分析我们知道，USB 重定向 U 盘识别过程进行了大量扇区数据的读取操作，通过抓包可以看出其协议封装，对 USB Mass Storage 的读写是按如下协议进行的：

Figure 6. USB Mass Storage protocol package
图 6. USB 大容量存储协议封装

从图 6 中可以看出，首先是磁盘 SCSI 系统层读取，然后经过 USB Mass Storage 层的封装，再经过 USB URB 的封装，这三层为常规 USB 设备读取过程协议。

通过对 SPICE 重定向通道得到其协议封装，这里通过图示来画出网络数据包的协议封装：

Figure 7. USB redirection channel protocol package
图 7. USB 重定向通道协议封装

从图 7 中可以看出，USB 数据是封装到 USBRedir 协议中进行传输的。经过 USBRedir 层的封装后，就可以经过 SPICE 封装通过 SPICE 通道发送，SPICE 协议是承载于 TCP 层之上的。

由此可知，在经过 QEMU 模拟的控制器后，先将磁盘读写和 USBMS 以及 USB URB 解析后转换成 USBRedir 协议，才经过 SPICE 协议进行发送。

3.2.2. 虚拟机内部磁盘读取分析

在虚拟机内部，通过微软提供的 Diskmon 磁盘读写监控工具，监控 USB 识别中的磁盘扇区读写，可以看出每次 IO 读取的数据量大小。

对比 Windows7 和 Windows10 发现，Windows10 每次 IO 读取的数据量最大可达 1 M 字节大小；而 Windows7 每次 IO 读取的数据量在 4 K 字节大小。

因此，磁盘读写 IO 是影响 U 盘识别速度的一大原因。

3.2.3. 虚拟机内部抓包分析

在虚拟机内部，通过 Wireshark 携带的 USBPcap 工具来抓取 USB 端口的数据可以知道，进行了大量的数据读取操作。

对比 Windows7 和 Windows10，发现，Windows10 每次读取的数据包长度为 64 K；而 Windows7 每次读取的数据包长度为 4 K。

根据协议封装，Windows10 在 IO 层面将 DISK.SYS 驱动的 1 次 IO 请求，由 USBSTOR.SYS 驱动拆分为 16 次对 USB 总线的请求，16 次 USB 总线请求的响应则由 DISK.SYS 驱动合并成了 1 次 IO 数据响应；而 Windows7 则没有进行 IO 的合并过程，DISK.SYS 驱动 1 次 IO 请求就是 4 K，USBSTOR.SYS 驱动也是按 4 K 请求的。

3.2.4. 从 USB 控制器角度分析

由于采用 USB2.0 和 USB3.0 的控制器对 IO 数据长度和 USB 数据包的长度没有影响。因此说明，控制器层面不会进行数据包的合并操作。

目前推断，这块应该是 USB3.0 控制器的特性。通过跟踪 QEMU 相关代码，确认 Windows7 采用 3.0 控制器，内部数据包长度也是 4 K。但是 Windows7 采用 3.0 控制器，识别时间要接近于 Windows10。但是随着 U 盘容量越大，Windows7 采用 3.0 的控制器，也没有 Windows10 的效果，说明 Windows10 对磁盘 IO 进行过优化。

根据以上的分析可以知道，影响 USB 重定向识别速度的因素，主要包括磁盘读写 IO 和 USB 控制器等。此外，如果在网络堵塞情况下，识别过程传输的大量未经压缩的数据也是导致识别速度偏慢的原因之一。

4. 基于 SPICE 协议的 USB 重定向识别优化

针对第 3 节分析出的影响当前 USB 重定向识别速度的各种因素，本节中将分别对这些问题提出解决方案。

从 USB 识别原理出发，通过分析调试并进行实际试验发现从以下 3 个方面进行优化对识别速度会有比较大的提升，分别为：

1) 虚拟机使用的 USB 控制器从传统的 2.0 控制器换成高速的 3.0 控制器；

2) 对重定向网络中传输的数据进行压缩，降低网络传输过程中带来的延迟，尤其在网络质量差时，效果比较好；

3) 虚拟机内部通过编写磁盘过滤驱动对 USB 读写的块大小进行优化。

4.1. 采用 3.0 控制器

对于采用 3.0 控制器在不限速条件下，识别时间明显要短很多，因此，如果采用 3.0 的控制器，在该条件下，对 USB 识别速度有较大提升。

xHCI (eXtensible Host Controller Interface)，是流行的 USB3.0 的接口标准，它在速度、节能、虚拟化等方面都比 USB2.0 的控制器中有了较大的提高。xHCI 支持所有种类速度的 USB 设备(USB 3.0 SuperSpeed, USB 2.0 Low-, Full-, and High-speed, USB 1.1 Low- and Full-speed)。

Libvirt 启动 xHCI 配置如图 8 所示：

```
<controller type='usb' index='0' model='nec-xhci' ports='15'>
  <address type='pci' domain='0x0000' bus='0x00' slot='0x07' function='0x0'/>
</controller>
```

Figure 8. Libvirt starts the xHCI configuration
图 8. Libvirt 启动 xHCI 配置

4.2. 数据压缩

由于主机会向 USB 设备读取大量的数据，为此，通过修改 SPICE 相关代码对数据进行压缩，以减少传输量，这里采用 LZ4 压缩算法来实现。LZ4 是一种无损压缩算法，最新版本的 LZ4 压缩速度为每核心 500 MB/s 以上，它拥有速度极快的解码器，同时 CPU 占用较低，总的来说，LZ4 压缩算法压缩效率较高[18]。

Compressor	Ratio	Compression	Decompression
memcpy	1.000	7300 MB/s	7300 MB/s
LZ4 fast 8 (v1.7.3)	1.799	**911 MB/s**	**3360 MB/s**
LZ4 default (v1.7.3)	**2.101**	**625 MB/s**	**3220 MB/s**
LZO 2.09	2.108	620 MB/s	845 MB/s
QuickLZ 1.5.0	2.238	510 MB/s	600 MB/s
Snappy 1.1.3	2.091	450 MB/s	1550 MB/s
LZF v3.6	2.073	365 MB/s	820 MB/s
Zstandard 1.1.1 -1	2.876	330 MB/s	930 MB/s
Zstandard 1.1.1 -3	3.164	200 MB/s	810 MB/s
zlib deflate 1.2.8 -1	2.730	100 MB/s	370 MB/s
LZ4 HC -9 (v1.7.3)	**2.720**	34 MB/s	**3240 MB/s**
zlib deflate 1.2.8 -6	3.099	33 MB/s	390 MB/s

Figure 9. Compression algorithm comparison diagram
图 9. 压缩算法对比图

对于文件传输这样的场景，不能容忍差错的出现，只能采用无损压缩算法，而从图 9 压缩算法的对比中，可以看出：LZ4 算法速度极快，提供很高的压缩和解压速度，其压缩比也最高，但其是以占用内存为代价的。目前较新版本的 SPICE 采用的 LZ4 压缩算法，经过测试，其压缩后，网络带宽会有所减少，平均情况下，可达 50%的带宽减少。如果在带宽受限条件下，对识别速率有一定的提升作用。

针对 USB 文件传输，采用下述流程进行有选择性的数据处理流程：

1) 监控 USB Redirect 模块数据传输接口，捕获 Bulk 类型数据；
2) 尝试采用 LZ4 default 的压缩接口对源数据进行压缩；
3) 压缩失败或者压缩后数据大于源数据空间，直接把源数据放进传输队列；
4) 压缩成功数据，修改传输数据头为 LZ4 的压缩类型，同时把压缩后数据放进传输队列；
5) 服务端处理 USB 数据，发现数据为 LZ4 的压缩类型，则进行 LZ4 解压，还原数据。

详细流程如图 10 所示：

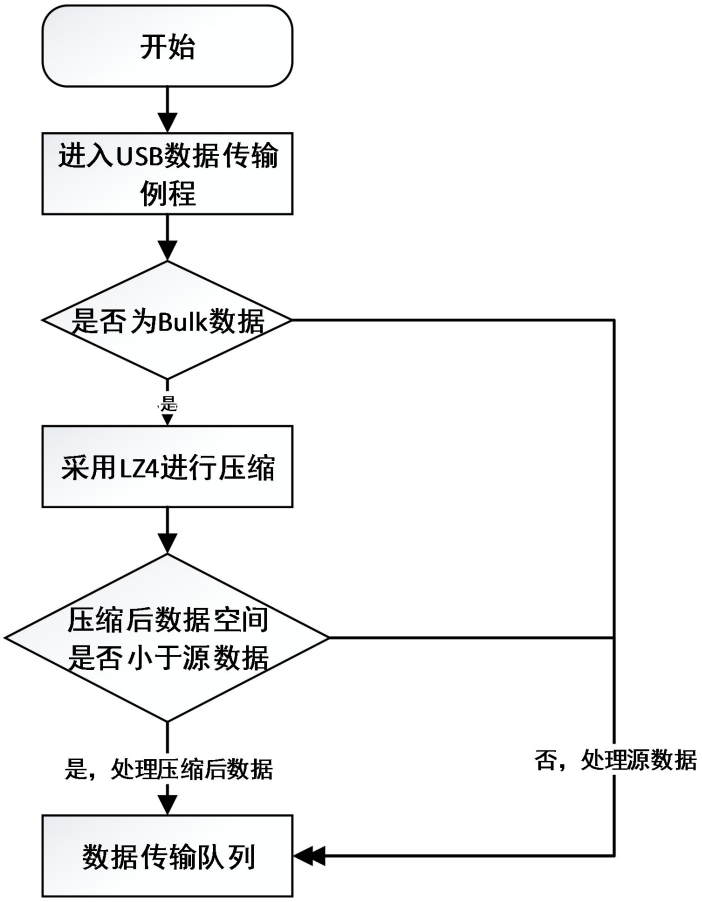

Figure 10. Data compression process flow chart
图 10. 数据压缩处理流程图

4.3. IO 优化

通过编写磁盘过滤驱动对 USB 存储设备读 IO 进行合并来优化 USB 存储设备识别过程中的读写速度。优化的核心点主要包含 2 个方面：1) 在添加设备例程中识别当前设备是否是 USB 设备，防止优化对非 USB 存储设备造成影响；2) 在读写例程中，识别出读数据操作，对小块数据读写进行预读和缓存。下面分别对上述两个方面进行说明。

4.3.1. 识别 USB 设备优化

识别 USB 设备优化的具体过程如下：判断当前设备的总线是否是 USB 总线，如果是则表示是 USB 设备，则创建过滤设备，否则直接跳过。

具体流程如图 11 所示。

4.3.2. 读数据优化

读数据优化的详细过程如下：判断当前发起的功能码是否为读操作，如果是读操作则判断当前读取的数据块是否为小块(小于 32 扇区的块)，如果数据块为小块，则进行预读缓存处理(预读 128 个扇区)，否则直接进行下发处理；判断当前发起的功能码是否为写操作，如果是写操作则判断写入的数据是否跟我们缓存的数据有重叠，如果有重叠则需进行更新并下发处理，否则直接下发处理。

具体流程如图 12 所示。

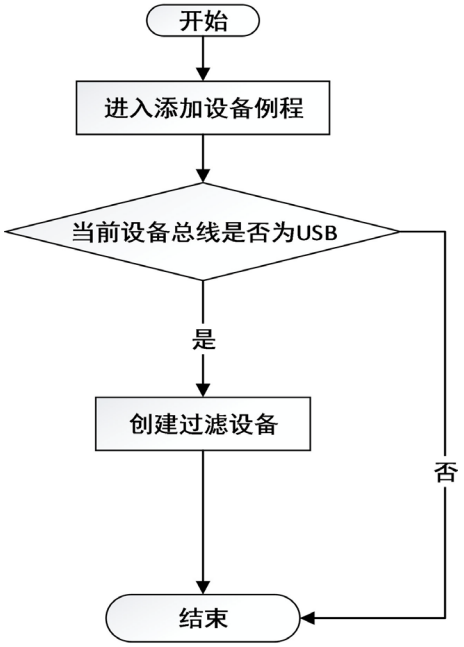

Figure 11. USB identification filter flow chart
图 11. USB 识别过滤流程图

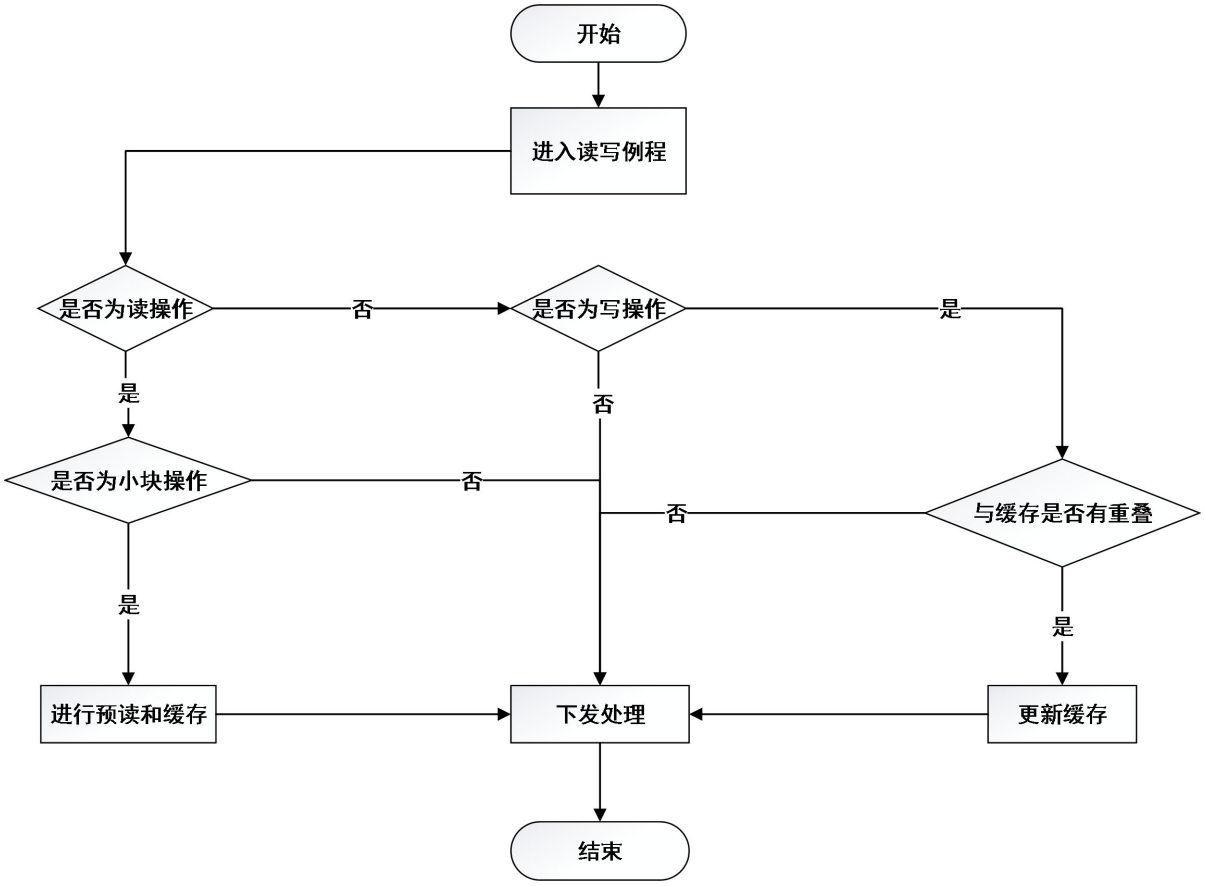

Figure 12. USB read data optimization flow chart
图 12. USB 读数据优化流程图

5. 实验结果与分析

本实验通过收集实际数据来验证我们采取的优化方案能否达到预期效果，收集的数据为优化前后USB 存储设备在进行重定向过程中 Windows7 系统正常识别出 USB 存储设备的时间，实验分别测试了 2 种不同网络场景下的实验数据，即网络不限速(内网环境)与限速 30 K/s (外网环境)。

5.1. 测试实验环境设置

1) 一台安装有 Fedora 系统的 Linux 客户端，其硬件为 Intel(R) Celeron(R) CPU 1037U @ 1.80 GHz，双核 4 G DDR 内存，系统安装 VDI 客户端软件；

2) 3 种容量的 USB 存储设备：一块普通 1 T 移动硬盘(3.0 接口)，一个普通 2.0 接口 16 G U 盘，一个 3.0 接口 32 G U 盘，磁盘格式均为 NTFS；

3) 一台安装有 VDI 服务端软件的服务器，其硬件为 Intel(R) Core(TM) i5-6400 CPU @ 2.70 GHz，四核 16G DDR 内存；

4) 在服务器上建立 Windows7 64 位系统的 KVM 虚拟机；

5) 配置好客户端和服务端的网络，保证千兆网络连通；

6) 网络设置，不限速：usb_transfer_limit speed_send = 0，speed_recv = 0；限速：usb_transfer_limit speed_send = 30 K/s，speed_recv = 30 K/s。

5.2. 实验过程

1) 采用现有方案收集实验数据

步骤一：KVM 虚拟机 USB 控制器设置为 2.0；

步骤二：对不同的 USB 存储设备，在不限速与限速 2 种网络环境下进行重定向并收集重定向识别时间数据。

2) 采用优化方案收集实验数据

步骤一：KVM 虚拟机 USB 控制器设置为 3.0，网络传输的数据启用压缩，虚拟机内安装我们的磁盘过滤驱动；

步骤二：对不同的 USB 存储设备，在不限速与限速 2 种网络环境下进行重定向并收集重定向识别时间数据。

5.3. 实验数据与分析

Table 1. Performance comparison before and after speed optimization for USB redirection recognition (no speed limit)
表 1. USB 重定向识别速度优化前后性能对比(不限速)

测试项(不限速)	优化前(秒)	优化后(秒)
3.0U 盘在 3.0USB 插口识别时间	不支持	7.98
3.0U 盘在 2.0USB 插口识别时间	16.62	12.26
2.0U 盘在 3.0USB 插口识别时间	8.02	4.41
2.0U 盘在 2.0USB 插口识别时间	9.24	4.53
3.0 移动硬盘在 3.0USB 插口识别时间	不支持	11.49
3.0 移动硬盘在 2.0USB 插口识别时间	20.18	12.35

Table 2. Performance comparison before and after speed optimization for USB redirection recognition (speed limit)
表 2. USB 重定向识别速度优化前后性能对比(限速)

测试项(限速 speed_send = 30 KB/s speed_recv = 30KB/s)	优化前(秒)	优化后(秒)
3.0U 盘在 3.0USB 插口识别时间	不支持	11.73
3.0U 盘在 2.0USB 插口识别时间	17.63	14.08
2.0U 盘在 3.0USB 插口识别时间	10.65	8.12
2.0U 盘在 2.0USB 插口识别时间	9.7	8.3
3.0 移动硬盘在 3.0USB 插口识别时间	不支持	21.75
3.0 移动硬盘在 2.0USB 插口识别时间	25.05	22.08

从表 1 实验数据分析可知，在网络不限速的条件下，不同的 USB 存储设备在虚拟机采用 WINDOWS 7 64 位系统时，重定向识别的时间在优化后有了较大的降幅，性能提升效果明显；由表 2 实验数据可以看出，由于网络环境受到限制，USB 存储设备重定向识别的速度整体偏慢，但是通过优化也获得了一定程度的提升。

从实验数据分析可知，通过方案的优化整体上能有效缩短 USB 设备识别时间，提升了用户使用体验，达到了预期效果。

6. 结束语

SPICE 协议作为开源的桌面传输协议，在桌面虚拟化解决方案中应用较为广泛，有着较好的市场前景。本文针对桌面云技术中，基于 SPICE 协议的 USB 重定向数据传输性能受限的问题，提出了优化 USB 重定向识别速度的解决方案，对 USB 重定向识别过程的传输数据，从网络协议层面、虚拟机内部磁盘读取和抓包、USB 控制器等方面进行分析，找准了 USB 重定向识别速度慢的原因。本文所述方案分别从 USB 控制器、数据压缩、磁盘读写 IO 三个方面来进行优化，最终，有效减少了 USB 存储设备识别的等待时间，提升了桌面云系统的可用性和用户体验。

基金项目

本项目受云南省教育厅科学研究基金项目(2018JS035)资助。

参考文献

[1] 陈鑫, 徐义臻, 郭禾, 于玉龙, 罗劼, 王宇新. 虚拟机可瞬时开启的私有桌面云架构[J]. 计算机应用, 2015(11): 3059-3062.

[2] Yan, L. (2011) Development and Application of Desktop Virtualization Technology. 2011 *IEEE 3rd International Conference on Communication Software and Networks* (*ICCSN*), Xi'an, 27-29 May 2011, 326-329. https://doi.org/10.1109/ICCSN.2011.6013725

[3] 张楠. 通过桌面云提升高校 IT 应用和管理水平[J]. 实验技术与管理, 2014(9): 126-128 + 138.

[4] 孙玉伟, 童新海, 张林惠, 杨春雨. 云桌面中 USB 设备重定向技术研究[J]. 信息网络安全, 2015(4): 78-85.

[5] 程庆年, 周冠宇. 云桌面 USB 重定向融合方法探讨与实现[J]. 移动通信, 2017, 41(14): 50-53.

[6] 李宝宇. 一种 USB 设备重定向的方法及系统[P]. 中国专利, CN102368231A. 2012-03-07.

[7] Zhou, Y.M. and Guo, H.H. (2012) A Research of USB Device Redirection Mechanism over IP Network in Desktop Cloud System. *Proceedings of the Advances in Intelligent Systems Research*, Lanzhou, 16-18 November 2012, 4. https://doi.org/10.2991/citcs.2012.53

[8] 苏妍, 云交互协议及其传输优化技术的研究与实现[D]: [硕士学位论文]. 成都: 电子科技大学, 2019.

[9] 丁洁, 郭辉辉, 唐帼英. 基于云端桌面 USB 设备重定向机制的研究[J]. 信息技术, 2015(4): 141-144.

[10] 王继刚, 郑纬民, 滕志猛, 钟卫东. 虚拟化环境下的 USB 设备访问方法[J]. 计算机应用, 2011, 31(5): 1439-1442.

[11] Wikipedia (2020) QEMU. https://wiki.qemu.org/Main_Page

[12] 郭新营. 桌面云平台在电子政务中的应用研究[D]: [硕士学位论文]. 南昌: 华东交通大学, 2016.

[13] Redhat (2017) Spice-GTK Reference Manual. https://www.spice-space.org/spice-gtk.html

[14] 王斌, 余佳齐, 李伟民, 朱翀, 盛津芳. SPICE 在透明桌面服务机制中的应用及优化研究[J]. 计算机科学, 2015(S1): 321-324+348.

[15] Redhat (2017) Spice for Newbies. https://www.spice-space.org/spice-for-newbies.html

[16] Redhat (2017) Spice Protocol. https://www.spice-space.org/spice-protocol.html

[17] 刘子杰, 文成玉, 薛雾. 基于 SPICE 协议的虚拟桌面技术[J]. 计算机系统应用, 2018, 27(4): 100-103.

[18] Github (2020) LZ4. https://lz4.github.io/lz4