Using USBIP to Create a Portable Remote USB Hub

Daniel Paradis and Bruce Segee
Electrical and Computer Engineering,
University of Maine,
Orono, Maine, United States

Abstract— *USB devices can be forwarded to remote computers with full functionality using the USBIP utility. One way that USBIP can be utilized is when it is paired with a cloud-based virtual machine. In this case, the cloud-based virtual machine can be a powerful system with assorted development tools. The development tools might include AVR or Android tool chains. An AVR programmer that is plugged into a Raspberry Pi 2 can be remotely forwarded to a cloud-based virtual machine by using USBIP. This allows the user to log onto the cloud-based virtual machine which has all the tools necessary to program an AVR while having the physical AVR programmer plugged into a local Raspberry Pi.*

Keywords

USBIP, SSH, AVR, Virtual Machines

Submission Type

Short Paper, CSCE-ICOMP

1. Introduction

Mobile devices like Chromebooks and tablets, are becoming cheaper, lighter, and more common, but usually lack support for advanced development tools such as Matlab[1], Android Studio, or AVR tool chains. Oftentimes, it is not practical to carry around a heavy, powerful computer, nor is it possible to always have direct physical access to a powerful computer. One way to circumvent this issue is by using a cloud-based virtual machine.

The cloud-based virtual machine offers several advantages for mobile users. Since the virtual machine is cloud-based, a static IP can be assigned to the virtual machine allowing users to access the virtual machine from anywhere with an Internet connection via SSH[2] or VNC. Programs like SSH or VNC give the user access to all the power and programs that the virtual machine while all the user has to have is a Chromebook, or an ultralight laptop. The problem with a setup like this is when the user wants to interface a hardware USB device, like an Android phone or a AVR programmer, with the cloud-based virtual machine where only remote access is available. This problem can be solved using USBIP, which allows USB devices to be forwarded remotely to any computer connected to the Internet, including a cloud-based virtual machine.

USBIP also offers a solution for professors who have a virtual machine with specific software that interacts with hardware. For example, a professor may want to make sure every student has access to a machine that has all the tools necessary to program an AVR. Not only does this require a Linux image, this also requires the Linux image to be set up properly with a patched version of AVRdude[4]. Having students install Linux, whether it be dual boot or otherwise can potentially pose problems and is not necessarily a reasonable request. However, having all students own a Raspberry Pi[3] is perfectly reasonable as they are affordable and can run Linux.

The professor could create an image that has all of the AVR programming tools and launch a cloud-based virtual machine for each student. Then the students could use USBIP to forward their AVR programmers from their Raspberry Pis to the VM. The students would then be able to use the cloud-based virtual machine to program their AVRs. The advantage of this is that the students would have to do minimal setup and could avoid doing other more complicated solutions such as dual boot, or installing the tools themselves. An example of a USBIP configuration using a Raspberry Pi, a laptop, and a cloud-based virtual machine can be seen in Figure 1

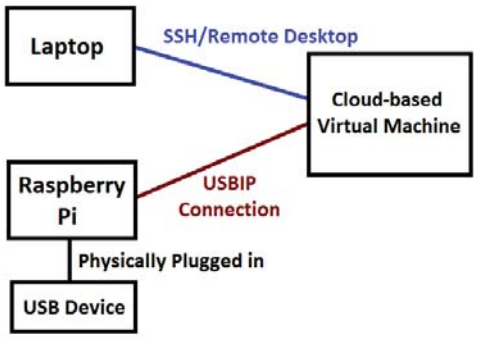

Fig. 1
A SIMPLE SETUP WHERE A USB DEVICE IS FORWARDED TO A CLOUD-BASED VIRTUAL MACHINE BY USBIP AND ACCESSED VIA SSH ON A LAPTOP.

The Raspberry Pi 2 and virtual machine set ups will be covered. The set ups consist of two parts, the USBIP set up, and the SSH tunnel set up. Future work will include other possible use cases for USBIP. The advantages and disadvantages of USBIP will also be discussed.

2. USBIP Setup

USBIP must be installed and set up properly first on both the client and host machines[5]. In the example highlighted in the introduction, the client machine would be the cloud-based virtual machine and the host would be the Raspberry Pi. USBIP is standard on newer versions of the Linux kernel and any desktop, laptop, or virtual machine running Linux can easily install USBIP on their systems. Installing USBIP on the Raspberry Pi is more troublesome due to the lack of certain packages on the Raspbian repository.

2.1 Virtual Machine

Getting the cloud-based virtual machine set up with USBIP is very simple. For this case, it is assumed that the system is Debian-based but other distributions like Arch or Fedora work similarly. As long as the virtual machine is running a new version of the Linux kernel (4.4+) USBIP can be installed by installing the linux-tools-generic package. An important thing to note is that the usbip package should *not* be installed. That package is outdated and will cause problems if installed. To make sure USBIP was installed properly the command usbip list -l should be run. A common problem is that when that command is run, the system says USBIP is not installed. This is because sometimes older versions of linux-tools-generic can be on the system and may cause problems with USBIP and should be removed with apt autoremove.

Once USBIP is installed, next the kernel modules usbip-core and vhci-hcd need to be inserted. This can done rather easily with modprobe, but, this is not an optimal solution as anytime the system is rebooted the modules will have to be inserted manually. To fix this, the names of the modules should be added at the end of the /etc/modules file so that they are inserted on system startup. After adding the modules to /etc/modules, the system should be rebooted and lsmod should be used to make sure that the usbip-core and vhci-hcd modules were inserted correctly.

After the modules have been verified to be inserted properly on boot, next, the usbip daemon must be started. Running the command usbip -D should start the usbip daemon. Similar to the modules, the daemon also will need to be manually started on reboot, but, there are several ways to get around this. One option would be to add the usbip daemon command to crontab with the time specified to reboot. Another option is to add the usbip -D command to the /etc/rc.local file. Other methods exist and any of them can be used to make sure the usbip daemon starts on system boot.

2.2 Raspberry Pi

Installing USBIP on the Raspberry Pi is not as simple as with a normal Linux system. The first roadblock is that the linux-tools-generic package does not exist on Raspbian[6]. However, the usbip package is available for Raspbian and unlike with the virtual machine, installing this package is the only way to get USBIP to work with the Raspberry Pi currently. To ensure that USBIP works properly, Raspbian should be fresh installed with the newest version. If Raspbian was installed prior to kernel version 4.4 and then updated, it can cause module problems and Raspbian will have to be freshly installed to fix them.

Once Raspbian has been fresh installed and the usbip package has been installed, next the modules must be inserted. This step is the same as with the virtual machine with the exception that an additional module, usbip-host, must also be inserted. It is also recommended that these modules be added to /etc/modules like with the virtual machine. The usbip daemon must also be running and should be set up so that it runs on boot which can be accomplished by using the same method described in the virtual machine section.

3. SSH Tunnels

Once USBIP has been set up it is ready to be used. However, in order to use USBIP the IP address of the host machine, which in this case is the Raspberry Pi, has to be known. It is not always possible to have an external IP for the Raspberry Pi since most users will have it connected to their home network. While a user can connect to devices on their home networks easily, if the user is off the home network, the Raspberry Pi is not accessible. In order to get around this problem, two SSH tunnels have to be created on the Raspberry Pi. These SSH tunnels will allow the cloud-based virtual machine to have access to the Raspberry Pi, even though the Raspberry Pi does not have an external IP. The ultimate goal is to make the Raspberry Pi essentially act as a portable remote USB hub. The user should be able to plug the Raspberry Pi in anywhere with Internet with no monitors, keyboards or mice, and be able to access the Raspberry Pi and its USB devices. An example of how an SSH tunnel works can be seen in Figure 2.

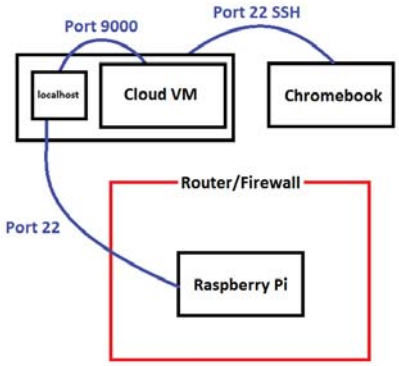

Fig. 2
AN EXAMPLE OF HOW AN SSH TUNNEL IS USED TO ALLOW A USER TO ACCESS A FIREWALL/ROUTER PROTECTED HOME NETWORK.

In this Figure, the Raspberry Pi is connecting its port 22 to the localhost of the cloud-based virtual machine. Then, the cloud-based virtual machine can go through port 9000 of its localhost to access port 22 of the Raspberry Pi. This configuration can be achieved by running the command ssh

paradis@204.197.1.108 -R 9000:localhost:22 on the Raspberry Pi where paradis@204.197.1.108 is the cloud-based virtual machine's username and ip. One thing to note is that any port can be used in place of port 9000 as long as it is not being used already for something else.

3.1 Setup and Purpose

The first SSH tunnel that needs to be set up is for port 22 which is for SSH. This SSH tunnel is set up with the -R option, which forwards a local port on the Raspberry Pi to a local port on a remote machine. In this case, port 22 on the Raspberry Pi is forwarded to port 9000 on the cloud-based virtual machine's port 9000. The way this tunnel works is that if the user SSHs to localhost on port 9000 on the cloud-based virtual machine, it will SSH to the Raspberry Pi. The purpose of this tunnel is to allow the user to connect to the cloud-based virtual machine and from there be able to SSH into the Raspberry Pi. This tunnel is not completely necessary, but, it is useful to be able to SSH to the Pi for troubleshooting and debugging.

The second SSH tunnel that needs to be set up is for port 3240 which USBIP uses to forward the USB devices. USBIP requires an IP in order for it to work. This is not a problem if both devices are on a local network, or if both devices have externally accessible IPs. However, the Raspberry Pi usually won't have an externally accessible IP for the above mentioned reasons. This tunnel allows the user to use localhost as the IP just like the other tunnel that was setup. The -R option is also used here and forwards port 3240 on the Raspberry Pi to port 3240 on the localhost of the cloud-based virtual machine.

3.2 Automation

In order to streamline this set up, everything, including the SSH tunnels, should be configured on boot. The USBIP daemon and USBIP modules have already been configured in this manner. However, SSH requires a network in order to work, so most normal methods of running commands on boot do not work. The best way to create SSH tunnels on startup is through the use of a systemd service. Systemd manages services on startup and while the system is running[7]. One of the advantages of creating a custom systemd service is that there are different parameters that a user can specify. These parameters can tell systemd to start services after the network is up and running and restart services that are broken. A systemd service should be created that establishes both SSH tunnels on bootup.

These services should wait for the network to be active before it attempts to create the SSH tunnel and should restart the service if it dies. This way, the SSH tunnels are guaranteed to be up and running if the Raspberry Pi is up and running with an active internet connection. One possible problem with setting up an SSH tunnel on boot is that the ip of the client machine has to be hard-coded into the systemd services. This is not a problem when using a cloud-based virtual machine. This is because the cloud-based virtual machine can be assigned a static ip which never changes. Sometimes it is possible to set up a static ip on a machine, but this is not always the case if the machine is on a school or work network that forces the user to have a dynamically changing ip.

4. Usage

Once the set up is complete the user has with a Raspberry Pi that can act similarly to a portable remote USB hub. Since most steps have been automated, the process of using the Raspberry Pi as a remote, portable, USB hub is rather simple. First, the user should plug in the Raspberry Pi and plug in any USB devices they want to forward to the cloud server. Next, the user should connect into the cloud-based virtual machine and from there SSH to the Raspberry Pi. Since an SSH tunnel is already there, this is as simple as SSHing to localhost over whichever port SSH was forwarded over. Once the user is on the Raspberry Pi, they should list the USB devices using the usbip list -l command and note what the busid of the USB device they want to forward is. The last step on the Raspberry Pi is to bind the device they want to forward using the usbip bind command.

Once the USB device has been bound, the cloud-based virtual machine can attach the device. To do this, the user issues the command usbip list -r localhost. Normally, this would list USB devices that have been bound on a remote IP. Since the IP of the Raspberry Pi is unknown, an SSH tunnel was setup on the Raspberry Pi for usbip. Therefore, localhost in this case refers to port 3240 on the Raspberry Pi. Once the user has listed and found the device they want to attach, they use the usbip attach command. In this case usbip –tcp-port=3240 attach -r localhost -b yourbusid would attach a device bound on the Raspberry Pi with busid yourbusid.

With these three simple commands, list, bind, and attach, a USB device that is physically plugged into the Raspberry Pi will show up as being plugged into the cloud-based virtual machine. This device is fully functional just as if it was being used locally.

4.1 Advantages

There are obviously many advantages to using USBIP. For one, USBIP allows a user to export any USB device to any machine that is connected to a network and is configured properly. This allows users to have USB devices physically plugged in locally, but, be working on the USB device using a much more powerful remote machine, such as a cloud-based virtual machine. Another advantage is that the program is very versatile since there are so many different devices that can be plugged into a USB port. Keyboards, mice, programmers, flash drives, phones, SD card adaptors and much more can all be plugged in through USB and forwarded to a remote computer using USBIP. Another major advantage of USBIP is that it can be installed on a Raspberry Pi and configured in a way that the Raspberry Pi can act as a remote USB hub. Raspberry Pis are cheap, light, small, and thus are a great choice for the application. Almost everybody can afford one and carrying a Raspberry Pi around is not a hassle. With the set up described in this paper, a user could carry around a Raspberry Pi and a

Chromebook and still have the power of a full desktop machine by making use of a cloud-based virtual machine.

4.2 Disadvantages

The one major disadvantage of USBIP is that it requires both systems to have Linux installed in order to run. One problem with this is that not every program can run on Linux. This can be a problem when a user wants to forward a USB device to a cloud-based virtual machine to take advantage of its power, but, the program only works on Windows or Mac OSX. USBIP used to have support for Windows, but it has been dropped relatively recently. Another disadvantage of USBIP is that it can be tricky to set up if the user is not aware of certain caveats. Most users that want to install a program will use their package manager. However, as covered earlier, the USBIP package is obsolete on regular versions of Linux (not the Raspberry Pi). This can be confusing as most online searches will tell the user to install the package.

5. Future Work

Due to the versatility of USBIP, there are many possibilities for future work. One possibility involves putting the Raspberry Pi in a remote location and hooking up a USB oscilloscope up to it. Then the user could forward the oscilloscope to a cloud-based virtual machine and do live data analysis with a powerful program like Matlab, which otherwise can not be run on the Raspberry Pi. This configuration can be seen in Figure 3.

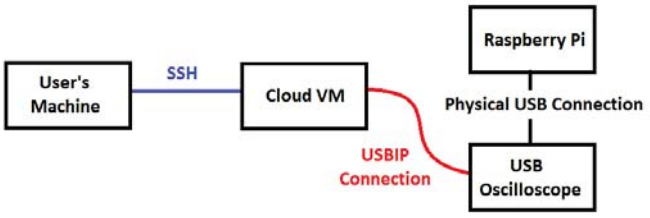

Fig. 3
AN EXAMPLE CONFIGURATION WHERE A USB OSCILLOSCOPE ON A REMOTE NETWORK IS FORWARDED TO A CLOUD-BASED VIRTUAL MACHINE FOR DATA PROCESSING.

This configuration would be ideal for someone who needed to collect data from remote locations and do large scale data processing on the collected data. One possibility would be if a user had multiple Raspberry Pis with USB sensors, or a USB oscilloscope spread all across the world and wanted to record and process weather data. This user could buy many inexpensive Raspberry Pis and clone the image that is already set up to all of them. Then they could simply go and put the Raspberry Pis in all the locations they wanted data from and they could all automatically connect the a single cloud-based virtual machine. From there, the user could log onto the cloud-based virtual machine and have access to all of the data being collected on the Raspberry Pis.

Another idea for future work would be enhancing the configuration described in this paper through the use of more automation. It is possible through the use of scripts and cron jobs to have the Raspberry Pi forward all of its USB devices to the cloud-based virtual machine and have the cloud-based virtual machine detect and attach them. It is open to discussion whether the user would want to forward all of its USB devices, or specific USB ports or specific USB devices. This addition would streamline the configuration further, eliminating all of the user's work aside from plugging the USB device in once configured.

6. Conclusion

Even though USBIP has been added to the Linux kernel, it is a relatively unknown and seldom used program. Lighter, cheaper, and more convenient devices like Chromebooks which are limited by the programs they can run can gain increased flexibility and power by using USBIP. This can be done by configuring a Raspberry Pi so that through SSH tunnels and startup processes it can act as a remote USB hub by automatically connecting to a remote cloud-based virtual machine. Once the Raspberry Pi is set up properly it can act as a plug-and-play portable, remote, USB hub.

There are many other ways that USBIP could be integrated into systems. Currently, it is a completely underutilized program despite being part of the Linux kernel. This is a relatively new addition however. With an idea that is as simple as being able to remotely forward USB devices to computers there are so many possible use cases that it is impossible to cover all of them. Once more people learn about USBIP and its capabilities it may become a tool that is as commonly used as SSH or a remote desktop sharing program.

References

[1] MATLAB version 9.2.0. Natick, Massachusetts: The MathWorks Inc., 2017.
[2] *OpenSSH*. OpenBSD Foundation, 2017. Print.
[3] "Raspberry Pi - Teach, Learn, And Make With Raspberry Pi". Raspberry Pi. N.p., 2017. Web. 21 Mar. 2017.
[4] "AVRDUDE - AVR Downloader/Uploader". Nongnu.org. N.p., 2017. Web. 21 Mar. 2017.
[5] Król, Piort. "Linux, Rpi And USB Over IP Updated - [3Mdeb Blog]". Blog.3mdeb.com. N.p., 2017. Web. 21 Mar. 2017.
[6] controller?, How. "How Do I Make My Raspberry Pi Act As A Wireless USB Controller?". Raspberrypi.stackexchange.com. N.p., 2017. Web. 21 Mar. 2017.
[7] "Persistent Reverse (NAT Bypassing) SSH Tunnel Access With Autossh - Raymii.Org". Raymii.org. N.p., 2017. Web. 21 Mar. 2017.

Acknowledgements: This work was partially supported by NSF Grants EEC-1460700 and DRC-1543040. Daniel Paradis is an NSF REU site participant.