Tethering an Android Smartphone to USB Devices

March 2011

BizDev@SecureCommConsulting.com

Veteran Owned Small Business
DUNS: 003083420
CAGE: 4SX48

http://www.securecommconsulting.com

1.1. Purpose

This paper presents a brief background of Universal Serial Bus (USB) and the architecture of USB drivers in the Linux operating system. The purpose of this paper is to provide a brief explanation of the technical issues involved in making an Android smart phone operate as a USB Host, and properly connect and interoperate with USB devices.

1.2 General Background

The Universal Serial Bus (USB) architecture includes a USB Host and a USB Device. The Host is the master and controls the communication between itself and all of its connected USB Devices. Normally[1], smart phones are considered USB devices, and are connected to personal computers (PCs) or laptops. However, a popular topic of late is the tethering of an Android smart phone to a USB device (i.e. the smart phone becomes the USB host). This is due to increased demand for smart phone applications that utilize external USB devices (such as mass storage devices, keyboards, and even external networking devices like radios). As a USB Device, a smart phone cannot connect to other USB devices; therefore, to enable smart phones to connect to either a USB device or a USB host, smart phone USB drivers must be modified to support USB OTG (On-The-Go) as depicted in Figure 1. USB OTG allows a component to take on the role of either a USB Host or a USB Device, whichever is best suited for the currently connected gadgets. Software development is required to achieve this. Further, if a secure and maintainable solution is desired, cooperation from smart phone vendors is needed to maintain this driver as part of the smart phone's kernel.

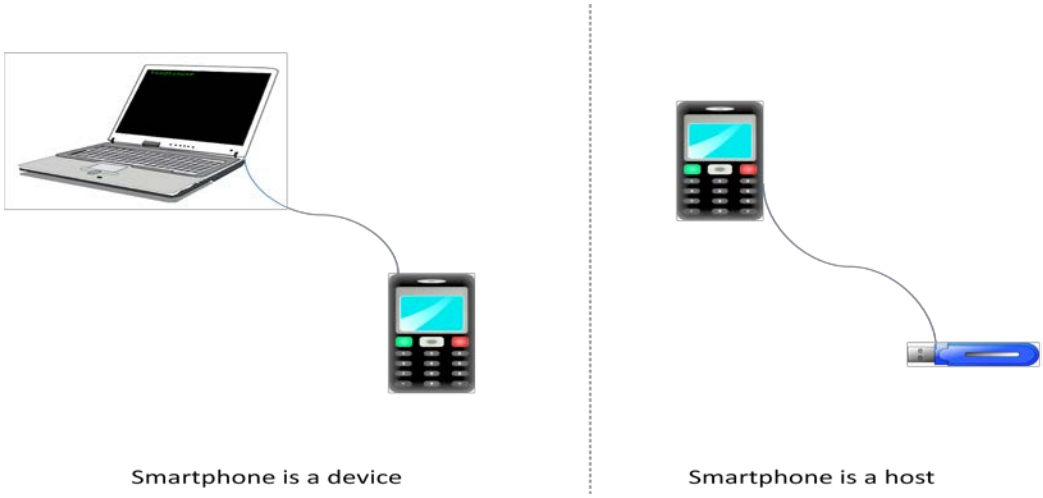

Figure 1: USB Host and Device

The good news is that Android application developers and users alike are requesting USB OTG support; and, given the highly competitive nature of the smart phone market, it is likely that

[1] At the date of this writing, commercially available Android smart phones connect only to USB hosts. Given the rapid advances in the smart phone space, it is anticipated that this will change in a few months from this writing.

smart phone vendors will start providing USB OTG in the next 6 -12 months. For those Android application developers who can't wait for the vendors to catch up with demand, this paper will explore the challenges involved in providing a custom USB OTG solution.

1.3 USB (Universal Serial Bus) History

According to the USB 2.0 specification, the host supplies 5V of power to a connected USB device over the USB cable in order to signal the device that a host is connected. PCs generally are much more powerful than smart phones, and are usually connected to an AC power source or have large batteries, and so were given the host role. Smart phones and many other gadgets were given the device role since they typically were much slower than the host and had much lower battery capacity. However, in a short period of time, smart phone technology has advanced, and the line between the PC and smart phone capabilities has blurred. The desire to have smart phones and other devices fill the role of the USB host, led to modification of the USB 2.0 specification to include the USB On-The-Go supplement.

1.4 Linux OS & Driver Architecture

The core of Android is the Linux operating system, and the services provided by Android are based upon the Linux kernel and Linux services. Android applications are actually Java applications, but use of the smart phone's underlying hardware is based upon Linux libraries, device drivers, and the Linux kernel. Support for USB services in an Android application requires the use of Linux device drivers to access the USB hardware.

The USB driver, shown in Figure 2, is a basic USB 1.1/USB 2.0 driver that supports Host mode only (not USB OTG) to connected devices. Support for USB on the Host is comprised of several device drivers, each of which logically communicates with an equivalent device driver on the USB Device. The Host Controller Driver is designed specifically for the Host Controller hardware that exists on the particular smart phone[2]. This is due to the fact that the Host Controller driver must directly manipulate the hardware registers, control registers, and data buffers in the Host Controller hardware. Since each Host Controller Chipset manufacturer has a different architecture for their Host Controller chipsets, the registers and buffers in each chipset are slightly different. The Host Controller driver encapsulates these differences, and prevents other components of the USB stack from being impacted by these hardware differences.

The Host Core driver implements standard USB functions, like configuration to accommodate the attached devices, buffer management, data transfer to devices, etc. These functions are common across all Host Controllers, and the interface to the Host Controller driver is used to

[2] It should be noted that similarities do exist between the different Host Controller drivers, and there are some *universal* host controller drivers that operate on several different chipsets. However, the specific Host Controller driver must be matched to the Host Controller chipset on the smart phone, and it may be the case that no open source (publicly available) Host Controller driver is available.

accomplish these functions. In order to prevent close coupling between the USB Core driver and the USB Host Controller driver, a Universal Host Controller driver interface was defined. This allows any USB Core driver to interoperate with any USB Host Controller driver that implements the UHCD interface.

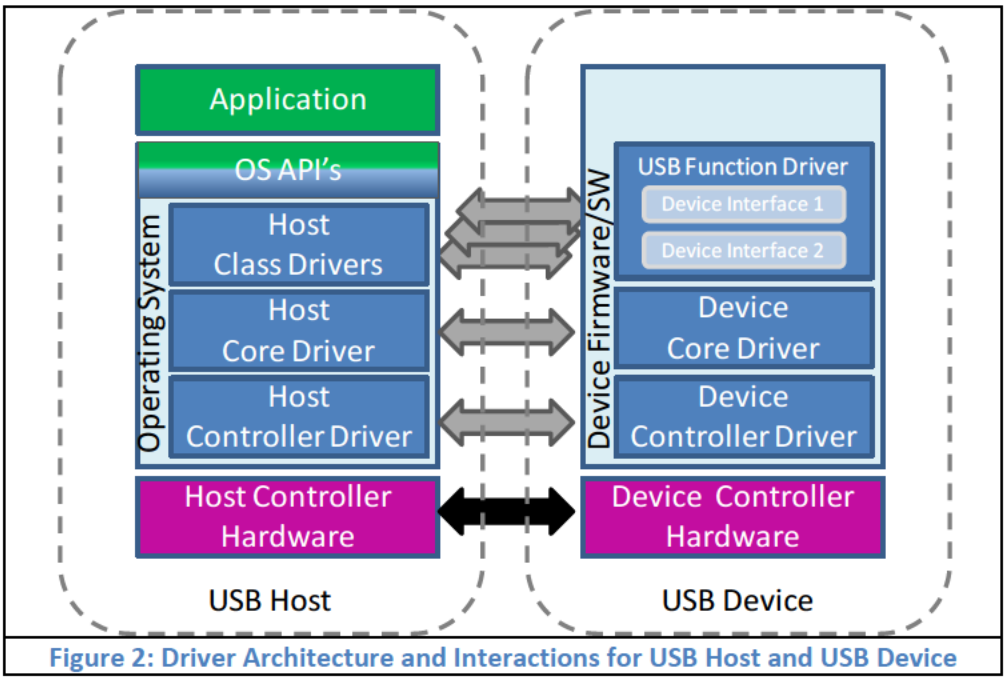

Figure 2: Driver Architecture and Interactions for USB Host and USB Device

Together, the USB Core Driver and the USB Host Controller driver provide enumeration, connection to devices, and data transfer. However, in order to access specific device capabilities, the USB Device interfaces must be accessed. Device interfaces are encapsulated in function drivers. When a USB device enumerates to the host, it lists the configurations that the device supports; each configuration includes one or more interfaces and the endpoint address to use to connect to that interface. These interfaces represent specific functions on the device, such as mass storage, or audio play-back, and are tied directly to software or hardware components on the device that implement that function.

The USB Host accesses the functionality on the device through the USB Function Drivers (which contain the interfaces). On the USB Host, Host Class drivers are used to communicate with the USB Device Function Drivers. There is typically only a single Host Class Driver for each *class* of interfaces on USB devices. For example, there are many different devices that are capable of mass storage, and each device has a particular way in which files must be read or written to the device. However, the mass storage interface to these devices is the same, and is handled by a single Host Class driver. The USB Host Class driver implements the generic interface requirements to communicate with the Device's Function Driver; the Function Driver handles the unique interfaces to the USB Device's hardware.

The Linux Operating System supports a series of class drivers to provide access to various device interfaces over USB. Examples of these class drivers include the Mass Storage Device Class, the Communications Device Class (CDC), and the Human Interface Device Class (HID). Within these classes, there are specializations of the class. For example, the CDC includes the CDC-EEM (Ethernet Emulation Model) and CDC-ECM (Ethernet Control Model). The different USB classes and the specifications that define them can be found on the USB Implementers Forum web site (http://www.usb.org/developers/devclass_docs).

USB On-The-Go is an amendment to the USB 2.0 specification, and defines an architecture that allows a USB enabled platform to operate as either a USB Host or USB Device, depending on how it is connected. A series of new protocols are included in USG-OTG to support the features of a combined host and device; some of these protocols include the ability to negotiate which party has the control of the USB bus (Host Negotiation Protocol), control of power usage by the USB bus (Suspend/Resume/Remote Wakeup protocol), and detection of attached hosts and devices (Attach Detection Protocol). These new capabilities require a different driver architecture that includes both the Host mode drivers as well as the Device mode drivers. Figure 3 shows an example USB-OTG driver architecture, which includes the On-The-Go protocols, and the host and device capabilities.

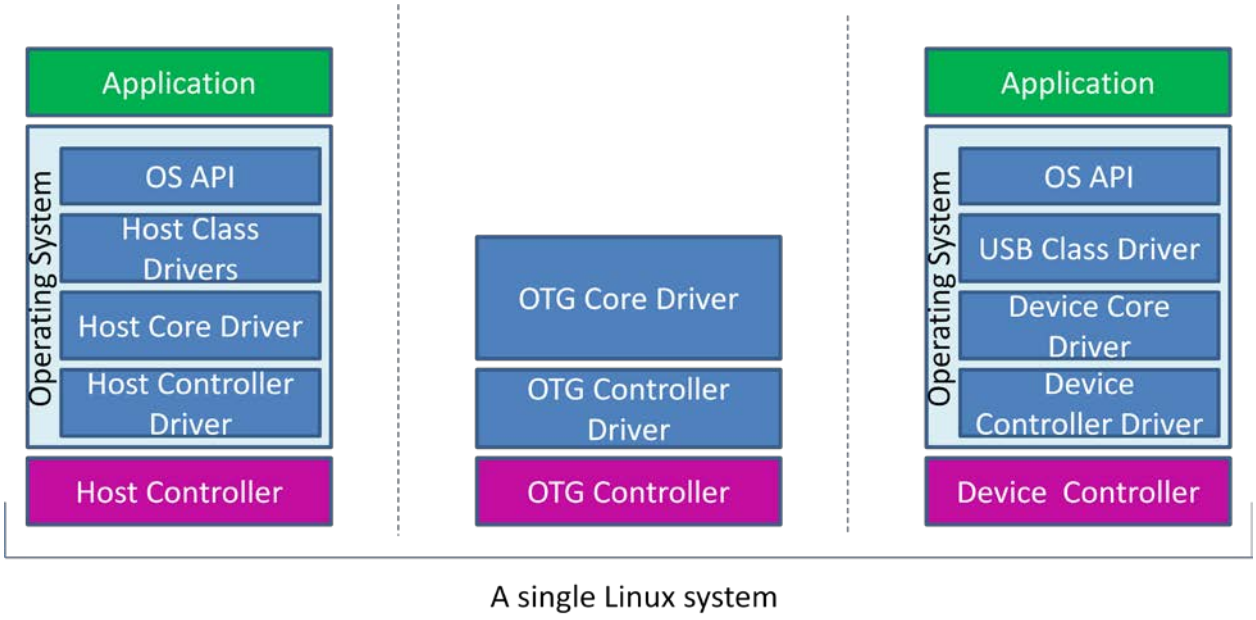

Figure 3: USB OTG Driver architecture in Linux (requires host, device, and OTG drivers)

1.5 Technical Issues

There are currently no Android smart phones on the market that support USB On-The-Go or USB Host Mode natively. Technically, all that is necessary to make a smart phone connect as a host would be to upgrade the USB software on the smart phone. However, there are several

issues that must be addressed. Below are the primary technical issues involved in enabling an Android smart phone to connect to a USB device.

1.5.1 USB Power

Some USB Devices that do not contain a power source (e.g. flash memory sticks) use power supplied by the USB Host, over the VCC pin of the USB cable, to power themselves. However, even if a USB Device has its own power source, the USB Host must still supply power over the VCC pin of the USB cable. This is because 5V DC on the VCC pin is the signal to the USB Device that a USB Host has connected to it. Therefore, in order to operate in host mode, a smart phone must be capable of providing 5V DC power over the USB VCC pin to signal the attached USB Device that a host has connected.

At the time of this writing there are no available commercial Android phones that supply power on their USB VCC pin[3]. This is likely due to the fact that a USB port that outputs 5V on the VCC pin would drain the battery of the smart phone faster. A potential, albeit temporary, solution is to provide power on the USB VCC pin from an external source, thereby tricking the USB Device that a Host is connected. This could be accomplished by using a USB hub or a special cable with a connection to a power supply.

1.5.2 USB Host & OTG Hardware, and Device Drivers

A USB host controller is a chip that provides host functions for a system (as shown in Figure 2 and Figure 3). For the smart phone to support USB Host Mode, the chipset in the phone must include USB Host Controller hardware. In many cases the host controller is integrated onto the CPU. However, to operate the smart phone in either host or device mode, the chipset on the smart phone must also support USB-OTG. Current hardware support for USB-OTG is fairly common, but driver software to support OTG chipsets is not openly available[4].

The host controller chip requires a driver for the Linux operating system in order to use the host functions. There are no known Android phones that ship pre-packaged with USB host controller drivers, even though the host controller hardware is present on the phone. A driver would need to be written for host functions to operate. An open source host controller driver is available for the Google Nexus One smart phone[5]. This open source driver will only allow the phone to operate in host mode (prohibiting operation in device mode). In order to operate in

[3] The original Motorola DROID reportedly was able to provide power over its connector using a hacked "dongle", but the droid is not sold anymore. Also the Motorola XOOM reportedly supplies power as well, but it is a tablet.
[4] Our researched indicates there is OTG hardware support in quite a few phones, including OMAP, Samsung S5PC110, S5PV210, and others.
[5] http://sven.killig.de/android/N1/2.2/usb_host/ - Although a driver has been written to support USB host mode on a Google Nexus One, this driver does not support USB On-The-Go. Consequently the Nexus One is locked in host mode and cannot communicate with another USB Host (such as a PC).

host or device mode, an OTG (On-The-Go) driver must be developed for the specific smart phone's chipset.

1.5.3 Class driver

In addition to having USB host mode support, a Class Driver must be available on the host and loaded to support the particular device function needed. For example, to communicate with a device via Ethernet, a CDC-EEM driver is needed. Android does not include many class drivers natively. However, there are many open source class drivers available for Linux, but they must be recompiled for the particular smart phone that is used. This requires a simple recompiling of the kernel, however, depending on the smart phone, the bootloader may prevent kernel modification (see Section 1.5.4). See Section 1.5.5 for impacts to Over-the-Air software updates.

1.5.4 Bootloader support

The bootloader is a piece of firmware/software that provides services to load the operating system when a smart phone is first powered on. Many smart phone manufacturers lock the bootloader to prevent third parties from loading custom kernels on their devices. In order to add a new Host Controller driver or a Class Driver, the kernel must be recompiled. The recompiled kernel must then be reloaded onto the smart phone, and the bootloader must load this new kernel (instead of the old one). Different methods of bootloader locking will prevent this new kernel from being loaded by the bootloader. This could be based upon the size of the kernel image or even a digital signature over the kernel itself. There are methods to unlock the bootloader, but some of these are unreliable, and in nearly every case, will void the warranty on the phone. It may also be in violation of the end user agreement with the cellular carrier to use a modified phone on their network. The Google Nexus One and Nexus S have bootloaders that can be unlocked, but other manufacturers are not as flexible.

1.5.5 Other considerations

In order to modify a phone to support USB Host mode, it is almost assured that the Android kernel will need to be recompiled with new drivers, or new kernel flags to support dynamically loadable class drivers that were not included in the original manufacture's kernel build. Installing the drivers requires the user to obtain root access on the phone. The process of obtaining root access opens several potential security risks to the user. In addition, the warranties of most phones are voided when 3rd party images are installed or when root access is obtained.

Many smart phone manufacturers continue to provide software updates to the Android Operating System by pushing them over-the-air. These updates usually include a new kernel and will overwrite the kernel that is currently on the phone. If a modified kernel is on the phone, and the manufacturer pushes a new version down, it will remove all the modifications

added to the custom kernel. Mechanisms to disable this automatic updating could be investigated, but how successful they would be is unknown.

The following are some additional open questions:

- Are voice or data services impacted when using a custom kernel?
- What proprietary drivers are phone manufactures using? Can these proprietary drivers operate with custom kernels and USB host/OTG mode?
- Will Android OS updates continue to be received when running a custom kernel?

In summary, the development required to connect an Android smart phone and USB devices includes driver and kernel changes. Since drivers are written and compiled for a specific piece of hardware, unique, although perhaps minor, driver modifications will be needed for each smart phone in the market. In addition, power must be provided to the USB connector to indicate to devices that the smart phone is operating as a USB Host. Once these issues are addressed, an Android smart phone can successfully function as a USB host and connect to a USB device. However, it is recommended that any solution not involving the smart phone vendor, be used only for demonstration purposes due to the root access and software updating issues mentioned above.