MPTCP on Android Marshmallow

Jakob Struye
University of Antwerp

March 5 2017

1 Introduction

This document gives an overview of the steps we undertook to enable Multipath TCP (MPTCP) on an Android smartphone running Android Marshmallow (6.0 - 6.0.1). Previous work had enabled this on Android KitKat (4.4 - 4.4.4). Due to substantial changes to the networking implementation starting from Android Lollipop (5.0), this method was no longer working.

2 Device

We use a Motorola Moto G (1st generation, LTE-capable) codename Peregrine with an unlocked bootloader. This method, with minimal modifications, should work with any phone with Android Marshmallow for which the kernel sources can be found, modified and flashed.

3 Kernel

MPTCP requires a modified MPTCP-aware Linux kernel. Each version of MPTCP is released only for one, recent version of the Linux kernel. The latest version (0.91) is based on kernel 4.1, while most Android Marshmallow distributions still run kernel 3.4. As backporting MPTCP is a large, tedious and error-prone job, this was considered out of scope for this internship. We instead based our work on an existing port of MPTCP v0.86.7 for the Nexus 5 Android KitKat (4.4.4) kernel. Using this port we could create an MPTCP patch for the 3.4 kernel and apply this to the Motorola Moto G's kernel with only a few minor conflicts which were easily resolved. We also found a backport of MPTCP v0.89.5 for the Nexus 5. The regular 0.89.5 release is based on kernel 3.14. Some analysis of the Nexus 5 version showed that this was not a 3.4 kernel with the 0.89.5 MPTCP functionality backported to it, but rather the 3.14 MPTCP-aware kernel with some of the Nexus 5 3.4 kernel's code ported to it. It is basically a 3.14 kernel pretending to be 3.4 to work with Android. This made it very difficult to create a patch for 3.4 kernels from it, so we did not work with this kernel.

4 Operating System

Using regular Android it was tedious to build and flash kernels. We instead opted to use CyanogenMod (CM) 13 which is based on Android Marshmallow. Now the kernel could be downloaded, recompiled and flashed in just a few commands[1]. Booting a fresh CM install only took a few additional minutes, while this could take up to half an hour with the stock Android ROM for the Moto G phone. The CM project has since been abandoned and some repositories are offline, although we still have the required code to build and flash the kernels. The project has been superseded by LineageOS, although we have not yet converted to it.

5 Enabling Both Interfaces

Some additional code is required to enable both the Wi-Fi and mobile interfaces simultaneously on Android. On KitKat this was achieved by the MultipathControl app[2] (requiring root). When both connections are available, Android prefers Wi-Fi so this was enabled by default. To enable mobile, the app first requests a high priority mobile connection. It then explicitly specifies the app wants to use this high priority mobile connection every few seconds to ensure it stays active. Whenever another app tries to send packets using the mobile connection's IP address (which is what the MPTCP kernel will do) the interface is enabled and this will work.

Starting from Android Lollipop the system calls used to achieve this were deprecated and no longer work. Having more than one interface enabled is now officially supported (the KitKat method was somewhat hacky), however we still need an app to enable the second interface. System calls to enable this were pointed out in an issue for the app's GitHub repository and partially implemented in another project [3] which reuses some of the MPTCP app's code despite not being MPTCP-related. This only implemented the first step in enabling both interfaces, so we had to implement the second step ourselves. Additionally some other deprecated calls had to be replaced. As Android will not allow invocation of some network-related calls on the main thread, this had to be performed in a separate thread. We simply start the thread and wait for it to finish on the main thread. This does not avoid blocking the main thread (which is what the restriction is meant to achieve) but we did not consider this to be an issue as this is a fairly quick call only performed once. Finally we noticed that even with the mobile interface enabled, other apps failed to open a socket on it. After some extensive kernel debugging this was traced back to the CONFIG_ANDROID_PARANOID_NETWORK flag. While this flag was

[1] `http://web.archive.org/web/20161224192821/https://wiki.cyanogenmod.org/w/Build_for_peregrine`

[2] `https://github.com/MPTCP-smartphone-thesis/MultipathControl`

[3] `https://github.com/tarunmangla/MultiInterface`

by default not defined in the Nexus 5's 3.4 kernel, it was enabled by default in the Moto G's kernel for Marshmallow. This flag was disabling apps from creating sockets on the mobile interface, as they were not in the AID_INET or AID_NET_RAW groups. We simply removed the flag.

6 Routing

A final hurdle was configuring routing. In KitKat routing rules were fairly simple, and simply adding a small routing table for each interface sufficed. Starting from Lollipop Android adopted more rigid routing rules. For an explanation of what each rule was intended for, we needed to analyze the source code of the netd system binary [4]. The 'implicit network' rules (priority 19000) here were interfering with MPTCP. These rules ensure that once an app requested a connection without explicitly requesting an interface, the app would continue to use that interface. With this enabled, apps such as browsers would not pick up on a change in default network until they were killed and restarted. When MPTCP is enabled in the app, we remove these rules. We added high priority (5000) rules that redirect requests with a specified source IP to the right interface. MPTCP requests for a second interface always specify that interface's IP address. Finally we needed to modify the default network rule already present (priority 22000). The modified rule is given priority 21500, giving it precedence over the regular default rule.

When an interface is restarted (e.g. because connection was lost) netd will rewrite the rules related to that interface, possibly interfering with our rules. To disable this we need a custom netd binary. Android will not boot when system binaries have been tampered with when SELinux is active. To set SELinux to permissive mode we need to recompile the kernel with a flag specifying this. There is no way to permanently set it to permissive otherwise. The netd source code is present within the CM sources. The directory containing the system binaries (/system) is usually mounted in read-only mode. The only way we managed to mount it as read-write was in recovery mode through the ADB shell. The system directory is not mounted there by default, and can be mounted from /dev/block/mmcblk0p34 on the Moto G.

7 DNS

When browsing the internet with MPTCP enabled we noticed DNS resolution was failing with the mobile network as default. Running tcpdump on the phone showed that DNS requests were either going to the Wi-Fi interface's DNS or

[4] `https://android.googlesource.com/platform/system/netd/+/android-5.0.0_r6/server/RouteController.cpp`

to localhost. The Wi-Fi DNS was set to the ISP's DNS servers, which do not reply to DNS requests not coming from one of the ISP's IP addresses. There is no way to set DNS servers programmatically. The only method we could find no longer works as of Lollipop. We instead add some iptables NAT rules redirecting all traffic on port 53 to 8.8.8.8, Google's public DNS server. Note that this disables having a secondary DNS server.

8 Summary: Installing MPTCP on Marshmallow

Follow these steps to enable MPTCP on Android Marshmallow. This will wipe the phone. Using CyanogenMod/LineageOS is not mandatory, but finding and flashing the kernel may require more effort without it. Note that non-CM operating systems are not rooted by default, which is required with MPTCP.

1. Install a CM13 ROM on the phone
2. Download CM13 sources for the phone including kernel (currently impossible, repositories down)
3. Apply our provided MPTCP patch to the kernel and solve any conflicts. (kernel-source-folder$ patch -p0 < mptcp.patch)
4. Build the kernel ($mka bootimage)
5. Flash the kernel (out/target/product/{PHONE CODENAME}$ fastboot flash boot boot.img)
6. Install the MultipathControl APK.