Mobile Application On-Device Testing at Google Scale

Denis Krupennikov
William Hester
Diana Cortes
Chris Cole

Date: March, 2022

Abstract

Over the years, the mobile devices landscape has grown to hundreds of OEMs and tens of thousands of device models. This landscape has made it difficult to develop quality mobile applications that are user-friendly, stable, and performant. To accelerate mobile development and to help developers build better performing, more stable apps, Google built a large Mobile Device Farm that allows developers to test their mobile applications. In this document we share lessons learned while building the Mobile Device Farm, including pitfalls and successes, to help the mobile development community leverage our experience and build better mobile apps. While we describe both Android and iOS, we primarily focus on the Android Open Source Project (AOSP) because it is highly diverse and dynamic. We've scaled from 10s of devices to 10s of thousands, and from hundreds of tests a day to millions.

Introduction **2**

Architecture overview **5**

Our journey **6**

Wi-Fi network **10**

Electromagnetic pulse-resistant rack **14**

USB hubs **19**

Extend device life and prevent fire hazards **21**
- Battery-swelling solution 21
- Heat solution 22

Android Open Source Project (AOSP) **22**
- Device resets and cleanup 22
- Solutions for device resetting 23
- ADB stability 24
- Battery charge management 25

Introduction

Building usable, stable, and performant mobile applications is becoming increasingly difficult. Currently, there are more than 50K Android device models (Fig. 1) produced by 5K OEMs, which are using 750 distinct systems on a chip. These devices have different form factors, screen sizes, and resolutions.

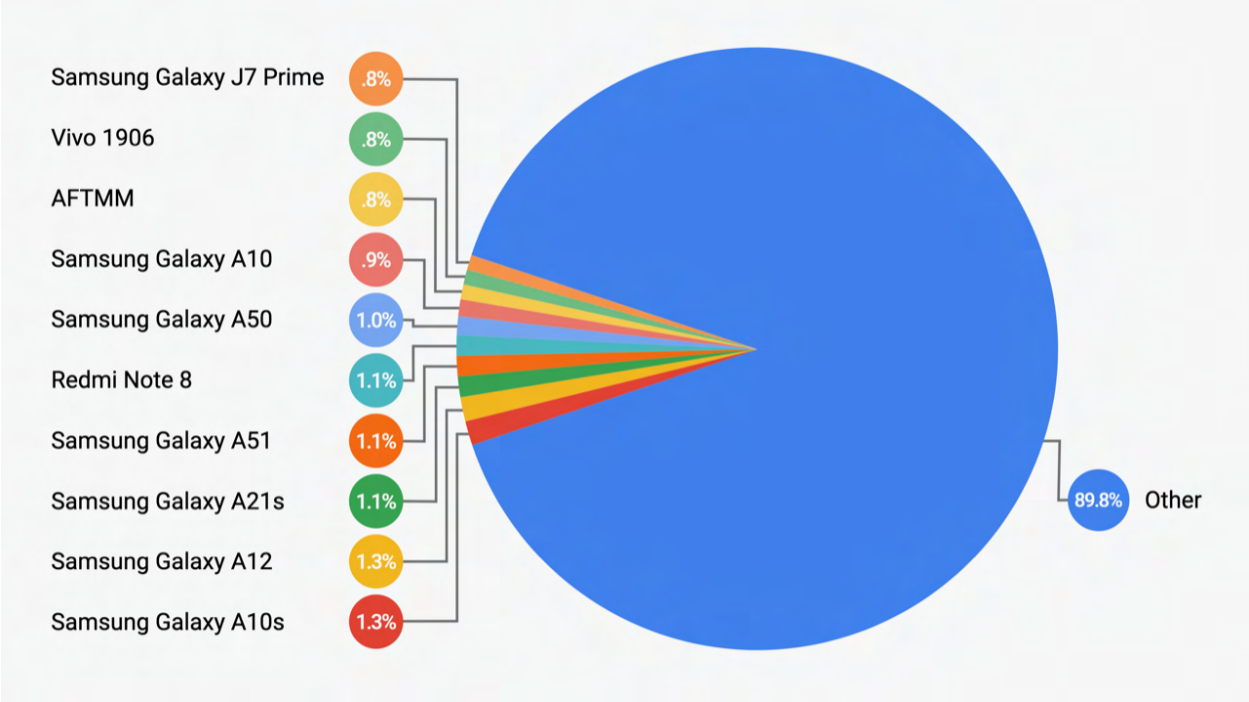

Fig. 1 - Top 10 Android models 2H2021 by AppBrain users (Source: AppBrain, Nov 2021)

To distinguish themselves in the market, each original equipment manufacturer (OEM) customizes user interface elements and introduces their own libraries to control device behavior, like operating system (OS) and app updates (Fig 2).

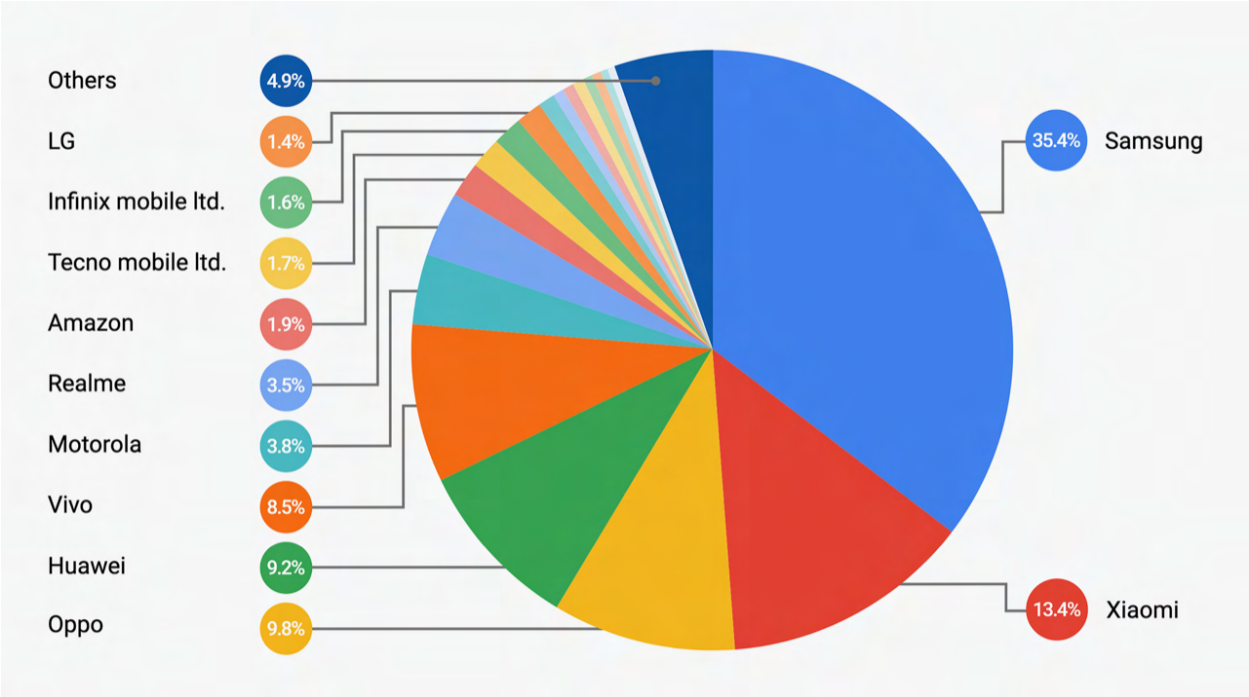

Fig. 2 - Market share per Android phone manufacturer 2H2021 by AppBrain users (Source: AppBrain, Nov 2021)

Android itself has a long tail of OS versions (Fig 3). The number of Android device models, distinct systems on a chip, and long tail of Android versions make the mobile landscape diverse and difficult to understand. The customization and diversity of Android helps Google serve 3 billion mobile users and a huge variety of needs, but the complexity of the Android ecosystem also makes it hard to build quality applications for user devices.

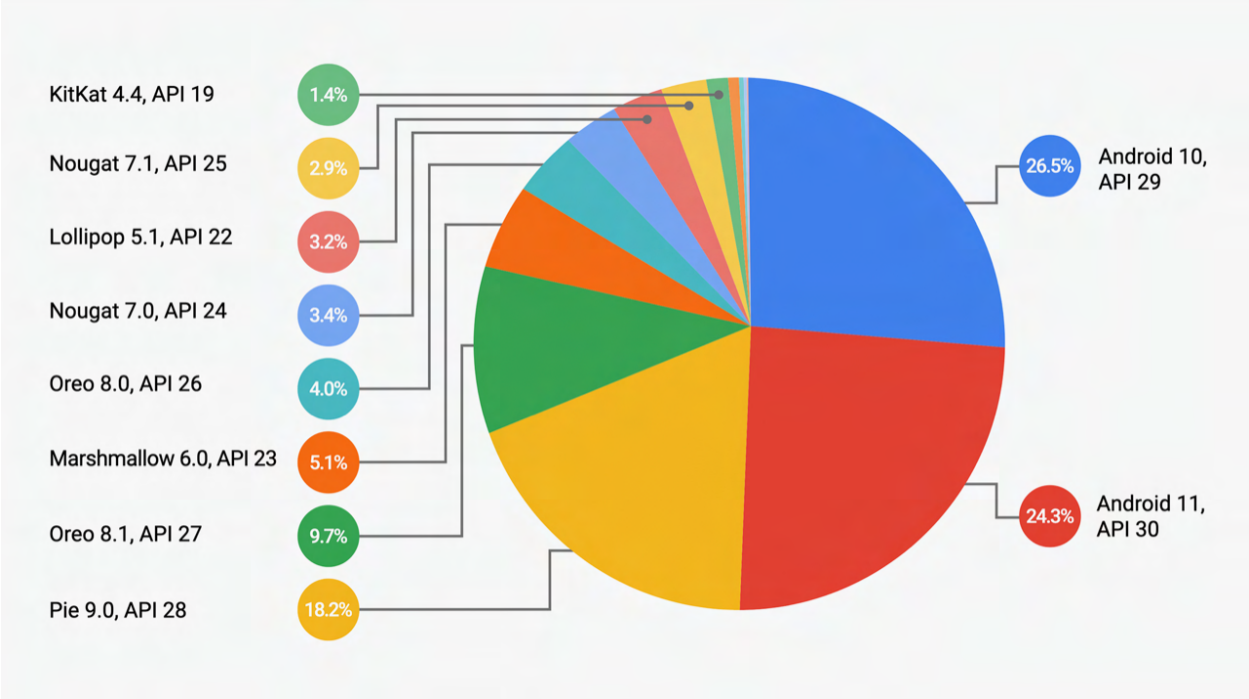

Fig. 3 - Android versions distribution 2H2021 (Source: Android Studio)

App developers face a difficult choice. They can test mobile apps on a limited number of devices and hope that their apps work on the untested devices, or they can test on many devices to cover most of their user base at great expense. Most developers try to find the middle ground, striking a balance between coverage and cost.

To scale internal mobile development and to help external developers build better mobile apps through improvements to AOSP, Google built the Mobile Device Farm. In this document, we discuss our journey to improving the mobile development landscape, including achievements like building electromagnetic pulse-resistant racks and improving AOSP to better integrate Android devices into testing frameworks.

Architecture overview

There are several different architectures for testing mobile applications on physical devices, the most typical of which is illustrated in Fig. 4. In this common architecture, a testing framework like Appium or EarlGrey is run on a host machine. The mobile device is connected to the host over a USB cable. The USB cable provides the power to the device and enables data communication between the host machine and the device to drive the test. To gain access to the internet, the mobile device is connected to a Wi-Fi Access Point, which in turn is connected to the internet.

Fig. 4 - Typical mobile test architecture.

This typical mobile test architecture is a simple design, which many real-world implementations augment with additional components. The host machine is usually connected to multiple devices. Since the host machine's primary role is to bridge I/O and coordinate the tests, a single low-power host machine can usually support many simultaneous tests. An external multiport USB hub facilitates this connection between a host machine and its mobile devices. The hub passes USB data but must also provide sufficient power to charge all connected devices. As the number of devices increases, more Wi-Fi access points must be deployed to spread the Wi-Fi traffic across the devices. However, as the number of devices and access points grows, so does Wi-Fi interference. To reduce interference, it is necessary to segment the devices and access points in Faraday Cages.

We will look at each of these architecture components and their complexity in this document.

Our journey

In 2015, the earliest version of the Google Device Farm had fewer than 1,000 devices connected to pre-built 30-port USB hubs and consumer-grade access points. As the number of devices increased, we found that devices were unable to remain connected to the Wi-Fi due to radio channel saturation. That first scaling bottleneck was the beginning of a multi-year journey to build a lab with tens of thousands of devices capable of running over one million tests a day.

To fundamentally improve our Wi-Fi design, we brought the network engineering team on board to build a reliable enterprise Wi-Fi network. This network was the first production Ultra High-Density (UHD) Wi-Fi network at Google built on Aruba Access Points with 24x7 NOC (Network Operations Center). For more information, see Wi-Fi Network.

After implementing the production Wi-Fi network, we grew the Mobile Device Farm almost 3x before the next scaling bottleneck became evident. Unfortunately, even with the high-quality Wi-Fi network, we had so many devices in the data center that devices started experiencing issues with Wi-Fi interference between devices. We had a choice to expand to more locations, which wouldn't be scalable in the long term, or to control Wi-Fi interference. between devices. We chose to control Wi-Fi interference.

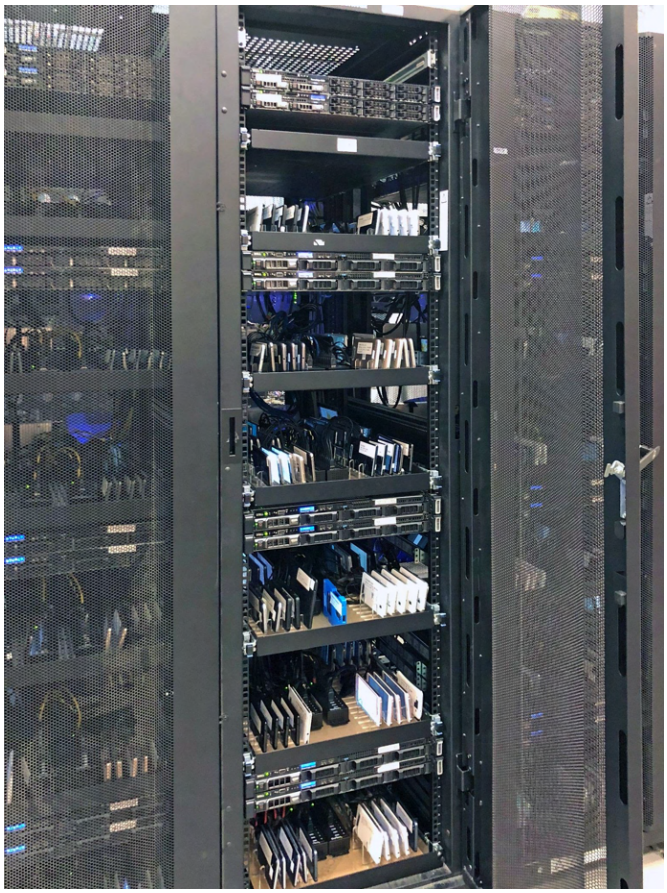

Fig. 5 - Unshielded device rack.

The best-known method for containing Radio Frequencies (RF) in an enclosed space is Faraday Cages. Unfortunately, Faraday Cages, which are often spec'd to federal standards and are used to shield equipment from electromagnetic pulse, are expensive and bulky. Moreover, we didn't require the complete isolation that Faraday Cages provide. Instead, we only needed a degree of RF signal attenuation sufficient to prevent devices in adjacent racks from significantly

interfering with Wi-Fi. We also needed the rack to fit the standard data center rack footprint, while ensuring sufficient air flow for cooling capability at a reasonable price. We weren't able to find a solution available on the market, so we decided to design and build it ourselves. For more information, see Electromagnetic pulse-resistant rack.

Working with the Google Technical Infrastructure Platforms team and external vendors, we built the electromagnetic pulse-resistant rack to sufficiently attenuate RF signals, and we placed three access points (APs) in each rack. This approach isolated Wi-Fi and other RF signals, like Bluetooth and cellular signals within each rack, eliminating network scaling limitations and allowing us to direct our attention to resolving other challenges, like device power needs.

As we have observed over the years, mobile devices are becoming increasingly power hungry. Devices are typically connected to USB hubs, which supply power. Device power needs increase the requirements for the USB hubs to deliver adequate power to each individual device. If a device runs a CPU/GPU-intensive application, the device might draw too much power from an individual USB port. Power consumption increases with each new device connected to the hub. If a device doesn't receive consistent, sufficient power, eventually the internal device battery drains and the device goes offline. We tried, but could not find, any USB hubs that met our needs. We then decided to build a custom hub to our specifications in partnership with the Google Technical Infrastructure Platforms team and external vendors. For more information, see USB hubs.

When mobile devices are used for production testing, they are connected to a USB hub, which continuously provides power to the device. Continuous charging causes the battery to always remain 100% charged, which eventually leads to overcharge and battery swelling. Overheating is another cause of battery swelling. App tests on devices are typically run continuously, which is not a typical consumer workload. Continuous tests cause devices to overheat, which in turn causes the batteries to swell. We discuss this in Extend device life and prevent fire hazards.

Another issue is the software environment on the device. Device stability and reliability are just as essential as the reliable and performant infrastructure. Consider, for example, that device storage needs to be cleaned after each use to support repeatable tests and to ensure that the artifacts do not cross-contaminate tests or leak information. Mobile devices are consumer-grade and aren't designed to be used for a production-grade Continuous Integration/Continuous Deployment (CI/CD) workload. These constraints created several challenges, which we discuss in AOSP.

This is the list of the challenges and our solutions so far:

Problem	**Solution**

Radio Frequency (Wi-Fi, Bluetooth,...) interference between thousands of devices and Access Points	- Ultra High Density (UHD) Wi-Fi network - Electromagnetic pulse-resistant rack
USB hubs powering a device don't have enough power and data connectivity stability	- Custom, high-power USB hub on the latest Infineon USB chip
Device overcharging and overheating cause the battery to swell, creating a fire hazard	- Visual device inspection and audit - Heat sinks attracted to the devices - Charging device battery to max 80%
Reliably clean up devices (erase all data, packages, and artifacts) between test runs	- Factory-resetting devices with the Test Harness Mode - Custom device re-imaging tools - Deleting data and packages
Android Debug Bridge (adb) stability and reliability	- System `adb` watchdog

The infrastructure for mobile device testing is continuously evolving to achieve better reliability and total cost of ownership (TCO). We plan to implement several new features to further improve the AOSP, which we will share in future publications.

Wi-Fi network

A stable and well-performing Wi-Fi network is a key component of any mobile device farm because it is the main way in which mobile devices communicate and access the internet. There are other ways to enable internet access for devices, like reverse tethering, but the known alternatives didn't meet our requirements. For example, you can reverse tether internet traffic from a device through the USB connection to the host machine, and from the host machine to the internet. By sending all traffic through a reverse tether, we can eliminate the Wi-Fi connection entirely and increase internet connection stability. While reverse tethering can be used for a subset of applications, it has drawbacks that are outside the scope of this document.

To address the need for a stable, well-performing Wi-Fi network, we partnered with Google Enterprise Network (GEN) Wi-Fi experts to build an Ultra High-Density (UHD) production Wi-Fi network. As opposed to a traditional high-density Wi-Fi network where humans use mobile and computing devices in large venues, an ultra high-density Wi-Fi network does not assume human interaction. UHD Wi-Fi networks are also effective because Wi-Fi clients operate in a much smaller space, with racked shelves in data centers.

The 802.11 standard supports the use of different RF ranges, including, but not limited to, 2.4 GHz, 5 GHz, and 6 GHz. The UHD Wi-Fi network was designed using 802.11ac-capable wireless access points to support 802.11n and 802.11ac mobile devices. These devices could

also be dual-band-capable (2.4GHz/5GHz) or single-band-capable, typically operating only on 2.4 GHz.

One of the biggest challenges of building UHD Wi-Fi networks is radio frequency (RF) spectrum availability, which can vary based on country-specific regulations. As illustrated in Fig. 6, the 2.4-GHz spectrum allows for three non-overlapping 22 MHz-wide channels, while the 5-GHz spectrum supports up to 24 non-overlapping, 20 MHz-wide channels. The re-use of channels within the same RF space results in co-channel interference (CCI) and the eventual performance degradation of a Wi-Fi network due to contention in the RF medium. For more information, see section 17.3.10.6 CCA requirements in the 802.11-2020 standard (4). Therefore, in an ideal situation, Wi-Fi networks should be implemented with little to no channel re-use.

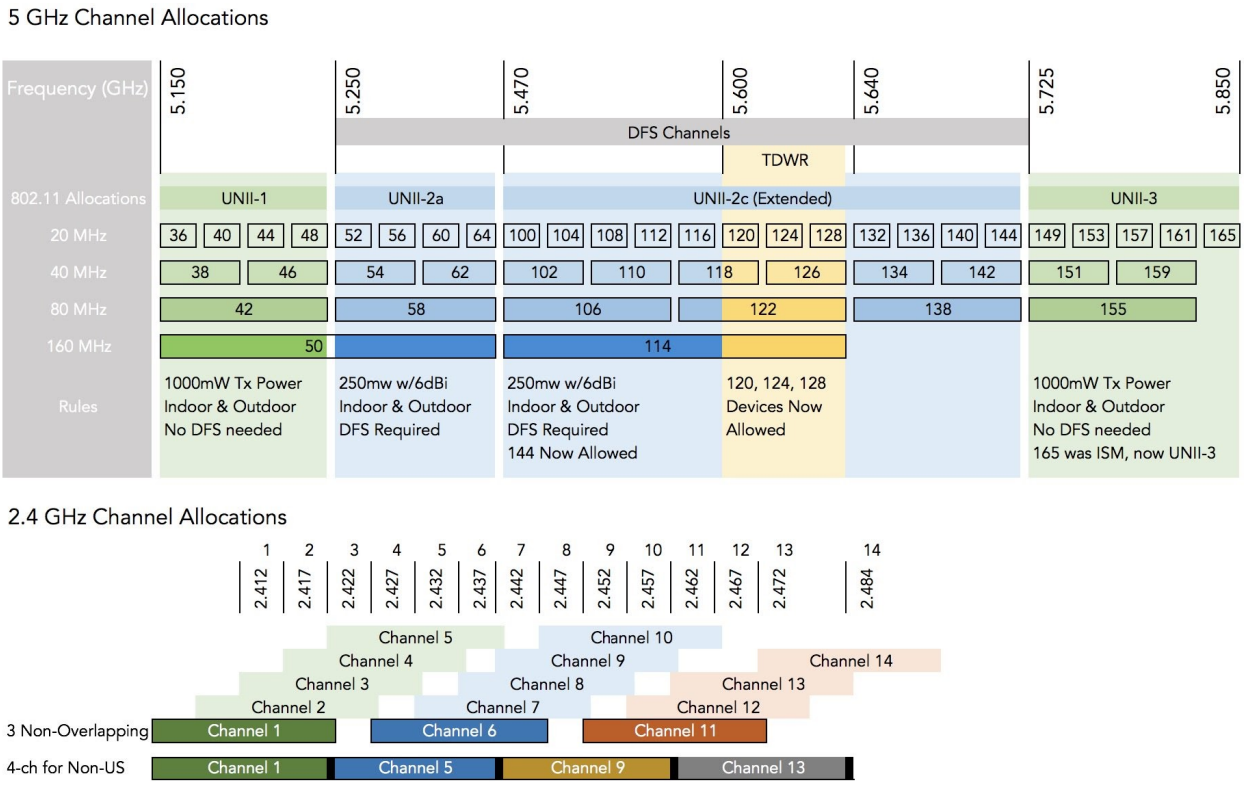

Fig. 6 - 2.4- and 5-GHz channels. Source: @KeithPParsons (WLAN Pros).

Mobile device farm deployments consist of many racks housing mobile devices in close proximity to each other. In such tight spaces, it is unavoidable to re-use channels in the 2.4-GHz and 5-GHz range. Channels are becoming even more saturated, especially in countries where part of the spectrum cannot be used. For example, the UNII-3 band is not available for use in Japan.

Today, as seen in Fig. 7, UHD Wi-Fi deployments with a rack include 3 802.11ac dual-radio access points installed inside the back door of the rack.

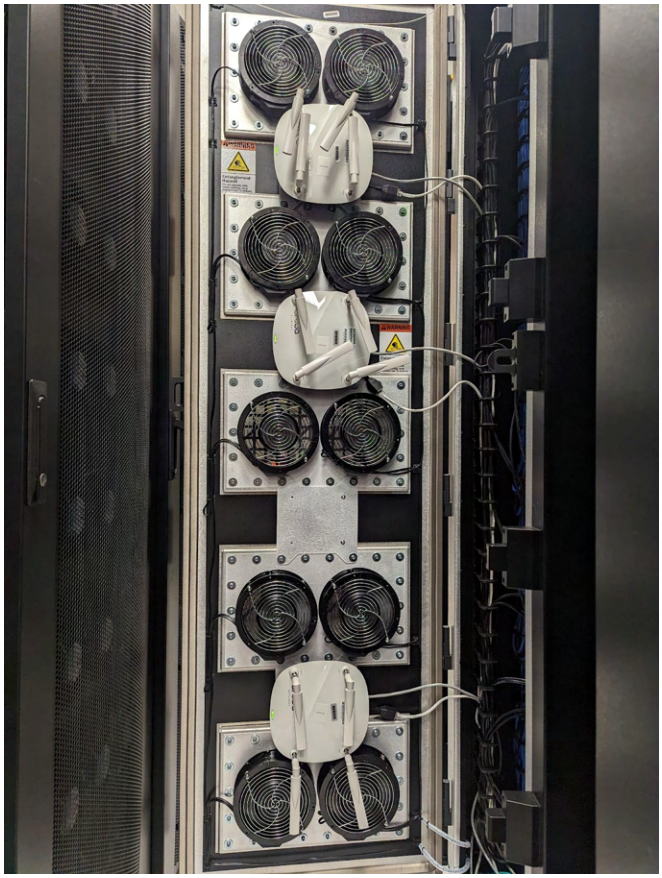

Fig. 7 - UHD Access Points mounting

Given the proximity of APs within a rack and the proximity of the racks to each other, APs are installed with 25dB RF attenuators and omni-directional dipole antennas to reduce the signal strength to the optimal range. Each set of three APs per rack has its 2.4-GHz radios configured on a static non-overlapping channel (1, 6, or 11). To further reduce the potential of adjacent channel interference, the 5-GHz radios on a rack's APs are configured using 20-MHz channels with at least ~80 MHz separation. Fig. 8 illustrates an example of this approach.

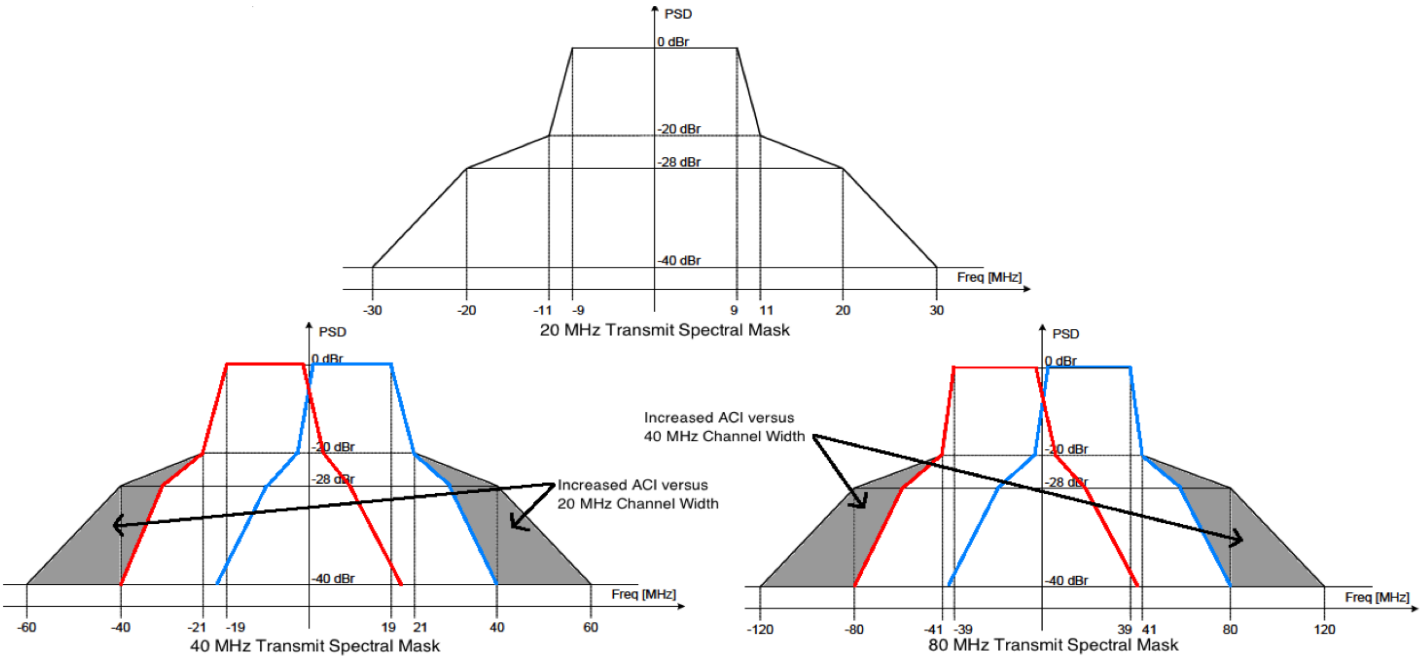

Fig. 8 - 20 MHz / 40 MHz / 80 MHz Spectral Mask

An example of UHD AP deployment and a static channel plan is shown in Fig. 9.

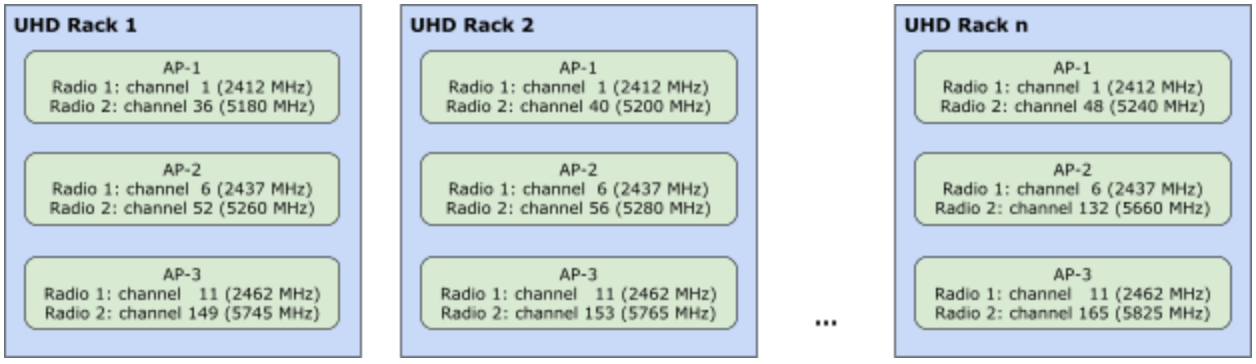

Fig. 9 - Sample AP deployment and channel plan

Currently, we are working to support both Wi-Fi 6 and Wi-Fi 6E mobile devices using the vendor-neutral OpenConfig architecture. This architecture will allow not only for higher performance, but it will also support expanded RF spectrum availability in the 6-GHz band and richer streaming telemetry data that can help validate network design and performance.

Electromagnetic pulse-resistant rack

In the early stages of development, our Mobile Device Farm had all devices and Wi-Fi access points deployed in the open racks (Fig. 5). In this scenario, the devices could see and talk to each other.

We partnered with the Google Technical Infrastructure Platforms team to better understand how to achieve our goal to get –82-dBm RF attenuation between racks, with minimal cost. As per section 17.3.10.6 of the IEEE-802.11-2016 standard, our goal was to reduce sensitivity of the Wi-Fi devices. This goal proved adequate to prevent devices and APs on the adjacent rack from interfering with each other. Over the course of six months, we built several prototypes to measure the effectiveness of different designs and to understand the constraints.

To design a shielded rack with high shielding performance, we considered the following:

Consideration	**Description**
Mechanical structure design	No slot/seams on the main frame of the rack except for doors, perforation panel, and cable penetration.
Perforation design	Should be based on the shielding performance: Honeycomb or other perforation panel and the size of the perforation cells.
The gasket design for the doors and other areas	The gasket thickness and the compression ratio should be carefully designed to meet the functional requirement. Gasket selection is important for the shielding performance, especially for the doors.
Cable penetration design	The RF field should not be leaked out or coupled in through the cables. The power filters should be added for power cables to attenuate the RF field coupled by a power cable. For signaling cables, optical fiber should be used and the penetration structure should be a waveguide opening instead of a planar opening, to prevent RF leaks.

Fig. 10 - Shielded racks experiments

We tested each design in the anechoic chamber (Fig. 11) and used the data to guide our decisions. Surprisingly, it was incredibly difficult to sufficiently attenuate RF in the rack. In particular, all parts of the rack had to have excellent electrical contact and shouldn't have any dielectrics, like paint, in between them. Dielectrics can easily conduct RF signals regardless of how tightly the parts are connected. The radio waves can travel on the surface of the paint or other dielectric and penetrate inside the rack, compromising RF-shielding.

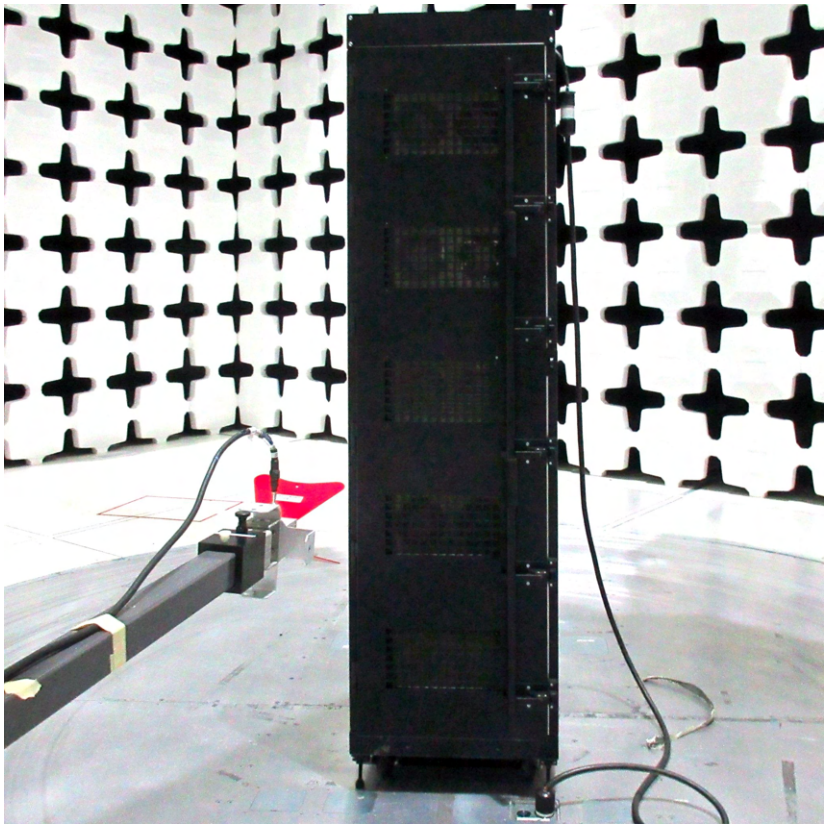

Fig. 11 - Testing the rack in the anechoic chamber.

In Fig. 11, we give an example to show the difficulty in preventing RF signals from penetrating the rack. Figs. 12 and 13 show Wi-Fi channel numbers for the 2.4-GHz band on the X-axis, and the RF signal strength in decibels per milliwatt (dBm) on the Y-axis. The RF signal strength is measured inside the rack, while the RF radiation source is located outside.

Fig. 12 shows nearly ideal RF attenuation inside the rack, which is mostly below -90 dBm. This attenuation met our objective, because the Wi-Fi APs and devices sensitivity threshold is –82 dBm.

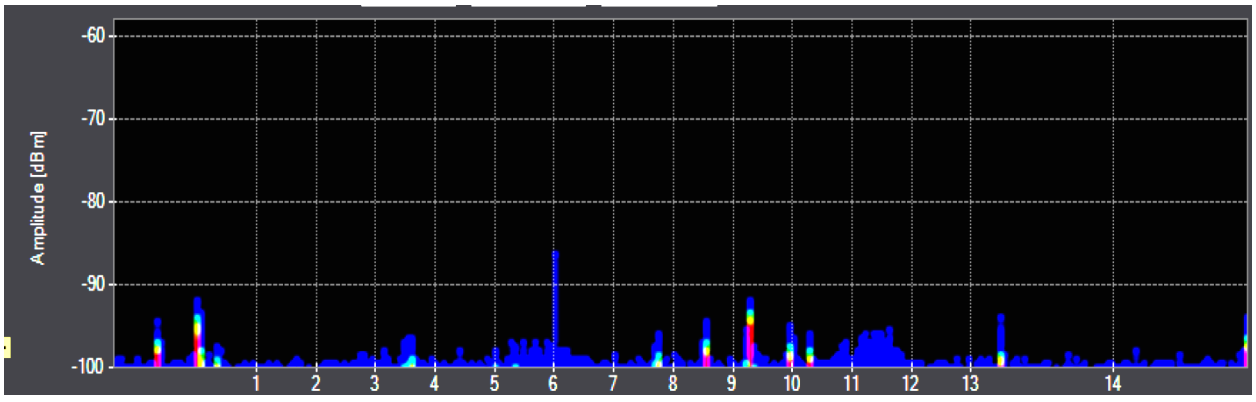

Fig. 12 - Very good RF attenuation. The X-axis represents channel numbers for the 2.4-GHz Wi-Fi band. The Y-axis represents the signal strength in dBm.

Fig. 13 shows RF signal strength inside the same rack with a single copper wire going into the rack. The RF signal easily penetrates the rack over a single copper wire, compromising RF shielding.

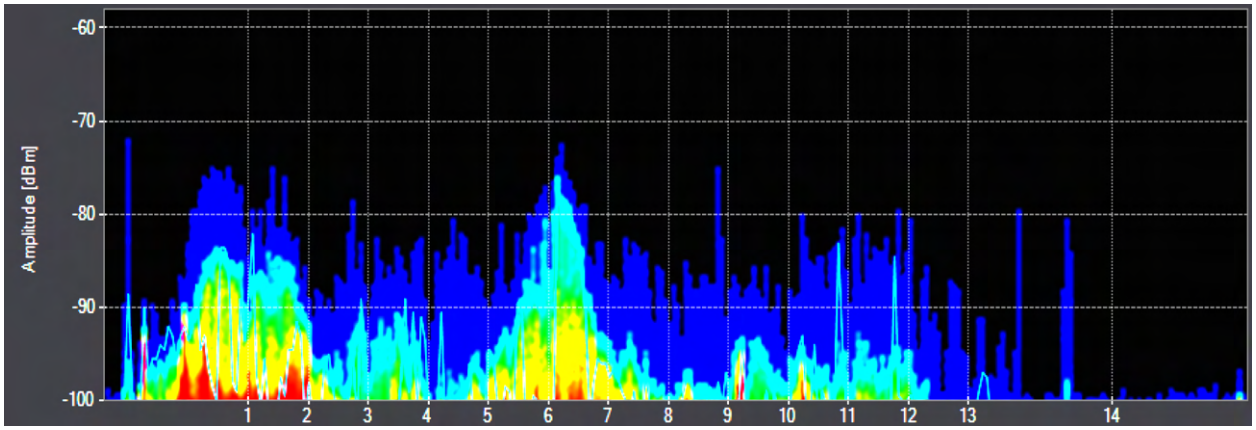

Fig. 13 - RF signal leaking into the rack over a single copper wire. The X-axis represents channel numbers for the 2.4-GHz Wi-Fi band. The Y-axis represents the signal strength in dBm.

After the initial rack design, we tested it in the anechoic chamber for RF leaks to iterate and continue improving the design. Fig. 14 shows where we added copper tape to prevent the leakage of 5-GHz Wi-Fi signal coming from the gasket of the door on the latch side. Copper tape in that area improved the shielding by ~10 dB.

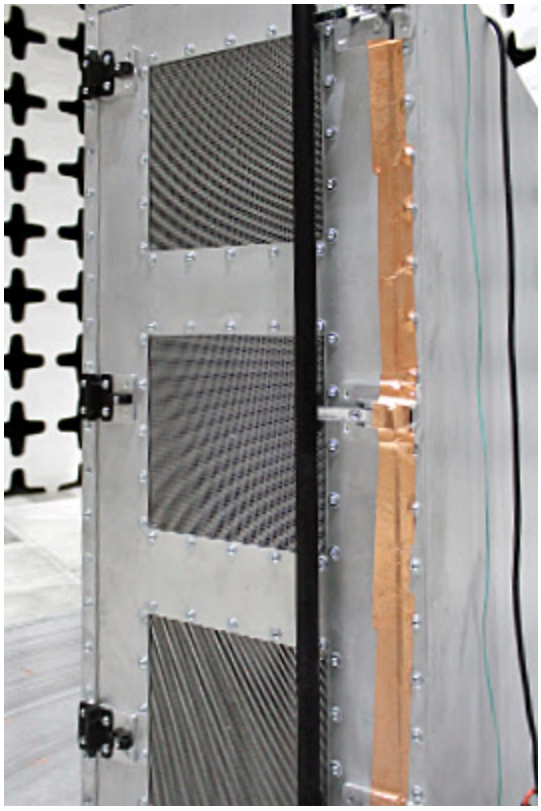

Fig. 14 - The leakage for 5 GHz is coming from the gasket of the door on the latch side. Copper tape in that area improved the shielding by ~10 dB.

After many experiments and failed attempts, we developed the final design and manufactured the rack for data center deployment, as shown in Fig. 15.

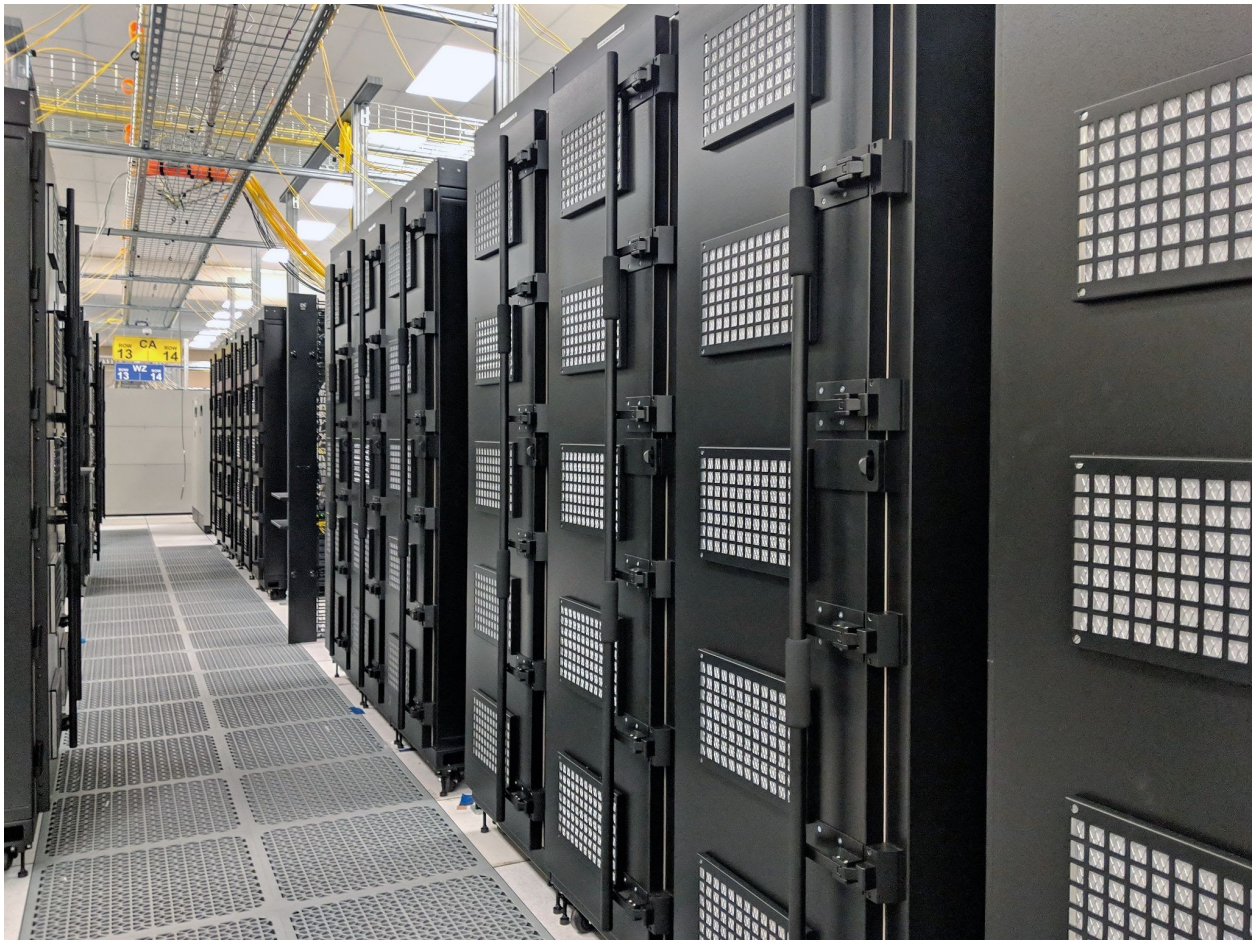

Fig. 15 - Electromagnetic pulse-resistant rack deployed in the Google production data centers.

To further build on our success, we worked with a second vendor to design an alternative rack, which is commercially available.

USB hubs

As we observed, mobile devices are becoming more power hungry. In the typical test setup, various devices are connected to a single USB hub, which provides data connection and power to the devices.

As our Mobile Device Farm evolved, we started looking for more powerful, high bandwidth and better RF-shielded USB hubs. The need for RF-shielding USB hubs is not a widely known issue and would benefit from further clarification. USB 3.0 operates on the same frequency as 2.4-GHz ISM band Wi-Fi, Bluetooth, and some other wireless device protocols, and it creates massive radio interference between the devices. For more information, see USB 3.0* Radio Frequency Interference Impact on 2.4-GHz Wireless Devices. Fig. 16 shows a massive RF leak from the USB 3.0 connector, which is right in the middle of the 2.4-GHz Wi-Fi band.

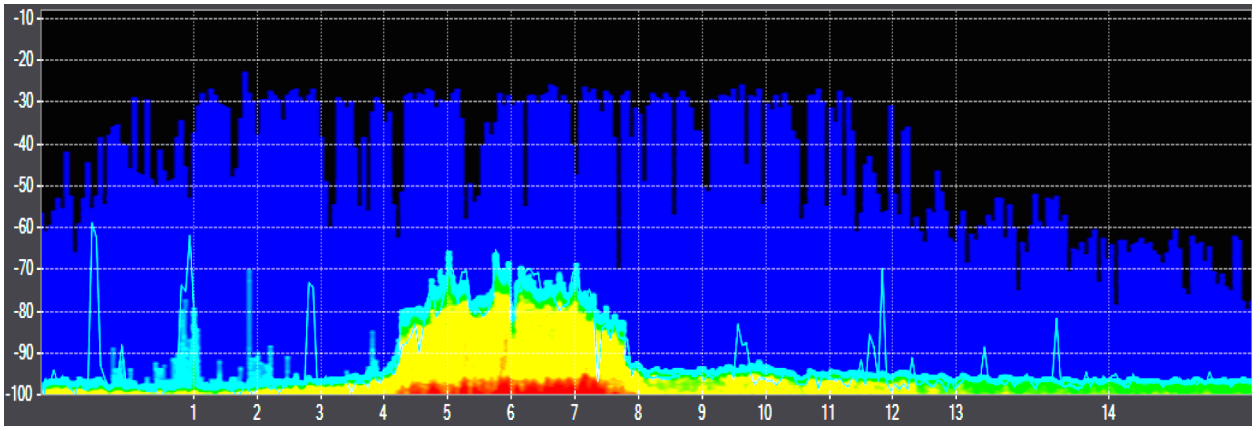

Fig. 16 - Massive USB 3.0 RF leak in the middle of the 2.4-GHz ISM band. The X-axis represents channel numbers for the 2.4-GHz Wi-Fi band. The Y-axis represents the signal strength in dB.

We needed to find a new USB hub that had sufficient RF shielding to prevent RF leakage from the USB ports. We combined the RF shielding requirement with the rest of our requirements for the USB hub:

- 40 USB ports per hub
- 15 W per port
- BC 1.1, 1.2 (Back Compatible)
- USB 2.0, 3.0, 3.2 Gen x2, 4.2
- Apple 1A/2.1A
- USB Power Delivery (PD) Rev. 1 Profile 1 (5V 2A)
- USB 3.1 Rev 1 data support
- 10Gbps uplink

We did not find a market solution that satisfied these requirements. Once again, we partnered with the Google Technical Infrastructure Platforms team and two external partners to design and build the prototype, including custom firmware. We focused on USB hub manageability, RF shielding, and data center safety. Recently, we completed the development of our new, advanced USB Hub, which is being rolled out to our infrastructure.

We are continuously looking for alternative, commercially available solutions. Since we started the project, there are new USB Hubs emerging on the market, but none of these product offerings fully satisfy our requirements.

Extend device life and prevent fire hazards

Battery-swelling solution

When mobile devices are used for production testing they are connected to a USB hub, which continuously provides power to the device. Continuous charging causes the battery to always remain 100% charged, which can eventually lead to overcharge and battery swelling. For more information, see Insight into the gassing problem of Li-ion batteries.

Swelling can damage the lithium-ion battery, causing it to overheat and burst into flames. Our Mobile Device Farm runs tens of thousands of devices in production, and the thermal runaway event of even a single device has the potential to cause a catastrophic fire hazard.

Fig. 17 - Visual inspection of the devices with swollen batteries, and a mobile device with swollen battery.

To prevent fire risk due to swollen batteries, we implemented the following procedural, mechanical, and software measures:

- We inspect all devices weekly and destroy all devices with swollen batteries.

- We position devices with sufficient gaps between them to create adequate separation to prevent a fire from jumping to adjacent devices.
- We ensure that our racks are rated to contain lithium-ion thermal runaway inside the rack, which prevents fire from spreading to other racks in the data center in a worst-case scenario.

Heat solution

Overheating is another cause of battery swelling. App tests on devices are typically run continually, which is not a typical consumer workload. Continuous tests cause devices to overheat, which in turn causes the batteries to swell. We continuously monitor device temperature and prevent tests from running if the internal device temperature exceeds 45°C.

We've also started experimenting with high heat-conductive ceramic heat sinks attached to mobile devices for more efficient heat dissipation. We use ceramic heat sinks instead of copper or aluminum to prevent the heat sinks from interfering with RF signals, like Wi-Fi and Bluetooth. So far, we've observed promising results.

Android Open Source Project (AOSP)

We've worked with our Android platform colleagues on AOSP features, including:

- Reliably reset the device to the known initial state.
- Remove any artifacts from previous tests from the device.
- Keep the device-host machine connection available at all times, even after the devices reset.
- Prevent any setup wizard dialogs after the device reset.

Device resets and cleanup

The key to reliably and consistently running tests on physical devices is the ability to reset the devices to a known initial state by removing data, configuration, or artifacts left over from previous tests. The ability to reset the devices to a known initial state is essential for test repeatability and to prevent data leaks between the tests.

Over the years, we implemented several solutions to reset devices, including device flashing and device cleanup, which we discuss in Solutions for device resetting.

When resetting a device, we have a few requirements. First, data must be cleaned up between tests. Our device lab runs as a multi-tenant environment, so it is essential to wipe out all data.

Our second requirement is that there is no setup wizard. Setup wizards vary between OEMs, increasing navigation complexity. To support a seamless device reset, we sought to skip the

setup wizard and other first-time setup screens that occur when factory-resetting the device, like keyboard dialogs that appear when a user first sets up a device.

Our third and most critical requirement is that we must retain the ability to control the device over USB. If we lose `adb` access to the device, we have to manually re-authorize access to the device. Considering that we have thousands of devices, manual reauthorization would require considerable effort. The conclusion is that we must be able to access the device using `adb` or `fastboot`.

`adb` lets us interact with a device as a standard end user, giving us our sole way of executing tests on a device. `adb` operates on keys stored on a connected computer, which creates an RSA public/private key pair and uses the key pairs to authorize the device each time it connects. Typically, USB debugging requires the user to authorize the connection in an on-screen dialog. When the connection is authorized, a public key corresponding to the host is stored on the device. It is essential to retain this key between device resets or else the `adb` connection is lost.

`fastboot` is the command-line utility for interacting with the Android bootloader. Its functionality includes installation of images, wiping data, and reboot. Most features of `fastboot`, however, require an unlocked bootloader. Many OEMs and carriers do not let users unlock their bootloaders. We primarily use `fastboot` to install new images and to reboot devices that fall into `fastboot` for some reason.

Our last requirement is that there is no human interaction with a device. If a device falls into a bad state, we need to be able to fix the issue programmatically. Sending a human to find a phone in a data center and to click a button doesn't scale well.

Solutions for device resetting

Mobile Device Lab started implementation of the device resetting solutions by building two methods to reset devices, device flashing, and device cleanup.

Device flashing works by creating a golden snapshot of the device and restoring it after each test. A snapshot is created by a binary backup of the data for each partition on the device. Device flashing requires that we are able to unlock the bootloader and install a custom recovery image. A recovery image is the part of the operating system that does the factory reset and has access to all files on the device. A custom recovery image is typically made by the Android community for popular devices with unlocked bootloaders, or by unlocking the bootloader. A custom recovery image gives us root access to all files on the device and the ability to back up and restore files.

However, finding a device that has an unlockable bootloader and a custom recovery image is hard, and it is getting harder as Android's security improves. Device flashing was a great solution for a time, and for Google devices that have unlockable bootloaders, but device flashing takes significant research and can be difficult to implement.

In response to device flashing limitations, we created a tool called *device cleanup*. We attempted to wipe all apps' data, delete all `/sdcard` files, and reset all settings except for the ones we could change without root access. However, the device cleanup process left artifacts behind, and attempting to erase data from the device while it was running did not always work, so we concluded that this approach was not reliable.

We needed a method for resetting devices that worked as reliably as device flashing, with the flexibility of device cleanup. Our first thought was, "How can we build in some functionality like device flashing?" However, making OEMs standardize on a solution for recovery images would be challenging, and there are security implications to forcing all app data to be backed up.

We wanted the simplest solution that would solve our needs. We did not want to mandate that each OEM use `fastboot` or our recovery image, or even that OEMs support our features from a recovery image.

Test Harness Mode implementation was the next logical step in our evolution, a reset method that is built into all Android Q and newer devices.

Test Harness Mode reset works entirely on pre-existing features that are mandated on all Android devices. To leverage Test Harness Mode, OEMs only need to correctly implement the Android Compatibility Definition Document (CDD) requirements. Test Harness Mode piggybacks on the Factory Reset Protection method of persisting data between factory resets to store the `adb` keys (the persistent data block). Test Harness Mode also uses a standard factory reset to wipe data from the device.

When a device is booted up immediately after a Test Harness Mode reset, the `testharness` system service reads persistent data starting before the factory reset. `testharness` uses the persistent data to set up the device, bypassing the setup wizard and writing the `adb` keys to the path where `adb` looks for them.

ADB stability

We also worked with the `adb` team to add a *watchdog* for the `adb` connection. If the device does not have an active `adb` connection for ten minutes – for example, if the device had an issue loading USB drivers – the device automatically kicks itself back into the bootloader. The watchdog lets us restore the connectivity to the device and interact with it using `fastboot`.

We have plans to make the AOSP more usable to groups running device labs, and we will continue to share these enhancements in the future.

Battery charge management

When the device is continuously connected to the power supply or is too hot, the device eventually causes battery degradation and swelling. It also reduces battery life and can potentially create a fire hazard.

Android implemented a new way of battery management, called Battery Defender, to improve battery life. When a device is too hot for a period of time or is continuously connected to the power for more than a specified length of time, Battery Defender limits the battery charge to 80% of capacity. The introduction of Battery Defender is expected to increase battery life and reduce the possibility of battery swelling.

Conclusion

Mobile Device Farm went a long way towards making mobile device testing both scalable and reliable. We scaled the Mobile Device Farm to hundreds of thousands of devices from top OEMs, and we currently run many millions of tests every month. We learned a lot along the way, unlocking the possibility of unlimited scale. We plan to continue this work, and we will share more findings in the future.

Call to action

- Follow the Firebase blog at firebase.googleblog.com for future updates on our work.
- Contact us at mobile-device-farm@google.com.

Further reading

- Firebase Test Lab
- Play Pre-Launch Report

ACKNOWLEDGEMENTS

Justin Broughton
Peter Doege
Tobi Siu
Adam Hicklin
Kevin Moran
Sumit Agrawal
Walter Susong
Niranjan Tulpule
Tara Vazir Ghaemmaghami
Andy Aneals
Robert Geraghty

Sinead orla Curley
Stanley Mui
Stephen Griffin
Tim Coble
Stephen Griffin
Bill Barry
Xu Gao
Thanh Tran
Mike Bostaph
Christopher Marrale
Chee Chung
Sam Phillips
Victoria Hurd
Eric Burnett

REFERENCES

1. Faraday Cage
2. Reverse tethering over `adb`/USB for Android
3. IEEE-802.11-2016 standard
4. USB 3.0* Radio Frequency Interference Impact on 2.4GHz Wireless Devices
5. Insight into the gassing problem of Li-ion batteries