EMTERIA

# Keeping Embedded Android Up To Date

## OTA challenges with professional Android devices

The AOSP and AAOS Meetup - 20th July 2022

# Igor Kalkov-Streitz

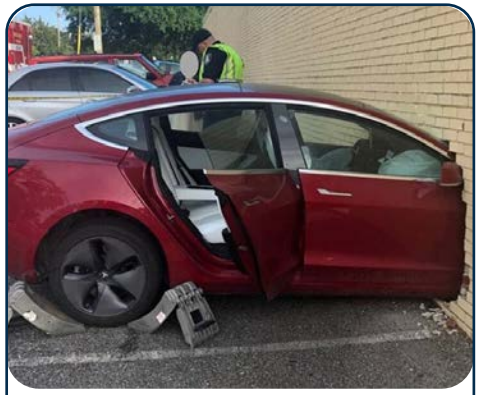
About me in short

- Located in Aachen, Germany
- Degree in Computer Science
- PhD in the Area of Software for Embedded Systems with focus on Real-Time Systems
  - Robotics: DARPA Robotics Challenge 2015
  - Real-Time Linux and Android
- One of the original maintainers of Android for RPi
  - https://github.com/RTAndroid
  - https://github.com/android-rpi
- Founded emteria in 2017
  - Focus on customizing the Android platform
  - Delivering Over-The-Air updates
  - Remote management capabilities

**OS updates are not optional. Retrofitting the update process is not impossible.**


Error-prone software


On-site maintenance



Exploited vulnerabilities

Regulatory compliance

EMTERIA

# AOSP only provides a small part of the solution

Many implementation details are device specific or require additional server infrastructure

- Over-The-Air (OTA) updates are well known from our smartphones, but typically requires user interaction on the device
- Companies have thousands of professional devices spread over large areas, sometimes with no users in front of them
- Rebuilding and reflashing is not feasible as for private use-cases

- The overall process seems easy
  - Step 1: Building an OTA package alongside an Android release
  - Step 2: Installation of an OTA package is also part of AOSP
  - Profit

- What is not included?
  - Device specific config (partitions, services)
  - Release specific config (versions, changelogs)
  - Package delivery (infrastructure, consistency)
  - Package installation (client, process)
  - …

# Building a (F)OTA infrastructure for Android

Some of the challenges to solve and things to consider

## Build

- Build environment
- Build automation
- Test automation
- Signing support
- CVE monitoring
- Dependency management
- Release versioning
- Release changelog
- Delta OTA support
- Partition layout
- Notification process
- Consistency validation
- …

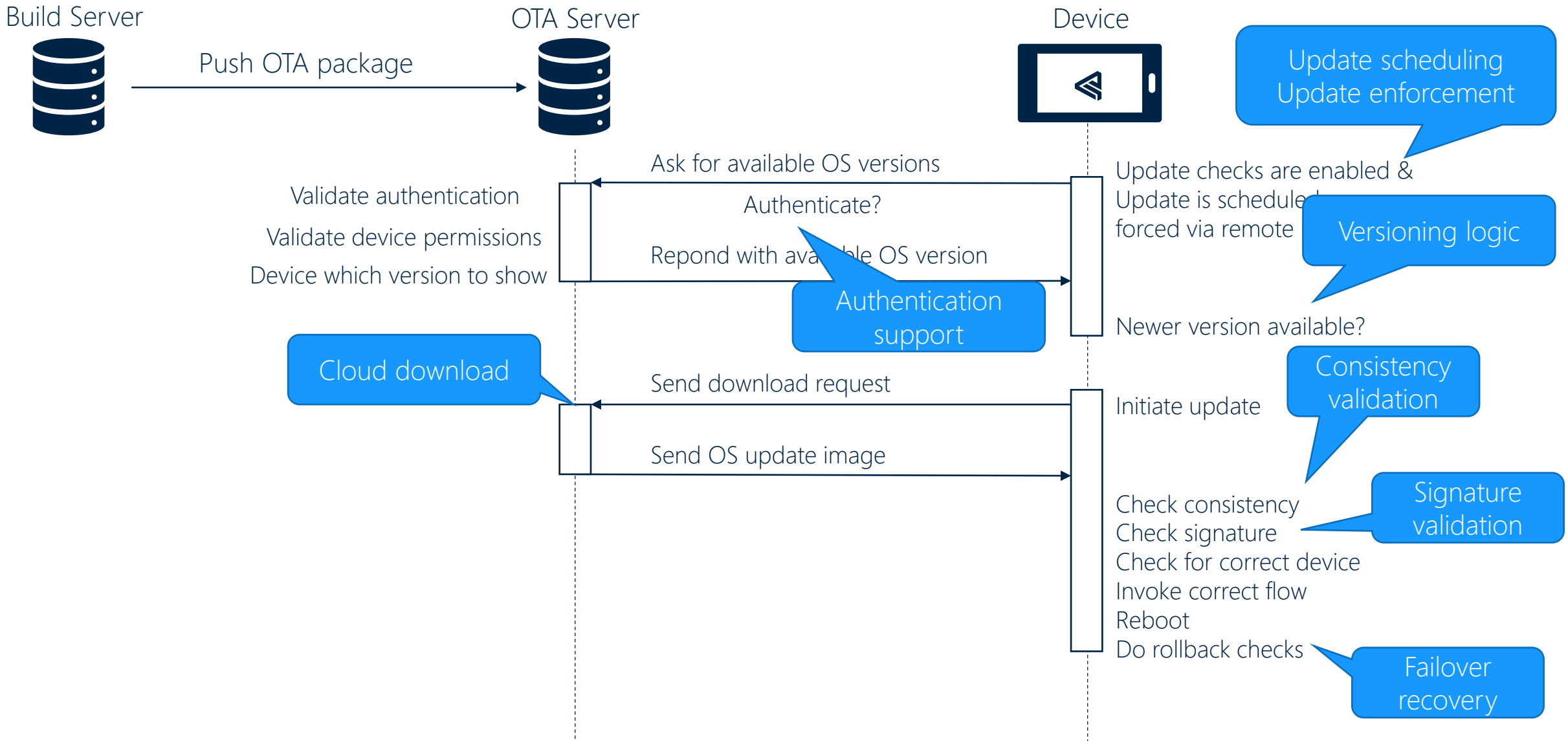
## Upload

- Hosting infrastructure
- Cloud storage
- Authentication mechanism
- Permission management
- Signature validation
- Consistency validation
- Multi-device support
- Versioning support
- Changelog support
- Release channels
- Administration UI/API
- …

## Install

- Client application
- Integration with AOSP
- Cloud download
- Authentication support
- Versioning logic
- Signature validation
- Consistency validation
- Update scheduling
- Update enforcement
- Failover recovery
- …

# Installing OTA Updates



# General considerations

- Secure connection using TLS over TCP/IP
  - Initiated by the device
  - If HTTP is still important (it shouldn't be) ->
- Release channel
  - Nightly, beta, stable
- Versioning
  - Semver or something customized?
- Incremental updates
  - Safes space as it contains only binary patches ->
  - Full OTA up to 500 MB incremental 1-60 MB
  - See https://source.android.com/devices/tech/ota/tools#incremental-updates
- Rollback protection
  - Updates with lower version numbers are refused.
  - Protects against already patched kernel exploits
  - Downgrade possible with "—downgrade" to ota_from_target_files

```
AndroidManifest.xml
<application
    …        android:usesCleartextTraffic="true">
```

```
ota_from_target_files –i \
    PREVIOUS-*-target_files.zip \
    dist_output/*-target_files.zip \
    incremental_ota_update.zip
```

# A/B (Seamless) System Updates

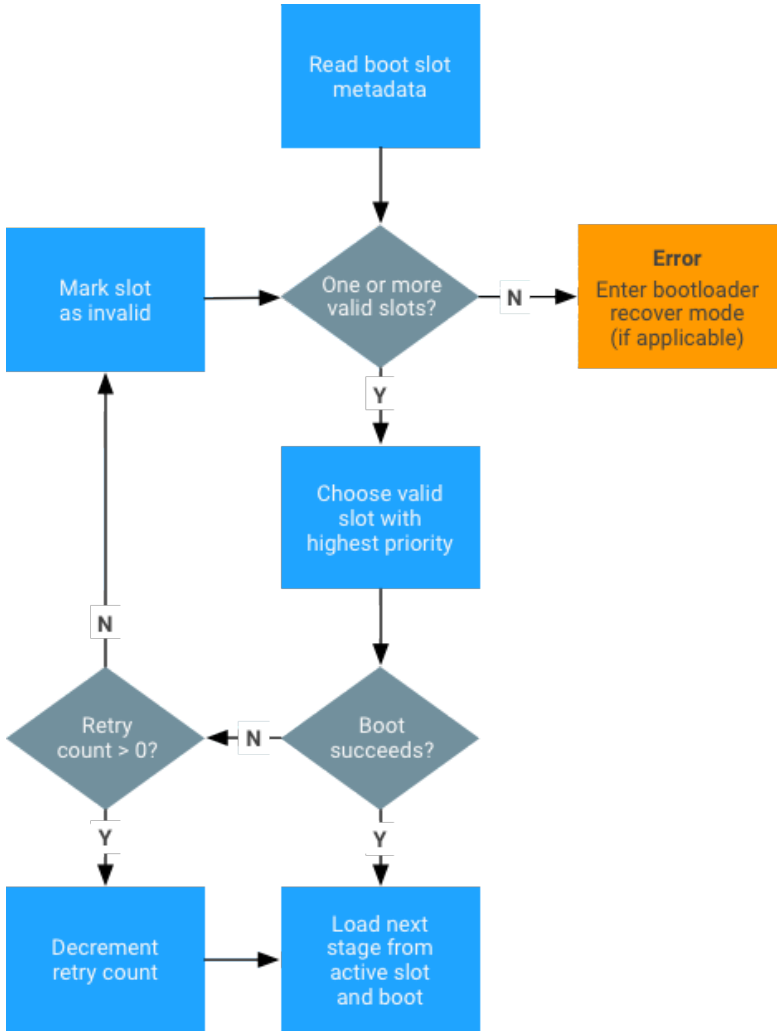
- Duplicate all important (system) partitions like system, vendor, boot, etc. with _a and _b suffixes (see https://source.android.com/devices/tech/ota/ab)
- Updates are installed on other partitions. If update fails, the previous version is still available on the current partitions and can be booted.
- Usually, can't change the partition layout through an OTA update
  - But it is possible to introduce dynamic partitions through OTA by adding `--retrofit_dynamic_partitions` to `ota_from_target_files` (see https://cs.android.com/android/platform/superproject/+/master:build/make/tools/releasetools/ota_from_target_files.py;l=67 )
  - This is automatically added for incremental OTA if the previous one doesn't support dynamic partitions, but the new version does

```
device.mk

AB_OTA_UPDATER := true // enable AB updates

AB_OTA_PARTITIONS := \ // select partitions which should be updated through an OTA
    boot \
    system \
    vendor

PRODUCT_PACKAGES += \  // add packages which will take care of updating the partition
    update_engine \  // takes care of installing the update to the target partition
    update_verifier  // verifies if the boot of the freshly updated partitions was successful
```

# Booting after A/B Update

- Boot process after update is installed to the second partition
- In case of a successful boot, the partition (slot) is marked valid
- Otherwise the system reboots and switches to a different slot

- update_engine is part of the AOSP
- No more „cache“ partition is required
- Additional „misc“partition is required for metadata
- Bootloader support required



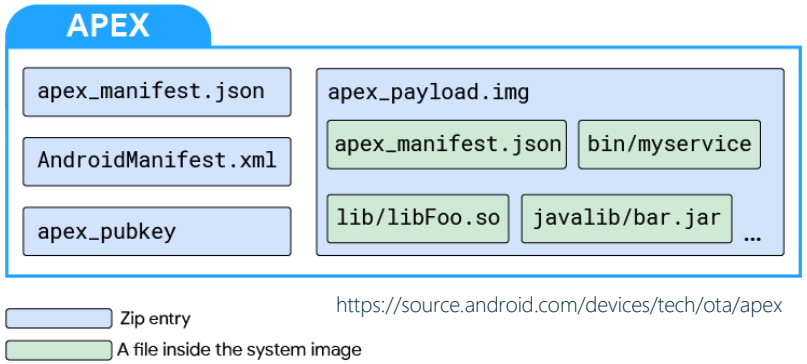

https://source.android.com/devices/tech/ota/ab/ab_implement

# Alternative to OS Updates

Android Pony Express (APEX)

```
device.mk
$(call inherit-product, $(SRC_TARGET_DIR)/product/updatable_apex.mk)
```

- Started with Android 10
- How can we update system components without building OTA → use app store / app infrastructure
  - Updates for native services, libraries, (HALs), runtime (ART), and class libraries.
- Full list of available APEX modules: https://source.android.com/devices/architecture/modular-system
  - Android Runtime (ART) (Available with Android 12)
  - Wi-Fi (Android 11)
  - Media Codecs (Android 10)
- Designed like an Android App
  - Apex_manifest.json - contains the package name and version, which identify an APEX file
  - AndroidManifest.xml - allows the APEX file to use APK-related tools and infrastructure such as ADB
  - Apex_payload.img - ext4 file system image backed by dm-verity. The image is mounted at runtime via a loopback device




https://source.android.com/devices/tech/ota/apex

# Alternative to OS Updates

APEX Manager (apexd) update flow

1. Download via a package installer, app, ADB, or other source
2. Package manager starts and hands over to **apexd** as soon as it recognize its an apex
3. The APEX manager verifies the APEX file
4. The internal database of the APEX manager is updated to reflect that the APEX file gets activated at next boot
5. The install requestor receives a broadcast upon successful package verification
6. The system must be rebooted to activate the new package
7. OS creates a loopback device from the APEX file
8. OS creates a device mapper block device on top of the loopback device.
9. OS mounts the device mapper block device onto a unique path (usually apex/name@version)

More information at https://source.android.com/devices/tech/ota/apex

# Emteria makes it easy to build and update Android remotely.