Overview Of Brillo and Weave

Bin Chen

v0.1 9/2016

Requirements of an IoT Device

Requirement	Description
OTA	ota for feature update and security patch
Security	Device, Communication, Cloud Security
HW I/O	Sensing, Actuator
Light Weight	For resource-constrained system
Connectivity	The `I` in Iot, all layer of Internet
Device Mgr	Provision, Operation, Monitor, Update
App Dev	Easy to development

Brillo/Weave, on top of Android, as one Solution

Requirement	Android	Brillo	Weave
OTA	Y		
Security	Y		+
App Dev	Java/C++	C/C++	
Light Weight		Y	
HW I/O		Y	+
Connectivity	Y		+
Media	Y	-	
Device Mgr			Y

Brillo

- Android based OS tailed for IoT Device
- High Level Architecture is same as Android Applications -> Clients <-> Service -> HAL -> Kernel
- Stripped:
  - No Java run time, framework, services
  - Stripped native system services
- Enhanced:
  - New services specific IoT device
  - Bdk for OS configuration and product image generation

Device System Requirements/Target

- CPU :
- ROM : 128M *
- RAM : 32M Here is what I got from a product, in terms of ROM requirement: ~130M

```
7297024  boot.img    6398160   cache.img       1246081  ramdisk.img

89238564  system.img  19202364  userdata.img
```

Supported Platforms

- Intel (X86), Edison(Dual-Core Atom 500M), minnowboard
- Qualcomm (Arm), dragonboard(MSM8916, QCore A53)
- Marvell (Arm), ABox Edge (IAP140, QCore A53)
- Freescale (Arm),
- Rockchip (Arm)
- Imagination (MIPS), Creatori41

Brillo is not targeted for MCU class device

Development Environment

- Language : C++/C
- Build : Android.mk, mm
- Flash : fastboot
- Update : adb
- Debug : gdb
- Libraries: A lot
- Bdk : Brillo Developer Kit

Brillo Developer Kit (BDK)

- Vendor `BSP` Management: download, update bsp
- `Product` Creation: add your value and customization here **Nice separation!**

```
bdk              AOSP common

install bsp      Vendor Specific

create product   Product specific
```

BSP Management

- brunch bsp download bsp_name(e.g edison)
- Following stuffs will be downloaded
  - device/$(soc_vendor) : product configure files e.g BoardConfig.mk
  - hardware/bsp/$(soc_vendor) : bsp source, e.g soc or peripheral hal/config
  - vendor/bsp/$(soc_vendor) : bsp prebuilt, e.g soc or peripheral firmwares
  - hardware/bsp/kernel/$(soc_vendor): vendor kernel (**now**)

Soc & Peripherals

- soc : e.g Qualcomm/msm8916, Intel/edison etc.
- peripheral : e.g audio, sensor, wifi

	soc	peripheral
source	A	C
binary	B	D

- A): hardware/bsp/$(soc_vendor)/soc/$(soc_name)
- B): vendor/bsp/$(soc_vendor)/hardware/soc/$(soc_name)
- C): hardware/bsp/$(peripheral_vendor)/peripheral/$(peripheral_name)
- D): vendor/bsp/$(peripheral_vendor)/hardware/peripheral/$(peripheral_name)

link

~~Vendor kernel~~ Brillo Common Kernel

All Brillo devices will be built from a single common kernel tree which tracks the latest upstream LTS, even on released devices.

Product Management

- brunch product create produce_name bsp_name
- Products should be created *outside* of BDK.
- Add your product specific service and configuration
- Build the product in the product directory and it will build corresponding bdk and bsp as well.

Directory structure

Brillo (AOSP Master)

```
├── bionic
├── bootable
├── build
├── device
├── external
├── filelist
├── frameworks
├── hardware
├── libnativehelper
├── out
├── prebuilts
├── product
├── system
└── tools
```

Android (AOSP Master)

```
├── art                 -
├── bionic
├── bootable
├── build
├── cts                 -
├── dalvik              -
├── developers          -
├── development         -
├── device
├── docs                -
├── external
├── filelist
├── frameworks
├── hardware
├── libcore             -
├── libnativehelper
├── ndk                 -
├── out
├── packages            -
├── pdk                 replaced by bdk
├── prebuilts
├── sdk                 -
├── system
├── toolchain           -
```

Brillo Specific Components/Services

- system/peripheralmanager/
- system/nativepower/
- system/webservd/
- system/weaved/
- system/firewalld/
- system/tpm
- system/connectivity/apmanager
- system/connectivity/shill
- system/connectivity/dhcp_client
- system/media/brillo/audio
- system/core/crash_report
- system/core/metricsd
- hardware/bsp/

Features and Services

Security

Device Security

- Open Source
- One top of linux kernel security
- UID/GID based sandboxing
- SELinux
- Crypto, Hardware-backed Keystore/keymaster
- Full Disk Encryption
- Verified/Secure Boot
- TEE

Channel Security

- Auth
- DTLS/TLS
- Crypto

OTA

- Same as Android OTA, with Brillo improvement
- Auto update: background download/updates, used chromium update_engine
- A/B update
  - Pros: Minimized downtime, can rollback
  - Cons: More partitions, bigger system size, complex bootloader
- Server Side : can either use Google infrastructure or roll your own
- Brillo Console: can select specific device, scheduled update.

System Services in Brillo

- 19 services, compared to ~120 in Android

```
# Same as Android System Service
media.radio: [android.hardware.IRadioService]
media.sound_trigger_hw: [android.hardware.ISoundTriggerHwService]
media.audio_policy: [android.media.IAudioPolicyService]
media.camera: [android.hardware.ICameraService]
media.resource_manager: [android.media.IResourceManagerService]
media.player: [android.media.IMediaPlayerService]
media.audio_flinger: [android.media.IAudioFlinger]
sensorservice: [android.gui.SensorServer]
android.security.keystore: [android.security.IKeystoreService]
power: [android.os.IPowerManager]

# New Brillo/Weave System Services
android.os.IPeripheralManager: [android.os.IPeripheralManager]
android.webservd.Server: [android.webservd.IServer]
weave_service: [android.weave.IWeaveServiceManager]
android.brillo.UpdateEngineService: [android.brillo.IUpdateEngine]
android.firewalld.Firewall: [android.firewalld.IFirewall]
trunks_service: [android.trunks.ITrunks]
android.brillo.brilloaudioservice.BrilloAudioService: [android.brillo.brilloaudioservice
android.brillo.metrics.IMetricsd: [android.brillo.metrics.IMetricsd]
android.brillo.metrics.IMetricsCollectorService: [android.brillo.metrics.IMetricsCollect
```

Compared with Android System Services

- All Java Services are removed.
  - No AMS, WMS, PMS...
- Some native System Service are kept
- Same are new for Brillo and Weave

System Services for Brillo

- Peripheral Manager
- Brillo Audio Service
- Update Engine
- Native Permission Service (Not used)

System Services for Weave

- Webservd
- Firewalld
- Weaved
- Metrics Services
- Crash Report (breakpad)

Update Engine

Functionality

- connect to the server,
- check update
- download the update
- verify
- apply

Updater WeaveService

one of the weave services, so not surprisingly can interact with cloud commands

Power Management

Brillo has c++ implementation for Power Manager Service

- service interface: android.os.IPowerManager
  - boot, shutdown, suspend, resume, wakelock
- service: nativepowerman daemon
- client:
  - Use Proxy directly, used by legacy Android component, e.g AudioFlinger
  - A new wrapper, PowerManagerClient, used by new Brillo components, e.g weaved)
- talks to sysfs: `/sys/power/state`, `/sys/power/wake_lock`, etc

HW I/O - Peripheral Manager

Provide platform independent API for accessing hardware I/O.

Architecture

- Client/Service, peripheralman daemon, Binder as IPC
- Single system service, providing single API for all protocols
- Be able to hook different "drivers" for the same protocol in the server
- Separate C client API for different protocols
- Possible to create bindings for high level languages

Supported Protocols

- GPIO
- I2c
- LED
- SPI
- UART

Example, I2C

```
BPeripheralManagerClient* client = BPeripheralManagerClient_new();
BI2cDevice* device;
BPeripheralManagerClient_openI2cDevice(client, "I2C0", 0x10, &device);
std::vector<uint8_t> buffer(40, 42);
uint32_t count = 0;
BI2cDevice_write(device, buffer.data(), 20, &count);
BI2cDevice_delete(device);
BPeripheralManagerClient_delete(client);
```

Sensors

- Same as Android Sensor Service, use sensor HAL SENSORS_HARDWARE_MODULE_ID
- Vendor implement the HAL, For Brillo, Intel is UMP which build on top of MRAA (/hardware/bsp/intel/peripheral/sensors/mraa/)
- Use NDK interface `sensor.h` `ASensorManager`

Connectivity

Same as Andriod Service and HAL

- Bluetooth
- Wifi
- NFC (Not yet)

Additional application layer services

- Shill
- apmanager
- dhcp

Media - Graphic, Display, Camera

- No Rendering (e.g OpenGL)
- No Display Service (i.e SurfaceFlinger)
- Has CameraService, Camera NDK in Android N

Media - Audio

- Microphone
- Speaker
- Brillo Audio Service

Brillo Audio Service

- Not a smart voice assistant as you might expect
- Used to get a listed connected audio devices
- Control volumes
- Be notified of new device status, or volume change

Webserverd

Android phone is usually used a client, however, IoT Device need to be run as both client (gathering & sending data) and server (accept commands). Therefore, a web server is needed for IoT OS.

Architecture

- A web server build on top of libmicrohttpd
- Client connect to server using Binder interface (switched from D-Bus)
  - android.webservd.IServer
  - android.webservd.IProtocolHandler
  - android.webservd.IRequestHandler
  - android.webservd.IHttpRequest
- Relations
  - Server can support several protocols, such http, https
  - For each protocol, there are several request handlers, each associate with distinctive URL and method pair
  - Each handler will take care of specific requests (send to that url/method)
- `webservd` is the daemon/service, `libwebserv` is the client
  - It is client's responsibility to implement the request handler

Example 1:

Example 1 - 1. Define your request handler

```
class PingRequestHandler : public RequestHandlerInterface {
    void HandleRequest(std::unique_ptr<Request> /* request */,
                       std::unique_ptr<Response> response) override {
    response->ReplyWithText(200, kResponse, brillo::mime::text::kPlain);
    }
};  // class PingRequestHandler
const char PingRequestHandler::kMethods[] = "";  // all methods
const char PingRequestHandler::kResponse[] = "Hello World!\n";
const char PingRequestHandler::kUrl[] = "/webservd-test-client/ping";
```

Example 1 - 2. Register your Request Handler to a Protocol Handler

```
webserver_ = Server::ConnectToServerViaBinder(
brillo::MessageLoop::current(),
base::Bind(&LogServerOnlineStatus, true /* online */),
base::Bind(&LogServerOnlineStatus, false /* offline */));
ProtocolHandler* http_handler = webserver_->GetDefaultHttpHandler();
http_handler->AddHandler(
    PingRequestHandler::kUrl,
    PingRequestHandler::kMethods,
    std::unique_ptr<RequestHandlerInterface>(new PingRequestHandler()));
```

Example 1 - 3. Run the Sever and test it

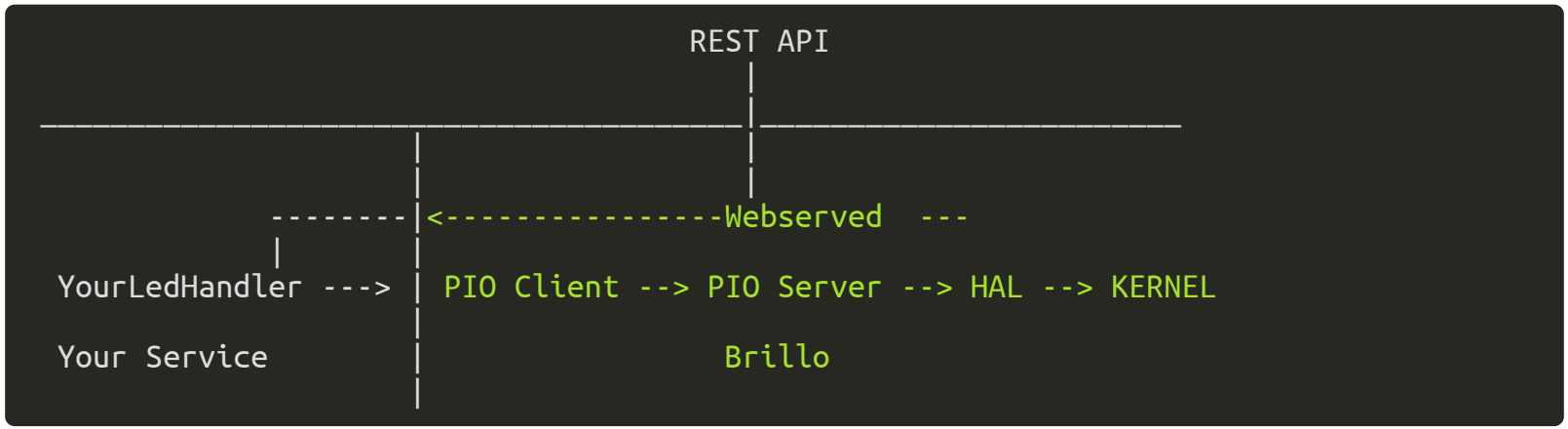
```
$ adb forward tcp:8080 tcp:80
$ adb shell start webservd_tclient
$ curl http://localhost:8080/webservd-test-client/ping
Hello World!
```

Example 2: Turn on/off the LED!

Previous example is quite naive since it does not interact with any System Service. Let's see what the **Hello World** example for an IoT project will look like in Brillo.

In your request handler, connect to Peripheral Manager by using the peripheral client API, turn on/off a GPIO.

A little diagram



firewallds

iptable is used underneath

Trunk Service and TPM

- both from Chromium OS

  > The Trunks TPM Library (TTL) is a set of types and functions used to interface with a Trusted Platform Module.

- contains encrypt operations/commands, overlapped functionality with keymaster and other TEE environment, such as ATF, or Trusty

- Didn't dive in.

Weave

Architecture/Components

external/libweave

The server side of weave.

- weavei::Device
  - The Entry point of weave service, weaved init and run a WeaveDevice
  - Components and Commands management
- weave::provider
  - httpClient, httpServer, Bluetooth, Wifi, Network, DnsServiceDiscovery
  - Weave is OS independent
- notification
  - xmpp channel, pull channel
- privet
  - Google Cloud command handler

Schema: Component, Traits, Command, State

- Component A device can have several Components, e.g 10 LEDs, 5 sensors)
- Traits Each `Component` will have several Traits (e.g onOff, color Trait for LED component; temperature for Sensor component). For each `Trait`, you can either set it through `Command`, or view it through `State`.
- Command & Command Handler Use Command to change a trait. e.g setOnOff() for "onOff" `Trait` of `Component` "led1"
- State Use `StateProperty` to get a trait state. e.g onOff `State` for onOff `Trait`.

Example - Traits

```
const char kTraits[] = R"({
  "onOff": {
    "commands": {
      "setConfig": {
        "minimalRole": "user",
        "parameters": {
          "state": {
            "type": "string",
            "enum": [ "on", "off" ]
          }
        }
      }
    },
    "state": {
      "state": {
        "isRequired": true,
        "type": "string",
        "enum": [ "on", "off" ]
      }
    }
  }
})";
```

Example - Add Command Handlers

```
device->AddTraitDefinitionsFromJson(kTraits);
for (size_t led_index = 0; led_index < led_states_.size(); led_index++) {
      std::string component_name = "led" + std::to_string(led_index + 1);
      device->AddComponent(component_name, {"onOff"}, nullptr);
      device->AddCommandHandler(
          component_name, "onOff.setConfig",
          base::Bind(&LedFlasherHandler::OnOnOffSetConfig,
                     weak_ptr_factory_.GetWeakPtr(), led_index));
      device->SetStateProperty(
          component_name, "onOff.state",
          base::StringValue{led_states_[led_index] ? "on" : "off"}, nullptr);
    }
  }
```

privet - The component that talks to Google Cloud

Those are the http request privet/weave will be handling! Source

```
AddHandler("/privet/info", &PrivetHandler::HandleInfo, AuthScope::kNone);
AddHandler("/privet/v3/pairing/start", &PrivetHandler::HandlePairingStart,
AddHandler("/privet/v3/pairing/confirm",
AddHandler("/privet/v3/pairing/cancel", &PrivetHandler::HandlePairingCancel,
AddSecureHandler("/privet/v3/auth", &PrivetHandler::HandleAuth,
AddSecureHandler("/privet/v3/accessControl/claim",
AddSecureHandler("/privet/v3/accessControl/confirm",
AddSecureHandler("/privet/v3/setup/start", &PrivetHandler::HandleSetupStart,
AddSecureHandler("/privet/v3/commands/execute",
AddSecureHandler("/privet/v3/commands/status",
AddSecureHandler("/privet/v3/commands/cancel",
AddSecureHandler("/privet/v3/commands/list",
AddSecureHandler("/privet/v3/checkForUpdates",
AddSecureHandler("/privet/v3/traits", &PrivetHandler::HandleTraits,
AddSecureHandler("/privet/v3/components", &PrivetHandler::HandleComponents
```

weavd/libweaved

- Add Android Binder interface
  - android.weave.IWeaveService
  - android.weave.IWeaveClient
  - android.weave.IWeaveCommand
  - android.weave.IWeaveServiceManager
  - android.weave.IWeaveServiceManagerNotificationListener
- Client library
- Use it to your component and command handler to weave
- buffet
  - Android implementation of `weave::provider`
  - webservd implements httpServer

Buffet

Implement all kinds of `weave::provider` needed by weave daemon

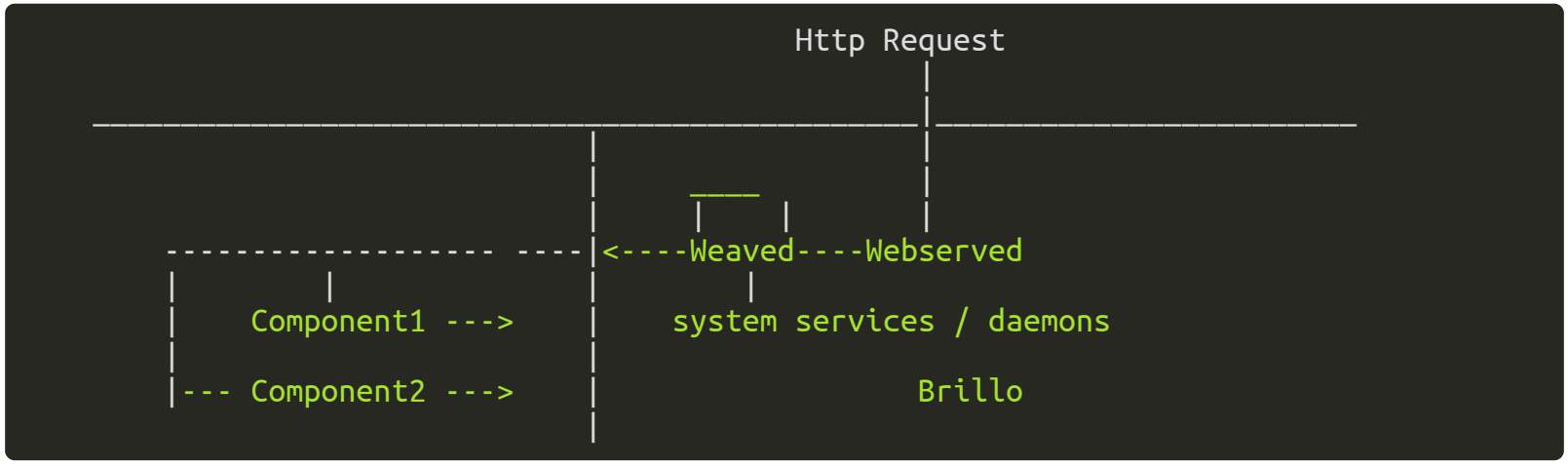
- `Shill` impl. `provider::Network` and `provider::Wifi` using `shill` daemon
- `Avahi` impl. `provider::mDns` using `avahi` daemon
- `WebservClint` impl. `provider::httpServer` using `webservd` daemon
- `BluetoothClient` impl. `provider::bluetooth` using `Flouride` daemon
- `httpClient` impl. `provider::httpClient` using `libbrillo/http`
- An `Encryptor` using `android::KeyStore`
- An `ApManager` using `chromium::apmanager` daemon

libbrillo

- utility: stream, daemon, string, message loop, binder watcher
- brillo::http
- brillo::minijail

The Diagram

- All the http requests will go to weaved first
- Generic device management commands will be handled by weaved/privet
- Specific device commands, i.e those added as component command, will be handled your component but still first dispatched by `webservd`
- Weaved will taking care of update/broadcast devices status change



Brillo without Weave, Or Brillo without Google Cloud?

Replace the Google cloud stuff with your own in the external/libweave.

Thank you.

pierr.chen At gmail.com