Low-Level API

Recommendation from prpl

Version 1.2

Revision History

Version	Date	Author	Description
0.5	2019.10.20	John Crispin (prpl) & Wouter Cloetens (Liberty Global)	v1 APIs
0.6	2019.10.28	Mirko Lindner (SmartRG)	Surrounding texts
1.0	2019-11-24	David Barr (Intel)	Approved text by the Prpl Technical Steering Committee
1.1	2020-01-29	Wouter Cloetens (Deutsche Telekom)	Cosmetic changes and addition of the 2017 approved WLAN API text. Fixed switchdev link.
1.2	2020-05-05	Wouter Cloetens (Deutsche Telekom)	Wireless: updated kernel documentation link. Appendix 1: updated linux-wireless document link.

Content

Introduction

This document identifies low-level APIs recommended by prpl to promote harmonization and convergence among SW platforms.

The recommended APIs should be supported by BSPs in order to best leverage the efforts from open-source software communities.

Harmonization and convergence is needed because the current landscape employs disparate abstraction layers from numerous different platform providers.

This proliferation of different abstraction layers has made porting efforts more difficult, harder to maintain, and hence unscalable to large numbers of CPE platforms.

Prpl therefore recommends instead the best practice of relying on standard kernel interfaces, wherever possible.

This recommendation lists those APIs corresponding to various BSP subsystems.

Some of the APIs are already proven and mature, and hence implementable now.

Others remain works-in progress, suitable for prototype development and verification.

While others are forward-looking toward the future.

Thus, prpl expects these recommendations to be updated over time—see <url>

Background

Introduction to the Low-Level API

In a world of growing complexity and a need by ISPs to launch new devices at an ever-faster pace, the industry is looking for ways to reduce product development times in an effort to cut time-to-market as much as possible.

Open API efforts such as prpl's High-Level API are designed to drastically reduce the effort and thereby time required to integrate an ISP's services on a new device and software platform.

However, a big driver in time-to-market is the effort to port any software stack to a new chipset, especially if this chipset comes from an SoC vendor the software integrator is unfamiliar with. Integrating new board support packages or SoC SDKs can be a lengthy and difficult process. A miscalculation or unexpected finding can threaten a project's overall success.

Historically, chipset SDKs are almost complete CPE stacks which were designed to help the hardware vendor to launch the product as quickly as possible. With the increase of services and customization required by each ISP, software vendors have begun to replace large parts of the SoC's SDK, some even only use the bare drivers. As long as the underlying architecture remains relatively unchanged, this is a valid and promising approach.

However, in addition to more services ISPs have moved to also require different chipsets and solutions from different SoC vendors to be used in their CPEs. Some do so because of procurement and continuity strategies, others simply because they deployed technologies that require a specific chipset to be used. In either case, the pressure to find ways to speed up product development increases even further.

Software vendors and hardware manufacturers are now faced with ISP customers who are searching for partners who can support more customization and services on a

greater number of devices using the largest variety of chipset vendors at the fastest pace possible.

To do so, industry representatives in prpl have agreed to establish a new standard integration boundary for software stacks on top of SoC SDKs. Instead of every software integrator integrating every driver and SDK feature, the SoC vendors are being asked to consolidate their APIs and offer all core features through open, publicly available and peer-reviewed kernel APIs - the *prpl Low-Level API*. Standardizing these interfaces will dramatically reduce the time it takes a software vendor to integrate a chipset maker's SDK and also result in accelerated innovation throughout the whole industry.

Of course, an Open API effort can never capture 100% of all functionality and SoC vendors will want to continue to innovate and differentiate. Therefore it is important to note that the *Low-Level API* is not meant to be the only API between CPE stack and SoC SDK, but it is meant to be the library of how standard features are to be exposed and made available to the higher layers.

Way of working

In order to make this transition to a harmonized API an evolution every company can participate in and benefit from rather than disruption, the industry representatives have designed a multi-step approach.

The first step is the creation of API proposals based on existing kernel APIs. Once ratified by prpl, representatives will engage with the relevant chipset vendors to seek their agreement and support for the proposed interfaces.

As part of the engagement, prpl will work with the chipset vendors to establish timelines for when their SDK will support which of the proposed interfaces

The final step is the creation of a validation tool that will help SoC vendors prove their conformity and help software integrators when adopting new chipsets and their SDKs.

This process will be repeated for each release of API candidates and may be accompanied by further naming and numbering schemes.

Purpose of this document

This document describes the first set of API candidates the prpl low-level API working group has identified. It is will serve as the baseline for the conversations with chipset vendors as described above. As such this document does not describe any formal API or requirement but is a working document.

Intended audience

This document is meant to be used by the technical teams at the SoC vendors to assess the direction of the Low-Level API and compare it with their own use of kernel APIs.

API

GPIO

Feature	Kernel Subsystem	Kernel API	Dependencies
GPIO	GPIO	sysfs/gpiolib	None
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/driver-api/gpio			

GPIO is a basic requirement for various other features and subsystems. The Vendor kernel shall provide support via the default kernel interface. If the silicon supports GPIO interrupts, these shall be supported using the kernel's IRQ subsystem.

Button

Feature	Kernel Subsystem	Kernel API	Dependencies
Button	GPIO / HID	GPIO / HID	GPIO
HID: https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/hid GPIO: https://git.openwrt.org/?p=openwrt/openwrt.git;a=tree;f=package/kernel/gpio-button-hotplug			

With the basic requirement of GPIO fulfilled, the kernel has several ways of exposing PIN connected buttons. For very simple tasks the GPIO hotplug trigger can be used. If required it is also possible to utilize the HID layer.

PWM

Feature	Kernel Subsystem	Kernel API	Dependencies
PWM	PWM	sysfs	None
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/driver-api/pwm.rst			

PWM is a basic requirement for controlling a LEDs brightness. If the silicon has a PWM block, it shall be supported by the kernels PWM layer.

LED

Feature	Kernel Subsystem	Kernel API	Dependencies
LED	LED	sysfs/leds	GPIO/PWM
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/leds			

LEDs shall be supported using the kernel's LED subsystem. This can be done via a custom, the leds-gpio or the leds-pwm driver. All LEDs shall follow the kernel's naming scheme.

I2C

Feature	Kernel Subsystem	Kernel API	Dependencies
I2C	I2C	ioctl/i2cdev	(GPIO)

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/i2c

I2C busses shall be supported using the kernel's I2C subsystem. This can be done via a custom or the i2c-gpio driver.

SPI

Feature	Kernel Subsystem	Kernel API	Dependencies
SPI	SPI	ioctl/spidev	(GPIO)
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/driver-api/spi.rst			

SPI busses shall be supported using the kernel's SPI subsystem. This can be done via a custom or the spi-gpio driver.

Sensors

Feature	Kernel Subsystem	Kernel API	Dependencies
Sensors	IIO	sysfs	(GPIO/I2C/SPI)
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/driver-api/iio			

Sensors support/required by the solution shall be supported using the kernels IIO subsystem.

Frequency Scaling

Feature	Kernel Subsystem	Kernel API	Dependencies
Frequency Scaling	cpu-freq	sysfs	None
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/cpu-freq			

All scaling of the SoC cores has to happen via the kernel's cpu-freq subsystem.

RFS/XPS

Feature	Kernel Subsystem	Kernel API	Dependencies
RFS/XPS	net	sysfs	None
https://www.kernel.org/doc/Documentation/networking/scaling.txt			

In the rare case that the Silicon supports frame steering in HW, it shall be hooked into the kernels standard configuration interface.

Flow table offload

Feature	Kernel Subsystem	Kernel API	Dependencies
Flow table offload	net	nftables/iptables	None
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/nf_flowtable.txt			

Prpl has funded a project to backport this feature to the commonly used vendor kernels. This shall be supported as an alternative pure SW patch for flow acceleration.

HW flow offload

Feature	Kernel Subsystem	Kernel API	Dependencies
HW flow offload	net	nftables/iptables	Flow table offload

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/nf_flowtable.txt

Silicon supporting HW accelerated networking shall hook this feature into the flow table offload infrastructure. Patches are currently being prepared for upstream and will be backported via prpl funding.

QoS queues

Feature	Kernel Subsystem	Kernel API	Dependencies
QoS queues	sched	tc/netlink	None

QoS queue setup shall be provided via the kernel's netlink interface.

QoS classification

Feature	Kernel Subsystem	Kernel API	Dependencies
QoS classification	netfilter	netlink	None

QoS traffic classification shall be provided via the kernel's netlink interface.

Wireless

Feature	Kernel Subsystem	Kernel API	Dependencies

wireless	net/wireless	cfg80211/nl80211	None
ll-api https://www.kernel.org/doc/html/latest/driver-api/80211/index.html			

Wireless drivers need to provide support for cfg80211/nl80211 in a manner that will allow an upstream hostapd/wpas to function properly.

See WLAN API for the full text of the recommendation.

DSA/switchdev

Feature	Kernel Subsystem	Kernel API	Dependencies
DSA/switchdev	DSA/PHY/bridge	netlink/ioctl/sysfs	V4.9
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/networking/switchdev.txt			

The Ethernet switch device driver model (switchdev) is an in-kernel driver model for switch devices which offload the forwarding (data) plane from the kernel. This is one of the two options for supporting the switch fabric.

swconfig

Feature	Kernel Subsystem	Kernel API	Dependencies
swconfig	switch	netlink	None
https://git.openwrt.org/?p=openwrt/openwrt.git;a=blob;f=target/linux/generic/files/drivers/net/phy/swconfig.c https://openwrt.org/docs/techref/swconfig			

Swconfig is the OpenWrt switch config layer that primarily allows the configuration of VLANs within the fabric.

MDB/mcast

Feature	Kernel Subsystem	Kernel API	Dependencies
MDB/mcast	net/bridge	MDB/notifier chain	(soft@home code drop)
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/net/bridge/br_mdb.c			

Multicast offloading should happen within the kernel by using MDB notifications if the silicon supports offloading.

Hardware cryptography

Feature	Kernel Subsystem	Kernel API	Dependencies
HW crypto	crypto	in-kernel/cryptodev	None
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/driver-api/80211			

The kernel crypto API offers a rich set of cryptographic ciphers as well as other data transformation mechanisms and methods to invoke these. This document contains a description of the API and provides example code.

Hardware RNG

Feature	Kernel Subsystem	Kernel API	Dependencies
HW RNG	net/DSA	rtnl/net	None
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/Documentation/driver-api/80211			

Random number generators need to be exposed via the crypto subsystem

MTD

Feature	Kernel Subsystem	Kernel API	Dependencies
MTD	MTD	ioctl	None

Vendor kernels may not provide their own bad block management layer but should rely on UBI to do the work.

Container privileged

Feature	Kernel Subsystem	Kernel API	Dependencies
Container - privileged			V3.8+

Container unprivileged

Feature	Kernel Subsystem	Kernel API	Dependencies
Container - unprivileged			v4.19+

Features not part of the LL-API

- It has been agreed, that DSL and TAPI will not be part of the LL-API. DSL will be abstracted from userland using xdsl.h. IOPSYS has already implemented a daemon for this task.

- It has been agreed, that TAPI/DECT will not be part of the LL-API. Vendors should provide a FOSS compliant userland implementation such as a functional asterisk channel driver.

Contacts

The following table describes the primary contacts to ask about the low-level API.

Contact person	Email address
John Crispin	john@phrozen.org
Wouter Cloetens	wouter@e2big.org

Appendix 1: WLAN API

Recommendations

The WLAN API recommendations are:

- cfg80211 driver for WLAN
- hostapd, wpa_supplicant, and iw support with WLAN management frames transmitted via nl80211

Scope

The primary focus is the control interface to radio and access point functionality, with STA (client) functionality as a lower priority. The data path is generally not a problem, as all known existing drivers model this as a Linux Ethernet device.

Availability of source code, and the license of, the device driver, microcode, and any firmware on an embedded offloaded processor, are out of scope.

The operating system scope is limited to Linux.

Reasoning

A lack of common interfaces for WLAN has a negative effect on all involved parties. Open source OpenWrt (or other open source router platform) developers are locked out from the use of officially supported drivers, which restricts which silicon solutions they can support. Router middleware vendors find themselves designing their own HAL's, and maintaining several implementations. WLAN is noted by middleware vendors as one of the most complex layers to maintain across chipset SDKs and even across versions of the same SDK.

While SDK users struggle with the lack of a common WLAN API, silicon vendors also maintain multiple interfaces. Those that support OpenWrt develop and maintain their own interface layer. Those that support RDK-B face the same effort with the RDK-B

HAL. Those that support other stacks, e.g. their own proprietary one, maintain yet another.

All parties can benefit from a commonly agreed upon interface. More devices supported by OpenWrt benefits the project's growth. Silicon vendors would only need to support a single interface for all Linux-based devices. Middleware vendors will still need some higher level management software, but the responsibility of the maintenance of the layer between this and the common API will be moved firmly to the middleware vendor. In addition, only one such implementation is needed. For example, a single RDK-B HAL implementation would support all drivers, and therefore, the silicon vendors would not need to maintain a HAL implementation for their chips.

Recommended APIs

The prpl Carrier Interest Group evaluated the Linux WLAN space and formally recommends the creation and use of drivers based on cfg80211. Cfg80211 is actively maintained by the Linux kernel community and technically and functionally vastly superior to the older ioctl-based wireless interfaces. Additional background on the reason for the kernel community's move from ioctl interfaces to cfg80211/nl80211 can be found at:

https://wireless.wiki.kernel.org/_media/en/developers/documentation/control.pdf.

Members of prpl Foundation are strongly encouraged to work with the Linux Wireless community to add missing features into cfg80211, especially features which could be used across multiple vendor implementations.

The standard WLAN driver delivery should include the cfg80211 management API. In the event that the driver is part of a board support package (BSP) for a SoC, the included Linux kernel should have cfg80211 enabled, as should the WLAN driver. The use of the cfg80211 by higher level management software, that may be included in the BSP, is recommended.

In addition to cfg80211 drivers, the Carrier Interest Group recommends the use of hostapd, wpa_supplicant and iw connected to the driver via nl80211. In particular,

WLAN management frames should be transmitted between the driver and hostapd/wpa_supplicant/iw via nl80211.

Vendor-specific extensions should be limited to hardware or implementation specific functionality that is distinctly not present in cfg80211, and does not lend itself to an extension of cfg80211 either.

Examples are control of embedded firmware, debugging interfaces to the lower layers of the implementation, and special calibration logic. Any interfaces that must be used by an operational control interface should be standardized. The recommended interface for vendor-specific extensions is cfg80211 vendor commands.

References

[1]