Whitepaper

LTS Kernel software may not be the best choice for IoT Infrastructure and Devices

This paper explains why embedded and mobile products have traditionally used Long Term Support (LTS) kernels, and why this may not be advisable for future connected devices. It also offers alternative methodologies for developers to consider. While the Linux kernel is used as the example for this paper, any software in the connected product stack has similar considerations.

Introduction

Over time it should be obvious that in a given software project the best, most tested, most stable release is the latest release. The counter is that if new software is being added, that software has less testing and therefore risks introducing instability into a stable product.

New software is introduced into projects for three reasons:

• Bug fixes	Designed to fix problems that have been found
• Security Updates	Designed to improve security or fix known flaws
• New Features	New functionality

In general the first two should improve existing product functionality. The third introduces new functionality that may or may not be advantageous to an existing product.

Embedded Devices Today - the LTS Kernel Approach

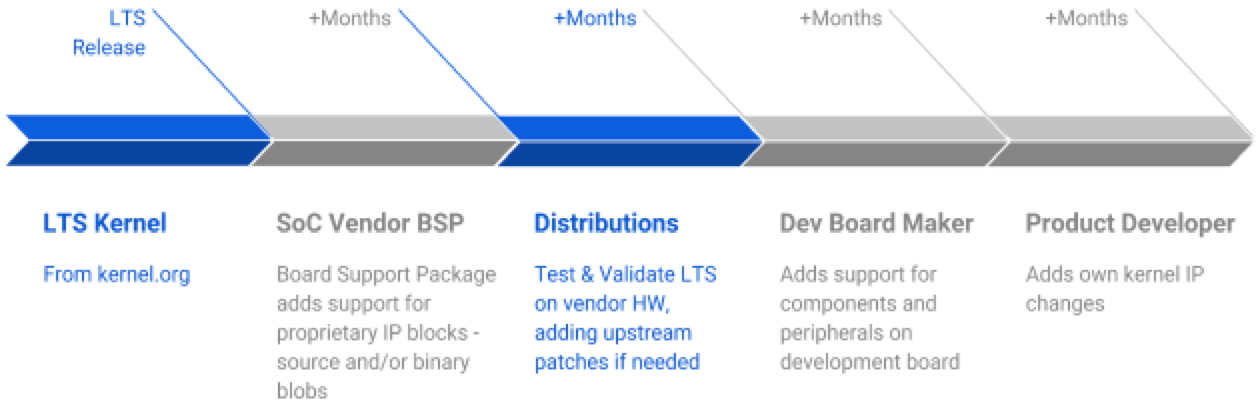
Approximately once a year the Linux kernel community designates a particular kernel release as LTS (Long Term Support). Typically, these kernels are maintained with backported key bug fixes and security updates for two- to four years after release[1]. As an outlier, the Linux 4.4 release from January 2016 is planned for maintenance until February 2022 (i.e., six years). Product developers see LTS releases as a good thing because a key complex component of the software stack is being maintained "for free" by the Linux community, reducing lifetime maintenance costs and increasing product stability.

This may not always be true.

[1] https://www.kernel.org/category/releases.html

What Really Happens

There is a long supply chain for software between the Linux kernel developers, System on Chip (SoC) makers, and an embedded/IoT end product build. The chain looks something like this:


LTS Release
+Months
+Months
+Months
+Months
LTS Kernel
From kernel.org
SoC Vendor BSP
Board Support Package adds support for proprietary IP blocks - source and/or binary blobs
Distributions
Test & Validate LTS on vendor HW, adding upstream patches if needed
Dev Board Maker
Adds support for components and peripherals on development board
Product Developer
Adds own kernel IP changes

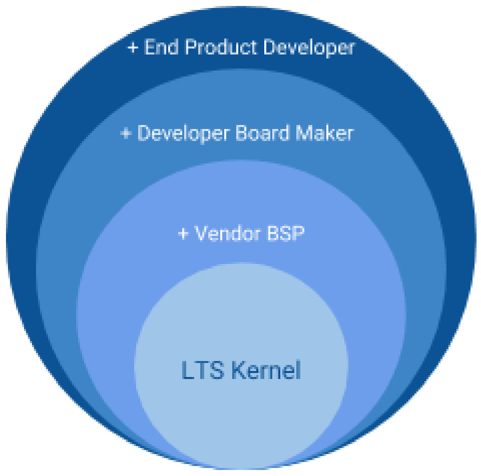
By the time the kernel is running in an end product it has had extensive changes that have taken many months of engineering by multiple parties:

- SoC vendors add kernel support for their own IP. This can be extensive with support for proprietary security, GPU, AI, networking, multimedia and peripheral IP
- Linux distributions such as Ubuntu, SuSE or RedHat test and stabilize the LTS kernel adding upstream patches if needed (sometimes they even choose a different kernel for their own "LTS")
- Next, component distributors or development board producers create their own derivative kernel with additional changes for the components they use externally to the SoC on a development board (for example USB, Bluetooth, PCIe peripherals etc.)
- Finally the end product developer may add additional changes to the kernel for their own purposes.

This development process can take a year or more - typically product developers are starting with kernels that are one- to two years old. Then, depending on the maintenance carried out further up the chain, the first task is often applying the LTS updates to the original LTS kernel. This may or may not go smoothly. If there are dependencies on any of the changes introduced by all of the third parties, they can take months to resolve.

The SnowFlake Problem

By this point every embedded product kernel is different. Mainstream SoC Vendor BSP code alone can add 50 percent or more lines of code to the kernel.



The end product developer is now responsible for testing and maintaining a unique kernel. Assuming everyone has complied with their GPL licensing obligations the source code to this unique kernel is available. However, many IP vendors (GPU, AI, Bluetooth, WiFi etc.) provide proprietary user space binary “blobs” for their IP, without source code. As the upstream kernel moves forward, these vendors often do not keep up, and so the end product vendor ends up having to maintain this kernel over the product lifetime with (single supplier) dependencies on multiple third parties.

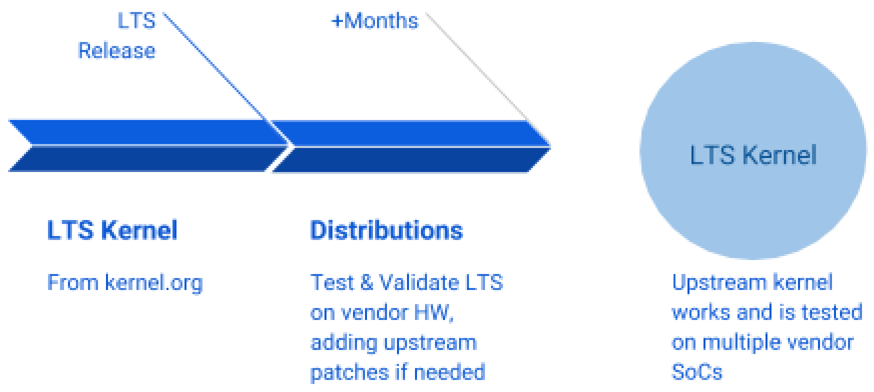
Testing and CI

Modern software testing uses Continuous Integration combined with automated testing to improve product quality. Every code check-in to a project can be automatically tested with regression, performance and functional testing. The result is that product quality can be continuously improved for end use cases, as each bug fix has an added regression test. Again, the best software is the most recent software.

However, the benefit of all this testing at the end product level in many cases only applies to that product, because every product kernel is different.

Embedded Devices vs. Enterprise Computing

Enterprise Computing runs mission critical systems across the world - from the stock market and banking systems, to government and city security and infrastructure systems. Enterprise Computing requires the highest level of stability. The kernel chain for Enterprise looks somewhat different:


LTS Release
+Months
LTS Kernel
LTS Kernel
From kernel.org
Distributions
Test & Validate LTS on vendor HW, adding upstream patches if needed
Upstream kernel works and is tested on multiple vendor SoCs

The distribution kernel (for example, RedHat) is maintained by the distribution vendor. The same kernel runs on a range of SoCs. New SoC support is delivered upstream (into linux.org) by the SoC vendor. The distro vendor then pulls from kernel.org updates into the distro, and makes the updates available to all, so that now a single kernel supports all end-products for a given architecture (typically datacenter servers).

Now, all testing on that kernel and all bugs found benefit everyone. The distro manages the kernel and testing, dramatically simplifying the task of the end-product developer.

This doesn’t work today on embedded devices because of the scale of the supply chain. There are hundreds of SoCs and thousands of peripheral devices for embedded products. SoC vendors can get products to market faster by modifying the kernel themselves. This locks in customers, who become reliant on the level of maintenance that the SoC vendor must then provide. Often SoC vendor kernels lag the latest Linux kernel by two or more years. Furthermore, many SoC vendors leave kernel maintenance to the product developer, who often does not have the tools or knowledge to determine if a particular upstream patch might cause a conflict with SoC vendor BSP code.

Updates in a Connected World

Traditionally, embedded products were built, tested for their purpose, delivered and maintained. Given good quality processes, bugs were rarely if ever found, let alone fixed.

The world has changed. Products today are far more complex, inter-operating systems. Sensors are gathering vast amounts of data, and delivering them through complex gateways and edge platforms, wired and wireless networks to cloud-based backend systems. Corporations, governments and bad actors are probing systems for weaknesses. Cybersecurity is becoming a key differentiator of value and product safety.

In a world of Connected Devices we need to be able to immediately update end products at scale to fix critical security flaws or bugs. This needs to be done “over the air” (OTA) - in most cases manual update of infrastructure devices is simply not feasible. It’s not just applications that need to be updated - problems like Spectre and Meltdown require software fixes at every level - firmware and kernel included.

OTA Updates & Latest Software

So how do we do this today? The idea of the LTS kernel is that when a security problem is found it is immediately fixed in the current kernel tree. Then, the fix must be back-ported to the LTS kernels. This can take the kernel community and maintainers days or even months - the older the kernel the harder the problem, as the kernel features used to create the current kernel fix may not even exist in older kernels.

After a period of time, the fix is made available in the LTS kernels. Next the distro will take the fix and make sure it is compatible with its own kernel value add and make it available in their distribution. This also can take days to months depending on the problem and the distro release and maintenance policies. Now SoC vendors, distributors and component vendors also must verify the patches, and test any dependencies in their own related code bases. Many simply don't have the resources to do this in a timely fashion, if at all, especially on products that might be two to three or more years old. Finally the product vendor may get fixes from some or all of the preceding chain entities and be able to apply a fix to its product.

The fact is that most embedded products simply don't get updated. Indeed, most products today don't even have the capability of being updated OTA.

We have created a fragmented software ecosystem that delays time to market, reduces interoperability, increases the cost of lifetime maintenance, and makes our end products more expensive and less secure in a connected world.

The best software is the latest software. Long Term Stable can really mean Long Term Unmaintainable, especially in the Connected Device world. All product software should be updatable from firmware to application, and all products should then benefit from the latest software. Zero day critical security fixes to any part of the stack then can be applied immediately and not days or months later, if at all.

A further benefit of building the infrastructure for secure OTA updates to an end product is that the product developer then also can choose to deliver new functionality over the product lifetime, extending utility and customer value.

How do we fix this?

Put simply, it should be as easy to update any Connected Device as it is to update an iPhone®. It should just happen. The product vendor should be able to determine whether an update is urgent and critical and must be done immediately for security or safety reasons, or whether it can wait for a normal “maintenance update” after further testing and/or certification, or whether it simply doesn’t apply to the product use case.

First, we need to do a better job at separating application code (product vendor value add) from the core platform. In the Android and iOS platform worlds, application developers build millions of applications on the OS platforms, which are themselves updated many times over the product lifetime. Because of the APIs between the platform and the applications, developers can create and maintain applications even while the core platform software is updated to improve stability, functionality and performance.

In the embedded world this can be hard. The architectural separation can come at a price in terms of software footprint. However, modern technologies such as Containers can enable updates, legacy and new software to be delivered to smart and infrastructure devices as easily as an update to an application on a mobile phone. Once we achieve this separation we can address the “application” updates more easily, and address the need for core platform updates without breaking application compatibility.

Next we need to build in security and secure update capability into the core platform.

Finally, we need a common platform built on the latest software, supported by component vendors. Then everyone can benefit from using the same software platform, that improves over time in quality, stability, performance and features. Billions of IoT devices will see substantial benefits by being based on the same core platform, not a different one for each product.

From a business perspective such a platform has to be open bringing the following benefits:

- Everyone can see the code - no black boxes
- Everyone can reproduce it, improve it, contribute to it and build upon it
- Everyone can test the same core software, resulting in quality through scale
- No-one is locked into a single vendor

Working Upstream

The term “upstream” refers to the latest software builds in an open source project. Upstream for the Linux kernel means working on the latest working kernel branches that are being developed and tested for future kernel releases (also called the “tip”).

IP, SoC and hardware device vendors deliver support for new devices into the Linux kernel by submitting their new features as patches to the Linux kernel maintainers. Once accepted, these patches become part of the Linux kernel and are carried forward as the kernel evolves. This reduces the maintenance cost for the vendors, while making their technology available to product developers at the earliest possible time, accelerating design wins and time to market.

Taking Responsibility

Who needs to do what?

IP vendors

- Ensure new IP software support is upstream, preferably in time for first silicon

SoC/MCU vendors

- Ensure new IP software support is upstream as early as possible
- Only use third-party IP that has committed upstream software support

Peripheral vendors

- Provide upstream software support for your devices

Distributors and Development Board Developers

- Provide and test your boards with upstream aligned software distributions/platforms
    - Linux and/or AOSP for 32/64bit SoCs
    - Open Source RTOS(es) such as Zephyr for 32-bit MCUs

Product Developers

- Leverage open platforms to deliver your IP, services and applications - do not duplicate platform functionality that you then need to maintain yourself
- Leverage the platform maintenance over your product lifetime to reduce your own cost of software maintenance as you deliver security fixes and, optionally, new functionality

Beyond LTS - microPlatforms

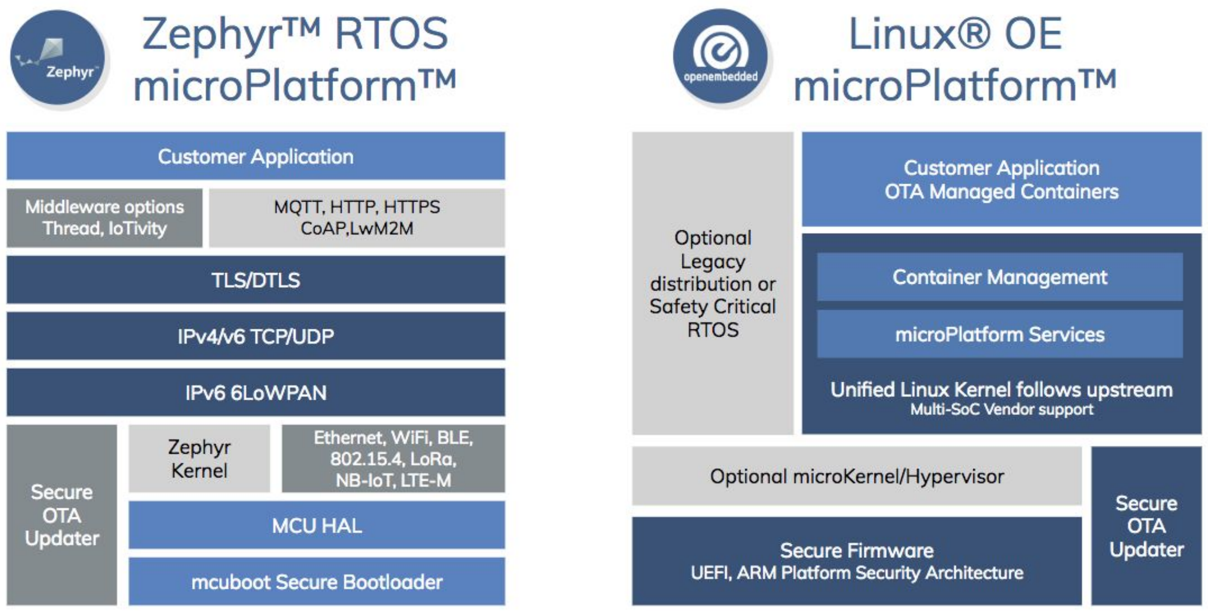
Most of today's Linux distributions are unsuitable for creating embedded devices. They are targeted at developers, not end products, and include many general purpose tools, libraries and packages for software developers, rather than end product functionality.

A software platform for use in end-products needs to be:

- Minimal
- Secure
- OTA updatable
- Cross device - e.g., multiple Arm SoCs supported
- Cross architecture - e.g., Arm, x86, RISC-V support
- Available for global use
- Provided with stable long-term APIs to product specific services and applications
- No lock in - users should be able to commercially independent of single vendors

Foundries.io

The Foundries.io microPlatforms are minimal, secure and OTA updatable open software platforms for building products using microcontrollers (Zephyr™ microPlatform™) or 32/64-bit SoCs (Linux® microPlatform™).


Zephyr
Zephyr™ RTOS microPlatform™
Customer Application
Middleware options Thread, IoTivity
MQTT, HTTP, HTTPS CoAP,LwM2M
TLS/DTLS
IPv4/v6 TCP/UDP
IPv6 6LoWPAN
Secure OTA Updater
Zephyr Kernel
Ethernet, WiFi, BLE, 802.15.4, LoRa, NB-IoT, LTE-M
MCU HAL
mcuboot Secure Bootloader
openembedded
Linux® OE microPlatform™
Optional Legacy distribution or Safety Critical RTOS
Customer Application OTA Managed Containers
Container Management
microPlatform Services
Unified Linux Kernel follows upstream
Multi-SoC Vendor support
Optional microKernel/Hypervisor
Secure Firmware UEFI, ARM Platform Security Architecture
Secure OTA Updater

Foundries.io microPlatforms

These platforms are based on latest stable upstream software from open source projects including mcuBoot, Zephyr Project RTOS, Tianocore UEFI, Linaro OP-TEE, Linux Kernel, OpenEmbedded and Yocto projects, Docker Containers and more.

They are continuously updated and tested end-to-end, gathering data from sensor devices running the Zephyr microPlatform, through gateways running the Linux microPlatform with gateways functions provided in sample Containers, to different Cloud providers.

The microPlatforms are available through low cost non-commercial or commercial per project (not per unit) subscriptions with continuous updates, enabling product developers to OTA update their products with platform bug fixes, security updates and new features for the product lifetime. Partner subscription options are available for unlimited internal use, to enable software development, product maintenance, microPlatform support and demonstration.

microPlatform Use Cases

Sensors & Controllers
Smart Sensors • Asset tags • Lights • Wearables • Controllers

Smart Devices
Home & Industrial control • Displays • PoS • Smart assistants • White goods

Edge/Fog/Automotive
Smart gateways • Routers • Fully Autonomous cars • Robots & Drones • Factory automation

DataCenter
Networking • Storage • OpenStack Clusters • Public & Private Cloud

Find out more and download the microPlatforms at foundries.io