

How Does an IoT OS Differ to a Traditional RTOS?

By Stefano Cadario and Andy Powers, Arm

February, 2021

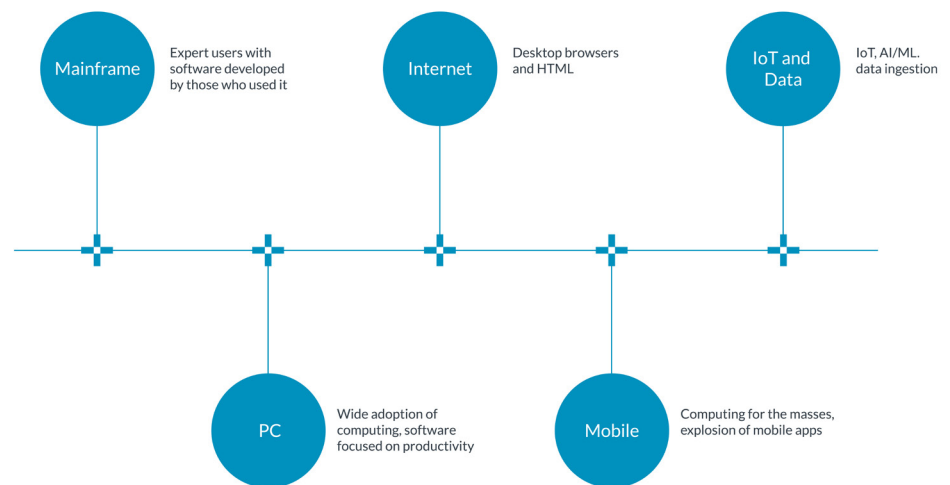
arm

White Paper

Introduction

Arm has been at the heart of computing for the past 30 years - the transition from personal computers to mobile phones and now to the incredibly powerful internet-connected computing devices we have today.

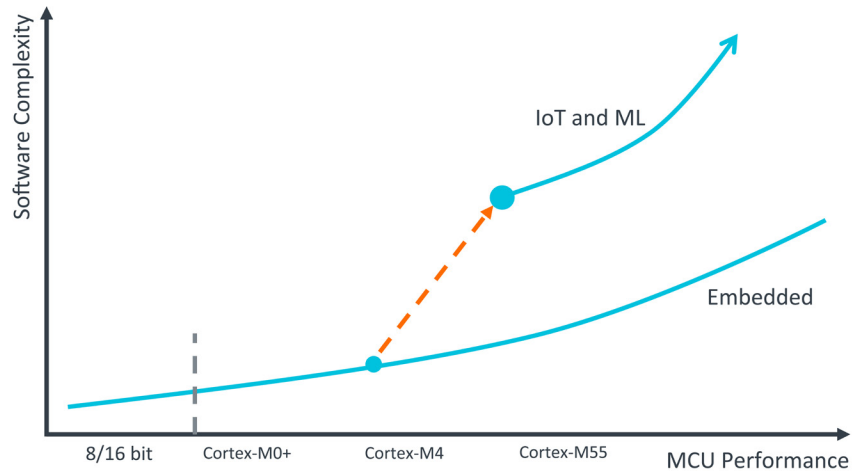
Figure 1: Fifth wave of computing



With the emergence of IoT, we believe the software landscape will change significantly for the embedded space as the requirement to manage devices in the field adds significant complexity to the traditional embedded software stack.

This increase in software complexity and how it impacts IoT software will be the topic for this paper. We will focus on the changes that have been made to [Mbed](#) as it evolves from a relatively straightforward microcontroller development platform to a feature rich IoT operating system, centered on simplifying IoT device development and deployment.

Figure 2: ML and IoT drive a disruptive jump in device complexity



Before the IoT, the main approaches to traditional embedded development were to develop custom bare metal firmware for specific hardware (in C or, in some cases, Assembly) or to use an RTOS kernel, like FreeRTOS or RTX, to get access to a few OS primitives such as threads and mutex. Then, it was a case of developing software to add the required functionalities. Often, it is more cost-effective to adapt existing software components either open source or commercially supported such as the [Keil MDK-Middleware](#) and fine-tuning them to your specific project needs

“The Mbed project began with a singular focus: to simplify software development for IoT.”

The Mbed project began with a singular focus: to simplify the software development for IoT applications running on 32-bit microcontrollers. At the time, 32-bit’s additional computing power was not seen as overly necessary, but the embedded space evolved, and today 32-bit architectures dominate the market. Mbed took a different approach to the traditional embedded development, driven by the belief that there was a developer ecosystem that wanted the simplified approach that a platform operating system brings.

Over the past decade, we believe that this approach has been validated by IoT developers and OEMs shipping products. The number of developers registered with an account on the Mbed website has consistently grown, and we now have more than half a million users. We are not alone in this, as our friends at Arduino have also taken Mbed OS as the foundation to enable 32-bit computing within their simplified approach to embedded development.

“For the IoT, we need to build new functionality on top of the RTOS to address the needs of connected devices.”

An Evolution From RTOS to IoT OS

So, what's the difference between a traditional RTOS application and one which is specialized for the demands of IoT? At a basic level, the fundamental OS services provided by an RTOS such as scheduler, threads, memory management, etc. are similarly available on an IoT OS. For example, Mbed OS leverages the RTOS kernel RTX to deliver those OS services that most embedded software requires.

However, for IoT, we need to build new functionality on top of the RTOS that addresses specific needs. These can be categorized into five main areas:



Connectivity - required to connect the device to cloud services



Security - to protect device integrity and the data on the device when stored and in transit



Storage - to hold data on the device until it needs to be communicated and to store new firmware images



Device management - the services required to manage the device's interaction with the cloud



Machine learning - to perform on-device processing before sending it to the cloud or to quickly react to stimuli

In the past few years, the Mbed team has focused development efforts on adding these fundamentals to the platform to make it a genuine IoT OS.



Connectivity

IoT encompasses a diverse set of use cases that have different bandwidth, power, and accessibility needs. This variety has led to a wide range of connectivity protocols being developed, adapted, and adopted for IoT. Many are new and designed from the outset to support the needs of IoT, with specific features optimized to accommodate low power, often compromising on bandwidth and latency; the following are commonly used in IoT devices.

- + **BLE**
- + **Cellular (2G, 3G, NB-IoT, CAT-M1 or 5G)**
- + **Ethernet**

“An IoT OS brings additional benefits, over-and-above those of a traditional RTOS with regards to connectivity, as the abstraction an IoT OS provides makes porting an application to a different protocol more straightforward.”

- + LoRaWAN
- + NFC
- + Thread
- + Wi-Fi
- + Wi-SUN
- + Zigbee

Connectivity and related optimizations can fundamentally change how the OS behaves. In NB-IoT use, for example, the Mbed team adapted how the OS can be brought out of a deep sleep, allowing the modem to wake it instead of automatically waking periodically to check the modem status, thus saving significant power.

An example such as this can be delivered using an RTOS. Still, we'd argue that an IoT OS brings additional benefits, over-and-above those of a traditional RTOS with regards to connectivity, as the abstraction an IoT OS provides makes porting an application to a different protocol more straightforward. However, there are no technical barriers that could prevent you from using an RTOS for IoT from a connectivity perspective.



Security

Once you start connecting IoT devices to public networks, then the need to secure them becomes essential, ensuring that hostile actors cannot compromise them. There is a wide range of examples of this, from bot attacks like the Mirai malware to state actors targeting critical infrastructure in rival states.

Arm invested heavily in the security features in Mbed OS. This started with the addition of a source of entropy to support a True Random Number Generator (TRNG) to deliver a solid underpinning for on-device cryptography. Additionally, Mbed provides a TLS library that secures data in transit between the device and the cloud. Finally, it incorporates support for the latest generation of Armv8-M architecture with [Arm TrustZone](#) and enables IoT products to comply with [PSA Certified](#). PSA Certified provides a standard framework to assess and certify the security of IoT devices and is an industry-wide initiative supported by Arm in conjunction with key partners.

A platform OS can abstract much of the complexity in these security capabilities, making it simpler for developers to use. Security is an area that often calls for specialist development skills, making it challenging to implement the security services that IoT demands, and historically many Mbed customers lacked this in-house capability. It is possible to achieve a secure system using an RTOS, but there is significant effort in implementing or integrating the capabilities involved piece-by-piece, and an additional challenge can come in maintaining a bespoke security architecture as new vulnerabilities and attacks are uncovered.

“Mbed includes robust and proven storage implementations, making application development that much quicker.”



One of the underlying components that is essential in IoT is a storage subsystem that equips a device with the software required to manage the receipt and storage of new binary images for firmware updates. This storage can use either the internal memory of the MCU or external flash connected via SPI.

This subsystem needs to offer a level of security to ensure that a firmware image cannot be replaced or modified while on the device, usually achieved by encrypting the image itself. The storage area needs to have the capacity to store two images at any given time; the existing software image that is to be replaced and the new, replacement image. Should installation fail, then a rollback mechanism is required to boot the device back up using the original image, thereby ensuring continuous operation.

In Mbed OS, we provide multiple layers of storage that stack to provide the user with the ability to configure the storage to meet their requirements best. Listing the supported interfaces from low-level to high-level:

- + BlockDevice: Access to a raw block-based storage device, such as (Q)SPI flash, SD card, EEPROM, or internal flash
- + FileSystem: POSIX-style file system
- + FileHandle: POSIX-style files.
- + KVStore: Key-value database.

Again, there is no software magic provided by an IoT OS like Mbed that cannot be implemented using a basic RTOS. It's just a case of Mbed including robust and proven implementations of this key IoT requirement, making application development for the adopting developer that much quicker.



Device Management

Another crucial part of the functionality required for IoT is device management through cloud services. This is the ability to deploy a device, commission it remotely, provision it with secure credentials, and then manage it throughout its deployment. Updating the device software as required and decommissioning it at the end of the device's working life are also crucial capabilities that are often overlooked as they have not been necessary for traditional embedded devices. This is where the difference between a traditional RTOS and an IoT OS is most apparent. Management by cloud services also brings new requirements into the embedded space, with protocols like LwM2M and MQTT and management of security credentials, and, most crucially of all, it fundamentally changes the embedded software development lifecycle.

“2020 was an important year for Mbed as we transformed Mbed OS into a cloud-neutral OS by adding more features for developers to connect to any cloud, providing more choice and flexibility.”

Previously, embedded device deployments were relatively simple, as a team would write the software required for their device, ensure through testing and validation that it worked as intended, then install the software on the device and deploy it. Once software was installed on the production line, the responsibility of the software team was complete - essentially, their work was done - as the device would be deployed and used until it was decommissioned or otherwise reached end-of-life. However, IoT brings a whole new area of responsibility to a software team as they need to be able to develop, test, verify and deploy software updates to the device for as long as it is actively deployed, at the very least to ensure security holes are patched in a timely fashion. That lifespan can extend well beyond ten years, and we have already seen customers stipulating long-term support in their requirements specifications.

For the traditional RTOS approach, where a developer brings in the required software components and libraries from multiple sources, integrates and optimizes them for their specific use case, and deploys the software, this long-term support requirement is very challenging. If we took a simple example, where the developer uses an RTOS kernel with an open-source BLE stack to deliver a device that connects to the cloud over a gateway, the maintenance costs of managing a software fork over five years are significant. This approach can even impact new product development: if the solution involves multiple products, it soon leads to a situation where the maintenance team consumes most if not all available software resources. In the mobile space, this is a common situation where some handset vendors struggle to keep upgrading existing devices to new versions of the [Android OS](#).

Beyond this, integrating embedded software with cloud services brings together two areas of software development: the embedded developer and the cloud developer. Previously, these two disciplines have had very little engagement. This new dynamic requires changes to development flows and has significant implications on the tooling required to develop and deploy IoT software successfully.

2020 was an important year for Mbed as we transformed Mbed OS into a cloud-neutral OS by adding more features for developers to connect to any cloud, providing more choice and flexibility. Mbed specifically added support for several leading public cloud services, including integrations for AWS, Google Cloud IoT, and Microsoft Azure IoT.

Mbed Studio's integrated software environment is online and is architected from the ground-up to meet the needs of IoT developers. It spans the embedded software developer needs, such as library management, drag-and-drop programming, and a debug interface. It also services the needs of the web/cloud developer, integrating REST APIs, firmware update packaging, and developer certificates. All neatly packaged in one extremely agile, simple-to-use integrated development environment (IDE).

Silicon vendors traditionally provide an SDK, some also include an IDE, to enable developers to build embedded devices on their MCU. However, what has become abundantly clear during our interactions with them is that IoT-specific workflows are incredibly challenging to support using the standard RTOS SDK approach.



Machine Learning

Not all data acquired from sensors in the field is transmitted to the cloud. There could be many reasons for it, but they can mainly be categorized in three buckets:

- + Delay
- + Transmission cost
- + Privacy

The first reason is the delay introduced when a quick reaction is required: as an example, if we have a safety mechanism on an industrial machine, it would be unwise to rely on the cloud connectivity to receive the signal to interrupt a potentially dangerous motion as a quick reaction is critical for the safety of the operator. The second reason is the cost, in terms of bandwidth and energy, to transmit all data to the cloud: using a similar example of an industrial machine, sending data measurements of multiple high-frequency accelerometers can easily produce Megabytes of data per second which would drain the battery and increase overall costs due to the requirements in bandwidth. Finally, privacy is now a strict requirement for many IoT devices and, if we take the example of a voice-activated assistant, we will not want to transmit everything the microphone captures to the cloud but limiting it only to the part the user wants to.

Although signal processing at the edge is far from being news, the increased capability of devices to run machine learning algorithms enabled the emergence of applications that weren't simply possible before. With a relatively small and low power Cortex-M4 device, it is even possible to perform relatively complex tasks such as recognizing [simple voice commands](#) or [movement](#).

The role of an IoT OS in this context is to provide the basic functionalities to simplify the deployment of ML frameworks such as TensorFlow or TVM. Abstracting some of the hardware specific functionalities required by the ML frameworks enables developers to pick their favorite framework without worrying too much about the hardware support and/or integration on the device.

Collaborating for the future

What started as a single development board, back in 2009, Mbed OS has become an entire ecosystem for IoT development, with a wide range of free development tools, thousands of code examples, and support for more than 150 microcontroller development boards. Mbed has continued to grow and thrive because of the contributions and engagement from our customers and the developer community.

As we move ahead with Mbed OS, we will be building on contributions and engagements from the developer community. With our partners, contributors and developers, we can shape the future of Mbed OS and, together, make the IoT a place where great ideas thrive.

“The role of an IoT OS is to provide the basic functionalities to simplify the deployment of ML frameworks. Abstracting some of the hardware specific functionalities required by the ML frameworks enables developers to deliver in ML without worrying about the hardware support and/or integration on the device.”

Conclusion

In our engagements with developers, partners, and customers, we have seen the challenges that IoT brings to embedded device development. There is widespread frustration that IoT projects are failing at the Proof-of-Concept phase, as that complexity manifests itself. A skilled embedded developer can address the complexities themselves, using an RTOS and hand-crafting software libraries and components, but the time this requires, in terms of development, verification, and on-going maintenance after the device is deployed, increases costs and impacts the viability of products and services.

For IoT to scale seamlessly and to be as genuinely transformational in all the ways we expect, we need to simplify the device-side software complexity and provide the connectivity protocols, the security libraries and services, and the storage. A highly integrated, proven, IoT-specific OS has been validated as an effective way to address this challenge.

Finally, the area that is key to unlocking the value of IoT is tight integration with cloud management services at unlimited scale. Effective integration empowers devices to connect and securely communicate with the business applications that are at the core of IoT. Solutions that facilitate this with minimal friction liberate developers to focus on extracting the maximum business value from IoT.

Resources:

- + [Get started with Mbed OS](#)
- + Join our monthly [Tech Forum](#) to contribute and help shape the future of Mbed OS
- + [Mbed OS on GitHub](#)
- + Share ideas on [new features](#) for Mbed OS



All brand names or product names are the property of their respective holders. Neither the whole nor any part of the information contained in, or the product described in, this document may be adapted or reproduced in any material form except with the prior written permission of the copyright holder. The product described in this document is subject to continuous developments and improvements. All particulars of the product and its use contained in this document are given in good faith. All warranties implied or expressed, including but not limited to implied warranties of satisfactory quality or fitness for purpose are excluded. This document is intended only to provide information to the reader about the product. To the extent permitted by local laws Arm shall not be liable for any loss or damage arising from the use of any information in this document or any error or omission in such information.